Contents

About This Book .......................................................................................................................... vii

CHAPTER 1 Commands ......................................................................................................... 1
   Overview ................................................................................................................................. 1
   alter database ...................................................................................................................... 7
   alter encryption key ........................................................................................................... 13
   alter role .............................................................................................................................. 19
   alter table ........................................................................................................................... 24
   begin...end ......................................................................................................................... 58
   begin transaction ................................................................................................................. 59
   break ................................................................................................................................... 60
   checkpoint ........................................................................................................................... 61
   close ..................................................................................................................................... 63
   commit ................................................................................................................................ 64
   compute clause ................................................................................................................... 66
   connect to...disconnect ................................................................................................... 75
   continue ............................................................................................................................... 78
   create archive database .................................................................................................... 79
   create database ............................................................................................................... 81
   create default .................................................................................................................... 89
   create encryption key ......................................................................................................... 92
   create existing table ......................................................................................................... 95
   create function .................................................................................................................. 101
   create function (SQLJ) ..................................................................................................... 104
   create index ....................................................................................................................... 107
   create plan ........................................................................................................................ 127
   create procedure ............................................................................................................... 129
   create procedure (SQLJ) .................................................................................................. 141
   create proxy_table ........................................................................................................... 145
   create role ........................................................................................................................ 149
   create rule ........................................................................................................................ 152
   create schema .................................................................................................................. 156
   create service .................................................................................................................... 158
   create table ....................................................................................................................... 163
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>create trigger</td>
<td>206</td>
</tr>
<tr>
<td>create view</td>
<td>221</td>
</tr>
<tr>
<td>dbcc</td>
<td>230</td>
</tr>
<tr>
<td>deallocate cursor</td>
<td>250</td>
</tr>
<tr>
<td>declare</td>
<td>251</td>
</tr>
<tr>
<td>declare cursor</td>
<td>253</td>
</tr>
<tr>
<td>delete</td>
<td>260</td>
</tr>
<tr>
<td>delete statistics</td>
<td>268</td>
</tr>
<tr>
<td>disk init</td>
<td>270</td>
</tr>
<tr>
<td>disk mirror</td>
<td>276</td>
</tr>
<tr>
<td>disk refit</td>
<td>280</td>
</tr>
<tr>
<td>disk reinit</td>
<td>281</td>
</tr>
<tr>
<td>disk remirror</td>
<td>285</td>
</tr>
<tr>
<td>disk resize</td>
<td>287</td>
</tr>
<tr>
<td>disk unmirror</td>
<td>289</td>
</tr>
<tr>
<td>drop database</td>
<td>292</td>
</tr>
<tr>
<td>drop default</td>
<td>294</td>
</tr>
<tr>
<td>drop encryption key</td>
<td>295</td>
</tr>
<tr>
<td>drop function</td>
<td>296</td>
</tr>
<tr>
<td>drop function (SQLJ)</td>
<td>297</td>
</tr>
<tr>
<td>drop index</td>
<td>298</td>
</tr>
<tr>
<td>drop procedure</td>
<td>300</td>
</tr>
<tr>
<td>drop role</td>
<td>302</td>
</tr>
<tr>
<td>drop rule</td>
<td>304</td>
</tr>
<tr>
<td>drop service</td>
<td>305</td>
</tr>
<tr>
<td>drop table</td>
<td>306</td>
</tr>
<tr>
<td>drop trigger</td>
<td>309</td>
</tr>
<tr>
<td>drop view</td>
<td>310</td>
</tr>
<tr>
<td>dump database</td>
<td>311</td>
</tr>
<tr>
<td>dump transaction</td>
<td>330</td>
</tr>
<tr>
<td>execute</td>
<td>346</td>
</tr>
<tr>
<td>fetch</td>
<td>353</td>
</tr>
<tr>
<td>goto label</td>
<td>359</td>
</tr>
<tr>
<td>grant</td>
<td>360</td>
</tr>
<tr>
<td>group by and having clauses</td>
<td>390</td>
</tr>
<tr>
<td>if...else</td>
<td>403</td>
</tr>
<tr>
<td>insert</td>
<td>406</td>
</tr>
<tr>
<td>kill</td>
<td>416</td>
</tr>
<tr>
<td>load database</td>
<td>418</td>
</tr>
<tr>
<td>load transaction</td>
<td>433</td>
</tr>
<tr>
<td>lock table</td>
<td>445</td>
</tr>
<tr>
<td>mount</td>
<td>447</td>
</tr>
<tr>
<td>online database</td>
<td>452</td>
</tr>
<tr>
<td>open</td>
<td>455</td>
</tr>
</tbody>
</table>
Contents

order by clause................................................................. 456
prepare transaction ....................................................... 462
print ............................................................................... 463
quiesce database .......................................................... 466
raiserror ....................................................................... 470
readtext ......................................................................... 475
reconfigure ................................................................. 480
remove java.................................................................. 481
reorg............................................................................ 483
return ............................................................................ 486
revoke .......................................................................... 489
rollback ........................................................................ 504
rollback trigger............................................................ 506
save transaction ............................................................ 507
select .............................................................................. 509
set ................................................................................ 538
set encryption passwd .................................................. 585
setuser ......................................................................... 588
shutdown ...................................................................... 590
truncate table .............................................................. 594
union operator ............................................................. 596
unmount ....................................................................... 600
update .......................................................................... 602
update all statistics ....................................................... 614
update index statistics .................................................. 616
update statistics .......................................................... 619
update table statistics .................................................... 625
use ............................................................................ 627
waitfor .......................................................................... 628
where clause ............................................................... 630
while ............................................................................. 637
writetext ...................................................................... 639

CHAPTER 2  Interactive SQL Commands ........................................... 643
  clear ........................................................................ 644
  configure ............................................................... 645
  connect .................................................................. 646
  disconnect ............................................................. 649
  exit .......................................................................... 650
  input ....................................................................... 651
  output ..................................................................... 656
  parameters ............................................................ 661
  read ....................................................................... 662
  set connection ....................................................... 664
Contents

set option................................................................. 665
start logging...................................................... 666
stop logging....................................................... 667
system............................................................... 668

Index ................................................................................................. 669
About This Book

The *Adaptive Server Reference Manual* includes four guides to Sybase® Adaptive Server® Enterprise and the Transact-SQL® language:

- *Building Blocks* describes the “parts” of Transact-SQL: datatypes, built-in functions, global variables, expressions and identifiers, reserved words, and SQLSTATE errors. Before you can use Transact-SQL successfully, you must understand what these building blocks do and how they affect the results of Transact-SQL statements.

- *Commands* (this book) provides reference information about the Transact-SQL commands, which you use to create statements.

- *Procedures* provides reference information about system procedures, catalog stored procedures, extended stored procedures, and `dbcc` stored procedures. All procedures are created using Transact-SQL statements.

- *Tables* is reference information about the system tables, which store information about your server, databases, users, and other details of your server. It also provides information about the tables in the `dbccdb` and `dbccalt` databases.

**Audience**

The *Adaptive Server Reference Manual* is intended as a reference tool for Transact-SQL users of all levels.

**How to use this book**

- Chapter 1, “Commands,” lists the Adaptive Server commands in a table that provides the name and a brief description. Each command name is listed in Table 1-1 on page 1.

- Chapter 2, “Interactive SQL Commands” describes Interactive SQL commands, which are entered in top pane of the Interactive SQL display. These commands are intended only for Interactive SQL and are not sent to Adaptive Server for execution.

Complex commands, such as `select`, are divided into subsections. For example, there are reference pages on the `compute` clause and on the `group by` and `having` clauses of the `select` command.

**Related documents**

The Adaptive Server Enterprise documentation set consists of the following:

Reference Manual: Commands
• The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Product Manuals Web site.

• The Installation Guide for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.

• What’s New in Adaptive Server Enterprise? – describes the new features in Adaptive Server version 15.0, the system changes added to support those features, and changes that may affect your existing applications.

• ASE Replicator User’s Guide – describes how to use the Adaptive Server Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.

• Component Integration Services User’s Guide – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.

• The Configuration Guide for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.

• Enhanced Full-Text Search Specialty Data Store User’s Guide – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.

• Glossary – defines technical terms used in the Adaptive Server documentation.


• Java in Adaptive Server Enterprise – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.

• Job Scheduler User’s Guide – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
• *Messaging Service User’s Guide* – describes how to use Real Time Messaging Services to integrate TIBCO Java Message Service and IBM WebSphere MQ messaging services with all Adaptive Server database applications.


• *Performance and Tuning Series* – a series of books that explain how to tune Adaptive Server for maximum performance:
  
  • *Basics* – the basics for understanding and investigating performance questions in Adaptive Server.
  
  • *Locking andConcurrency Control* – describes how the various locking schemas can be used for improving performance in Adaptive Server, and how to select indexes to minimize concurrency.
  
  • *Query Processing and Abstract Plans* – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.
  
  • *Physical Database Tuning* – describes how to manage physical data placement, space allocated for data, and the temporary databases.
  
  • *Monitoring Adaptive Server with sp_sysmon* – describes how to monitor Adaptive Server’s performance with `sp_sysmon`.
  
  • *Improving Performance with Statistical Analysis* – describes how Adaptive Server stores and displays statistics, and how to use the `set statistics` command to analyze server statistics.
  
  • *Using the Monitoring Tables* – describes how to query Adaptive Server’s monitoring tables for statistical and diagnostic information.
  
• *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, data types, and utilities in a pocket-sized book (regular size when viewed in PDF format).

• *Reference Manual* – is a series of four books that contains the following detailed Transact-SQL information:

  • *Building Blocks* – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
• **Commands** – Transact-SQL commands.

• **Procedures** – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.

• **Tables** – Transact-SQL system tables and dbcc tables.

• **System Administration Guide** –
  - **Volume 1** – provides an introduction to the basics of system administration, including a description of configuration parameters, resource issues, character sets, sort orders, and diagnosing system problems. The second part of this book is an in-depth description of security administration.
  - **Volume 2** – includes instructions and guidelines for managing physical resources, mirroring devices, configuring memory and data caches, managing multiprocessor servers and user databases, mounting and unmounting databases, creating and using segments, using the reorg command, and checking database consistency. The second half of this book describes how to back up and restore system and user databases.

• **System Tables Diagram** – illustrates system tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.

• **Transact-SQL User’s Guide** – documents Transact-SQL, the Sybase enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.

• **Troubleshooting Series** (for release 15.0) –
  - **Troubleshooting: Error Messages Advanced Resolutions** – contains troubleshooting procedures for problems that you may encounter when using Sybase Adaptive Server Enterprise. The problems addressed here are those which the Sybase Technical Support staff hear about most often
• **Troubleshooting and Error Messages Guide** – contains detailed instructions on how to resolve the most frequently occurring Adaptive Server error messages. Most of the messages presented here contain error numbers (from the master.sysmessages table), but some error messages do not have error numbers, and occur only in Adaptive Server’s error log.

• **User Guide for Encrypted Columns** – describes how configure and use encrypted columns with Adaptive Server

• **Using Adaptive Server Distributed Transaction Management Features** – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

• **Using Sybase Failover in a High Availability System** – provides instructions for using Sybase Failover to configure an Adaptive Server as a companion server in a high availability system.

• **Unified Agent and Agent Management Console** – describes the Unified Agent, which provides runtime services to manage, monitor and control distributed Sybase resources.

• **Utility Guide** – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.

• **Web Services User’s Guide** – explains how to configure, use, and troubleshoot Web Services for Adaptive Server.

• **XA Interface Integration Guide for CICS, Encina, and TUXEDO** – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.

• **XML Services in Adaptive Server Enterprise** – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

• The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
• The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the SyBooks Installation Guide on the Getting Started CD, or the README.txt file on the SyBooks CD for instructions on installing and starting SyBooks.

• The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications
on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click Certification Report.

3 In the Certification Report filter select a product, platform, and timeframe and then click Go.

4 Click a Certification Report title to display the report.

❖ Finding the latest information on component certifications

1 Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.
Creating a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

2. Click MySybase and create a MySybase profile.

Finding the latest information on EBFs and software maintenance

2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.
   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following sections describe conventions used in this manual.

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and most syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented. Complex commands are formatted using modified Backus Naur Form (BNF) notation.

Table 1 shows the conventions for syntax statements that appear in this manual:
### Table 1: Font and syntax conventions for this manual

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command names, procedure names, utility names, database names, datatypes, and other keywords display in sans serif font.</td>
<td><code>select sp_configure</code></td>
</tr>
<tr>
<td>Book names, file names, variables, and path names are in italics.</td>
<td><em>System Administration Guide</em></td>
</tr>
<tr>
<td></td>
<td><em>sql.ini file</em></td>
</tr>
<tr>
<td></td>
<td><em>column_name</em></td>
</tr>
<tr>
<td></td>
<td><em>SYBASE/ASE directory</em></td>
</tr>
<tr>
<td>Variables—or words that stand for values that you fill in—when they are part of a query or statement, are in italics in Courier font.</td>
<td><code>select column_name from table_name where search_conditions</code></td>
</tr>
<tr>
<td>Type parentheses as part of the command.</td>
<td><code>compute row_aggregate (column_name)</code></td>
</tr>
<tr>
<td>Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.</td>
<td>::=</td>
</tr>
<tr>
<td>Braces mean that you must choose at least one of the enclosed options. Do not type the braces.</td>
<td><code>{cash, check, credit}</code></td>
</tr>
<tr>
<td>Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.</td>
<td>`[cash</td>
</tr>
<tr>
<td>The comma means you may choose as many of the options shown as you want. Separate your choices with commas as part of the command.</td>
<td><code>cash, check, credit</code></td>
</tr>
<tr>
<td>The pipe or vertical bar (</td>
<td>) means you may select only one of the options shown.</td>
</tr>
<tr>
<td>An ellipsis (...) means you can repeat the last unit as many times as you like.</td>
<td>`buy thing = price [cash</td>
</tr>
</tbody>
</table>

- Syntax statements (displaying the syntax and all options for a command) appear as follows:
  ```
  sp_dropdevice [device_name]
  ```
- For a command with more options:
  ```
  select column_name from table_name where search_conditions
  ```
In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

  ```sql
  select * from publishers
  ```

- Examples of output from the computer appear as follows:

  pub_id | pub_name               | city   | state
  ------ | -----------------------|--------|--------
  0736   | New Age Books          | Boston | MA     
  0877   | Binnet & Hardley       | Washington | DC
  1389   | Algodata Infosystems   | Berkeley | CA

  (3 rows affected)

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, `SELECT`, `Select`, and `select` are the same.

Adaptive Server sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see the *System Administration Guide*.

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Adaptive Server HTML documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

**Note** You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.
Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.
CHAPTER 1

Commands

This volume describes commands, clauses, and other elements used to construct a Transact-SQL statement.

Overview

Table 1-1 provides a brief description of the commands in this chapter.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter database on page 7</td>
<td>Increases the amount of space allocated to a database. Allowed with an archive database.</td>
</tr>
<tr>
<td>alter encryption key on page 13</td>
<td>Changes the default property of key. Creates and drops key copies for users. Changes owner of key. Recovers lost passwords.</td>
</tr>
<tr>
<td>alter role on page 19</td>
<td>Defines mutually exclusive relationships between roles, adds, drops, and changes passwords for roles.</td>
</tr>
<tr>
<td>alter table on page 24</td>
<td>Adds new columns; adds, changes, or drops constraints, changes constraints; partitions or unpartitions an existing table. Creates computed columns. Encrypts or decrypts existing data. Adds encrypted columns. Adds, drops, or replaces decrypt defaults for encrypted columns. Changes the encryption key on an encrypted column.</td>
</tr>
<tr>
<td>begin...end on page 58</td>
<td>Encloses a series of SQL statements so that control-of-flow language, such as if...else, can affect the performance of the whole group.</td>
</tr>
<tr>
<td>begin transaction on page 59</td>
<td>Marks the starting point of a user-defined transaction.</td>
</tr>
<tr>
<td>break on page 60</td>
<td>Causes an exit from a while loop. break is often activated by an if test.</td>
</tr>
<tr>
<td>checkpoint on page 61</td>
<td>Writes all dirty pages (pages that have been updated since they were last written) to the database device. Allowed with an archive database.</td>
</tr>
<tr>
<td>close on page 63</td>
<td>Deactivates a cursor.</td>
</tr>
<tr>
<td>commit on page 64</td>
<td>Marks the ending point of a user-defined transaction.</td>
</tr>
<tr>
<td>compute clause on page 66</td>
<td>Generates summary values that appear as additional rows in the query results.</td>
</tr>
<tr>
<td>connect to...disconnect on page 75</td>
<td>Specifies the server to which a passthrough connection is required.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>continue on page 78</td>
<td>Causes the while loop to restart. continue is often activated by an if test.</td>
</tr>
<tr>
<td>create archive database on page 79</td>
<td>Creates an archive database.</td>
</tr>
<tr>
<td>create database on page 81</td>
<td>Creates a new database.</td>
</tr>
<tr>
<td>create default on page 89</td>
<td>Specifies a value to insert in a column (or in all columns of a user-defined datatype) if no value is explicitly supplied at insert time.</td>
</tr>
<tr>
<td>create encryption key on page 92</td>
<td>Creates a named key for column encryption.</td>
</tr>
<tr>
<td>create existing table on page 95</td>
<td>Confirms that the current remote table information matches the information that is stored in column_list, and verifies the existence of the underlying object.</td>
</tr>
<tr>
<td>create function on page 101</td>
<td>Creates a user-defined function, which is a saved Transact-SQL routine that returns a specified value.</td>
</tr>
<tr>
<td>create function (SQLJ) on page 104</td>
<td>Creates a user-defined function by adding a SQL wrapper to a Java static method.</td>
</tr>
<tr>
<td>create index on page 107</td>
<td>Creates an index on one or more columns in a table. Creates a function-based index.</td>
</tr>
<tr>
<td>create plan on page 127</td>
<td>Creates an abstract query plan.</td>
</tr>
<tr>
<td>create procedure on page 129</td>
<td>Creates a stored procedure that can take one or more user-supplied parameters.</td>
</tr>
<tr>
<td>create procedure (SQLJ) on page 141</td>
<td>Creates a SQLJ stored procedure by adding a SQL wrapper to a Java static method.</td>
</tr>
<tr>
<td>create proxy_table on page 145</td>
<td>Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.</td>
</tr>
<tr>
<td>create role on page 149</td>
<td>Creates a user-defined role.</td>
</tr>
<tr>
<td>create rule on page 152</td>
<td>Specifies the domain of acceptable values for a particular column or for any column of a user-defined datatype.</td>
</tr>
<tr>
<td>create schema on page 156</td>
<td>Creates a new collection of tables, views, and permissions for a database user.</td>
</tr>
<tr>
<td>create service on page 158</td>
<td>Wraps the supplied SQL statement in a stored procedure with the specified name and parameters.</td>
</tr>
<tr>
<td>create table on page 163</td>
<td>Creates new tables and optional integrity constraints. Creates a table containing computed columns and function-based indexes. Creates a table containing encrypted columns and decrypt defaults.</td>
</tr>
<tr>
<td>create trigger on page 206</td>
<td>Creates a trigger, which is a type of stored procedure often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.</td>
</tr>
<tr>
<td>create view on page 221</td>
<td>Creates a view, which is an alternative way of looking at the data in one or more tables.</td>
</tr>
<tr>
<td>dbcc on page 230</td>
<td>Checks the logical and physical consistency of a database. Use dbcc regularly as a periodic check or if you suspect any damage.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deallocate cursor on page 250</td>
<td>Makes a cursor inaccessible and releases all memory resources committed to that cursor. Allowed with an archive database.</td>
</tr>
<tr>
<td>declare on page 251</td>
<td>Declares the name and type of local variables for a batch or procedure.</td>
</tr>
<tr>
<td>declare cursor on page 253</td>
<td>Defines a cursor. Allowed with an archive database.</td>
</tr>
<tr>
<td>delete on page 260</td>
<td>Removes rows from a table.</td>
</tr>
<tr>
<td>delete statistics on page 268</td>
<td>Removes statistics from the sysstatistics system table.</td>
</tr>
<tr>
<td>disk init on page 270</td>
<td>Makes a physical device or file usable by Adaptive Server.</td>
</tr>
<tr>
<td>disk mirror on page 276</td>
<td>Creates a software mirror that immediately takes over when the primary device fails.</td>
</tr>
<tr>
<td>disk refit on page 280</td>
<td>Rebuilds the master database’s sysusages and sysdatabases system tables from information contained in sysdevices. Use disk refit after disk reinit as part of the procedure to restore the master database.</td>
</tr>
<tr>
<td>disk reinit on page 281</td>
<td>Rebuilds the master database’s sysdevices system table. Use disk reinit as part of the procedure to restore the master database.</td>
</tr>
<tr>
<td>disk remirror on page 285</td>
<td>Reenables disk mirroring after it is stopped by failure of a mirrored device or temporarily disabled by the disk unmirror command.</td>
</tr>
<tr>
<td>disk resize on page 287</td>
<td>Dynamically increases the size of database devices.</td>
</tr>
<tr>
<td>disk unmirror on page 289</td>
<td>Disables either the original device or its mirror, allowing hardware maintenance or the changing of a hardware device.</td>
</tr>
<tr>
<td>drop database on page 292</td>
<td>Removes one or more databases from an Adaptive Server. Allowed with an archive database.</td>
</tr>
<tr>
<td>drop default on page 294</td>
<td>Removes a user-defined default.</td>
</tr>
<tr>
<td>drop encryption key on page 295</td>
<td>Removes an encryption key and its key copies.</td>
</tr>
<tr>
<td>drop function on page 296</td>
<td>Removes one or more user-defined functions from the current database.</td>
</tr>
<tr>
<td>drop function (SQLJ) on page 297</td>
<td>Removes a SQLJ function.</td>
</tr>
<tr>
<td>drop index on page 298</td>
<td>Removes an index from a table in the current database.</td>
</tr>
<tr>
<td>drop procedure on page 300</td>
<td>Removes user-defined stored procedures.</td>
</tr>
<tr>
<td>drop role on page 302</td>
<td>Removes a user-defined role.</td>
</tr>
<tr>
<td>drop rule on page 304</td>
<td>Removes a user-defined rule.</td>
</tr>
<tr>
<td>drop service on page 305</td>
<td>Removes a user-defined Web service from the current database. Both the metadata and the corresponding stored procedure are removed.</td>
</tr>
<tr>
<td>drop table on page 306</td>
<td>Removes a table definition and all of its data, indexes, triggers, and permission specifications from the database.</td>
</tr>
<tr>
<td>drop trigger on page 309</td>
<td>Removes a trigger.</td>
</tr>
</tbody>
</table>
### Overview

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop view on page 310</td>
<td>Removes one or more views from the current database.</td>
</tr>
<tr>
<td>dump database on page 311</td>
<td>Makes a backup copy of the entire database, including the transaction log, in a form that can be read in with load database. Dumps and loads are performed through Backup Server.</td>
</tr>
<tr>
<td>dump transaction on page 330</td>
<td>Makes a copy of a transaction log and removes the inactive portion.</td>
</tr>
<tr>
<td>execute on page 346</td>
<td>Runs a system procedure, a user-defined stored procedure, or a dynamically constructed Transact-SQL command. Allowed with an archive database.</td>
</tr>
<tr>
<td>fetch on page 353</td>
<td>Returns a row or a set of rows from a cursor result set. In scrollable cursors, fetch orientation keywords specify the position of the row to fetch. Allowed with an archive database.</td>
</tr>
<tr>
<td>goto label on page 359</td>
<td>Branches to a user-defined label.</td>
</tr>
<tr>
<td>grant on page 360</td>
<td>Assigns permissions to users or to user-defined roles.</td>
</tr>
<tr>
<td>group by and having clauses on page 390</td>
<td>Used in select statements to divide a table into groups and to return only groups that match conditions in the having clause.</td>
</tr>
<tr>
<td>if...else on page 403</td>
<td>Imposes conditions on the execution of a SQL statement.</td>
</tr>
<tr>
<td>insert on page 406</td>
<td>Adds new rows to a table or view.</td>
</tr>
<tr>
<td>kill on page 416</td>
<td>Kills a process.</td>
</tr>
<tr>
<td>load database on page 418</td>
<td>Loads a backup copy of a user database, including its transaction log. Allowed with an archive database.</td>
</tr>
<tr>
<td>lock table on page 445</td>
<td>Explicitly locks a table within a transaction. Allowed with an archive database.</td>
</tr>
<tr>
<td>mount on page 447</td>
<td>Attaches the database to the destination or secondary Adaptive Server.</td>
</tr>
<tr>
<td>online database on page 452</td>
<td>Marks a database available for public use after a normal load sequence and, if needed, upgrades a loaded database and transaction log dumps to the current version of Adaptive Server. Allowed with an archive database.</td>
</tr>
<tr>
<td>open on page 455</td>
<td>Opens a cursor for processing.</td>
</tr>
<tr>
<td>order by clause on page 456</td>
<td>Returns query results in the specified columns in sorted order.</td>
</tr>
<tr>
<td>prepare transaction on page 462</td>
<td>Used by DB-Library™ in a two-phase commit application to see if a server is prepared to commit a transaction.</td>
</tr>
<tr>
<td>print on page 463</td>
<td>Prints a user-defined message on the user’s screen.</td>
</tr>
<tr>
<td>quiesce database on page 466</td>
<td>Suspends and resumes updates to a specified list of databases.</td>
</tr>
<tr>
<td>raiserror on page 470</td>
<td>Prints a user-defined error message on the user’s screen and sets a system flag to record that an error condition has occurred.</td>
</tr>
<tr>
<td>readtext on page 475</td>
<td>Reads text, unitext, and image values, starting from a specified offset and reading a specified number of bytes or characters. Allowed with an archive database.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reconfigure on page 480</td>
<td>Currently has no effect; included to allow existing scripts to run without modification. In earlier versions, you were required to execute reconfigure after sp_configure, to implement new configuration parameter settings.</td>
</tr>
<tr>
<td>remove java on page 483</td>
<td>Removes one or more Java-QL classes, packages, or JARs from a database. Use when Java is enabled in the database.</td>
</tr>
<tr>
<td>reorg on page 486</td>
<td>Reclaims unused space on pages, removes row forwarding, or rewrites all rows in the table to new pages, depending on the option used.</td>
</tr>
<tr>
<td>return on page 489</td>
<td>Exits from a batch or procedure unconditionally, optionally providing a return status. Statements following return are not executed.</td>
</tr>
<tr>
<td>revoke on page 489</td>
<td>Revokes permissions or roles from users or roles.</td>
</tr>
<tr>
<td>rollback on page 504</td>
<td>Rolls a user-defined transaction back to the last savepoint inside the transaction or to the beginning of the transaction.</td>
</tr>
<tr>
<td>rollback trigger on page 506</td>
<td>Rolls back the work done in a trigger, including the update that caused the trigger to fire, and issues an optional raiserror statement.</td>
</tr>
<tr>
<td>save transaction on page 507</td>
<td>Sets a savepoint within a transaction.</td>
</tr>
<tr>
<td>select on page 509</td>
<td>Retrieves rows from database objects. Allowed with an archive database.</td>
</tr>
<tr>
<td>set on page 538</td>
<td>Sets Adaptive Server query-processing options for the duration of the user’s work session. Can be used to set some options inside a trigger or stored procedure. Can also be used to activate or deactivate a role in the current session.</td>
</tr>
<tr>
<td>set encryption passwd on page 586</td>
<td>Sets a password for a named key or column at the session level.</td>
</tr>
<tr>
<td>setuser on page 588</td>
<td>Allows a Database Owner to impersonate another user.</td>
</tr>
<tr>
<td>shutdown on page 590</td>
<td>Shuts down Adaptive Server or a Backup Server™. This command can be issued only by a System Administrator.</td>
</tr>
<tr>
<td>truncate table on page 594</td>
<td>Removes all rows from a table.</td>
</tr>
<tr>
<td>union operator on page 596</td>
<td>Returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the all keyword is specified.</td>
</tr>
<tr>
<td>unmount on page 600</td>
<td>Shuts down the database and drops it from the Adaptive Serve, and deactivates and drops devices.</td>
</tr>
<tr>
<td>update on page 602</td>
<td>Changes data in existing rows, either by adding data or by modifying existing data; updates all statistics information for a given table; updates information about the number of pages in each partition for a partitioned table; updates information about the distribution of key values in specified indexes.</td>
</tr>
<tr>
<td>update all statistics on page 614</td>
<td>Updates all statistics information for a given table.</td>
</tr>
<tr>
<td>update index statistics on page 616</td>
<td>Updates the statistics for all columns in an index.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>update statistics on page 619</strong></td>
<td>Updates information about the distribution of key values in specified indexes, for all columns in an index, table, or partition.</td>
</tr>
<tr>
<td><strong>update table statistics on page 625</strong></td>
<td>Updates statistics that are stored in systabstats table, such as rowcount, cluster ratios, and so on.</td>
</tr>
<tr>
<td><strong>use on page 627</strong></td>
<td>Specifies the database with which you want to work. Allowed with an archive database.</td>
</tr>
<tr>
<td><strong>waitfor on page 628</strong></td>
<td>Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.</td>
</tr>
<tr>
<td><strong>where clause on page 630</strong></td>
<td>Sets the search conditions in a select, insert, update, or delete statement.</td>
</tr>
<tr>
<td><strong>while on page 637</strong></td>
<td>Sets a condition for the repeated execution of a statement or statement block. Statements execute repeatedly, as long as the specified condition is true.</td>
</tr>
<tr>
<td><strong>writetext on page 639</strong></td>
<td>Permits nonlogged, interactive updating of an existing text, unitext, or image column.</td>
</tr>
</tbody>
</table>
alter database

Description
Increases the amount of space allocated to a database, as well as to the modified pages section of an archive database.

Syntax
```
alter database database_name
   [on {default | database_device} [= size]]
   [, database_device [= size]]...
   [log on {default | database_device} [= size]]
   [, database_device [= size]]...
   [with override]
   [for load]
   [for proxy_update]
```

Parameters
- **database_name**
  - is the name of the database. The database name can be a literal, a variable, or a stored procedure parameter.

- **on**
  - indicates a size and location for the database extension. If you have your log and data on separate device fragments, use this clause for the data device and the log on clause for the log device.

- **default**
  - indicates that alter database can put the database extension on any default database devices (as shown by the `sp_helpdevice` stored procedure in Chapter 1, “System Procedures,” in Reference Manual: Procedures). To specify a size for the database extension without specifying the exact location, use:

  ```
  on default = size
  ```

  To change a database device’s status to default, use `sp_diskdefault`.

- **database_device**
  - is the name of the database device on which to locate the database extension. A database can occupy more than one database device with different amounts of space on each. Add database devices to Adaptive Server with `disk init`. 

The `alter database` command is used to allocate additional space to the database extension. The amount of space to allocate is specified by the `size` parameter. The `size` parameter can use various unit specifiers, such as ‘k’ or “K” (kilobytes), “m” or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends always including a unit specifier. If a unit specifier is not included, the value is presumed to be in megabytes.

If no size is specified, `alter database` extends the database by 1MB or 4 allocation units, whichever is larger. The table below describes the minimum amounts:

<table>
<thead>
<tr>
<th>Server’s logical page size</th>
<th>Database extended by</th>
</tr>
</thead>
<tbody>
<tr>
<td>2K</td>
<td>1MB</td>
</tr>
<tr>
<td>4K</td>
<td>1MB</td>
</tr>
<tr>
<td>8K</td>
<td>2MB</td>
</tr>
<tr>
<td>16K</td>
<td>4MB</td>
</tr>
</tbody>
</table>

### Examples

#### Example 1
Adds 1MB to a 2K-page-size database `mydb` on a default database device:

```
alter database mydb
```

#### Example 2
Adds 3MB to the space allocated for the `pubs2` database on the database device named `newdata`:

```
alter database pubs2 size '3M'
```
alter database pubs2 on newdata = 3

Example 3 Adds 10MB of space for data on userdata1 and 2MB for the log on logdev:

    alter database production
    on userdata1 = "10M"
    log on logdev = '2.5m'

Usage

Using alter database for archive databases

You can use alter database to add space to the modified pages section of the archive database at any time, not only when space runs out. Increasing the space in the modified pages section allows a suspended command to resume operation. The syntax is:

    alter database database_name
    [ on database_device [= size]
    [. database_device [= size]]...]

Restrictions

• Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.

• Adaptive Server reports an error if the total size of all fixed-length columns, plus the row overhead, is greater than the table’s locking scheme and page size allows.

• Because Adaptive Server allocates space for databases for create database and alter database in chunks of 256 logical pages, these commands round the specified size down to the nearest multiple of allocation units.

• You can specify the size as a float datatype, however, the size is rounded down to the nearest multiple of the allocation unit.

• Although Adaptive Server does create tables in the following circumstances, you will receive errors about size limitations when you perform data manipulation language operations:
  • If the length of a single variable-length column exceeds the maximum column size.
  • For DOL tables, if the offset of any variable-length column other than the initial column exceeds the limit of 8191 bytes.

• If Adaptive Server cannot allocate the requested space, it comes as close as possible per device and prints a message telling how much space has been allocated on each database device.
You must be using the master database, or executing a stored procedure in the master database, to use alter database.

You can expand the master database only on the master device. An attempt to use alter database to expand the master database to any other database device results in an error message. Here is an example of the correct statement for modifying the master database on the master device:

```sql
alter database master on master = 1
```

Each time you allocate space on a database device with create database or alter database, that allocation represents a device fragment, and the allocation is entered as a row in sysusages.

If you use alter database on a database that is in the process of being dumped, the alter database command cannot complete until the dump finishes. Adaptive Server locks the in-memory map of database space use during a dump. If you issue an alter database command while this in-memory map is locked, Adaptive Server updates the map from the disk after the dump completes. If you interrupt alter database, Adaptive Server instructs you to run sp_dbremap. If you do not run sp_dbremap, the space you added does not become available to Adaptive Server until the next time you restart the server.

You can use alter database on database_device on an offline database.

Back up master after allocating more space

- Back up the master database with dump database after each use of alter database. This makes recovery easier and safer in case master becomes damaged.

- If you use alter database and do not back up master, you may be able to recover the changes with disk refit.

Placing the log on a separate device

- To increase the amount of storage space allocated for the transaction log when you have used the log on extension to create database, give the name of the log’s device in the log on clause when you issue the alter database command.

- If you did not use the log on extension of create database to place your logs on a separate device, you may not be able to recover fully in case of a hard disk crash. In this case, you can extend your logs by using alter database with the log on clause, then using sp_logdevice to move the log to its own devices.
Getting help on space usage

- To see the names, sizes, and usage of device fragments already in use by a database, execute `sp_helpdb dbname`.
- To see how much space the current database is using, execute `sp_spaceused`.

The system and default segments

- The system and default segments are mapped to each new database device included in the on clause of an alter database command. To unmap these segments, use `sp_dropsegment`.
- When you use `alter database` (without `override`) to extend a database on a device already in use by that database, the segments mapped to that device are also extended. If you use the `override` clause, all device fragments named in the on clause become system/default segments, and all device fragments named in the log on clause become log segments.

Using alter database to awaken sleeping processes

- If user processes are suspended because they have reached a last-chance threshold on a log segment, use `alter database` to add space to the log segment. The processes awaken when the amount of free space exceeds the last-chance threshold.

Using for proxy_update

- If the `for proxy_update` clause is entered with no other options, the size of the database is not extended; instead, the proxy tables, if any, are dropped from the proxy database and re-created from the metadata obtained from the path name specified during `create database ... with default_location = 'pathname'`.
- If this command is used with other options to extend the size of the database, the proxy table synchronization is performed after the size extensions are made.
- The purpose of this `alter database` extension is to provide the Database Administrator with an easy-to-use, single-step operation with which to obtain an accurate and up-to-date proxy representation of all tables at a single remote site.
Resynchronization is supported for all external data sources, not just the primary server in a HA-cluster environment. Also, a database need not have been created with the for proxy_update clause. If a default storage location has been specified, either through the create database command or with sp_defaultloc, the metadata contained within the database can be synchronized with the metadata at the remote storage location.

To make sure databases are synchronized correctly so that all the proxy tables have the correct schema to the content of the primary database you just reloaded, you may need to run the for proxy_update clause on the server hosting the proxy database.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
alter database permission defaults to the Database Owner. System Administrators can also alter databases.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>alter</td>
<td>alter database</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – alter size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if a set proxy is in effect</td>
</tr>
</tbody>
</table>

See also

Commands  
create database, disk init, drop database, load database

System procedures  
sp_adsegment, sp_dropsegment, sp_helpdb, sp_helpsegment, sp_logdevice, sp_renamedb, sp_spaceused
alter encryption key

Description
Changes the current password for an encryption key.

For more information about encrypted columns, see the Encrypted Column Users Guide.

Syntax
```
alter encryption key [[database.][owner.]] keyname
   { [ as | not default ]
      | [ with passwd 'password' | system_encr_passwd | login_passwd ]
      modify encryption
      { [ with passwd 'password' | system_encr_passwd | login_passwd ]
          add encryption [ with passwd 'password' ]
          for user user_name
          [ for login_association | for recovery ]
      | drop encryption for
      { user user_name | key recovery }
      | [ with passwd 'password'
          recover encryption with passwd 'password'
      | modify owner user_name
   }
```

Parameters
- `keyname` is the name for a column encryption key.
- `as [not] default` indicates that the database default property should be assigned to, or unassigned from, this key.
with passwd ['password' | system_encr_passwd | login_password]
specifies the current password Adaptive Server uses to decrypt the column encryption key, and a new password for one of the following purposes:

- Modify the encryption of a key or a key copy. The key owner can add key copies for individual users that are accessible through a private password or a login password.
- Encrypt a newly-added key copy
- Recover the encryption key after losing a password

Adaptive Server supports the following of passwords for keys:

- password – a character string up to 255 bytes long.
- login_passwd – tells Adaptive Server to use the session’s login password.
- system_encr_passwd – is the system encryption password for the current database.

If you do not specify with passwd, the default is system_encr_passwd.

modify encryption
indicates you are modifying the encryption key or key copy.

add encryption
adds encrypted key copy for a designated user.

for user user_name
specifies the user for whom you are adding or dropping a key copy.

for login_association
indicates that the key copy being added is encrypted by the assigned user’s login password during his or her first access to this key.

for recovery
indicates this key copy is for recovery purposes.

drop encryption
indicates that you are dropping the key copy for the specified user.

recover encryption
makes the base key accessible to a new password.

When you create a key using create encryption key, Adaptive Server saves the key in encrypted form, along with the key’s properties, as a row in sysencryptkeys. This row represents the base key. The key owner can choose to allow access to encrypted data exclusively through the base key.
modify owner
  changes the key’s owner to the specified user.

Examples

Example 1 Changes my_key to the default encryption key:

    alter encryption key my_key as default

You must have the sso_role or keycustodian_role to change the default
property of a key. If the command above is executed by:

• The system security officer, Adaptive Server removes the default property
  unconditionally from the previous default key, if one exists.
• The key custodian, he or she must own my_key. The key custodian must
  own the previous default key, if one exists.

To remove the default property from my_key, the SSO or the key custodian as
owner of the key, executes:

    alter encryption key my_key as not default

If my_key is not the default key, this command returns an error.

Example 2 Changes the password to the important_key encryption key:

    alter encryption key important_key with passwd 'oldpassword'
    modify encryption with passwd 'newpassword'

If this command is executed by:

• The key owner – the command re-encrypts the base key
• The user assigned a key copy – the command re-encrypts that key copy.

Example 3 Changes the password on a key copy to the current session’s login
password:

    alter encryption key important_key
    modify encryption
    with passwd login_passwd

This command can be executed only by a user who has been assigned a key
copy:

Example 4 Changes the password for the important_key encryption key to the
system password:

    alter encryption key important_key
    with passwd 'ReallyBigSecret'
    modify encryption with passwd system_encr_passwd
This command can be executed only by the key owner or a user with sso_role, and is allowed only if a key has no key copies. It modifies the encryption of the base key.

**Example 5** Changes the password for the `important_key` encryption key from the system encryption password to a new password (because the system encryption password is the default password, it does not need to be specified in the statement):

```sql
alter encryption key important_key
  modify encryption
    with passwd 'ReallyNewPassword'
```

**Example 6** Adds encryption for user “ted” for the `important_key` encryption key with the password `just4now`:

```sql
alter encryption key important_key
  with passwd 'TopSecret'
  add encryption with passwd 'just4now'
    for user 'ted'
```

You must be a key owner or a user with the sso_role to execute this command. Adaptive Server uses the password “TopSecret” to decrypt the base key, making a copy of the raw key and encrypting it for user “ted” using the password “just4now.”

**Example 7** Modifies the encryption for user “ted” to use a new password. Only “ted” can execute this command:

```sql
alter encryption key important_key
  with passwd 'just4now'
  modify encryption
    with passwd 'TedsOwnPassword'
```

**Example 8** Drops encryption for user “ted” for the `important_key` encryption key (you must have the sso_role or be the key owner to execute this command):

```sql
alter encryption key important_key
  drop encryption for user 'ted'
```

**Example 9** Modifies the owner of `important_key` to new owner, “tinnap” (you must have the sso_role or be the key owner to execute this command):

```sql
alter encryption key important_key modify owner tinnap
```

**Example 9** Sets up the recovery key copy and use it for key recovery after losing a password:

1. The key custodian originally creates a new encryption key protected by a password.
create encryption key key1 for AES passwd 'loseitl8ter'

2 The key custodian adds a special encryption key recovery copy for key1 for user “charlie.”

```sql
alter encryption key key1 with passwd 'loseitl8ter'
    add encryption
    with passwd 'temppasswd'
    for user charlie
    for recovery
```

3 “charlie” assigns a different password to the recovery copy and saves this password in a locked drawer:

```sql
alter encryption key key1
    with passwd 'temppasswd'
    modify encryption
    with passwd 'finditl8ter'
    for recovery
```

4 If the key custodian loses the password for base key, he can obtain the password from “charlie” and recover the base key from the recovery copy of the key using:

```sql
alter encryption key key1
    with passwd 'finditl8ter'
    recover encryption
    with passwd 'newpasswd'
```

Usage

- If the SSO issues `alter encryption key` to set the key as the database default, the specified key replaces any existing key as the default.

- Keys are owned and managed by users with `keycustodian_role` or by users explicitly granted permission for the `create encryption key` command. Keys are used by all users who have permissions to process and see the data from encrypted columns. How Adaptive Server protects keys affects how they are accessed:

  - The key owner creates the key for encryption by the system encryption password – When users access the encrypted data, Adaptive Server decrypts the base key using the system encryption password. The key owner does not create individual key copies for users.

  - The key custodian encrypts the base key with an explicit password – Rather than create key copies, the key custodian shares this password with all users who process encrypted data. Users or applications must supply this password with the `set encryption passwd` to access data. See set encryption passwd.
If the key custodian issues `alter encryption key` to set a key as the database default, the specified key and the current default key (if it exists) must be owned by the key custodian.

If you do not include the `with passwd` parameter with `alter encryption`, Adaptive Server uses the system-encryption password.

You cannot use the system-encryption password to alter the base key of a key that has copies, and you cannot encrypt copies of keys with the system encryption password.

Users implicitly modify only their own key-copies.

If you specify `for login_association`, Adaptive Server temporarily encrypts the key copy with the system encryption password.

You cannot specify `for recovery` and `login_association` for the same key copy.

Permissions You must be:

- The system security officer or a user with the `keycustodian_role` to execute `alter encryption key` as default or not default. This permission cannot be granted to other users.
- The system security officer or the key owner to use `alter encryption key` to add or drop key copies, recover the key, and modify the key owner.
- The system security officer or the key owner to execute `alter encryption key` to modify the password of the base key. You must be the user assigned the key copy to modify the key copy password. You implicitly have permission to modify your own key copy.

Auditing For information about auditing encrypted columns, see Chapter 6, “Auditing Encrypted Columns,” in the *Encrypted Columns Users Guide*.

See also `create encryption key`, `drop encryption key`, and `sp_encryption`.
**alter role**

**Description**

Defines mutually exclusive relationships between roles; adds, drops, and changes passwords for roles; specifies the password expiration interval, the minimum password length, and the maximum number of failed logins allowed for a specified role. alter role is also used to lock and unlock roles.

**Syntax**

```sql
alter role role1 {add | drop} exclusive
   (membership | activation) role2

alter role role_name [add passwd "password" | drop passwd] [lock | unlock]

alter role {role_name | "all overrides"}
   set {passwd expiration | min passwd length | max failed_logins} option_value
```

**Parameters**

- **role1**
  is one role in a mutually exclusive relationship.
  - **add**
    adds a role in a mutually exclusive relationship; adds a password to a role.
  - **drop**
    drops a role in a mutually exclusive relationship; drops a password from a role.
  - **exclusive**
    makes both named roles mutually exclusive.
  - **membership**
    does not allow you to grant users both roles at the same time.
  - **activation**
    allows you to grant a user both roles at the same time, but does not allow the user to activate both roles at the same time.

- **role2**
  is the other role in a mutually exclusive relationship.

- **role_name**
  is the name of the role for which you want to add, drop, or change a password. Use `role_name` to specify the password expiration interval, the minimum password length, and the maximum number of failed logins.

- **passwd**
  adds or drops a password to a role.
**password**

is the password to add to a role. You cannot use variables for passwords. For rules on passwords, see Chapter 14, “Managing Adaptive Server Logins, Database Users, and Client Connections,” in the System Administration Guide, Volume 1.

**lock**

locks the specified role.

**unlock**

unlocks the specified role.

**all overrides**

applies the setting that follows to the entire server rather than to a specific role.

**set**

activates the option that follows it.

**passwd expiration**

specifies the password expiration interval in days. It can be any value between 0 and 32767, inclusive.

**min passwd length**

specifies the minimum length allowed for the specified password.

**max failed logins**

specifies the maximum number of failed login attempts allowed for the specified password.

**option_value**

specifies the value for **passwd expiration**, **min passwd length**, or **max failed logins**. To set all overrides, set the value of **option_value** to -1.

---

**Examples**

**Example 1** Defines **intern_role** and **specialist_role** as mutually exclusive at the membership level:

```
alter role intern_role add exclusive membership
specialist_role
```

**Example 2** Defines roles as mutually exclusive at the membership level and at the activation level:

```
alter role specialist_role add exclusive membership
intern_role
alter role intern_role add exclusive activation
surgeon_role
```

**Example 3** Adds a password to an existing role:
alter role doctor_role add passwd "physician"

**Example 4** Drops a password from an existing role:

```
alter role doctor_role drop passwd
```

**Example 5** Locks the role physician_role:

```
alter role physician_role lock
```

**Example 6** Unlocks the role physician_role:

```
alter role physician_role unlock
```

**Example 7** Changes the maximum number of failed logins allowed for physician_role to 5:

```
alter role physician_role set max failed_logins 5
```

**Example 8** Sets the minimum password length for physician_role, an existing role, to five characters:

```
alter role physician_role set min passwd length 5
```

**Example 9** Overrides the minimum password length of all roles:

```
alter role "all overrides" set min passwd length -1
```

**Example 10** Removes the overrides for the maximum failed logins for all roles:

```
alter role "all overrides" set max failed_logins -1
```

---

**Usage**

- The `alter role` command defines mutually exclusive relationships between roles and adds, drops, and changes passwords for roles.
- The `all overrides` parameter removes the system overrides that were set using `sp_configure` with any of the following parameters:
  - `passwd expiration`
  - `max failed_logins`
  - `min passwd length`
- Dropping the role password removes the overrides for the password expiration and the maximum failed logins options.

**Mutually exclusive roles**

- You need not specify the roles in a mutually exclusive relationship or role hierarchy in any particular order.
- You can use mutual exclusivity with role hierarchy to impose constraints on user-defined roles.
Mutually exclusive membership is a stronger restriction than mutually exclusive activation. If you define two roles as mutually exclusive at membership, they are implicitly mutually exclusive at activation.

If you define two roles as mutually exclusive at membership, defining them as mutually exclusive at activation has no effect on the membership definitions. Mutual exclusivity at activation is added and dropped independently of mutual exclusivity at membership.

You cannot define two roles as mutually exclusive property after granting both roles to users or roles. Revoke either granted role from existing grantees before attempting to define the roles as mutually exclusive at the membership level.

If two roles are defined as mutually exclusive at activation, the System Security Officer can assign both roles to the same user, but the user cannot activate both roles at the same time.

If the System Security Officer defines two roles as mutually exclusive at activation, and users have already activated both roles or, by default, have set both roles to activate at login, Adaptive Server makes the roles mutually exclusive, but issues a warning message naming specific users with conflicting roles. The users’ activated roles do not change.

Changing passwords for roles

To change the password for a role, first drop the existing password, then add the new password, as follows:

```
alter role doctor_role drop passwd
alter role doctor_role add passwd "physician"
```

Note Passwords that existed before Adaptive Server version 12.x and that attached to user-defined roles do not expire.

---

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Only a System Security Officer can execute alter role.

Auditing
Values in event and extrainfo columns of sysaudits are:
### References

#### Commands

- **create role**
- **drop role**
- **grant**
- **revoke**
- **set**

#### Documents

For more information on altering roles, see the *System Administration Guide*.

#### Functions

- `mut_excl_roles`
- `proc_role`
- `role_contain`
- `role_id`
- `role_name`

#### System procedures

- `sp_activeroles`
- `sp_displaylogin`
- `sp_displayroles`
- `sp_modifylogin`

### Event Audit Option

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 85    | roles        | create role, drop role, alter role, grant role, or revoke role | - *Roles* – Current active roles  
- *Keywords or options* – NULL  
- *Previous value* – NULL  
- *Current value* – NULL  
- *Other information* – NULL  
- *Proxy information* – Original login name, if set proxy in effect |

See also

**Commands**  
create role, drop role, grant, revoke, set

**Documents**  
For more information on altering roles, see the *System Administration Guide*.

**Functions**  
`mut_excl_roles`, `proc_role`, `role_contain`, `role_id`, `role_name`

**System procedures**  
`sp_activeroles`, `sp_displaylogin`, `sp_displayroles`, `sp_modifylogin`
**alter table**

**Description**

- Adds new columns to a table; drops or modifies existing columns; adds, changes, or drops constraints; changes properties of an existing table; enables or disables triggers on a table.

- Supports adding, dropping, and modifying computed columns and to enable the materialized property, nullability, or definition of an existing computed column to be changed.

- Partitions and repartitions a table with specified partition strategy, or add partitions to a table with existing partitions. Syntax for altering table partitions is listed separately. See the `alter table` syntax for partitions.

**Syntax**

```sql
alter table [[database.][owner.]]table_name  
  {add column_name datatype  
  [default {constant_expression | user | null}]
  [identity | null | not null]
  [off row | in row]
  [[constraint constraint_name]  
  {[unique | primary key]  
  [clustered | nonclustered]
  [asc | desc]
  [with {fillfactor = pct,  
      max_rows_per_page = num_rows,  
      reservepagegap = num_pages}]
  [on segment_name]  
  | references [[database.][owner.]]ref_table  
  {[ref_column]}  
  [match full]  
  | check (search_condition)  
  [encrypt [with key_name] [decrypt_default value]],  
  [, next_column]...  
  | add {[constraint constraint_name]  
  {[unique | primary key]  
  [clustered | nonclustered]
  (column_name [asc | desc], column_name [asc | desc]...)}
  [with {fillfactor = pct,  
      max_rows_per_page = num_rows,  
      reservepagegap = num_pages}]
  [on segment_name]  
  | foreign key (column_name [, column_name]...)  
  references [[database.][owner.]]ref_table  
  {[ref_column] [, ref_column]...}  
  [match full]  
  | check (search_condition)  
  | drop {column_name [, column_name]...  
  | constraint constraint_name}  
  | modify column_name  
  [encrypt [with key_name] [decrypt_default value]],}
```

24

Adaptive Server Enterprise
CHAPTER 1  Commands

datatype [null | not null]

| next_column]
| replace column_name
| default (constant_expression | user | null)
| decrypt_default (constant_expression | null)
| drop decrypt_default
| lock {allpages | datarows | datapages} )
| with exp_row_size=num_bytes
| partition number_of_partitions
| unpartition
| partition_clause
| add_partition_clause

alter table syntax for partitions:

partition_clause::=
partition by range (column_name[, column_name]...)
((partition_name) values <= (constant | MAX)
[, (constant | MAX)] ...) [on segment_name]
| [partition_name) values <= (constant | MAX)
[, (constant | MAX)] ...) [on segment_name]...

 partition by hash (column_name[, column_name]...)
{ (partition_name) [on segment_name]
[, partition_name [on segment_name]]...}
| number_of_partitions
| [on (segment_name[, segment_name] ...)]

| partition by list (column_name)
((partition_name) values (constant[, constant] ...)
[on segment_name]
| [partition_name) values (constant[, constant] ...)
| [on segment_name]...)

| partition by roundrobin
{ (partition_name) [on segment_name]
[, partition_name [on segment_name]]...}
| number_of_partitions
| [on (segment_name[, segment_name] ...)]

add_partition_clause::=
add partition
{ (partition_name) values <= (constant | MAX)
[, (constant | MAX)]...)
[on segment_name]
| [partition_name) values <= (constant | MAX)
[, (constant | MAX)] ...) [on segment_name]...

| (partition_name) values (constant[, constant] ...)
[on segment_name]
alter table

[, [partition_name] values (constant[, constant] ...)
[on segment_name]] ...)}

alter table syntax for computed columns

alter table
  add column_name {compute | as}
  computed_column_expression...
  [materialized | not materialized]
drop column_name
modify column_name {null | not null |
  {materialized | not materialized} [null | not null] | 
  {compute | as} computed_column_expression
  [materialized | not materialized]
  [null | not null]}

alter table syntax for dropping partitions

alter table table_name drop partition
  partition_name [, partition_name]...

Parameters

 table_name
  is the name of the table to change. Specify the database name if the table is
  in another database, and specify the owner’s name if more than one table of
  that name exists in the database. The default value for owner is the current
  user, and the default value for database is the current database.

add
  specifies the name of the column or constraint to add to the table. If
  Component Integration Services is enabled, you cannot use add for remote
  servers.

column_name
  is the name of a column in that table. If Java is enabled in the database, the
  column can be a Java-SQL column.

datatype
  is any system datatype except bit or any user-defined datatype except those
  based on bit.

If Java is enabled in the database, datatype can be the name of a Java class
installed in the database, either a system class or a user-defined class. See
Java in Adaptive Server Enterprise for more information.
default

specifies a default value for a column. If you specify a default and the user
does not provide a value for this column when inserting data, Adaptive
Server inserts this value. The default can be a constant_expression, user (to
insert the name of the user who is inserting the data), or null (to insert the null
value).

Adaptive Server generates a name for the default in the form of
tabname_colname_objid, where tabname is the first 10 characters of the table
name, colname is the first 5 characters of the column name, and objid is the
object ID number for the default. Setting the default to null drops the default.

If Component Integration Services is enabled, you cannot use default for
remote servers.

constant_expression

is a constant expression to use as a default value for a column. It cannot
include global variables, the name of any columns, or other database objects,
but can include built-in functions. This default value must be compatible
with the datatype of the column.

user

specifies that Adaptive Server should insert the user name as the default if
the user does not supply a value. The datatype of the column must be
char(30), varchar(30), or a type that Adaptive Server implicitly converts to
char; however, if the datatype is not char(30) or varchar(30), truncation may
occur.

null | not null

specifies the Adaptive Server behavior during data insertion if no default
exists.

null specifies that a column is added that allows nulls. Adaptive Server
assigns a null value during inserts if a user does not provide a value.

The properties of a bit-type column must always be not null.

not null specifies that a column is added that does not allow nulls. Users must
provide a non-null value during inserts if no default exists.

If you do not specify null or not null, Adaptive Server uses not null by default.
However, you can switch this default using sp_dboption to make the default
compatible with the SQL standards. If you specify (or imply) not null for the
newly added column, a default clause is required. The default value is used
for all existing rows of the newly added column, and applies to future inserts
as well.
encrypt [with key_name]  
 specifies an encrypted column and the key used to encrypt it. You may use encrypt [with key_name] on columns with any system datatype other than text and image.

decrypt_default constant_expression  
 specifies that this column returns a default value for users who do not have decrypt permissions, and constant_expression is the value Adaptive Server returns on select statements instead of the encrypted value. The value can be NULL on nullable columns only. If the decrypt_value cannot be converted to the column’s data type, Adaptive Server catches the conversion error only when it executes the query.

identity  
 indicates that the column has the IDENTITY property. Each table in a database can have one IDENTITY column with a datatype of:

- Exact numeric and scale of 0, or
- Any of the integer datatypes, including signed or unsigned bigint, int, smallint, or tinyint.

IDENTITY columns are not updatable and do not allow nulls.

IDENTITY columns store sequential numbers, such as invoice numbers or employee numbers that are automatically generated by Adaptive Server. The value of the IDENTITY column uniquely identifies each row in a table.

off row | in row  
 specifies whether the Java-SQL column is stored separately from the row, or in storage allocated directly in the row.

The storage for an in row column cannot exceed 16K bytes, depending on the page size of the database server and other variables. The default value is off row.

constraint  
 introduces the name of an integrity constraint. If Component Integration Services is enabled, you cannot use constraint for remote servers.
constraint_name

is the name of the constraint. It must conform to the rules for identifiers and be unique in the database. If you do not specify the name for a table-level constraint, Adaptive Server generates a name in the form of 
tabname_colname_objectid, where tabname is the first 10 characters of the table name, colname is the first 5 characters of the column name, and objectid is the object ID number for the constraint. If you do not specify the name for a unique or primary key constraint, Adaptive Server generates a name in the format tabname_colname_tabindid, where tabindid is a string concatenation of the table ID and index ID.

Constraints do not apply to the data that already exists in the table at the time the constraint is added.

unique

constrains the values in the indicated column or columns so that no two rows can have the same non-null value. This constraint creates a unique index that can be dropped only if the constraint is dropped. You cannot use this option along with the null option described above.

primary key

constrains the values in the indicated column or columns so that no two rows can have the same value and so that the value cannot be NULL. This constraint creates a unique index that can be dropped only if the constraint is dropped.

classed | nonclustered

specifies that the index created by a unique or primary key constraint is a clustered or nonclustered index. clustered is the default (unless a clustered index already exists for the table) for primary key constraints; nonclustered is the default for unique constraints. There can be only one clustered index per table. See create index for more information.

asc | desc

specifies whether the index is to be created in ascending (asc) or descending (desc) order. The default is ascending order.
with fillfactor=pct

specifies how full to make each page when Adaptive Server creates a new index on existing data. “pct” stands for percentage. The fillfactor percentage is relevant only when the index is created. As data changes, pages are not maintained at any particular level of fullness.

**Warning!** Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

The default for fillfactor is 0; this is used when you do not include with fillfactor in the create index statement (unless the value has been changed with sp_configure). When specifying a fillfactor, use a value between 1 and 100.

A fillfactor of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes. There is seldom a reason to change the fillfactor.

If the fillfactor is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A fillfactor of 100 makes sense only for read-only tables—tables to which no data will ever be added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A fillfactor of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small fillfactor values cause each index (or index and data) to take more storage space.
max_rows_per_page = num_rows
limits the number of rows on data pages and the leaf-level pages of indexes. Unlike fillfactor, the max_rows_per_page value is maintained until it is changed with sp_chgattribute.

If you do not specify a value for max_rows_per_page, Adaptive Server uses a value of 0 when creating the index. When specifying max_rows_per_page for data pages, use a value between 0 and 256. The maximum number of rows per page for nonclustered indexes depends on the size of the index key; Adaptive Server returns an error message if the specified value is too high.

For indexes created by constraints, a max_rows_per_page setting of 0 creates clustered indexes with full pages and nonclustered indexes with full leaf pages. A setting of 0 leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes.

If max_rows_per_page is set to 1, Adaptive Server creates both clustered and nonclustered leaf index pages with one row per page at the leaf level. You can use this to reduce lock contention on frequently accessed data.

Low max_rows_per_page values cause Adaptive Server to create new indexes with pages that are not completely full, use more storage space, and may cause more page splits.

Warning! Creating a clustered index with max_rows_per_page can affect the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

reservepagegap = num_pages
specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations for the index created by the constraint. For each specified num_pages, an empty page is left for future expansion of the table. Valid values are 0 – 255. The default value, 0, leaves no empty pages.
on segment_name

specifies the segment on which the index exists or is to be placed. When using on segment_name, the logical device must already have been assigned to the database with create database or alter database, and the segment must have been created in the database with sp_addsegment. See your System Administrator or use sp_helpsegment for a list of the segment names available in your database.

If you specify clustered and use the on segment_name option, the entire table migrates to the segment you specify, since the leaf level of the index contains the actual data pages.

When using on segment_name for partitions, it specifies the segment on which to place the partition.

references

specifies a column list for a referential integrity constraint. You can specify only one column value for a column constraint. By including this constraint with a table that references another table, any data inserted into the referencing table must already exist in the referenced table.

To use this constraint, you must have references permission on the referenced table. The specified columns in the referenced table must be constrained by a unique index (created by either a unique constraint or a create index statement). If no columns are specified, there must be a primary key constraint on the appropriate columns in the referenced table. Also, the datatypes of the referencing table columns must exactly match the datatype of the referenced table columns.

If Component Integration Services is enabled, you cannot use references for remote servers.

foreign key

specifies that the listed columns are foreign keys in this table whose matching primary keys are the columns listed in the references clause.

ref_table

is the name of the table that contains the referenced columns. You can reference tables in another database. Constraints can reference up to 192 user tables and internally generated worktables. Use the system procedure sp_helpconstraint to check a table’s referential constraints.

ref_column

is the name of the column or columns in the referenced table.
match full
  specifies that if all values in the referencing columns of a referencing row are:
  - Null – the referential integrity condition is true.
  - Non-null values – if there is a referenced row where each corresponding column is equal in the referenced table, then the referential integrity condition is true.

If they are neither, then the referential integrity condition is false when:
  - All values are non-null and not equal, or
  - Some of the values in the referencing columns of a referencing row are non-null values, while others are null.

check
  specifies a search_condition constraint that Adaptive Server enforces for all the rows in the table. If Component Integration Services is enabled, you cannot use check for remote servers.

search_condition
  is a Boolean expression that defines the check constraint on the column values. These constraints can include:
  - A list of constant expressions introduced with in
  - A set of conditions, which may contain wildcard characters, introduced with like
    An expression can include arithmetic operations and Transact-SQL functions. The search_condition cannot contain subqueries, aggregate functions, parameters, or host variables.

next_column
  includes additional column definitions (separated by commas) using the same syntax described for a column definition.

drop
  specifies the name of a column or constraint to drop from the table. If Component Integration Services is enabled, you cannot use drop for remote servers.

modify
  specifies the name of the column whose datatype or nullability you are changing.
**alter table**

**replace**

specifies the column whose default value you want to change with the new value specified by a following `default` clause. If Component Integration Services is enabled, you cannot use `replace` for remote servers.

**enable | disable trigger**

enables or disables a trigger. For more information, see the *System Administration Guide*.

**lock datarows | datapages | allpages**

changes the locking scheme to be used for the table.

**with exp_row_size=num_bytes**

specifies the expected row size. You can only apply this parameter:

- To datarows and datapages locking schemes.
- To tables with variable-length rows.
- When `alter table` performs a data copy, such as with `alter table add` or `modify`. You cannot use `with exp_row_size=num_bytes` with `alter table lock change` operations.

Valid values are 0, 1, and any value between the minimum and maximum row length for the table. The default value is 0, which means a server-wide setting is applied.

**partition number_of_partitions**

adds `(number_of_partitions - 1)` empty partitions to an unpartitioned table (round-robin partitioned table with a single partition). Thus, the total number of partitions for the table becomes `number_of_partitions`. Even if Component Integration Services (CIS) is enabled, you cannot use `partition` for remote servers.

**unpartition**

changes a round-robin partitioned table without indexes, to an unpartitioned table. Even if CIS is enabled, you cannot use `unpartition` for remote servers.

**partition by range**

specifies records are to be partitioned according values in the partitioning column or columns. Each partitioning column value is compared with sets of user-supplied upper and lower bounds to determine partition assignment.

**column_name**

when used in the `partition_clause`, specifies a partition key column. A partition key column cannot be an encrypted column.
partition_name
specifies the name of a new partition on which table records are to stored. Partition names must be unique within the set of partitions on a table or index. Partition names can be delimited identifiers if set quoted_identifier is on. Otherwise, they must be valid identifiers.

If partition_name is omitted, Adaptive Server creates a name in the form table_name_partition_id. Adaptive Server truncates partition names that exceed the allowed maximum length.

values <= constant | MAX
specifies the inclusive upper bound of values for a named partition. Specifying a constant value for the highest partition bound imposes an implicit integrity constraint on the table. The keyword MAX specifies the maximum value in a given datatype.

on segment_name
when used in the partition_clause, specifies the segment on which the partition is to be placed. When using on segment_name, the logical device must already have been assigned to the database with create database or alter database, and the segment must have been created in the database with sp_addsegment. See your System Administrator or use sp_helpsegment for a list of the segment names available in your database.

partition by hash
specifies records are to be partitioned by a system-supplied hash function. The function computes the hash value of the partition keys that specify the partition to which records are assigned.

partition by list
specifies records are to be partitioned according to literal values specified in the named column. The partition key contains only one column. You can list up to 250 constants as the partition values for each list partition.

partition by round-robin
specifies records are to be partitioned in a sequential manner. A round-robin partitioned table has no partitioning key. Neither the user nor the optimizer knows in which partition a particular record resides.

add partition
applies only to range- or list-partitioned tables:

* For range-partitioned tables – adds one or more partitions to the upper end of a range partitioned table.
* For list-partitioned tables – adds one or more partitions with a new set of values.
**compute** | **as**

adds or drops a new computed column. Follow the same rules defined for the existing create table command and the alter table add rules.

**computed_column_expression**
is any valid T-SQL expression that does not contain columns from other tables, local variables, aggregate functions, or subqueries. It can be one or a combination of column name, constant, function, global variable, or case expression, connected by one or more operators. You cannot cross-reference between computed columns except when virtual computed columns reference materialize computed columns. An encrypted column cannot be referenced in a **computed_column_expression**.

**materialized** | **not materialized**
reserved keywords in the modify clause that specify whether the computed column is materialized, or physically stored in the table. By default, a computed column is **not materialized** (that is, not physically stored in the table). You can also use this piece of syntax to change the definitions of existing virtual computed columns; that is, to materialize them.

**table_name** **drop partition partition_name [, partition_name]...**
drops one or more list or range partitions. You cannot use alter table to drop a hash or round-robin partition.

For each partition you drop, Adaptive Server:

- Deletes all data on the partition.
- Deletes the partition definition from the system catalog.
- Drops all corresponding local index partitions that refer to this data partition.
- Regenerates the partition condition object of the base table and each local index.
- Deletes all statistics information on this partition.
- Rebuilds all global indexes.

**Note** If you attempt to drop a partition from a table that is referenced by another table, and the partition to be dropped and the referencing table are not empty, the command fails because of possible violations with the foreign-key constraint, and Adaptive Server displays error message 13971.

---

**Examples**

**Example 1** Adds a column to a table. For each existing row in the table, Adaptive Server assigns a NULL column value:
alter table publishers
add manager_name varchar (40) null

Example 2 Adds an IDENTITY column to a table. For each existing row in the table, Adaptive Server assigns a unique, sequential column value. The IDENTITY column could be type numeric or integer, and a scale of zero. The precision determines the maximum value ($10^5 - 1$, or 99,999) that can be inserted into the column:

```sql
alter table sales_daily
add ord_num numeric (5,0) identity
```

Example 3 Adds a primary key constraint to the authors table. If there is an existing primary key or unique constraint on the table, you must drop the existing constraint first (see next example):

```sql
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
```

Example 4 Drops the au_identification constraint:

```sql
alter table authors
drop constraint au_identification
```

Example 5 Creates an index on authors; the index has a reservepagegap value of 16, leaving 1 empty page in the index for each 15 allocated pages:

```sql
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
with reservepagegap = 16
```

Example 6 Removes the default constraint on the phone column in the authors table. If the column allows NULL values, NULL is inserted if no column value is specified. If the column does not allow NULL values, an insert that does not specify a column value fails:

```sql
alter table authors
replace phone default null
```

Example 7 Modifies the emp table to encrypt the ssn column and specifies decrypt default:

```sql
alter table emp modify ssn encrypt with key1
decrypt_default '000-00-0000'
```

Example 8 Decrypts credit card data that is longer sensitive:

```sql
alter table stolen_ccards
decrypt ccard
```
If card was encrypted by a key protected by a user-defined password, precede
this command with the set encryption key command.

**Example 9** Adds an encrypted column to an existing table. Because keyname
is omitted, Adaptive Server looks for the database default encryption key:

```sql
alter table sales_mgr
    add bonus money null encrypt
```

**Example 10** Creates an encryption key and encrypt ssn column in existing
employee table.

```sql
set encryption passwd '4evermore' for key ssn_key
alter table employee modify ssn
    encrypt with ssn_key
```

If ssn in the above example were an existing encrypted column encrypted by
‘key1’ the alter table would cause Adaptive Server to decrypt ssn using ‘key1’
and re-encrypt ssn using ‘ssn_key’.

**Example 11** Adds a decrypt default to the salary column, which is already
encrypted:

```sql
alter table employee replace salary
    decrypt_default $0.00
```

**Example 12** Removes the decrypt default for salary without removing the
encryption property:

```sql
alter table employee replace salary drop
    decrypt_default
```

**Example 13** Changes an unpartitioned table to a range-partitioned table with
three partitions, each of which is on a different segment:

```sql
alter table titles partition by range (total_sales)
    (smallsales values <= (500) on seg1,
     mediumsales values <= (5000) on seg2,
     bigsales values <= (25000) on seg3)
```

**Example 14** Adds another range partition to the titles table:

```sql
alter table titles add partition
    (vbigsales values <= (40000) on seg4)
```

**Example 15** Changes the locking scheme for the titles table to datarows
locking:

```sql
alter table titles lock datarows
```

**Example 16** Adds the not-null column author_type to the authors table with a
default of primary_author:
alter table authors
  add author_type varchar (20)
  default "primary_author" not null

Example 17  Drops the advance, notes, and contract columns from the titles table:

    alter table titles
    drop advance, notes, contract

Example 18  Modifies the city column of the authors table to be a varchar(30) with a default of NULL:

    alter table authors
    modify city varchar (30) null

Example 19  Modifies the stor_name column of the stores table to be NOT NULL. Its datatype, varchar(40), remains unchanged:

    alter table stores
    modify stor_name not null

Example 20  Modifies the type column of the titles table and changes the locking scheme of the titles table from allpages to datarows:

    alter table titles
    modify type varchar (10)
    lock datarows

Example 21  Modifies the notes column of the titles table from varchar(200) to varchar(150), changes the default value from NULL to NOT NULL, and specifies an exp_row_size of 40:

    alter table titles
    modify notes varchar (150) not null
    with exp_row_size = 40

Example 22  Adds, modifies, and drops a column, and then adds another column in one query. Alters the locking scheme and specifies the exp_row_size of the new column:

    alter table titles
    add author_type varchar (30) null
    modify city varchar (30)
    drop notes
    add sec_advance money default 1000 not null
    lock datarows
    with exp_row_size = 40

Example 23  Add a virtual computed column:
**alter table**

```sql
alter table authors
  add fullname compute au_fname + ' ' + au_lname
```

**Example 24** Changes a virtual computed column to a materialized computed column:

```sql
alter table authors modify fullname materialized
```

**Usage**

- Before you add, modify, or drop columns on a table, run the `sp_depends` stored procedure to see if there are any stored procedures that depend on the table you are changing. If such stored procedures exist, Sybase recommends that you drop, then re-create the stored procedures as necessary after changing table schema.

- If stored procedures using `select *` reference a table that has been altered, no new columns appear in the result set, even if you use the `with recompile` option. You must drop the procedure and re-create it to include these new columns. Otherwise, the wrong results can be caused by the `insert...select` statement of `insert into table1 select * from table2` in the procedure when the tables have been altered and new columns have been added to the tables.

- When the table owner uses `alter table`, Adaptive Server disables access rules during the execution of the command and enables them upon completion of the command. The access rules are disabled to avoid filtering of the table data during `alter table`.

- If you specify `clustered and use the on segment_name` option, the entire table migrates to the segment you specify, since the leaf level of the index contains the actual data pages.

- `alter table` performs error checking for check constraints before it alters the table.

- When using `on segment_name` for partitions, the logical device must already have been assigned to the database with `create database` or `alter database`, and the segment must have been created in the database with `sp_addsegment`. See your System Administrator or use `sp_helpsegment` for a list of the segment names available in your database.

**Restrictions**

**Warning!** Do not alter the system tables.

- You cannot add a column of datatype `bit` to an existing table if you specify a default value. This default value must be 0 or 1.

- The maximum number of columns in a table is:
• 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables
• 254 for variable-length columns in an APL table
• 1024 for variable-length columns in a DOL table
• `alter table` raises an error if the number of variable-length columns in an APL table exceeds 254.
• All compiled objects should be dropped and recreated after changing a table’s lock schema.
• The maximum length for in-row Java columns is determined by the maximum size of a variable-length column for the table’s schema, locking style, and page size.
• When converting a table to a different locking scheme, the data in the source table cannot violate the limits of the target table. For example, if you attempt to convert a DOL table with more than 254 variable-length columns to an APL table, `alter table` fails because an APL table is restricted to having no more than 254 variable-length columns.
• Columns with fixed-length data (for example `char`, `binary`, and so on) have the maximum sizes shown in Table 1-2:

<table>
<thead>
<tr>
<th>Locking scheme</th>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL tables</td>
<td>2K (2048 bytes)</td>
<td>1962</td>
<td>1960 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4010</td>
<td>4008 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8106</td>
<td>8104 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16298</td>
<td>16296 bytes</td>
</tr>
<tr>
<td>DOL tables</td>
<td>2K (2048 bytes)</td>
<td>1964</td>
<td>1958 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4012</td>
<td>4006 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8108</td>
<td>8102 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300</td>
<td>16294 bytes – if table does not include any variable length columns.</td>
</tr>
</tbody>
</table>

* This size includes six bytes for the row overhead and two bytes for the row length field

• The maximum number of bytes of variable-length data per row depends on the locking scheme for the table. The following describes the maximum size columns for an APL table:
The following describes the maximum size columns for a DOL table:

<table>
<thead>
<tr>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2K (2048 bytes)</td>
<td>1960</td>
<td>1960</td>
</tr>
<tr>
<td>4K (4096 bytes)</td>
<td>4008</td>
<td>4008</td>
</tr>
<tr>
<td>8K (8192 bytes)</td>
<td>8104</td>
<td>8157</td>
</tr>
<tr>
<td>16K (16384 bytes)</td>
<td>16296</td>
<td>16227</td>
</tr>
</tbody>
</table>

- You cannot use `alter table` to add a declarative or check constraint and then insert data into the table in the same batch or procedure. Either separate the `alter` and `insert` statements into two different batches or procedures, or use `execute` to perform the actions separately.

- You cannot use the following variable in `alter table` statements that include defaults:

```sql
declare @a int
select @a = 2
alter table t2 add c3 int
default @a
```

Doing so results in error message 154, which says, “Variable is not allowed in default.”

- SQL user-defined functions are not currently supported with `create proxy table`, `create table at remote server`, or `alter table`.

**Note** The execution of SQL functions requires the syntax `username.functionname()`.

**alter table and encrypted columns**

You cannot use `alter table` to encrypt or decrypt a column belonging to a clustered or placement index. To encrypt or decrypt this column, drop the index, alter the column, and re-create the index.

You cannot use `alter table` to encrypt or decrypt a column if the table has a trigger defined. To decrypt the column, drop the trigger, alter the column, then re-create the trigger.
You can encrypt these data types:

- `int`, `smallint`, `tinyint`
- `unsigned int`, `unsigned smallint`, `unsigned tinyint`
- `bigint`, `unsigned bigint`
- `decimal` and `numeric`
- `float4` and `float8`
- `money`, `smallmoney`
- `date`, `time`, `smalldatetime`, `datetime`
- `char` and `varchar`
- `unichar`, `univarchar`
- `binary` and `varbinary`
- `bit`

The underlying data type of encrypted data on disk is `varbinary`. Null values are not encrypted.

Modifying the data type of the encrypted column belonging to a clustered or placement index results in the index being out of order, and `alter table` displays an error. To modify the type:

1. Drop the index
2. Modify the type
3. Recreate the index

`alter table` reports an error if you:

- Modify a computed column to be an encrypted column
- Modify a column for encryption in which the column is referenced in an expression used by a computed column
- Modify an encrypted column to be a computed column
- Modify a column to be a computed column in which the expression references an encrypted column
- Encrypt a column that is a member of a functional index
- Specify an encrypted column as a partition key
alter table

- Modify a column that is used as a partition key for encryption

**Note** Referential integrity between encrypted columns is supported when the columns are encrypted with the same key. For details, refer to *Encrypted Columns Guide*, chapter “Encrypting Data.”

Getting information about tables
- For information about a table and its columns, use `sp_help`.
- To rename a table, execute `sp_rename` (do not rename the system tables).
- For information about integrity constraints (unique, primary key, references, and check) or the default clause, see `create table` in this chapter.

Specifying ascending or descending ordering in indexes
- Use the `asc` and `desc` keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the `order by` clause of queries eliminates the sorting step during query processing. For more information, see Chapter 13, “Indexing for Performance” in the *Performance and Tuning Guide: Basics*.

Using cross-database referential integrity constraints
- When you create a cross-database constraint, Adaptive Server stores the following information in the `sysreferences` table of each database:

<table>
<thead>
<tr>
<th>Information stored in <code>sysreferences</code></th>
<th>Columns with information about the referenced table</th>
<th>Columns with information about the referencing table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key column IDs</td>
<td>refkey1 through refkey16</td>
<td>fokey1 through fokey16</td>
</tr>
<tr>
<td>Table ID</td>
<td>reftabid</td>
<td>tableid</td>
</tr>
<tr>
<td>Database ID</td>
<td>pmrydbid</td>
<td>frgndbid</td>
</tr>
<tr>
<td>Database name</td>
<td>pmrydbname</td>
<td>frgndbname</td>
</tr>
</tbody>
</table>

- When you drop a referencing table or its database, Adaptive Server removes the foreign-key information from the referenced database.
- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains the referenced table, or
  - Rename either database with `sp_renamedb`.

You must first remove the cross-database constraint with `alter table`.
• Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases could cause database corruption.

• The sysreferences system table stores the name and the ID number of the external database. Adaptive Server cannot guarantee referential integrity if you use load database to change the database name or to load it onto a different server.

**Warning!** Before dumping a database to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

Changing defaults

• You can create column defaults in two ways: by declaring the default as a column constraint in the create table or alter table statement, or by creating the default using the create default statement and binding it to a column using sp_bindefault.

• You cannot replace a user-defined default bound to the column with sp_bindefault. Unbind the default with sp_unbinddefault first.

• If you declare a default column value with create table or alter table, you cannot bind a default to that column with sp_bindefault. Drop the default by altering it to NULL, then bind the user-defined default. Changing the default to NULL unbinds the default and deletes it from the sysobjects table.

Setting space management properties for indexes

• The space management properties fillfactor, max_rows_per_page, and reservepagegap in the alter table statement apply to indexes that are created for primary key or unique constraints. The space management properties affect the data pages of the table if the constraint creates a clustered index on an allpages-locked table.

• Use sp_chgattribute to change max_rows_per_page or reservepagegap for a table or an index, to change the exp_row_size value for a table, or to store fillfactor values.

• Space management properties for indexes are applied:
When indexes are re-created as a result of an alter table command that changes the locking scheme for a table from allpages locking to data-only locking or vice versa. See “Changing locking schemes” on page 55 for more information.

When indexes are automatically rebuilt as part of a reorg rebuild command.

To see the space management properties currently in effect for a table, use sp_help. To see the space management properties currently in effect for an index, use sp_helpindex.

The space management properties fillfactor, max_rows_per_page, and reservepagegap help manage space usage for tables and indexes in the following ways:

- fillfactor leaves extra space on pages when indexes are created, but the fillfactor is not maintained over time. It applies to all locking schemes.
- max_rows_per_page limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables.
- reservepagegap specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to all locking schemes.

You can store space management properties for tables and indexes so that they are applied during alter table and reorg rebuild commands.

The following table shows the valid combinations of space management properties and locking schemes. If an alter table command changes the table so that the combination is not compatible, the values stored in the stored in system tables remain there, but are not applied during operations on the table. If the locking scheme for a table changes so that the properties become valid, then they are used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>allpages</th>
<th>datapages</th>
<th>datarows</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_rows_per_page</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>reservepagegap</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fillfactor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>exp_row_size</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The following table shows the default values and the effects of using the default values for the space management properties.
Conversion of `max_rows_per_page` to `exp_row_size`

- If a table has `max_rows_per_page` set, and the table is converted from allpages locking to data-only locking, the value is converted to an `exp_row_size` value before the `alter table...lock` command copies the table to its new location. The `exp_row_size` is enforced during the copy. The following table shows how the values are converted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Effect of using the default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max_rows_per_page</code></td>
<td>0</td>
<td>Fits as many rows as possible on the page, up to a maximum of 256</td>
</tr>
<tr>
<td><code>reservepagegap</code></td>
<td>0</td>
<td>Leaves no gaps</td>
</tr>
<tr>
<td><code>fillfactor</code></td>
<td>0</td>
<td>Fully packs leaf pages</td>
</tr>
</tbody>
</table>

### Using `reservepagegap`

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The `reservepagegap` keyword causes these commands to leave empty pages so that future page allocations take place close to the page that is being split or to the page from which a row is being forwarded.

- The `reservepagegap` value for a table is stored in `sysindexes`, and is applied when the locking scheme for a table is changed from allpages locking to data-only locking or vice versa. To change the stored value, use `sp_chgattribute` before running `alter table`.

- `reservepagegap` specified with the `clustered` keyword on an allpages-locked table overwrites any value previously specified with `create table` or `alter table`.
Partitioning tables for improved performance

- You can partition an unpartitioned table or repartition an already partitioned table using the `partition by` clause. The task requires data copy; all data rows are redistributed according to the specified partition criteria. The task may be run in parallel if the Adaptive Server is configured for parallel processing. You must set the `select into/bulkcopy/pllsort` option to `true`. If the table has indexes, you must drop the indexes before you can change:
  - An unpartitioned table into a semantic-partitioned table.
  - The partitioning type.
  - The partitioning key – You need not drop indexes to change other attributes of the partitions, such as number of partitions, partition bounds, or partition location; the indexes are built automatically. See `create table` on page 163 for more information on partition key and bound restrictions.
- You can use the `add partition` clause to add empty partitions to list- or range-partitioned tables, but not to hash or round-robin partitioned tables.
  
  For range-partitioned tables, you can add new partitions only to the high end of the partition conditions. If the last existing partition has the maximum boundary (values <= (MAX)), you cannot add new partitions.

- The partition `number_of_partition` and `unpartition` clause are provided for compatibility with versions of Adaptive Server earlier than 15.0. You can use `partition number_of_partition` clause only on unpartitioned tables to add \((\text{number_of_partition}-1)\) empty round-robin partitions; existing data is placed on the first partition, with subsequent data distributed among all partitions. If the table has a global clustered index, Adaptive Server places subsequent data rows in the first partition. To redistribute the data, drop and re-create the index.

  **Note** These commands do not require data movement. However, because Adaptive Server performs a number of internal steps, the commands, especially when executed on large tables, do not occur instantly. To avoid data corruption, do not interrupt the operation while you partition or unpartition the table.

You can use the unpartition clause only on round-robin-partitioned tables without indexes.

- You cannot partition system tables.
• You cannot partition remote proxy tables.

• You cannot issue the partition-related alter table commands within a user-defined transactions.

• You cannot change a table’s partitioning properties using the partition by clause if there are active open cursors on the table.

• After using the partition by clause, you must perform a full database dump before you can use dump transaction.

• You cannot drop a column that is part of a partitioning key.

• Alter key columns with care. In some cases, modifying the datatype of a key column might redistribute data among partitions. See the Transact-SQL User's Guide for details.

• Changing a table’s partitioning properties increments the schema count, which causes existing stored procedures that access this table to recompile the next time they are executed.

Using computed columns

• When you add a new computed column without specifying nullability and the materialization property, the default option is nullable and not materialized.

• When you add a new materialized computed column, the computed_column_expression is evaluated for each existing row in the table, and the result is stored in the table.

• You cannot add new computed columns and add or modify their base columns at the same time.

• You can modify the entire definition of an existing computed column. This is a quick way to drop the computed column and add a new one with the same name. Such a column behaves like a new computed column: its defaults are not materialized and nullable, if you do not specify these options.

• You can modify the materialization property of an existing computed column without changing its other properties, such as the expression that defines it or its nullability.

• When you modify a not-null, materialized computed column into a virtual column, you must specify “null” in the modify clause.

• When you modify a computed column that is not materialized, to materialize it, the computed_column_expression is evaluated for each existing row in the table, and the result is stored in the table.
If you modify existing columns that are index keys, the index is rebuilt.

You cannot modify a materialized computed column into a virtual column if it has been used as an index key; you must first drop the index.

You cannot modify a regular column to become a computed column, or a computed column to become a regular column.

You cannot modify or drop the base column referenced by a computed column.

You cannot drop a computed column if it is used as an index key.

### Adding IDENTITY columns

- When adding a numeric or integer IDENTITY column to a table, make sure the column precision is large enough to accommodate the number of existing rows. If the number of rows exceeds 10 \( \text{precision} - 1 \), Adaptive Server prints an error message and does not add the column.

- When adding an IDENTITY column to a table, Adaptive Server:
  - Locks the table until all the IDENTITY column values have been generated. If a table contains a large number of rows, this process may be time-consuming.
  - Assigns each existing row a unique, sequential IDENTITY column value, beginning with the value 1.
  - Logs each insert operation into the table. Use `dump transaction` to clear the database’s transaction log before adding an IDENTITY column to a table with a large number of rows.

- Each time you insert a row into the table, Adaptive Server generates an IDENTITY column value that is one higher than the last value. This value takes precedence over any defaults declared for the column in the `alter table` statement or bound to it with `sp_bindefault`.

### Altering table schema

- `add`, `drop`, or `modify`, and `lock` sub-clauses are useful in changing an existing table’s schema. A single statement can contain any number of these sub-clauses, in any order, as long as the same column name is not referenced more than once in the statement.

- If stored procedures using `select *` reference a table that has been altered, no new columns appear in the result set, even if you use the `with recompile` option. You must drop the procedure and re-create it to include these new columns.
To ensure that triggers fire properly, you must drop and re-create all triggers on an altered table after you perform an add, drop, modify, or lock operation.

Adaptive Server issues an error message if you add a not null column with alter table.

You cannot drop all the columns in a table. Also, you cannot drop the last remaining column from a table (for example, if you drop four columns from a five-column table, you cannot then drop the remaining column). To remove a table from the database, use drop table.

Data copy is required:
- To drop a column
- To add a NOT NULL column
- For most alter table ... modify commands

Use set noexec on and showplan on options to determine if a data copy is required for a particular alter table command.

You can specify a change in the locking scheme for the modified table with other alter table commands (add, drop, or modify) when the other alter table command requires a data copy.

If alter table performs a data copy, select into /bulkcopy/pllsort must be turned on in the database that includes the table whose schema you are changing.

The modified table retains the existing space management properties (max_rows_per_page, fillfactor, and so on) and indexes of the table.

alter table that requires a data copy does not fire any triggers.

You can use alter table to change the schema of remote proxy tables created and maintained by Component Integration Services (CIS). For information about CIS, see the Component Integration Services User's Guide.

You cannot perform a data copy and add a table level or referential integrity constraint in the same statement.

You cannot perform a data copy and create a clustered index in the same statement.

If you add a not null column, you must also specify a default clause. This rule has one exception: if you add a user-defined type column, and the type has a default bound to it, you need not specify a default clause.
You can always add, drop, or modify a column in all-pages locked tables. However, there are restrictions for adding, dropping, or modifying a column in a data-only locked table, which are described in the following table:

<table>
<thead>
<tr>
<th>Type of index</th>
<th>All pages locked, partitioned table</th>
<th>All pages locked, unpartitioned table</th>
<th>Data-only locked, partitioned table</th>
<th>Data-only locked, unpartitioned table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustered</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nonclustered</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

If you need to add, drop, or modify a column in a data-only-locked table partitioned with a clustered index, you can:

a. Drop the clustered index.

b. Alter the data-only locked table.

c. Re-create the clustered index.

You cannot add a NOT NULL Java object as a column. By default, all Java columns always have a default value of NULL, and are stored as either varbinary strings or as image datatypes.

You cannot modify a partitioned table that contains a Java column if the modification requires a data copy. Instead, first unpartition the table, execute `alter table`, then repartition the table.

You cannot drop the key column from an index or a referential integrity constraint. To drop a key column, first drop the index or referential integrity constraint, then drop the key column. See the *Transact-SQL User’s Guide* for more information.

You can drop columns that have defaults or rules bound to them. Any column-specific defaults are also dropped when you drop the column. You cannot drop columns that have check constraints or referential constraints bound to them. Instead, first drop the check constraint or referential constraint, then drop the column. Use `sp_helpconstraint` to identify any constraints on a table, and use `sp_depends` to identify any column-level dependencies.

You cannot drop a column from a system table. Also, you cannot drop columns from user tables that are created and used by Sybase-provided tools and stored procedures.
You can generally modify the datatype of an existing column to any other datatype if the table is empty. If the table is not empty, you can modify the datatype to any datatype that is explicitly convertible to the original datatype.

You can:
- Add a new IDENTITY column.
- Drop an existing IDENTITY column.
- Modify the size of an existing IDENTITY.

See the Transact-SQL User's Guide for more information.

Altering the schema of a table increments the schema count, causing existing stored procedures that access this table to be renormalized the next time they are executed. Changes in datatype-dependent stored procedures or views may fail with datatype normalization type errors. You must update these dependent objects so they refer to the modified schema of the table.

Restrictions for modifying a table schema
- You cannot run alter table from inside a transaction.
- Altering a table’s schema can invalidate backups that you made using bcp. These backups may use a tables schema that is no longer compatible with the table’s current schema.
- You can add NOT NULL columns with check constraints, however, Adaptive Server does not validate the constraint against existing data.
- You cannot change the locking scheme of a table using the alter table . . . add, drop, or modify commands if the table has a clustered index and the operation requires a data copy. Instead you can
  a. Drop the clustered index.
  b. Alter the table’s schema.
  c. Re-create the clustered index.
- You cannot alter a table’s schema if there are any active open cursors on the table.

Restrictions for modifying text and image columns
- You can only add text or image columns that accept null values.
To add a text or image column so it contains only non-null values, first add a column that only accepts null values and then update it to the non-null values.

- You can modify a column from text datatype only to the following datatypes:
  - [n]char
  - [n]varchar
  - unichar
  - univarchar
  - nchar
  - nvarchar
- You can modify a column from image datatype only to binary or varbinary.
- You cannot add a new text or image column and then drop an existing text or image column in the same statement.
- You cannot modify a column to either text or image datatype.

Modifying tables with unitext columns

The following restrictions apply when you use alter table to modify unitext columns:

- You can add a new unitext column that accepts NULL values.
- You can modify a column from unitext only to the following datatypes:
  - [n]char
  - [n]varchar
  - unichar
  - univarchar
  - binary
  - varbinary
- You cannot modify a column to the unitext datatype.
- You cannot add a unitext column and drop an existing unitext column in the same statement.
Changing locking schemes

- **alter table** supports changing from any locking scheme to any other locking scheme. You can change:
  - From allpages to datapages or vice versa
  - From allpages to datarows or vice versa
  - From datapages to datarows or vice versa

- Before you change from allpages locking to a data-only locking scheme, or vice versa, use **sp_dboption** to set the database option **select into/bulkcopy/pllsort** to true, then run **checkpoint** in the database if any of the tables are partitioned and the sorts for the indexes require a parallel sort.

- After changing the locking scheme from allpages-locking to data-only locking or vice versa, the use of the **dump transaction** command to back up the transaction log is prohibited; you must first perform a full database dump.

- When you use **alter table...lock** to change the locking scheme for a table from allpages locking to data-only locking or vice versa, Adaptive Server makes a copy of the table’s data pages. There must be enough room on the segment where the table resides for a complete copy of the data pages. There must be space on the segment where the indexes reside to rebuild the indexes.

  Clustered indexes for data-only-locked tables have a leaf level above the data pages. If you are altering a table with a clustered index from allpages-locking to data-only-locking, the resulting clustered index requires more space. The additional space required depends on the size of the index keys.

  Use **sp_spaceused** to determine how much space is currently occupied by the table, and use **sp_helpsegment** to see the space available to store the table.

- When you change the locking scheme for a table from allpages locking to datapages locking or vice versa, the space management properties are applied to the tables, as the data rows are copied, and to the indexes, as they are re-created. When you change from one data-only locking scheme to another, the data pages are not copied, and the space management properties are not applied.
• If a table is partitioned, changing the locking scheme performs a partition-to-partition copy of the rows. It does not balance the data on the partitions during the copy.

• When you change the locking scheme for a table, the alter table...lock command acquires an exclusive lock on the table until the command completes.

• When you use alter table...lock to change from datapages locking to datarows locking, the command does not copy data pages or rebuild indexes. It only updates system tables.

• Changing the locking scheme while other users are active on the system may have the following effects on user activity:
  • Query plans in the procedure cache that access the table are recompiled the next time they are run.
  • Active multistatement procedures that use the table are recompiled before continuing with the next step.
  • Ad hoc batch transactions that use the table are terminated.

**Warning!** Changing the locking scheme for a table while a bulk copy operation is active can cause table corruption. Bulk copy operates by first obtaining information about the table and does not hold a lock between the time it reads the table information and the time it starts sending rows, leaving a small window of time for an alter table...lock command to start.

Adding Java-SQL columns
• If Java is enabled in the database, you can add Java-SQL columns to a table. For more information, see *Java in Adaptive Server Enterprise.*

• The declared class (datatype) of the new Java-SQL column must implement either the Serializable or Externalizable interface.

• When you add a Java-SQL column to a table, the Java-SQL column cannot be specified:
  • As a foreign key
  • In a references clause
  • As having the UNIQUE property
  • As the primary key
• If in row is specified, the value stored cannot exceed 16K bytes, depending on the page size of the data server.

• If off row is specified, then the column cannot be:
  • Referenced in a check constraint
  • Referenced in a select that specifies distinct
  • Specified in a comparison operator, in a predicate, or in a group by clause

Standards
ANSI SQL – Compliance level: Transact-SQL extension.


Permissions
alter table permission defaults to the table owner; it cannot be transferred except to the Database Owner, who can impersonate the table owner by running the setuser command. A System Administrator can also alter user tables.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>alter</td>
<td>alter table</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – add column, drop column, modify column, replace column, add constraint, or drop constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if a set proxy is in effect</td>
</tr>
</tbody>
</table>

See also Commands create index, create table, dbcc, drop database, dump transaction, insert, setuser

System procedures sp_chgattribute, sp_help, sp_helpartition, sp_rename
**begin...end**

**Description**
Encloses a series of SQL statements so that control-of-flow language, such as `if...else`, can affect the performance of the whole group.

**Syntax**
```
begin
    statement block
end
```

**Parameters**
- `statement block` is a series of statements enclosed by `begin` and `end`.

**Examples**

**Example 1** Without `begin` and `end`, the `if` condition would cause execution of only one SQL statement:

```
if (select avg (price) from titles) < $15
begin
    update titles
    set price = price * $2
    select title, price
    from titles
    where price > $28
end
```

**Example 2** Without `begin` and `end`, the `print` statement would not execute:

```
create trigger deltitle
on titles
for delete
as
if (select count (*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) > 0
begin
    rollback transaction
    print "You can't delete a title with sales."
end
else
    print "Deletion successful--no sales for this title."
```

**Usage**
- `begin...end` blocks can nest within other `begin...end` blocks.

**Standards**
- ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
- `begin...end` permission defaults to all users. No permission is required to use it.

**See also**
- `Commands` `if...else`
**begin transaction**

**Description**  
Marks the starting point of a user-defined transaction.

**Syntax**  
```
begin transaction [transaction_name]
```

**Parameters**  
`transaction_name`  
is the name assigned to this transaction. Transaction names must conform to the rules for identifiers. Use transaction names only on the outermost pair of nested `begin transaction/commit` or `begin transaction/rollback` statements.

**Examples**  
Explicitly begins a transaction for the `insert` statement:
```
begin transaction
    insert into publishers (pub_id) values ('9999')
commit transaction
```

**Usage**  
- Define a transaction by enclosing SQL statements and system procedures within the phrases `begin transaction` and `commit`. If you set chained transaction mode, Adaptive Server implicitly invokes a `begin transaction` before the following statements: `delete`, `insert`, `open`, `fetch`, `select`, and `update`. You must still explicitly close the transaction with a `commit`.

- To cancel all or part of a transaction, use the `rollback` command. The `rollback` command must appear within a transaction; you cannot roll back a transaction after it is committed.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
`begin transaction` permission defaults to all users. No permission is required to use it.

**See also**  
`Commands` commit, rollback, save transaction
**break**

**Description**

Causes an exit from a while loop. break is often activated by an if test.

**Syntax**

```
while logical_expression
    statement
    break
    statement
    continue
```

**Parameters**

- `logical_expression` is an expression (a column name, constant, any combination of column names and constants connected by arithmetic or bitwise operators, or a subquery) that returns TRUE, FALSE, or NULL. If the logical expression contains a `select` statement, enclose the `select` statement in parentheses.

**Examples**

If the average price is less than $30, double the prices. Then, select the maximum price. If it is less than or equal to $50, restart the while loop and double the prices again. If the maximum price is more than $50, exit the while loop and print a message:

```
while (select avg (price) from titles) < $30
begin
    update titles
      set price = price * 2
    select max (price) from titles
    if (select max (price) from titles) > $50
        break
    else
        continue
end
begin
    print "Too much for the market to bear"
end
```

**Usage**

- break causes an exit from a while loop. Statements that appear after the keyword end, which marks the end of the loop, are then executed.
- If two or more while loops are nested, the inner break exits to the next outermost loop. First, all the statements after the end of the inner loop run; then, the next outermost loop restarts.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

break permission defaults to all users. No permission is required to use it.

**See also**

*Commands* continue, while
checkpoint

Description
Writes all dirty pages (pages that have been updated since they were last written) to the database device.

Syntax
checkpoint [all | [dbname[, dbname, dbname, ........]]]

Examples
Writers all dirty pages in the current database to the database device, regardless of the system checkpoint schedule:

    checkpoint

Usage
• You can use checkpoint with an archive database, however, the checkpoint process does not checkpoint an archive database automatically.
• Use checkpoint only as a precautionary measure in special circumstances.
• sp_dboption automatically defaults to using checkpoint when you change a database option.
• You can specify the database or databases to run checkpoint.
• To run checkpoint all against all databases, including system and temp databases, you must have the sa_role or oper_role.
• If you do not have the sa_role or oper_role, the checkpoint all runs only against those databases you own.

Automatic checkpoints
• Checkpoints caused by the checkpoint command supplement automatic checkpoints, which occur at intervals calculated by Adaptive Server on the basis of the configurable value for maximum acceptable recovery time.
• checkpoint shortens the automatic recovery process by identifying a point at which all completed transactions are guaranteed to have been written to the database device. A typical checkpoint takes about 1 second, although checkpoint time varies depending on the amount of activity on Adaptive Server.
• The automatic checkpoint interval is calculated by Adaptive Server on the basis of system activity and the recovery interval value in the system table syscurconfigs. The recovery interval determines checkpoint frequency by specifying the maximum amount of time it should take for the system to recover. Reset this value by executing sp_configure.
• You can configure Adaptive Server with multiple checkpoint processes. This allows Adaptive Server with multiple engines to checkpoint tasks more frequently, thereby shortening the automatic recovery process.
If the housekeeper task can flush all active buffer pools in all configured caches during the server’s idle time, it wakes up the checkpoint task. The checkpoint task determines whether it can checkpoint the database.

Checkpoints that occur as a result of the housekeeper task are known as **free checkpoints**. They do not involve writing many dirty pages to the database device, since the housekeeper task has already done this work. They may improve recovery speed for the database.

---

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Checkpoint permission defaults to the Database Owner. It cannot be transferred.

See also

**System procedures**  sp_configure, sp_dboption
close

Description  Deactivates a cursor.

Syntax  close cursor_name

Parameters  cursor_name
is the name of the cursor to close.

Examples  Closes the cursor named authors_crsr:

    close authors_crsr

Usage  • The close command essentially removes the cursor’s result set. The cursor position within the result set is undefined for a closed cursor.

• Adaptive Server returns an error message if the cursor is already closed or does not exist.

Standards  ANSI SQL – Compliance level: Entry-level compliant.

Permissions  close permission defaults to all users. No permission is required to use it.

See also  Commands  deallocate cursor, declare cursor, fetch, open
commit

Description
Marks the ending point of a user-defined transaction.

Syntax
commit [tran | transaction | work] [transaction_name]

Parameters
tran | transaction | work
specifies that you want to commit the transaction or the work. If you specify
tran, transaction, or work, you can also specify the transaction_name.

transaction_name
is the name assigned to the transaction. It must conform to the rules for
identifiers. Use transaction names only on the outermost pair of nested begin
transaction/commit or begin transaction/rollback statements.

Examples
After updating the royalty per entries for the two authors, insert the savepoint
percent changed, then determine how a 10 percent increase in the book’s price
would affect the authors’ royalty earnings. The transaction is rolled back to the
savepoint with the rollback transaction command:

    begin transaction royalty_change
    update titleauthor
      set royaltyper = 65 from titleauthor, titles
      where royaltyper = 75
      and titleauthor.title_id = titles.title_id
      and title = "The Gourmet Microwave"
    update titleauthor
      set royaltyper = 35 from titleauthor, titles
      where royaltyper = 25
      and titleauthor.title_id = titles.title_id
      and title = "The Gourmet Microwave"
    save transaction percentchanged
    update titles
      set price = price * 1.1
      where title = "The Gourmet Microwave"
    select (price * total_sales) * royaltyper
      from titles, titleauthor
      where title = "The Gourmet Microwave"
      and titles.title_id = titleauthor.title_id
    rollback transaction percentchanged
    commit transaction
Usage

- Define a transaction by enclosing SQL statements and system procedures with the phrases `begin transaction` and `commit`. If you set the chained transaction mode, Adaptive Server implicitly invokes a `begin transaction` before the following statements: `delete`, `insert`, `open`, `fetch`, `select`, and `update`. You must still explicitly enclose the transaction with a `commit`.

- To cancel all or part of an entire transaction, use the `rollback` command. The `rollback` command must appear within a transaction. You cannot roll back a transaction after the `commit` has been entered.

- If no transaction is currently active, the `commit` or `rollback` statement has no effect on Adaptive Server.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

The commit transaction and commit tran forms of the statement are Transact-SQL extensions.

Permissions

Commit permission defaults to all users.

See also

| Commands | begin transaction, rollback, save transaction |
compute clause

Description
Generates summary values that appear as additional rows in the query results.

Syntax
```
start_of_select_statement
    compute row_aggregate (column_name)
    [, row_aggregate (column_name)]...
    [by column_name [, column_name]...]
```

Parameters
- `row_aggregate` is one of the following:
  - `sum` – is the total of values in the (numeric) column
  - `avg` – is the average of values in the (numeric) column
  - `min` – is the lowest value in the column
  - `max` – is the highest value in the column
  - `count` – is the number of values in the column as an integer
  - `count` – is the number of values in the column as a bigint

- `by` calculates the row aggregate values for subgroups. Whenever the value of
  the by item changes, row aggregate values are generated. If you use by, you
  must use order by.

  Listing more than one item after by breaks a group into subgroups and
  applies a function at each level of grouping.

Examples

**Example 1** Calculates the sum of the prices of each type of cook book that
costs more than $12:

```
select type, price
from titles
where price > $12
    and type like "%cook"
order by type, price
compute sum (price) by type

type      price
---------  --------
mod_cook  19.99
```

```
Example 2  Calculates the sum of the prices and advances for each type of cook book that costs more than $12, with one compute clause applying several aggregate functions to the same set of grouping columns:

```
select type, price, advance
from titles
where price > $12
  and type like "%cook"
order by type, price
compute sum (price), sum (advance) by type
```

```
type      price     advance
--------- ----------- ----------
mod_cook   19.99     0.00
sum       sum
           --------- -----------
           19.99     0.00

type      price     advance
--------- ----------- ----------
trad_cook 14.99     8,000.00
trad_cook 20.95     7,000.00
sum       sum
           --------- -----------
           35.94     15,000.00
```

(5 rows affected)

Example 3  Calculates the sum of the prices and maximum advances of each type of cook book that costs more than $12, with one compute clause applying several aggregate functions to the same set of grouping columns:

```
select type, price, advance
from titles
where price > $12
  and type like "%cook"
order by type, price
compute sum (price), max (advance) by type
```

```
type      price     advance
--------- ----------- ----------
trad_cook 14.99     max = 8,000.00
trad_cook 20.95     max = 7,000.00
sum       sum
           --------- -----------
           35.94     max = 15,000.00
```

(5 rows affected)
**Example 4**  Breaks on type and pub_id and calculates the sum of the prices of psychology books by a combination of type and publisher ID:

```sql
select type, pub_id, price
from titles
where price > $10
  and type = "psychology"
order by type, pub_id, price
compute sum (price) by type, pub_id
```

<table>
<thead>
<tr>
<th>type</th>
<th>pub_id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>psychology</td>
<td>0736</td>
<td>10.95</td>
</tr>
<tr>
<td>psychology</td>
<td>0736</td>
<td>19.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.94</td>
</tr>
</tbody>
</table>

(5 rows affected)
Chapter 1  Commands

Example 5  Calculates the grand total of the prices of psychology books that cost more than $10 in addition to calculating sums by type and pub_id, using more than one compute clause to create more than one group:

```sql
select type, pub_id, price
from titles
where price > $10
    and type = "psychology"
order by type, pub_id, price
compute sum (price) by type, pub_id
compute sum (price) by type

<table>
<thead>
<tr>
<th>type</th>
<th>pub_id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>psychology</td>
<td>0736</td>
<td>10.95</td>
</tr>
<tr>
<td>psychology</td>
<td>0736</td>
<td>19.99</td>
</tr>
</tbody>
</table>

  sum
  --------
  30.94

<table>
<thead>
<tr>
<th>type</th>
<th>pub_id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>psychology</td>
<td>0877</td>
<td>21.59</td>
</tr>
</tbody>
</table>

  sum
  --------
  21.59
  sum
  --------
  52.53

(6 rows affected)

Example 6  Calculates the grand totals of the prices and advances of cook books that cost more than $10:

```sql
select type, price, advance
from titles
where price > $10
    and type like "%cook"
compute sum (price), sum (advance)

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_cook</td>
<td>19.99</td>
<td>0.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>8,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>4,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>7,000.00</td>
</tr>
</tbody>
</table>

  sum  sum
  --------  -----------
  67.88  19,000.00
```
Example 7 Calculates the sum of the price of cook books and the sum of the price used in an expression:

```sql
select type, price, price*2
from titles
where type like "%cook"
compute sum (price), sum (price*2)
```

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_cook</td>
<td>19.99</td>
<td>39.98</td>
</tr>
<tr>
<td>mod_cook</td>
<td>2.99</td>
<td>5.98</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>41.90</td>
</tr>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>23.90</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>29.98</td>
</tr>
</tbody>
</table>

sum

<table>
<thead>
<tr>
<th>============</th>
<th>============</th>
</tr>
</thead>
<tbody>
<tr>
<td>70.87</td>
<td>141.74</td>
</tr>
</tbody>
</table>

Usage

- The `compute` clause allows you to see the detail and summary rows in one set of results. You can calculate summary values for subgroups, and you can calculate more than one aggregate for the same group.

- You can use `compute` without `by` to generate grand totals, grand counts, and so on. `order by` is optional if you use the `compute` keyword without `by`. See Example 6.

- If you use `compute by`, you must also use an `order by` clause. The columns listed after `compute by` must be identical to or a subset of those listed after `order by` and must be in the same left-to-right order, start with the same expression, and not skip any expressions. For example, if the `order by` clause is `order by a, b, c`, the `compute by` clause can be any (or all) of these:

  - `compute by a, b, c`
  - `compute by a, b`
  - `compute by a`

Restrictions

- You cannot use more than 127 aggregate columns in a `compute` clause.

- You cannot use a `compute` clause in a cursor declaration.

- You can compute summary values for both expressions and columns. Any expression or column that appears in the `compute` clause must appear in the `select` list.
• Aliases for column names are not allowed as arguments to the row aggregate in a compute clause, although you can use them in the select list, the order by clause, and the by clause of compute.

• In a select statement with a compute clause, the order of columns in the select list overrides the order of the aggregates in the compute clause. OpenClient, JDBC, and DBLibrary programmers must be aware of this in order to put the aggregate results in the right place. See “Row Aggregates” for an example.

• You cannot use select into in the same statement as a compute clause, because statements that include compute do not generate normal tables.

• If a compute clause includes a group by clause:
  • The compute clause cannot contain more than 255 aggregates.
  • The group by clause cannot contain more than 255 columns.
  • Columns included in a compute clause cannot be longer than 255 bytes.

**compute results appear as a new row or rows**

• The aggregate functions ordinarily produce a single value for all the selected rows in the table or for each group, and these summary values are shown as new columns. For example:

```sql
select type, sum (price), sum (advance)
from titles
where type like "%cook"
group by type
type
------------- --------- ----------
mod_cook 22.98 15,000.00
trad_cook 47.89 19,000.00
```

(2 rows affected)

• The compute clause allows you to retrieve detail and summary rows with one command. For example:

```sql
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum (price), sum (advance) by type
type price advance
------------- ---------- ----------------
mod_cook 2.99 15,000.00
mod_cook 19.99 0.00
```
### Compute Result:

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>4,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>8,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>7,000.00</td>
</tr>
</tbody>
</table>

(7 rows affected)

Table 1-4 lists the output and grouping of different types of compute clauses.

**Table 1-4: compute by clauses and detail rows**

<table>
<thead>
<tr>
<th>Clauses and grouping</th>
<th>Output</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>One compute clause, same function</td>
<td>One detail row</td>
<td>1, 2, 4, 6, 7</td>
</tr>
<tr>
<td>One compute clause, different functions</td>
<td>One detail row per type of function</td>
<td>3</td>
</tr>
<tr>
<td>More than one compute clause, same grouping columns</td>
<td>One detail row per compute clause; detail rows together in the output</td>
<td>Same results as having one compute clause with different functions</td>
</tr>
<tr>
<td>More than one compute clause, different grouping columns</td>
<td>One detail row per compute clause; detail rows in different places, depending on the grouping</td>
<td>5</td>
</tr>
</tbody>
</table>

### Case sensitivity

- If your server has a case-insensitive sort order installed, compute ignores the case of the data in the columns you specify. For example, given this data:

```sql
select * from groupdemo
lname amount
Smith 10.00
smith 5.00
SMITH 7.00
Levi 9.00
Lévi 20.00
```

compute by on lname produces these results:

```sql
select lname, amount from groupdemo
```
order by lname
compute sum (amount) by lname

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levi</td>
<td>9.00</td>
</tr>
</tbody>
</table>

Compute Result:

|        | 9.00   |

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lévi</td>
<td>20.00</td>
</tr>
</tbody>
</table>

Compute Result:

|        | 20.00  |

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>5.00</td>
</tr>
<tr>
<td>SMITH</td>
<td>7.00</td>
</tr>
<tr>
<td>Smith</td>
<td>10.00</td>
</tr>
</tbody>
</table>

Compute Result:

|        | 22.00  |

The same query on a case- and accent-insensitive server produces these results:

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levi</td>
<td>9.00</td>
</tr>
<tr>
<td>Lévi</td>
<td>20.00</td>
</tr>
</tbody>
</table>

Compute Result:

|        | 29.00  |

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>5.00</td>
</tr>
<tr>
<td>SMITH</td>
<td>7.00</td>
</tr>
<tr>
<td>Smith</td>
<td>10.00</td>
</tr>
</tbody>
</table>

Compute Result:
compute clause

22.00

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

See also
Commands  group by and having clauses, select
Functions  avg, count, max, min, sum
connect to...disconnect

Description

Component Integration Services only
Connects to the specified server and disconnects the connected server.

Syntax

Syntax 1 This syntax is sent to Adaptive Server verbatim. Use this syntax with Component Integration Services (CIS) to create a passthru to a different server.

```sql
connect to server_name
disconnect
[from ASE]
[all]
[connection_name]
```

Syntax 2 This syntax opens a new JDBC-level connection to Adaptive Server, and does not use CIS. You can specify the arguments in any order. If you do not include an arguments, Adaptive Server prompts you for connection parameters.

```sql
connect
[to ASE engine_name]
[database database_name]
[as connection_name]
[user user_id]
[identified by password]]
```

Syntax 3 This syntax opens a new JDBC-level connection to Adaptive Server. This syntax does not use CIS:

```sql
connect using connect_string
```

Parameters

**server_name**

is the server to which a passthrough connection is required.

**from ASE**

disconnects from the current Adaptive Server.

**all**

disconnects from all Adaptive Servers.

**connection_name**

disconnects from the specified connection.

**engine_name**

connect to the specified engine.

**database_name**

connects to the specified database.

**connection_name**

connect to the configured connection.
connect to...disconnect

`user_id`
connect to the user with this ID.

`connection_string`
connects using a predetermine connection string.

**Examples**

**Example 1** Establishes a passthrough connection to the server named SYBASE:

`connect to SYBASE`

**Example 2** Disconnects the connected server:

`disconnect`

**Example 3** Disconnects from all servers:

`disconnect all`

**Usage**

- `connect to` specifies the server to which a passthrough connection is required. Passthrough mode enables you to perform native operations on a remote server.
  - `server_name` must be the name of a server in the `sys.servers` table, with its server class and network name defined.
  - When establishing a connection to `server_name` on behalf of the user, Component Integration Services uses one of the following identifiers:
    - A remote login alias described in `sys.attributes`, if present
    - The user’s name and password
  - In either case, if the connection cannot be made to the specified server, Adaptive Server returns an error message.
  - For more information about adding remote servers, see `sp_addserver`.
  - After making a passthrough connection, Component Integration Services bypasses the Transact-SQL parser and compiler when subsequent language text is received. It passes statements directly to the specified server, and converts the results into a form that can be recognized by the Open Client interface and returned to the client program.
  - To close the connection created by the `connect to` command, use the `disconnect` command. You can use this command only after the connection has been made using `connect to`.
  - You can abbreviate the `disconnect` command to `disc`.
  - The `disconnect` command returns an error unless `connect to` has been previously issued and the server is connected to a remote server.
Standards  
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions  
Permission to use the connect to command must be explicitly granted by the System Administrator. The syntax is:

```
grant connect to user_name
```

The System Administrator can grant or revoke connect permission to public globally while in the master database. If the System Administrator wants to grant or revoke connect to permission for a particular user, the user must be a valid user of the master database, and the System Administrator must first revoke permission from public as follows:

```
use master
go
revoke connect from public

sp_adduser fred

grant connect to fred

sp_adduser fred
```

Auditing  
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 90    | security     | connect to                | • Roles – Current active roles  
        |               |                           | • Keywords or options – connect to  
        |               |                           | • Previous value – NULL  
        |               |                           | • Current value – NULL  
        |               |                           | • Other information – NULL  
        |               |                           | • Proxy information – Original login name, if set proxy in effect |

See also  
**Commands**  
create existing table, grant

**System procedures**  
sp_addserver, sp_autoconnect, sp_helpserver,  
sp_passthru, sp_remotesql, sp_serveroption
continue

Description
Repeats the while loop. continue is often activated by an if test.

Syntax
```
while boolean_expression
   statement
   break
   statement
continue
```

Examples
If the average price is less than $30, double the prices. Then, select the maximum price. If it is less than or equal to $50, restart the while loop and double the prices again. If the maximum price is more than $50, exit the while loop and print a message:

```
while (select avg (price) from titles) < $30
begin
   update titles
   set price = price * 2
   select max (price) from titles

   if (select max (price) from titles) > $50
      break
   else
      continue
end

begin
print "Too much for the market to bear"
end
```

Usage
- continue restarts the while loop, skipping any statements after continue.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
continue permission defaults to all users. No permission is required to use it.

See also
Commands break, while
create archive database

Description
Creates an archive database.

Syntax
create archive database db_name
   [on db_device [= size]]
   [ , db_device [= size] ] ...
   with scratch_database = db_name

Parameters

on
specifies the modified pages section. Adaptive Server requires traditional
database storage to store modified pages. Use the on clause to specify the
location and size of the modified pages section.

db device
specifies the database device on which you want to create your modified
pages section.

size
specifies the size of the modified pages section you want to create. If you
omit size, 5120 pages are allocated.

with scratch_database
(required) specifies the name of an existing database in which information
about the archive database is maintained. The sysaltusages system table,
which maps logical pages in the archive database onto physical pages, is
stored in the scratch database.

Examples
The following syntax could be a typical archive database command sequence.

1 Create the scratch database if necessary, using the create database
   command.

   create database scratchdb
       on datadev1 = 100
       log on logdev1 = 50

   This creates a 150MB traditional database called scratchdb.

2 Use sp_dboption to designate the database you just created as a scratch
   database:

   sp_dboption "scratchdb", "scratch database", "true"

3 Create the archive database.

   create archive database archivedb
       on datadev2 = 20
       with scratch_database = scratchdb
This creates an archive database called archivedb, with a 20MB modified pages section.

4 Materialize your archive database using load database:

   load database archivedb
   from "/dev/dumps/050615/proddb_01.dmp"
   stripe on "/dev/dumps/050615/proddb_02.dmp"

5 Bring the database online:

   online database archivedb

6 Check the consistency of the archive database using dbcc commands. For example:

   dbcc checkdb(archivedb)

7 Load a transaction log dump using load tran and restore objects from the archive database using select into or bcp.

   load tran archivedb
   from "/dev/dumps/050615/proddb1_log_01.dmp"
   load tran archivedb
   from "/dev/dumps/050615/proddb1_log_02.dmp"
   online database archivedb
   select * into proddb.dbo.orders from
   archivedb.dbo.orders
   load tran archivedb
   from "/dev/dumps/050615/proddb1_log_03.dmp"
   online database archivedb

Usage

create archive database permission defaults to System Administrators, who can transfer it to users listed in the sysusers table of the master database. However, create archive database permission is often centralized to maintain control over database storage allocation.

create archive database permission is not included in the grant all command.

See also
create database

Description
Creates a new database.

Syntax
create [temporary] database database_name
    [on {default | database_device} [= size]
    [, database_device [= size]]...]
    [log on database_device [= size]
    [, database_device [= size]]...]
    [with {dbid = number, default_location = ".pathname", override}]
    [for {load | proxy_update}]

Parameters
temporary
indicates that you are creating a temporary database.

database_name
is the name of the new database. It must conform to the rules for identifiers
and cannot be a variable.

on
indicates a location and size for the database.

default
indicates that create database can put the new database on any default
database devices, as shown in sysdevices.status. To specify a size for the
database without specifying a location, use:

    on default = size

To change a database device’s status to “default,” use sp_diskdefault.

database_device
is the logical name of the device on which to locate the database. A database
can occupy different amounts of space on each of several database devices.
To add database devices to Adaptive Server, use disk init.

size
is the amount of space to allocate to the database extension. You can use the
following unit specifiers, using uppercase, lowercase, single and double
quotes interchangeably: ‘k’ or “K” (kilobytes), “m” or ‘M’ (megabytes), “g”
or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you
always include a unit specifier. Quotes are optional if you do not include a
unit specifier. However, you must use quotes if you include a unit specifier.
If you do not provide a unit specifier, the value provided is presumed to be
in megabytes.

log on
specifies the logical name of the device for the database logs. You can
specify more than one device in the log on clause.
create database

with
can be specified in any order. You must specify at least one of the following options when you use the with clause:

- **with dbid = number** – specifies the dbid for the new database. If you do not explicitly specify the dbid, the server assigns an unused dbid.

- **with default_location** – specifies the storage location of new tables. If you also specify the for proxy_update clause, one proxy table for each remote table or view is automatically created from the specified location.

- **with override** – forces Adaptive Server to accept your device specifications, even if they mix data and transaction logs on the same device, thereby endangering up-to-the-minute recoverability for your database. If you attempt to mix log and data on the same device without using this clause, the create database command fails. If you mix log and data, and use with override, you are warned, but the command succeeds.

for load

invokes a streamlined version of create database that you can use only for loading a database dump. See “Using the for load option” on page 87 for more information.

for proxy_update

automatically gets metadata from the remote location and creates proxy tables. You cannot use for proxy_update unless you also specify with default_location.

**Examples**

**Example 1** Creates a database named pubs:

```sql
create database pubs
```

**Example 2** Creates a 4MB database named pubs:

```sql
create database pubs
on default = 4
```

If you do not provide a unit specifier for size, the value provided for pubs is presumed to be in megabytes.

**Example 3** Creates a database named pubs with 3MB on the datadev device and 2MB on the moredatadev device:

```sql
create database pubs
on datadev = "3M", moredatadev = '2.0m'
```

**Example 4** Creates a database named pubs with 3MB of data on the datadev device and a 0.5GB log on the logdev device:
create database pubs
   on datadev='3m'
   log on logdev='0.5g'

Example 5  Creates a proxy database named proxydb but does not automatically create proxy tables:

create database proxydb
   with default_location
   "UNITEST.pubs.dbo."

Example 6  Creates a proxy database named proxydb and automatically creates proxy tables:

create database proxydb
   on default = "4M"
   with default_location
   "UNITEST.pubs2.dbo."
   for proxy_update

Example 7  Creates a proxy database named proxydb, and retrieves the metadata for all of the remote tables from a remote database:

create database proxydb
   on default = 4
   with default_location
   "UNITEST.pubs2." for proxy_update

Example 8  Creates a database called pubs with dbid 15:

create database pubs with dbid = 15

Example 9  Creates a temporary database called mytempdb1, with 3MB of data on the datadev device and 1MB of log on the logdev device:

create temporary database mytempdb1
   on datadev = '3m' log on logdev = '1M'

Example 10 Creates a table with one materialized computed column:

create table mytitles
   (title_id tid not null,
    title varchar (80) not null,
    type char (12) not null,
    pub_id char (4) null,
    price money null,
    advance money null,
    total_sales int null,
    notes varchar (200) null,
    pubdate datetime not null,
create database

sum_sales compute price * total_sales materialized)

Usage

- Use create database from the master database.

- You can specify the size as a float datatype, however, the size is rounded
down to the nearest multiple of the allocation unit.

- If you do not explicitly state the size of the database, the size is determined
by the size of the model database. The minimum size that you can create a
database is four allocation units.

- Because Adaptive Server allocates space for databases for create database
and alter database in chunks of 256 logical pages, these commands round
the specified size down to the nearest multiple of allocation units.

- If you do not include a unit specifier, Adaptive Server interprets the size
in terms of megabytes of disk space, and this number is converted to the
logical page size the server uses.

- If you do not specify a location and size for a database, the default location
is any default database devices indicated in master..sysdevices. The
default size is the larger of the size of the model database or the default
database size parameter in sysconfigures.

System Administrators can increase the default size by using sp_configure
to change the value of default database size and restarting Adaptive Server.
The default database size parameter must be at least as large as the model
database. If you increase the size of the model database, you must also
increase the default size.

If Adaptive Server cannot give you as much space as you want where you
have requested it, it comes as close as possible, on a per-device basis, and
prints a message telling how much space was allocated and where it was
allocated. The maximum size of a database is system-dependent.

- If you create a proxy database using:

  create database mydb on my_device
  with default_location = "pathname" for proxy_update

  The presence of the device name is enough to bypass size calculation, and
  this command may fail if the default database size (the size of the model
database) is not large enough to contain all of the proxy tables.

  To allow CIS to estimate database size, do not include any device name or
  other option with this command:

  create database mydb
  with default_location = "pathname" for proxy_update
Restrictions

- Adaptive Server can manage as many as 32,767 databases.
- The dbid should always be greater than zero and less than the maximum dbid of 32,767.
- Adaptive Server can create only one database at a time. If two database creation requests collide, one user sees this message:
  
  ```
  model database in use: cannot create new database
  ```
- Each time you allocate space on a database device with create database or alter database, that allocation represents a device fragment, and the allocation is entered as a row in sysusages.
- The maximum number of named segments for a database is 32. Segments are named subsets of database devices available to a particular Adaptive Server. For more information on segments, see the System Administration Guide.

Temporary databases

- You cannot use either with default_location or for proxy_update parameters with the create temporary database command, and doing so generates an error, such as the following two examples:

  ```
  1> create temporary database tbl with default_location
      "remSERVER.mydb.."

  Msg 102, Level 15, State 7:
  Server 'ebi_SUS_AS125x_SUN32', Line 1:
  Incorrect syntax near 'create temporary database'.

  1> create temporary database tbl with default_location
      "remSERVER.mydb.." for proxy_update

  Msg 102, Level 15, State 7:
  Server 'ebi_SUS_AS125x_SUN32', Line 1:
  Incorrect syntax near 'create temporary database'.
  ```
- The temporary status of a database, which is set during the creation of the temporary database, is indicated by value 0x00000100 (256 decimal) of the status3 field of a sysdatabases entry.
- In addition to all options inherited from model, a temporary database, like the system tempdb, has the following database options set:
  - select into/bulkcopy
  - trunc log on chkpt
create database

- As with system tempdb, the guest user is added to the temporary database, and create table permission is granted to PUBLIC.

- Unused pages are not cleared during creation of the temporary database, since a temporary database is re-created every time the server is restarted.

New databases created from model

- Adaptive Server creates a new database by copying the model database.

- You can customize model by adding tables, stored procedures, user-defined datatypes, and other objects, and by changing database option settings. New databases inherit these objects and settings from model.

- To guarantee recoverability, create database must clear every page that was not initialized when the model database was copied. This may take several minutes, depending on the size of the database and the speed of your system.

  If you are creating a database to load a database dump into it, you can use the for load option to skip the page-clearing step. This makes database creation considerably faster.

Ensuring database recoverability

- Back up the master database each time you create a new database. This makes recovery easier and safer if master is damaged.

  Note If you create a database and fail to back up master, you may be able to recover the changes with disk reinit.

- The with override clause allows you to mix log and data segments on a single device. However, for full recoverability, the device or devices specified in log on should be different from the physical device that stores the data. In the event of a hard disk crash, you can recover the database from database dumps and transaction logs.

  You can create a small database on a single device that is used to store both the transaction log and the data, but you must rely on the dump database command for backups.

- The size of the device required for the transaction log varies according to the amount of update activity and the frequency of transaction log dumps. As a rule of thumb, allocate to the log device 10 – 25 percent of the space you allocate to the database itself. It is best to start small, since space allocated to a transaction log device cannot be reclaimed and cannot be used for storing data.
Using the *for load* option

You can use the *for load* option for recovering from media failure or for moving a database from one machine to another, if you have not added to the database with *sp_addsegment*. Use *alter database* for load to create a new database in the image of the database from which the database dump to be loaded was made. For a discussion of duplicating space allocation when loading a dump into a new database, see the *System Administration Guide*.

- When you create a database using the *for load* option, you can run only the following commands in the new database before loading a database dump:
  - `alter database for load`
  - `drop database`
  - `load database`

After you load the database dump into the new database, you can also use some *dbcc* diagnostic commands in the databases. After you issue the online database command, there are no restrictions on the commands you can use.

- A database created with the *for load* option has a status of “don’t recover” in the output from *sp_helpdb*.

Getting information about databases

- To get a report on a database, execute *sp_helpdb*.
- For a report on the space used in a database, use *sp_spaceused*.

Using *with default_location* and *for proxy_update*

Without the *for proxy_update* clause, the behavior of the with *default_location* clause is the same as that provided by *sp_defaultloc*—a default storage location is established for new and existing table creation, but automatic import of proxy table definitions is not done during the processing of create database.

- If *for proxy_update* is specified with no *default_location*, an error is reported.
- When a proxy database is created (using the *for proxy_update* option), Component Integration Services is called upon to:
  - Provide an estimate of the database size required to contain all proxy tables representing the actual tables and views found in the primary server’s database. This estimate is the number of database pages needed to contain all proxy tables and indexes. The estimate is used if no size is specified, and no database devices are specified.
create database

- Create all proxy tables representing the actual tables and views found in the companion server’s database.
- Grant all permissions on proxy tables to public.
- Add the guest user to the proxy database.
- The database status is set to indicate that this database 'Is_A_Proxy’. This status is contained in master.dbo.sysdatabases.status3.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
create database permission defaults to System Administrators, who can transfer it to users listed in the sysusers table of the master database. However, create database permission is often centralized to maintain control over database storage allocation.

If you are creating the sybsecurity database, you must be a System Security Officer.

create database permission is not included in the grant all command.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>create</td>
<td>create database</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also
Commands: alter database, disk init, drop database, dump database, load database, online database

System procedures: sp_changedbowner, sp_diskdefault, sp_helpdb, sp_logdevice, sp_renamedb, sp_spaceused
create default

Description
Specifies a value to insert in a column (or in all columns of a user-defined datatype) if no value is explicitly supplied at insert time.

Syntax
create default [owner.]default_name
as constant_expression

Parameters

- **default_name** is the name of the default. It must conform to the rules for identifiers and cannot be a variable. Specify the owner’s name to create another default of the same name owned by a different user in the current database. The default value for **owner** is the current user.

- **constant_expression** is an expression that does not include the names of any columns or other database objects. It can include global variables and built-in functions that do not reference database objects. Enclose character and date constants in quotes and use a “0x” prefix for binary constants.

Examples

**Example 1** Creates a default called D1 that uses the @@spid global variable:

```
create default D1 as @@spid
```

**Example 2** Defines a default value. Now, bind it to the appropriate column or user-defined datatype with sp_bindefault:

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedflt, "authors.phone"
```

The default takes effect only if there is no entry in the phone column of the authors table. No entry is different from a null value entry. To get the default, issue an insert command with a column list that does not include the column that has the default.

**Example 3** Creates a default value, todays_date, that inserts the current date into the columns to which it is bound:

```
create default todays_date as getdate ()
```

Usage

- Bind a default to a column or user-defined datatype—but not a Adaptive Server-supplied datatype—with sp_bindefault.

- You can bind a new default to a datatype without unbinding the old one. The new default overrides and unbinds the old one.

- create default performs error checking for check constraints before it creates the default.

- To hide the source test of a default, use sp_hidetext.
Restrictions

- You can create a default only in the current database.
- You cannot combine `create default` statements with other statements in a single batch.
- You must drop a default with `drop default` before you create a new one of the same name; you must unbind a default with `sp_unbindefault`, before you drop it.

Datatype compatibility

- Adaptive Server generates an error message when it attempts to insert a default value that is not compatible with the column’s datatype. For example, if you bind a character expression such as “N/A” to an integer column, any insert that does not specify the column value fails.
- If a default value is too long for a character column, Adaptive Server either truncates the string or generates an exception, depending on the setting of the `string_rtruncation` option. For more information, see the `set` command.

Getting information about defaults

- Default definitions are stored in `syscomments`.
- After a default is bound to a column, its object ID is stored in `syscolumns`. After a default is bound to a user-defined datatype, its object ID is stored in `systypes`.
- To rename a default, use `sp_rename`.
- For a report on the text of a default, use `sp_helptext`.

Defaults and rules

- If a column has both a default and a rule associated with it, the default value must not violate the rule. A default that conflicts with a rule cannot be inserted. Adaptive Server generates an error message each time it attempts to insert such a default.

Defaults and nulls

- If a column does not allow nulls, and you do not create a default for the column, when a user attempts to insert a row but does not include a value for that column, the insert fails and Adaptive Server generates an error message.

Table 1-5 illustrates the relationship between the existence of a default and the definition of a column as NULL or NOT NULL.
Table 1-5: Relationship between nulls and column defaults

<table>
<thead>
<tr>
<th>Column null type</th>
<th>No entry, no default</th>
<th>No entry, default exists</th>
<th>Entry is null, no default</th>
<th>Entry is null, default exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>Null inserted</td>
<td>Default value inserted</td>
<td>Null inserted</td>
<td>Null inserted</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>Error, command fails</td>
<td>Default value inserted</td>
<td>Error, command fails</td>
<td>Error, command fails</td>
</tr>
</tbody>
</table>

Specifying a default value in `create table`

- You can define column defaults using the `default` clause of the `create table` statement as an alternative to using `create default`. However, these column defaults are specific to that table; you cannot bind them to other tables. See `create table` and `alter table` for information about integrity constraints.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Use the `default` clause of the `create table` statement to create ANSI SQL-compliant defaults.

Permissions

`create default` permission defaults to the Database Owner, who can transfer it to other users.

Auditing

Values in `event` and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 14    | create       | create default             | • Roles – Current active roles  
|       |              |                            | • Keywords or options – NULL  
|       |              |                            | • Previous value – NULL       
|       |              |                            | • Other information – NULL    
|       |              |                            | • Current value – NULL        
|       |              |                            | • Proxy information – Original login name, if set proxy in effect |

See also

- **Commands**: `alter table`, `create rule`, `create table`, `drop default`, `drop rule`

- **System procedures**: `sp_bindefault`, `sp_help`, `sp_helptext`, `sp_rename`, `sp_unbindefault`
**create encryption key**

**Description**

Creates user-specified passwords on keys. All the information related to keys and encryption is encapsulated by `create encryption key`, which allows you to specify the encryption algorithm and key size, the key's default property, an optional user-specified password to encrypt the key, as well as the use of an initialization vector or padding during the encryption process.

Adaptive Server uses Security Builder Crypto™ for key generation and encryption.

**Syntax**

```
create encryption key [database.[owner.]] keyname [as default]
   [ for algorithm_name ]
   [ with { [ keylength num_bits ]
   [ passwd 'password_phrase' ]
   [ init_vector {NULL | random } ]
   [ pad {NULL | random } ] } ]
```

**Parameters**

`keyname` must be unique in the user’s table, view, and procedure name space in the current database. Specify the `database` name if the key is in another database; specify the `owner` name if this key name is not unique in the database. The default value for `owner` is the current user, and the default value for `database` is the current database. Only the system security officer can create keys for other users.

You cannot create temporary keys with names starting with “#” as the first character.

`as default` allows the system security officer or the key custodian to create a database default key for encryption. The existence of an encryption key on a database default column enables the table creator to specify encryption without using a keyname on `create table`, `alter table` and `select into`. Adaptive Server uses the default key from the same database. The default key may be changed. See `alter encryption key` on page 13.

`for algorithm_name` specifies the algorithm you are using. Advanced Encryption Standard (AES) is the only algorithm supported. AES supports key sizes of 128 bits, 192 bits, and 256 bits and a block size of 16 bytes.

`keylength num_bits` the size, in bits, of the key to be created. For AES, valid key lengths are 128, 192, and 256 bits. The default keylength is 128 bits.
password_phrase

  is a quoted alphanumeric string of up to 255 bytes in length that Adaptive
  Server uses to generate the key used to encrypt the column encryption key
  (the key encryption key).

init_vector random

  specifies use of an initialization vector during encryption. When an
  initialization vector is used by the encryption algorithm, the ciphertext of
  two identical pieces of plaintext are different, which prevents a cryptanalyst
  from detecting patterns of data. Use of an initialization vector can add to the
  security of your data.

  An initialization vector has some performance implications. Index creation,
  and optimized joins and searches, can be performed only on a column whose
  encryption key does not specify an initialization vector.

  The default is to use an initialization vector, that is, init_vector random. Use
  of an initialization vector implies using a cipher block chaining (CBC) mode
  of encryption; setting init_vector null implies the electronic code book (ECB)
  mode.

init_vector null

  omits the use of an initialization vector when encrypting. This makes the
column suitable for supporting an index.

pad null

  is the default. It omits random padding of data. You cannot use padding if
  the column must support an index.

pad random

  data is automatically padded with random bytes before encryption. You can
  use padding instead of an initialization vector to randomize the ciphertext.
  Padding is suitable only for columns whose plaintext length is less than half
  the block length. For the AES algorithm the block length is 16 bytes.

Examples

Example 1  Specifies a 256-bit key called “safe_key” as the database default
key. The system security officer enters:

create encryption key safe_key as default for AES with
  keylength 256

Example 2  Creates a 128-bit key called “salary_key” for encrypting columns
using random padding:

  create encryption key salary_key for AES with
    init_vector null pad random
Example 3  Creates a 192-bit key named “mykey” for encrypting columns using an initialization vector:

    create encryption key mykey for AES with keylength 192
    init_vector random

Example 4  Shows how to use passwords on keys, and the key custodian’s function in setting up encryption. This example creates a key protected by a user-specified password:

    create encryption key key1 with passwd
    'Worlds1Biggest6Secret'

You must enter user-specified passwords that protect keys before accessing a column encrypted by the key. For more information about `set encryption passwd`, see Reference Manual: Commands.

Usage  Adaptive Server does not save the user-specified password. It saves a string of validating bytes known as the “salt” in `sysencryptkeys.eksalt`, which allows Adaptive Server to recognize whether a password used on a subsequent encryption or decryption operation is legitimate for a key. You must supply the password to Adaptive Server before you can access any column encrypted by keyname.

Permissions  The system security officer and the key custodian have implicit permission to create encryption keys. The system security officer may grant that permission to other users.

Default keys can be created by the system security officer and the key custodian only.

See also  Commands  `alter encryption key`, `drop encryption key`, `grant`, `revoke`

Documentation  For information about auditing, see Chapter 6, “Auditing Encrypted Columns,” in the User Guide for Encrypted Columns.
create existing table

**Description**  
Component Integration Services only  
Creates a proxy table, then retrieves and stores metadata from a remote table and places the data into the proxy table. Allows you to map the proxy table to a table, view, or procedure at a remote location.

The preferred method of creating proxy tables is the create proxy_table command, which eliminates the need to define the column definitions.

**Syntax**  
create existing table `table_name` (column_list)
  [on `segment_name`]
  [[external {table | procedure | file | connection_type}] at `pathname`
  [column delimiter "string"]]

**Parameters**  
`table_name`  
specifies the name of the table for which you want to create a proxy table.

`column_list`  
specifies the name of the column list that stores information about the remote table.

`on segment_name`  
specifies the segment that contains the remote table.

`external`  
specifies that the object is a remote object.

`table`  
specifies that the remote object is a table or a view. The default is `external table`.

`procedure`  
specifies that the remote object is a stored procedure.

`file`  
specifies that the remote object is a file.

`connection_type`  
determines whether a remote procedure call uses the current or a separate connection. The valid values are:

- non_transactional – is a separate connection is used to execute the RPC.
- transactional – is the existing connection is used to execute the RPC.

The default behavior is transactional.
create existing table

at _pathname_

specifies the location of the remote object. _pathname_ takes the form:

server_name.dbname.owner.object, where:

- **server_name** (required) – is the name of the server that contains the remote object.
- **dbname** (optional) – is the name of the database managed by the remote server that contains this object.
- **owner** (optional) – is the name of the remote server user that owns the remote object.
- **object** (required) – is the name of the remote table, view, or procedure.

_column delimiter_

used to separate fields within each record when accessing flat files, column delimiters. The column delimiter can be up to 16 bytes long.

_string_

the column delimiter string can be any character sequence, but if the string is longer than 16 bytes, only the first 16 bytes are used. The use of column delimiter for proxy tables mapped to anything but files results in a syntax error.

**Examples**

Example 1 Creates the proxy table authors:

```sql
create existing table authors
{
  au_id  id,
  au_lname  varchar (40) NOT NULL,
  au_fname  varchar (20) NOT NULL,
  phone    char (12),
  address  varchar (40) NULL,
  city     varchar (20) NULL,
  state    char (2) NULL,
  zip      char (5) NULL,
  contract bit
}
at "nhserver.pubs2.dbo.authors"
```

Example 2 Creates the proxy table syb_columns:

```sql
create existing table syb_columns
{
  id     int,
  number smallint,
  colid  tinyint,
  status tinyint,
```
CHAPTER 1 Commands

Example 3 Creates a proxy table named `blurbs` for the `blurbs` table at the remote server `SERVER_A`:

```sql
create existing table blurbs
(
    author_id id not null,
    copy text not null
)
at "SERVER_A.db1.joe.blurbs"
```

Example 4 Creates a proxy table named `rpc1` for the remote procedure named `p1`:

```sql
create existing table rpc1
(
    column_1 int,
    column_2 int
)
external procedure
at "SERVER_A.db1.joe.p1"
```

Usage

- `create existing table` does not create a new table unless the remote object is a file. Instead, Component Integration Services checks the table mapping to confirm that the information in `column_list` matches the remote table, verifies the existence of the underlying object, and retrieves and stores metadata about the remote table.

- If the host data file or remote server object does not exist, the command is rejected with an error message.

- If the object exists, the system tables `sysobjects`, `syscolumns`, and `sysindexes` are updated. The verification operation requires these steps:
create existing table

a The nature of the existing object is determined. For host data files, this requires determining file organization and record format. For remote server objects, this requires determining whether the object is a table, a view, or an RPC.

b For remote server objects (other than RPCs), column attributes obtained for the table or view are compared with those defined in the column_list.

c Index information from the host data file or remote server table is extracted and used to create rows for the sysindexes system table. This defines indexes and keys in Adaptive Server terms and enables the query optimizer to consider any indexes that might exist on this table.

- The on segment_name clause is processed locally and is not passed to a remote server.

- After successfully defining an existing table, issue update statistics for the table. This allows the query optimizer to make intelligent choices regarding index selection and join order.

- Component Integration Services allows you to create a proxy table with a column defined as NOT NULL even though the remote column is defined as NULL. It displays a warning to notify you of the mismatch.

- The location information provided by the at keyword is the same information that is provided by sp_addobjectdef. The information is stored in the sysattributes table.

- Component Integration Services inserts or updates a record in the systabstats catalog for each index of the remote table. Since detailed structural statistics are irrelevant for remote indexes, only a minimum number of columns are set in the systabstats record—id, indid, and rowcnt.

- External files cannot be of datatypes text, image, or Java ADTs.

Datatype conversions

- When using create existing table, you must specify all datatypes with recognized Adaptive Server datatypes. If the remote server tables reside on a class of server that is heterogeneous, the datatypes of the remote table are automatically converted into the specified Adaptive Server types when the data is retrieved. If the conversion cannot be made, Component Integration Services does not allow the table to be defined.
• The Component Integration Services User’s Guide contains a section for each supported server class and identifies all possible datatype conversions that are implicitly performed by Component Integration Services.

Changes by server class
• All server classes allow you to specify fewer columns than there are in the table on the remote server.
• All server classes match the columns by name.
• All server classes allow the column type to be any datatype that can be converted to and from the datatype of the column in the remote table.

Remote procedures
• When the proxy table is a procedure-type table, you must provide a column list that matches the description of the remote procedure’s result set. create existing table does not verify the accuracy of this column list.
• No indexes are created for procedures.
• Component Integration Services treats the result set of a remote procedure as a virtual table that can be sorted, joined with other tables, or inserted into another table using insert or select. However, a procedure type table is considered read-only, which means you cannot issue the following commands against the table:
  • alter table
  • create index
  • delete
  • insert
  • truncate table
  • update
• Begin the column name with an underscore (_) to specify that the column is not part of the remote procedure’s result set. These columns are referred to as parameter columns. For example:

```sql
create existing table rpc1
(
    a int,
    b int,
    c int,
    _p1 int null,
    _p2 int null
)
```
In this example, the parameter columns \_p1 and \_p2 are input parameters. They are not expected in the result set, but can be referenced in the query:

```
    select a, b, c from t1
    where \_p1 = 10 and \_p2 = 20
```

Component Integration Services passes the search arguments to the remote procedure as parameters, using the names @p1 and @p2.

- Parameter-column definitions in a create existing table statement:
  - Must allow a null value.
  - Cannot precede regular result columns—they must appear at the end of the column list.

- If a parameter column is included in a select list and is passed to the remote procedure as a parameter, the return value is assigned by the where clause.

- If a parameter column is included in a select list, but does not appear in the where clause or cannot be passed to the remote procedure as a parameter, its value is NULL.

- A parameter column can be passed to a remote procedure as a parameter if the Adaptive Server query processor considers it a searchable argument. A parameter column is considered a searchable argument if it is not included in any or predicates. For example, the or predicate in the second line of the following query prevents the parameter columns from being used as parameters:

```
    select a, b, c from t1
    where \_p1 = 10 or \_p2 = 20
```

Encrypted Columns

create existing table automatically updates syscolumns with any encrypted column metadata from the remote table. You cannot include the encrypt keyword in the column list for a create existing table command.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

create existing table permission defaults to the table owner and is not transferable.

See also

Commands  alter table, create table, create proxy_table, drop index, insert, order by clause, set, update
create function

Description

Creates a user-defined function, which is a saved Transact-SQL routine that returns a specified value.

Syntax

create function [ {owner_name} ] function_name
( [ { [ @parameter_name [as] parameter_datatype [ = default ] } ]
[ ,...n ] ]
) returns return_datatype
[ with recompile ]
as
[begin]
  function_body
  return scalar_expression
[end]

Parameters

owner_name

is the name of the user ID that owns the user-defined function. Must be an existing user ID.

function_name

is the name of the user-defined function. Function names must conform to the rules for identifiers and must be unique within the database and to its owner. Function names cannot be the same as other Adaptive Server functions.

Note

To reference or invoke a user-defined function, specify the owner_name function_name, followed by parentheses (see the BONUS function in the “Examples” section below for an example). Specify expressions as arguments for all the parameters within the parentheses. You cannot specify the parameter names in the argument list when you invoke a function. You must supply argument values for all of the parameters, and the argument values must be in the same sequence in which the parameters are defined in the create function statement. When a function’s parameter has a default value, you must specify the keyword “default” when calling the function to get the default value.
create function

@parameter_name

is the parameter in the user-defined function. You can declare one or more parameters in a create function statement. A function can have a maximum of 2,047 parameters. The value of each declared parameter must be supplied by the user when the function is executed, unless you define a default for the parameter.

Specify a parameter name using an “at” sign (@) as the first character. The parameter name must conform to the rules for identifiers. Parameters are local to the function. You can use the same parameter names in other functions.

If a parameter has a default value, the user must specify the keyword “default” when they call the function to get the default value.

parameter_datatype

is the data type of the parameter. All scalar data types and Java abstract datatypes (ADTs) can be used as a parameter for user-defined functions. However, user-defined functions do not support the timestamp, text, image and unitext.

with recompile

indicates that Adaptive Server never saves a plan for this function; instead, a new plan is created each time the function is first referenced in a SQL statement. Use with recompile when you expect the execution of this function is atypical, and it will require a new plan.

return_datatype

is the return value of a scalar, user-defined function. It can be any of the scalar data types and Java ADTs except text, image, unitext and timestamp.

scalar_expression

specifies the scalar value the scalar function returns.

You can invoke scalar-valued functions where scalar expressions are used, including computed columns and check constraint definitions.

function_body

specifies a series of T-SQL statements, which together do not produce a side effect but define the value of the function. function_body is used only in scalar functions and multi-statement table-valued functions. In scalar functions, function_body is a series of T-SQL statements that evaluate to a scalar value.

Examples

Creates a user-defined function named bonus:

create function BONUS(@salary int, @grade int, @dept_id int)
returns int
as
begin
    declare @bonus int
    declare @cat int
    set @bonus = 0
    select  @cat = dept_cat from department
      where dept_id = @dept_id

    if (@cat < 10)
      begin
        set @bonus = @salary *15/100
      end
    else
      begin
        set @bonus = @salary * 10/100
      end
    return @bonus
end

Usage

- If the owner of the user-defined function also owns all the database objects referenced inside, then all the other users who have execute permission on the function are automatically granted access permissions to all the referenced objects when they execute the function.

- When a function is created, Adaptive Server checks to see if it is a SQL user-defined function or a SQLJ user-defined function. If it is the latter, Adaptive Server checks for “sa” permissions. If it is a SQL function Adaptive Server checks for create function privileges.

Permissions

create function permission defaults to the database owner, who can transfer it to other users.

Owners of functions have execute permission on their functions. Other users do not have execute permissions unless execute permissions on the specific function are granted to them.

Auditing

See also
create function (SQLJ)

Description
Creates a user-defined function by adding a SQL wrapper to a Java static method. Can return a value defined by the method.

Syntax
```
create function [owner.]sql_function_name
    ([ sql_parameter_name sql_datatype
        [((length)| (precision[, scale ]))]
        [, sql_parameter_name sql_datatype
        [((length)| (precision[, scale ]))]
        ...]])
    returns sql_datatype
        [((length)| (precision[, scale ]))]
    [modifies sql data]
    [returns null on null input |
    called on null input]
    [deterministic | not deterministic]
    [exportable]
    language java
    parameter style java
    external name 'java_method_name
        [([java_datatype[, java_datatype
        ...]])]'
```

Parameters
- `sql_function_name` is the Transact-SQL name of the function. It must conform to the rules for identifiers and cannot be a variable.

- `sql_parameter_name` is the name of an argument to the function. The value of each input parameter is supplied when the function is executed. Parameters are optional; a SQLJ function need not take arguments. Parameter names must conform to the rules for identifiers. If the value of a parameter contains non-alphanumeric characters, it must be enclosed in quotes. This includes object names qualified by a database name or owner name, since they include a period. If the value of the parameter begins with a numeric character, it also must be enclosed in quotes.

- `sql_datatype` is the Transact-SQL datatype of the parameter. See create procedure on page 129 for more information about these parameters.

- `returns sql_datatype` specifies the result datatype of the function.
modifies sql data
indicates that the Java method invokes SQL operations, reads, and modifies
SQL data in the database. This is the default and only implementation. It is
included for syntactic compatibility with the ANSI standard.

deterministic | not deterministic
included for syntactic compatibility with the ANSI standard. Not currently
implemented.

exportable
specifies that the procedure is to be run on a remote server using the
Adaptive Server OmniConnect™ feature. Both the procedure and the
method it is built on must reside on the remote server.

language java
specifies that the external routine is written in Java. This is a required clause
for SQLJ functions.

parameter style java
specifies that the parameters passed to the external routine at runtime are
Java parameters. This is a required clause for SQLJ functions.

external
indicates that create function defines a SQL name for an external routine
written in a programming language other than SQL.

name
specifies the name of the external routine (Java method). The specified
name—java_method_name [ java_datatype[,...]]—is a
character-string literal and must be enclosed in single quotes.

java_method_name
specifies the name of the external Java method.

java_datatype
specifies a Java datatype that is mappable or result-set mappable. This is the
Java method signature.

Examples
This example creates a function square_root that invokes the
java.lang.Math.sqrt() method:

```
create function square_root
  (input_number double precision) returns
double precision
  language java parameter style java
  external name 'java.lang.Math.sqrt'
```
create function (SQLJ)

Usage

- You cannot create a SQLJ function with the same name as an Adaptive Server built-in function.
- You can create user-defined functions (based on Java static methods) and SQLJ functions with the same class and method names.

**Note** Adaptive Server searching order ensures that the SQLJ function is always found first.

- You can include a maximum of 31 parameters in a create function statement.
- When a function is created, Adaptive Server checks to see if it is a SQL user-defined function or a SQLJ user-defined function. If it is the latter, Adaptive Server checks for “sa” permissions. If it is a SQL function Adaptive Server checks for create function privileges.

Permissions

Only the Database Owner or user with sa role can execute create function. The Database Owner or sa cannot transfer permission for create function.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 97    | install      | create function           | • *Roles* – Current active roles  
• *Keywords or options* – NULL  
• *Previous value* – NULL  
• *Current value* – NULL  
• *Other information* – NULL  
• *Proxy information* – Original login name, if a set proxy is in effect |

See also

See *Java in Adaptive Server Enterprise* for more information about create function.

Commands  
create function (SQLJ), drop function (SQLJ)

System procedures  
sp_depends, sp_help, sp_helpjava, sp_helpprotect
create index

Description

Creates an index on one or more columns in a table, computed or non-computed. Creates partitioned indexes.

Allows computed columns, like ordinary columns, to be index keys, and creates function-based indexes. A function-based index has one or more expressions as its index key.

The existing create index syntax can create indexes on computed columns, but some new syntax is needed to create function-based indexes.

Syntax

create [unique] [clustered | nonclustered] index index_name
on [[database.]owner.]table_name
  (column_expression [asc | desc]
  [, column_expression [asc | desc]]...)
  [with {fillfactor = pct,
    max_rows_per_page = num_rows,
    reservepagegap = num_pages,
    consumers = x, ignore_dup_key, sorted_data,
    [ignore_dup_row | allow_dup_row],
    statistics using num_steps values}]
  [on segment_name]
  [index_partition_clause]

Syntax to create index partitions

index_partition_clause::= 
  [local index [partition_name [on segment_name]
  [, partition_name [on segment_name]...]]]

Syntax for function-based indexes

create [unique | nonclustered] index index_name
on [[database.] owner.] table_name
  (column_expression [asc | desc]
  [, column_expression [asc | desc]]...
create index

Parameters

unique

prohibits duplicate index values (also called “key values”). The system checks for duplicate key values when the index is created (if data already exists), and each time data is added with an insert or update. If there is a duplicate key value or if more than one row contains a null value, the command fails, and Adaptive Server prints an error message giving the duplicate entry.

Warning! Adaptive Server does not detect duplicate rows if a table contains any non-null text, unitext, or image columns.

update and insert commands, which generate duplicate key values, can succeed if you create your index using the allow_dup_row option.

Composite indexes (indexes in which the key value is composed of more than one column) can also be unique.

The default is nonunique. To create a nonunique clustered index on a table that contains duplicate rows, specify allow_dup_row or ignore_dup_row. See “Duplicate rows” on page 119.

When you create a unique local index on range-, list-, and hash-partitioned tables, the index key list is a superset of the partition-key list.

clustered

means that the physical order of rows on the current database device is the same as the indexed order of the rows. The bottom, or leaf level, of the clustered index contains the actual data pages. A clustered index almost always retrieves data faster than a nonclustered index. Only one clustered index per table is permitted. See “Creating clustered indexes” on page 118.

If clustered is not specified, nonclustered is assumed.

nonclustered

means that the physical order of the rows is not the same as their indexed order. The leaf level of a nonclustered index contains pointers to rows on data pages. You can have as many as 249 nonclustered indexes per table.

index_name

is the name of the index. Index names must be unique within a table, but need not be unique within a database.
`table_name`

is the name of the table in which the indexed column or columns are located. Specify the database name if the table is in another database, and specify the owner’s name if more than one table of that name exists in the database. The default value for `owner` is the current user, and the default value for `database` is the current database.

`column_expression`

is a valid Transact-SQL expression that references at least one base column, and does not contain columns from other tables, local and global variables, aggregate functions, or subqueries.

Note `column_expression` replaces the `column_name` variable used in Adaptive Server before version 15.0.

`asc | desc`

specifies whether the index is to be created in ascending or descending order for the column specified. The default is ascending order.
fillfactor

specifies how full Adaptive Server makes each page when it creates a new index on existing data. The fillfactor percentage is relevant only when the index is created. As the data changes, the pages are not maintained at any particular level of fullness.

The value you specify is not saved in sysindexes for display by sp_helpindex or for later use by the reorg command. Use sp_chgattribute to create stored fillfactor values.

The default for fillfactor is 0; this is used when you do not include with fillfactor in the create index statement (unless the value has been changed with sp_configure). When specifying a fillfactor, use a value between 1 and 100.

A fillfactor of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes. There is seldom a reason to change the fillfactor.

If the fillfactor is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A fillfactor of 100 makes sense only for read-only tables—tables to which no data is ever added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A fillfactor of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small fillfactor values cause each index (or index and data) to occupy more storage space.

---

**Warning!** Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.
max_rows_per_page

limits the number of rows on data pages and the leaf-level pages of indexes. Unlike fillfactor, the max_rows_per_page value is maintained until it is changed with sp_chgattribute.

If you do not specify a value for max_rows_per_page, Adaptive Server uses a value of 0 when creating the table. Values for tables and clustered indexes range from 0 to 183K on a 2K page, to 0 to 1486 on a 16K page.

The maximum number of rows per page for nonclustered indexes depends on the size of the index key. Adaptive Server returns an error message if the specified value is too high.

A max_rows_per_page value of 0 creates clustered indexes with full pages and nonclustered indexes with full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes.

If max_rows_per_page is set to 1, Adaptive Server creates both clustered and nonclustered indexes with one row per page at the leaf level. Use low values to reduce lock contention on frequently accessed data. However, low max_rows_per_page values cause Adaptive Server to create new indexes with pages that are not completely full, uses more storage space, and may cause more page splits.

If Component Integration Services is enabled, you cannot use max_rows_per_page for remote servers.

with reservepagegap = num_pages

specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations. For each specified num_pages, an empty page is left for future expansion of the index. Valid values are 0 – 255. The default is 0.

with consumers

specifies the number of consumer processes that should perform the sort operation for creating the index. The actual number of consumer processes used to sort the index may be different than the specified number depending on the number of worker processes available and the number of data partitions.
ignore_dup_key

cancels attempts of duplicate key entry into a table that has a unique index (clustered or nonclustered). Adaptive Server cancels the attempted insert or update of a duplicate key with an informational message. After the cancellation, the transaction containing the duplicate key proceeds to completion.

You cannot create a unique index on a column that includes duplicate values or more than one null value, whether or not ignore_dup_key is set. If you attempt to do so, Adaptive Server prints an error message that displays the first of the duplicate values. You must eliminate duplicates before Adaptive Server can create a unique index on the column.

ignore_dup_row

allows you to create a new, nonunique clustered index on a table that includes duplicate rows. ignore_dup_row deletes the duplicate rows from the table, and cancels any insert or update that would create a duplicate row, but does not roll back the entire transaction. See “Duplicate rows” on page 119 for more information.

allow_dup_row

allows you to create a nonunique clustered index on a table that includes duplicate rows, and allows you to duplicate rows with update and insert statements. See “Duplicate rows” on page 119 for an explanation of how to use these options.

sorted_data

speeds creation of clustered indexes or unique nonclustered indexes when the data in the table is already in sorted order (for example, when you have used bcp to copy data that has already been sorted into an empty table). See “Using the sorted_data option to speed sorts” on page 121 for more information.
with statistics using `num_steps` values specifies the number of steps to generate for the histogram used to optimize queries. If you omit this clause:

- The default value is 20, if no histogram is currently stored for the leading index column.
- The current number of steps is used, if a histogram for the leading column of the index column already exists.

If you specify 0 for `num_steps`, the index is re-created, but the statistics for the index are not overwritten in the system tables.

The actual number of steps may differ from the one you specify; if the histogram steps specified with `num_steps` is $M$, and the histogram tuning factor parameter is $N$, then the actual steps are between $M$ and $M*N$, depending on the number of frequency cells that exist in the distribution.

`on segment_name` creates the index on the named segment. Before using the `on segment_name` option, initialize the device with `disk init`, and add the segment to the database using `sp_addsegment`. See your System Administrator, or use `sp_helpsegment` for a list of the segment names available in your database.

There are two locations where you can use `on segment_name`:

- Immediately before the `index_partition_clause` – defines a global default which will be used for all partitions where the segment is not explicitly defined in the `index_partition_clause`
- Within that clause itself – allows you to specify a segment for each individual partition

See the examples section for an example that uses `on segment_name` in both locations.

`local index` specifies, for semantically partitioned tables, an index that is always equipartitioned with its base table; that is, the table and index share the same partitioning key and partitioning criteria. For round-robin partitioned tables, a local index means that index keys in each of its index partitions refer to data rows in one and only one table partition.

For both semantically partitioned tables and round-robin partitioned tables, each table partition has only one corresponding index partition.
**partition_name** specifies the name of a new partition on which indexes are to stored. Partition names must be unique within the set of partitions on a table or index. Partition names can be delimited identifiers if set quoted_identifier is on. Otherwise, they must be valid identifiers.

If **partition_name** is omitted, Adaptive Server creates a name in the form `table_name_partition_id`. Adaptive Server truncates partition names that exceed the allowed maximum length.

---

**Example 1** Creates an index named `au_id_ind` on the `au_id` column of the `authors` table:

```sql
create index au_id_ind on authors (au_id)
```

**Example 2** Creates a unique clustered index named `au_id_ind` on the `au_id` column of the `authors` table:

```sql
create unique clustered index au_id_ind on authors (au_id)
```

**Example 3** Creates an index named `ind1` on the `au_id` and `title_id` columns of the `titleauthor` table:

```sql
create index ind1 on titleauthor (au_id, title_id)
```

**Example 4** Creates a nonclustered index named `zip_ind` on the `zip` column of the `authors` table, filling each index page one-quarter full and limiting the sort to 4 consumer processes:

```sql
create nonclustered index zip_ind on authors (postalcode) with fillfactor = 25, consumers = 4
```

**Example 5** Creates an index with ascending ordering on `pub_id` and descending order on `pubdate`:

```sql
create index pub_dates_ix on titles (pub_id asc, pubdate desc)
```

**Example 6** Creates an index on `title_id`, using 50 histogram steps for optimizer statistics and leaving 1 empty page out of every 40 pages in the index:

```sql
create index title_id_ix on titles (title_id) with reservepagegap = 40, statistics using 50 values
```

**Example 7** Creates a local, clustered index on a partitioned `salesdetail` table. The `clust_idx` index inherits the partition strategy, partition key, and partition bounds of `salesdetail`. 
Example 8  Creates a nonpartitioned, nonclustered global index on a partitioned sales table, which is partitioned by range on the date column.

```sql
create nonclustered index global_idx
on sales (order_num)
```

Example 9  First, creates a table, pback_sales, with three data partitions:

```sql
create table pback_sales (c1 int, c2 int,
c3 varchar (20)) partition range (c1)
(p1 c1 values <= (10),
p2 c1 values <= (20),
p3 c1 values <= (MAX))
```

Then, creates a local, function-based index on partition p1:

```sql
create index fc_idx on pback_sales (c1*c2) local index p1
```

Example 10  Creates a function-based index:

```sql
create index sum_sales on mytitles (price * total_sales)
```

Example 11  Specifies the on segment_name clause both before and after the partition name:

```sql
use tempdb
go
if not exists(select 1 from tempdb..syssegments where name = 'seg1')
    exec sp_addsegment seg1,tempdb,master
    go
if not exists(select 1 from tempdb..syssegments where name = 'seg2')
    exec sp_addsegment seg2,tempdb,master
    go
if not exists(select 1 from tempdb..syssegments where name = 'seg3')
    exec sp_addsegment seg3,tempdb,master
    go
if not exists(select 1 from tempdb..syssegments where name = 'seg4')
    exec sp_addsegment seg4,tempdb,master
    go
if exists(select 1 from sysobjects where name = 't1')
    drop table t1
    go
create table t1 (a int, b varchar(30)) partition by roundrobin (p1 on seg1, p2 on seg2)
    go
create index t1_i1 on t1 (a) local index
    go
create index t1_i2 on t1 (a) on seg3 local index ip1 on seg4
    go
```
sp_help t1
go

Provides the following output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Object_type</th>
<th>Create_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>dbo</td>
<td>user table</td>
<td>Aug 7 2008 11:14AM</td>
</tr>
</tbody>
</table>

(1 row affected)

<table>
<thead>
<tr>
<th>Column_name</th>
<th>Type</th>
<th>Length</th>
<th>Prec</th>
<th>Scale</th>
<th>Nulls</th>
<th>Default_name</th>
<th>Rule_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>int</td>
<td>4</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>b</td>
<td>varchar</td>
<td>30</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Object has the following indexes

<table>
<thead>
<tr>
<th>index_name</th>
<th>index_keys</th>
<th>index_description</th>
<th>index_max_rows_per_page</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1_i1</td>
<td>a</td>
<td>nonclustered</td>
<td>0</td>
</tr>
<tr>
<td>t1_i2</td>
<td>a</td>
<td>nonclustered</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows affected)

<table>
<thead>
<tr>
<th>index_ptn_name</th>
<th>index_ptn_seg</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1_i1_952063116</td>
<td>default</td>
</tr>
<tr>
<td>t1_i1_968063173</td>
<td>default</td>
</tr>
<tr>
<td>ip1</td>
<td>seg4</td>
</tr>
<tr>
<td>t1_i2_1000063287</td>
<td>seg3</td>
</tr>
</tbody>
</table>

(4 rows affected)

No defined keys for this object.

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>partition_type</th>
<th>partitions</th>
<th>partition_keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>base table</td>
<td>roundrobin</td>
<td>2</td>
<td>NULL</td>
</tr>
</tbody>
</table>

(1 row affected)

<table>
<thead>
<tr>
<th>partition_name</th>
<th>partition_id</th>
<th>pages</th>
<th>row_count</th>
<th>segment</th>
<th>create_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>920063002</td>
<td>1</td>
<td>0</td>
<td>seg1</td>
<td>Aug 7 2008 11:14AM</td>
</tr>
<tr>
<td>p2</td>
<td>936063059</td>
<td>1</td>
<td>0</td>
<td>seg2</td>
<td>Aug 7 2008 11:14AM</td>
</tr>
</tbody>
</table>

Partition Conditions
NULL

<table>
<thead>
<tr>
<th>Avg_pages</th>
<th>Max_pages</th>
<th>Min_pages</th>
<th>Ratio(Max/Avg)</th>
<th>Ratio(Min/Avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not applicable to tables with allpages lock scheme.

<table>
<thead>
<tr>
<th>exp_row_size</th>
<th>reservepagegap</th>
<th>fillfactor</th>
<th>max_rows_per_page</th>
<th>identity_gap</th>
<th>ascinserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(1 row affected)

<table>
<thead>
<tr>
<th>concurrency_opt_threshold</th>
<th>optimistic_index_lock</th>
<th>dealloc_first_txtpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(1 row affected)
(return status = 0)

Usage

- Run update statistics periodically if you add data to the table that changes the distribution of keys in the index. The query optimizer uses the information created by update statistics to select the best plan for running queries on the table.

- If the table contains data when you create a nonclustered index, Adaptive Server runs update statistics on the new index. If the table contains data when you create a clustered index, Adaptive Server runs update statistics on all the table’s indexes.

- Index all columns that are regularly used in joins.

- When Component Integration Services is enabled, the create index command is reconstructed and passed directly to the Adaptive Server associated with the table.

create index and stored procedures

Adaptive Server automatically recompiles stored procedures after executing create index statements. Although adhoc queries that you start before executing create index still continue to work, they do not take advantage of the new index.

In Adaptive Server versions 12.5 and earlier, create index was ignored by cached stored procedures.
Creating indexes efficiently

- Indexes speed data retrieval, but can slow data updates. For better performance, create a table on one segment and create its nonclustered indexes on another segment, when the segments are on separate physical devices.

- Adaptive Server can create indexes in parallel if a table is partitioned and the server is configured for parallelism. It can also use sort buffers to reduce the amount of I/O required during sorting. For more information, see Chapter 9, “Parallel Sorting,” in the Performance and Tuning Guide: Optimizer and Abstract Plans.

- Create a clustered index before creating any nonclustered indexes, since nonclustered indexes are automatically rebuilt when a clustered index is created.

- When using parallel sort for data-only-locked tables, the number of worker processes must be configured to equal or exceed the number of partitions, even for empty tables. The database option select into/bulkcopy/plsort must also be enabled.

Creating clustered indexes

- A table “follows” its clustered index. When you create a table, use the on segment_name extension to create clustered index, the table migrates to the segment where the index is created.

  If you create a table on a specific segment, then create a clustered index without specifying a segment, Adaptive Server moves the table to the default segment when it creates the clustered index there.

  Because text, unitext, and image data is stored in a separate page chain, creating a clustered index with on segment_name does not move text and image columns.

- To create a clustered index, Adaptive Server duplicates the existing data; the server deletes the original data when the index is complete. Before creating a clustered index, use sp_spaceused to make sure that the database has at least 120 percent of the size of the table available as free space.

- The clustered index is often created on the table’s primary key (the column or columns that uniquely identify the row). You can record the primary key in the database (for use by front-end programs and sp_depends) using sp_primarykey.

- To allow duplicate rows in a clustered index, specify allow_dup_row.
Creating indexes on encrypted columns

You can create an index on an encrypted column if you specify the encryption key without any initialization vector or random padding. Adaptive Server issues an error if you execute `create index` on an encrypted column with an initialization vector or random padding. Indexes on encrypted columns are useful for equality and non-equality matches, but are not useful for matching case-insensitive data, or for range searches of any data.

**Note** You cannot use an encrypted column in an expression for a functional index

Specifying ascending or descending ordering in indexes

- Use the `asc` and `desc` keywords after index column names to specify the sorting order for the index keys. Creating indexes so that columns are in the same order specified in the `order by` clause of queries eliminates the sorting step during query processing. For more information, see Chapter 6, “Indexing for Performance,” in the *Performance and Tuning Guide: Locking*.

Space requirements for indexes

- Space is allocated to tables and indexes in increments of one extent, or eight pages, at a time. Each time an extent is filled, another extent is allocated. Use `sp_spaceused` to display the amount of space allocated and used by an index.

- In some cases, using the `sorted_data` option allows Adaptive Server to skip copying the data rows as described in Table 1-8 on page 121. In these cases, you need only enough additional space for the index structure itself. Depending on key size, this is usually about 20 percent of the size of the table.

Duplicate rows

- The `ignore_dup_row` and `allow_dup_row` options are not relevant when you are creating a nonunique, nonclustered index. Adaptive Server attaches a unique row identification number internally in each nonclustered index; duplicate rows are not a problem even for identical data values.

- `ignore_dup_row` and `allow_dup_row` are mutually exclusive.

- In all-pages-locked tables, nonunique clustered index allows duplicate keys, but does not allow duplicate rows unless you specify `allow_dup_row`. This behavior differs for data-only-locked tables, and is described in detail on Table 1-6.
allow_dup_row allows you to create a nonunique, clustered index on a table that includes duplicate rows. If a table has a nonunique, clustered index that was created without the allow_dup_row option, you cannot create new duplicate rows using the insert or update command.

If any index in the table is unique, the requirement for uniqueness takes precedence over the allow_dup_row option. You cannot create an index with allow_dup_row if a unique index exists on any column in the table.

The ignore_dup_row option is also used with a nonunique, clustered index. The ignore_dup_row option eliminates duplicates from a batch of data. ignore_dup_row cancels any insert or update that would create a duplicate row, but does not roll back the entire transaction.

Table 1-6 illustrates how allow_dup_row and ignore_dup_row affect attempts to create a nonunique, clustered index on a table that includes duplicate rows and attempts to enter duplicate rows into a table.

<table>
<thead>
<tr>
<th>Option setting</th>
<th>Create an index on a table that has duplicate rows</th>
<th>Insert duplicate rows into a table with an index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither option set</td>
<td>create index fails.</td>
<td>insert fails.</td>
</tr>
<tr>
<td>allow_dup_row set</td>
<td>create index completes.</td>
<td>insert completes.</td>
</tr>
<tr>
<td>ignore_dup_row set</td>
<td>Index is created but duplicate rows are deleted; error message.</td>
<td>All rows are inserted except duplicates; error message.</td>
</tr>
</tbody>
</table>

Table 1-7 shows which index options you can use with the different types of indexes:

<table>
<thead>
<tr>
<th>Index type</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustered</td>
<td>ignore_dup_row</td>
</tr>
<tr>
<td>Unique, clustered</td>
<td>ignore_dup_key</td>
</tr>
<tr>
<td>Nonclustered</td>
<td>None</td>
</tr>
<tr>
<td>Unique, nonclustered</td>
<td>ignore_dup_key</td>
</tr>
</tbody>
</table>

Using unique constraints in place of indexes

As an alternative to create index, you can implicitly create unique indexes by specifying a unique constraint with the create table or alter table statement. The unique constraint creates a clustered or nonclustered unique index on the columns of a table. These implicit indexes are named after the constraint, and they follow the same rules for indexes created with create index.
• You cannot drop indexes supporting unique constraints using the drop index statement. They are dropped when the constraints are dropped through an alter table statement or when the table is dropped. See create table for more information about unique constraints.

Using the sorted_data option to speed sorts

• The sorted_data option can reduce the time needed to create an index by skipping the sort step and by eliminating the need to copy the data rows to new pages in certain cases. The speed increase becomes significant on large tables and increases to several times faster in tables larger than 1GB.

If sorted_data is specified, but data is not in sorted order, Adaptive Server displays an error message, and the command fails.

Creating a nonunique, nonclustered index succeeds, unless there are rows with duplicate keys. If there are rows with duplicate keys, Adaptive Server displays an error message, and the command fails.

• The effects of sorted_data for creating a clustered index depend on whether the table is partitioned and whether certain other options are used in the create index command. Some options require data copying, if used at all, for nonpartitioned tables and sorts plus data copying for partitioned tables, while others require data copying only if you use:

  • The ignore_dup_row option
  • The fillfactor option
  • The on segmentname clause to specify a segment that is different from the segment where the table data is located
  • The max_rows_per_page clause to specify a value that is different from the value associated with the table

• Table 1-8 shows when the sort is required and when the table is copied for partitioned and nonpartitioned tables.

<table>
<thead>
<tr>
<th>Options</th>
<th>Partitioned table</th>
<th>Unpartitioned table</th>
</tr>
</thead>
<tbody>
<tr>
<td>No options specified</td>
<td>Parallel sort necessary only for creating a clustered index on a round-robin partitioned table; copies data, distributing evenly on partitions; creates index tree.</td>
<td>Either parallel or nonparallel sort; copies data, creates index tree.</td>
</tr>
<tr>
<td>with sorted_data only or with</td>
<td>Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.</td>
<td>Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.</td>
</tr>
<tr>
<td>sorted_data on same_segment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Specifying the number of histogram steps

- Use the with statistics clause to specify the number of steps for a histogram for the leading column of an index. Histograms are used during query optimization to determine the number of rows that match search arguments for a column.

- To re-create an index without updating the values in sysstatistics for a column, use 0 for the number of steps. This avoids overwriting statistics that have been changed with optdiag.

- If you specify the histogram tuning factor parameter with a value, then create index uses anywhere between 20 and M*20 steps, depending on the number of frequency cells that have been isolated. The default is 20, but you can specify a different number with the using step values option.

Space management properties

- fillfactor, max_rows_per_page, and reservepagegap help manage space on index pages in different ways:
  - fillfactor applies to indexes for all locking schemes. For clustered indexes on allpages-locked tables, it affects the data pages of the table. On all other indexes, it affects the leaf level of the index.
  - max_rows_per_page applies only to index pages of allpages-locked tables.
  - reservepagegap applies to tables and indexes for all locking schemes.

- reservepagegap affects space usage in indexes when:
  - The index is created.
  - reorg commands on indexes are executed.
  - Nonclustered indexes are rebuilt after creating a clustered index.

- When a reservepagegap value is specified in a create clustered index command, it applies to:
  - The data and index pages of allpages-locked tables
  - Only the index pages of data-only-locked tables
The **num_pages** value specifies a ratio of filled pages to empty pages on the leaf level of the index so that indexes can allocate space close to existing pages, as new space is required. For example, a **reservepagegap** of 10 leaves 1 empty page for each 9 used pages.

**reservepagegap** specified along with create clustered index on an allpages-locked table overwrites any value previously specified with create table or alter table.

You can change the space management properties for an index with **sp_chgattribute**. Changing properties with **sp_chgattribute** does not immediately affect storage for indexes on the table. Future large scale allocations, such as reorg rebuild, use the **sp_chgattribute** value.

The **fillfactor** value set by **sp_chgattribute** is stored in the **fill_factor** column in **sysindexes**. The **fillfactor** is applied when an index is re-created as a result of an alter table...lock command or a reorg rebuild command.

**Index options and locking modes**

- **Table 1-9** shows the index options supported for allpages-locked and data-only-locked tables. On data-only-locked tables, the **ignore_dup_row** and **allow_dup_row** options are enforced during create index, but are not enforced during insert and update operations. Data-only-locked tables always allow the insertion of duplicate rows.

<table>
<thead>
<tr>
<th>Index type</th>
<th>Allpages-locked table</th>
<th>Data-only-locked table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>During index creation</td>
<td>During inserts</td>
</tr>
<tr>
<td>Clustered</td>
<td>allow_dup_row, ignore_dup_row</td>
<td>allow_dup_row, ignore_dup_row</td>
</tr>
<tr>
<td>Unique clustered</td>
<td>ignore_dup_key</td>
<td>ignore_dup_key</td>
</tr>
<tr>
<td>Nonclustered</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Unique nonclustered</td>
<td>ignore_dup_key</td>
<td>ignore_dup_key</td>
</tr>
</tbody>
</table>

**Table 1-10** shows the behavior of commands that attempt to insert duplicate rows into tables with clustered indexes, and when the clustered indexes are dropped and re-created.

<table>
<thead>
<tr>
<th>Options</th>
<th>Allpages-locked table</th>
<th>Data-only-locked table</th>
</tr>
</thead>
<tbody>
<tr>
<td>No options specified</td>
<td>Insert fails with error message 2615. Re-creating the index succeeds.</td>
<td>Insert succeeds. Re-creating the index fails with error message 1508.</td>
</tr>
<tr>
<td>allow_dup_row</td>
<td>Insert and re-creating the index succeed.</td>
<td>Insert and re-creating the index succeed.</td>
</tr>
</tbody>
</table>
create index

<table>
<thead>
<tr>
<th>Options</th>
<th>Allpages-locked table</th>
<th>Data-only-locked table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignore_dup_row</td>
<td>Insert fails with “Duplicate row was ignored” message. Re-creating the index succeeds.</td>
<td>Insert succeeds. Re-creating the index deletes duplicate rows.</td>
</tr>
</tbody>
</table>

Using the `sorted_data` option on data-only-locked tables

- You can use the `sorted_data` option to `create index` only immediately following a bulk copy operation into an empty table. Once data modifications to that table cause additional page allocations, you cannot use the `sorted_data` option.

- Specifying different values for space management properties may override the sort suppression functionality of the `sorted_data`.

Getting information about tables and indexes

- Each index—including composite indexes—is represented by one row in `sysindexes`.

- For information about the order of the data retrieved through indexes and the effects of an Adaptive Server installed sort order, see the `order by` clause.

- For information about a table’s indexes, execute `sp_helpindex`. For information about index partitions, you can also execute `sp_helppartitions`.

- Each index partition and data partition is represented by one row in `syspartitions`.

Creating indexes on computed columns

- You can use materialized computed columns as index keys, as though they were regular columns.

- To convert a virtual column to a materialized column and index it, use `alter table modify` with the `materialized` option before executing `create index`.

- A computed column need not be deterministic to be used as an index key; however, you must be careful about the possible impact of a nondeterministic column on the queries that reference it.

Creating partitioned indexes

- A local index inherits the partition strategies, partition columns, and partition bounds (for range and list partitions) of the base table.

- Adaptive Server maintains local indexes, rebuilding the local index if the base table is repartitioned with a different partition key.

- Adaptive Server supports:
For range-, hash-, and list-partitioned tables, clustered indexes are always local. Adaptive Server creates a local clustered index whether or not “local index” is included in the syntax.

Creating function-based indexes

- You can create indexes directly on expressions.
- The expression must be deterministic.
- Because Adaptive Server does not verify the deterministic property of the expression index key, the user is responsible for maintaining the property. A change in this property can cause unexpected results.
- As a function-based index key must be deterministic, its result is preevaluated, and reused without reevaluation. Adaptive Server assumes all function-based index keys to be deterministic and uses their preevaluated values when they are referenced in a query; they are reevaluated only when the values of their base columns are changed.
- An index can have multiple function-based index keys or a combination of function-based index keys and regular columns.
- Expressions used as index keys must be deterministic. An expression key is different from a computed column index key, which needs to be evaluated only once, and does not require the deterministic property. An expression, however, must be reevaluated upon each occurrence of the expression in a specified query, and must always return the same result.
- If a user-defined function that is referenced by a function-based index is dropped or becomes invalid, any operations that call that function fail.
- Adaptive Server does not support clustered function-based indexes.
- You cannot create a function-based index with the sorted_data option.
- Once you create an index key on an expression, subsequent queries recognize the expression as an index key only if the expression is exactly the same as the expression used to create the index key.
- All insert, delete, and update operations on base columns cause Adaptive Server to update the value of function-based index keys automatically.

<table>
<thead>
<tr>
<th>Index type</th>
<th>Table type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local clustered and nonclustered partitioned indexes</td>
<td>Partitioned tables</td>
</tr>
<tr>
<td>Global, clustered, unpartitioned indexes</td>
<td>Round-robin partitioned tables</td>
</tr>
<tr>
<td>Global, nonclustered, unpartitioned indexes</td>
<td>All partitioned tables</td>
</tr>
</tbody>
</table>
**create index**

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
create index permission defaults to the table owner and is not transferable.

**Auditing**
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 104   | create       | create index              | • Roles – Current active roles  
|       |              |                            | • Keywords or options – NULL  
|       |              |                            | • Previous value – NULL       
|       |              |                            | • Current value – NULL        
|       |              |                            | • Other information – Name of the index  
|       |              |                            | • Proxy information – Original login name, if a set proxy is in effect |

See also **Commands** alter table, create table, drop index, insert, order by clause, set, update

**System procedures** sp_addsegment, sp_chgattribute, sp_helpcomputedcolumn, sp_helpindex, sp_helpsegment, sp_spaceused

**Utilities** optdiag
create plan

Description
Creates an abstract plan.

Syntax
create plan query plan
   [into group_name]
   [and set @new_id]

Parameters
query
is a string literal, parameter, or local variable containing the SQL text of a query.

plan
is a string literal, parameter, or local variable containing an abstract plan expression.

into group_name
specifies the name of an abstract plan group.

and set @new_id
returns the ID number of the abstract plan in the variable.

Examples
Example 1 Creates an abstract plan for the specified query:
create plan "select * from titles where price > $20" " (t_scan titles)"

Example 2 Creates an abstract plan for the query in the dev_plans group, and returns the plan ID in the variable @id:

```
declare @id int
create plan "select au_fname, au_lname from authors
   where au_id = '724-08-9931' "
   " (i_scan au_id_ix authors)"
into dev_plans
   and set @id
   select @id
```

Usage
- create plan saves the abstract plan in the group specified with into. If no group name is specified, it saves the plan in the currently active plan group.

- Queries and abstract plans specified with create plan are not checked for valid SQL syntax and plans are not checked for valid abstract plan syntax. Also, the plan is not checked for compatibility with the SQL text. You should immediately check all plans created with create plan for correctness by running the query specified in the create plan statement.

- If another query plan in the group has the same SQL text, replace mode must be enabled with set plan replace on. Otherwise, the create plan command fails.
create plan

- You must declare @new_id before using it in the and set clause.
- The abstract plan group you specify with into must already exist.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

create plan permission defaults to all users. No permission is required to use it.

See also

Commands

set plan

Documentation

*Performance and Tuning Guide: Optimizer and Abstract Plans* for information on abstract plans.

System procedures

sp_add_qpgroup, sp_find_qplan, sp_help_qplan, sp_set_qplan
create procedure

Description
Creates a stored procedure or an extended stored procedure (ESP) that can take one or more user-supplied parameters.

Note For syntax and usage information about the SQLJ command for creating procedures, see create function (SQLJ) on page 104.

Syntax
```
create procedure [owner.]procedure_name[number]
    [([@parameter_name datatype [(length) | (precision [, scale])] = default[output])]
    [,. @parameter_name datatype [(length) | (precision [, scale])] = default[output]...]]
    [with recompile]
    as (SQL_statements | external name dll_name)
```

Parameters
- `procedure_name` is the name of the procedure. It must conform to the rules for identifiers and cannot be a variable. Specify the owner’s name to create another procedure of the same name owned by a different user in the current database. The default value for `owner` is the current user.
- `number` is an optional integer used to group procedures of the same name so that they can be dropped together with a single `drop procedure` statement. Procedures used in the same application are often grouped this way. For example, if the procedures used with the application named `orders` are named `orderproc;1`, `orderproc;2`, and so on, the following statement drops the entire group:

```
drop proc orderproc
```

Once procedures have been grouped, individual procedures within the group cannot be dropped. For example, the following statement is not allowed:

```
drop procedure orderproc;2
```

You cannot group procedures if you are running Adaptive Server in the **evaluated configuration**. The evaluated configuration requires that you disallow procedure grouping so that every stored procedure has a unique object identifier and can be dropped individually. To disallow procedure grouping, a System Security Officer must use `sp_configure` to reset `allow procedure grouping`. For more information about the evaluated configuration, see the *System Administration Guide*. 


create procedure

*parameter_name* is the name of an argument to the procedure. The value of each parameter is supplied when the procedure is executed. Parameter names are optional in `create procedure` statements—a procedure is not required to take any arguments.

Parameter names must be preceded by the @ sign and conform to the rules for identifiers. A parameter name, including the @ sign, can be a maximum of 30 characters, and larger for identifiers. Parameters are local to the procedure: the same parameter names can be used in other procedures.

If the value of a parameter contains nonalphanumeric characters, it must be enclosed in quotes. This includes object names qualified by a database name or owner name, since they include a period. If the value of a character parameter begins with a numeric character, it also must be enclosed in quotes.

*datatype([(length) | (precision [, scale])]]

is the datatype of the parameter. See “User-defined datatypes” on page 43 in Chapter 1, “System and User-Defined Datatypes” of *Reference Manual: Building Blocks* for more information about datatypes. Stored procedure parameters cannot have a datatype of text, unitext, or image or a user-defined datatype whose underlying type is text, unitext, or image.

The char, varchar, unichar, univarchar, nchar, nvarchar, binary, and varbinary datatypes should include a *length* in parentheses. If you omit the length, Adaptive Server truncates the parameter value to 1 character.

The float datatype expects a binary *precision* in parentheses. If you omit the precision, Adaptive Server uses the default precision for your platform.

The numeric and decimal datatypes expect a *precision* and *scale*, enclosed in parentheses and separated by a comma. If you omit the precision and scale, Adaptive Server uses a default precision of 18 and a scale of 0.

*default*

defines a default value for the procedure’s parameter. If a default is defined, a user can execute the procedure without giving a parameter value. The default must be a constant. It can include the wildcard characters (%, _, [ ], and [^

The default can be NULL. The procedure definition can specify that some action be taken if the parameter value is NULL (see Example 3).
output indicates that the parameter is a return parameter. Its value can be returned to the execute command that called this procedure. Use return parameters to return information to the calling procedure.

To return a parameter value through several levels of nested procedures, each procedure must include the output option with the parameter name, including the execute command that calls the highest level procedure.

The output keyword can be abbreviated to out.

with recompile
means that Adaptive Server never saves a plan for this procedure; a new plan is created each time it is executed. Use this optional clause when you expect that the execution of a procedure is atypical—that is, when you need a new plan. The with recompile clause has no impact on the execution of an extended stored procedure.

SQL statements specify the actions the procedure is to take. You can include any number and kind of SQL statements, with the exception of create view, create default, create rule, create procedure, create trigger, and use.

create procedure SQL statements often include control-of-flow language, including one or more of the following: declare; if...else; while; break; continue; begin...end; goto label; return; waitfor; /* comment */. They can also refer to parameters defined for the procedure.

The SQL statements can reference objects in another database, as long as they are properly qualified.

external name creates an extended stored procedure. You cannot use the number parameter with as external name.

dll_name specifies the name of the dynamic link library (DLL) or shared library containing the functions that implement the extended stored procedure. The dll_name can be specified with no extension or with a platform-specific extension, such as dll on Windows NT or .so on Sun Solaris. If you specify the extension, enclose the entire dll_name in quotation marks.

Examples

Example 1 Given a table name, the procedure showind displays its name and the names and identification numbers of any indexes on any of its columns:

```sql
create procedure showind @tabname varchar (30) as begin
    select sysobjects.name, sysindexes.name, indid
```
create procedure

from sysindexes, sysobjects
where sysobjects.name = @tabname
and sysobjects.id = sysindexes.id

Here are the acceptable syntax forms for executing showind:

execute showind titles
execute showind @tabname = "titles"

Or, if this is the first statement in a file or batch:

showind titles

Example 2 This procedure displays information about the system tables if the user does not supply a parameter:

create procedure
showindsind @table varchar (30) = "sys%"
as
select sysobjects.name, sysindexes.name, indid
from sysindexes, sysobjects
where sysobjects.name like @table
and sysindexes.id = sysobjects.id

Example 3 This procedure specifies an action to be taken if the parameter is NULL (that is, if the user does not give a parameter):

create procedure
showindsindnew @table varchar (30) = null
as
if @table is null
print "Please give a table name"
else
select sysobjects.name, sysindexes.name, indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysindexes.id = sysobjects.id

Example 4 This procedure multiplies two integer parameters and returns the product in the output parameter, @result:

create procedure mathtutor @mult1 int, @mult2 int,
@result int output
as
select @result = @mult1 * @mult2

If the procedure is executed by passing it 3 integers, the select statement performs the multiplication and assigns the values, but does not print the return parameter:

mathtutor 5, 6, 32
Example 5 In this example, both the procedure and the execute statement include output with a parameter name so that the procedure can return a value to the caller:

```sql
declare @guess int
select @guess = 32
exec mathtutor 5, 6, @result = @guess output

(1 row affected)
(return status = 0)

Return parameters:

@result
-----------
30
```

The output parameter and any subsequent parameters in the execute statement, `@result`, must be passed as:

- `@parameter = value`
- The value of the return parameter is always reported, whether or not its value has changed.
- `@result` does not need to be declared in the calling batch because it is the name of a parameter to be passed to `mathtutor`.
- Although the changed value of `@result` is returned to the caller in the variable assigned in the execute statement (in this case, `@guess`), it is displayed under its own heading (`@result`).

Example 6 You can use return parameters in additional SQL statements in the batch or calling procedure. This example shows how to use the value of `@guess` in conditional clauses after the execute statement by storing it in another variable name, `@store`, during the procedure call. When return parameters are used in an execute statement that is part of a SQL batch, the return values are printed with a heading before subsequent statements in the batch are executed.

```sql
declare @guess int
declare @store int
select @guess = 32
select @store = @guess
execute mathtutor 5, 6, @result = @guess output
select Your_answer = @store, Right_answer = @guess
if @guess = @store
```
create procedure

  print "Right-o"
else
  print "Wrong, wrong, wrong!"

(1 row affected)
(1 row affected)
(return status = 0)

Return parameters:

<table>
<thead>
<tr>
<th>@result</th>
<th>Right_answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

(1 row affected)
Wrong, wrong, wrong!

Example 7 Creates an extended stored procedure named xp_echo, which takes an input parameter, @in, and echoes it to an output parameter, @out. The code for the procedure is in a function named xp_echo, which is compiled and linked into a DLL named sqlsrvdll.dll:

```sql
create procedure xp_echo @in varchar (255),
    @out varchar (255) output
as external name "sqlsrvdll.dll"
```

Usage

- To avoid seeing unexpected results due to changes in settings, run set rowcount 0 as your initial statement before executing create procedure. The scope of set is limited to just your create procedure command, and resets to your previous setting once the procedure exits.
- After a procedure is created, you can run it by issuing the execute command along with the procedure’s name and any parameters. If a procedure is the first statement in a batch, you can give its name without the keyword execute.
- You can use sp_hidetext to hide the source text for a procedure, which is stored in syscomments.
- When a stored procedure batch executes successfully, Adaptive Server sets the @@error global variable to 0.

Restrictions

- The maximum number of parameters that a stored procedure can have is 2048.
• The maximum number of local and global variables in a procedure is limited only by available memory.

• The maximum amount of text in a stored procedure is 16MB.

• You cannot combine a create procedure statement with other statements in a single batch.

• You can create a stored procedure only in the current database, although the procedure can reference objects from other databases. Most objects referenced in a procedure must exist at the time you create the procedure. However, you can include statements like drop table, create index, or truncate table. These are allowed in a create procedure statement even if the underlying object does not exist when you create the procedure.

You can create an object within a procedure, then reference it, provided the object is created before it is referenced.

You cannot use alter table in a procedure to add a column and then refer to that column within the procedure.

• If you use select * in your create procedure statement, the procedure (even if you use the with recompile option to execute) does not pick up any new columns you may have added to the table. You must drop the procedure and re-create it. Otherwise, the wrong results can be caused by the insert...select statement of insert into table1 select * from table2 in the procedure when new columns have been added to both tables.

• Within a stored procedure, you cannot create an object (including a temporary table), drop it, then create a new object with the same name. Adaptive Server creates the objects defined in a stored procedure when the procedure is executed, not when it is compiled.

**Warning!** Certain changes to databases, such as dropping and re-creating indexes, can cause object IDs to change. When object IDs change, stored procedures recompile automatically, and can increase slightly in size. Leave some space for this increase.

Extended stored procedures

• If you use the as external name syntax, create procedure registers an extended stored procedure (ESP). Extended stored procedures execute procedural language functions rather than Transact-SQL commands.

• On Windows NT – an ESP function should not call a C runtime signal routine. This can cause XP Server to fail, because Open Server™ does not support signal handling on Windows NT.
To support multithreading, ESP functions should use the Open Server
sr_v_yield function, which suspends and reschedules the XP Server thread
to allow another thread of the same or higher priority to execute.

The DLL search mechanism is platform-dependent. On Windows NT, the
sequence of a DLL file name search is as follows:

a  The directory from which the application is loaded
b  The current directory
c  The system directory (SYSTEM32)
d  Directories listed in the PATH environment variable

If the DLL is not in the first three directories, set the PATH to include the
directory in which it is located.

On UNIX platforms, the search method varies with the particular platform.
If it fails to find the DLL or shared library, it searches $SYBASE/lib.

Absolute path names are not supported.

System procedures

System Administrators can create new system procedures in the
sybsystemprocs database. System procedure names must begin with the
characters “sp_”. These procedures can be executed from any database by
specifying the procedure name; it is not necessary to qualify it with the
sybsystemprocs database name. For more information about creating
system procedures, see the System Administration Guide.

System procedure results may vary depending on the context in which
they are executed. For example, sp_foo, which executes the db_name ()
function, returns the name of the database from which it is
executed. When executed from the pubs2 database, it returns the value
“pubs2”:

```
use pubs2
sp_foo
-----------------------------
pubs2
```

When executed from sybsystemprocs, it returns the value
“sybsystemprocs”:

```
use sybsystemprocs
sp_foo
-----------------------------
sybsystemprocs
```
Nested procedures

- Procedure nesting occurs when one stored procedure calls another.
- If you execute a procedure that calls another procedure, the called procedure can access objects created by the calling procedure.
- The nesting level increments when the called procedure begins execution and decrements when the called procedure completes execution. Exceeding the maximum of 16 levels of nesting causes the transaction to fail.
- You can call another procedure by name or by a variable name in place of the actual procedure name.
- The current nesting level is stored in the @@nestlevel global variable.

Procedure return status

- Stored procedures can return an integer value called a **return status**. The return status either indicates that the procedure executed successfully or specifies the type of error that occurred.
- When you execute a stored procedure, it automatically returns the appropriate status code. Adaptive Server currently returns the following status codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Procedure executed without error</td>
</tr>
<tr>
<td>-1</td>
<td>Missing object</td>
</tr>
<tr>
<td>-2</td>
<td>Datatype error</td>
</tr>
<tr>
<td>-3</td>
<td>Process was chosen as deadlock victim</td>
</tr>
<tr>
<td>-4</td>
<td>Permission error</td>
</tr>
<tr>
<td>-5</td>
<td>Syntax error</td>
</tr>
<tr>
<td>-6</td>
<td>Miscellaneous user error</td>
</tr>
<tr>
<td>-7</td>
<td>Resource error, such as out of space</td>
</tr>
<tr>
<td>-8</td>
<td>Non-fatal internal problem</td>
</tr>
<tr>
<td>-9</td>
<td>System limit was reached</td>
</tr>
<tr>
<td>-10</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-11</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-12</td>
<td>Table or index is corrupt</td>
</tr>
<tr>
<td>-13</td>
<td>Database is corrupt</td>
</tr>
<tr>
<td>-14</td>
<td>Hardware error</td>
</tr>
</tbody>
</table>

Codes -15 through -99 are reserved for future use.
Users can generate a user-defined return status with the return statement. The status can be any integer other than 0 through -99. The following example returns “1” when a book has a valid contract and “2” in all other cases:

```sql
create proc checkcontract @titleid tid
as
if (select contract from titles where
    title_id = @titleid) = 1
    return 1
else
    return 2
checkcontract @titleid = "BU1111"
    (return status = 1)
checkcontract @titleid = "MC3026"
    (return status = 2)
```

If more than one error occurs during execution, the code with the highest absolute value is returned. User-defined return values take precedence over system-defined values.

Object identifiers

- To change the name of a stored procedure, use `sp_rename`.
- To change the name of an extended stored procedure, drop the procedure, rename and recompile the supporting function, then re-create the procedure.
- If a procedure references table names, column names, or view names that are not valid identifiers, you must set `quoted_identifier` on before the `create procedure` command and enclose each such name in double quotes. The `quoted_identifier` option does not need to be on when you execute the procedure.
- You must drop and re-create the procedure if any of the objects it references have been renamed.
- Inside a stored procedure, object names used with the `create table` and `dbcc` commands must be qualified with the object owner’s name if other users are to make use of the stored procedure. For example, user “mary,” who owns the table `marytab`, should qualify the name of her table inside a stored procedure (when it is used with these commands) if she wants other users to be able to execute it. This is because the object names are resolved when the procedure is run. When another user tries to execute the procedure, Adaptive Server looks for a table called `marytab` owned by the user “mary” and not a table called `marytab` owned by the user executing the stored procedure.
Thus, if marytab is not qualified, and user “john” tries to execute the procedure, Adaptive Server looks for a table called marytab owned by the owner of the procedure (“mary,” in this case) or by the Database Owner if the user table does not exist. For example, if the table mary.marytab is dropped, the procedure references dbo.marytab.

Object names used with other statements (for example, select or insert) inside a stored procedure need not be qualified because the names are resolved when the procedure is compiled.

Temporary tables and procedures

- You can create a procedure to reference a temporary table if the temporary table is created in the current session. A temporary table created within a procedure disappears when the procedure exits. For more information, see the Transact-SQL User’s Guide.

- System procedures such as sp_help work on temporary tables, but only if you use them from tempdb.

Setting options in procedures

- You can use the set command inside a stored procedure. Most set options remain in effect during the execution of the procedure, then revert to their former settings.

  However, if you use a set option (such as identity_insert) which requires the user to be the object owner, a user who is not the object owner cannot execute the stored procedure.

Getting information about procedures

- For a report on the objects referenced by a procedure, use sp_depends.

- To display the text of a create procedure statement, which is stored in syscomments, use sp_helptext with the procedure name as the parameter. You must be using the database where the procedure resides when you use sp_helptext. To display the text of a system procedure, execute sp_helptext from the sybsystemprocs database.

- To see a list of system extended stored procedures and their supporting DLLs, use sp_helpextendedproc from the sybsystemprocs database.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

create procedure permission defaults to the Database Owner, who can transfer it to other users.

Permission to use a procedure must be granted explicitly with the grant command and may be revoked with the revoke command.
Permissions on objects at procedure creation  When you create a procedure, Adaptive Server makes no permission checks on objects, such as tables and views, that are referenced by the procedure. Therefore, you can create a procedure successfully even though you do not have access to its objects. All permission checks occur when a user executes the procedure.

Permissions on objects at procedure execution  When the procedure is executed, permission checks on objects depend upon whether the procedure and all referenced objects are owned by the same user.

- If the procedure’s objects are owned by different users, the invoker must have been granted direct access to the objects. For example, if the procedure performs a select from a table that the user cannot access, the procedure execution fails.

- If a procedure and its objects are owned by the same user, however, special rules apply. The invoker automatically has “implicit permission” to access the procedure’s objects even though the invoker could not access them directly. Without having to grant users direct access to your tables and views, you can give them restricted access with a stored procedure. In this way, a stored procedure can be a security mechanism. For example, invokers of the procedure might be able to access only certain rows and columns of your table.

A detailed description of the rules for implicit permissions is discussed in the System Administration Guide.

Auditing  Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 11    | create       | create procedure           | • Roles – Current active roles  
|       |              |                             | • Keywords or options – NULL |
|       |              |                             | • Previous value – NULL     |
|       |              |                             | • Current value – NULL      |
|       |              |                             | • Other information – NULL  |
|       |              |                             | • Proxy information – Original login name, if set proxy in effect |

See also  Commands: begin...end, break, continue, declare, drop procedure, execute, goto label, grant, if...else, return, select, waitfor, while

System procedures: sp_addextendedproc, sp_helpextendedproc, sp_helptext, sp_hidetext, sp_rename

Adaptive Server Enterprise
create procedure (SQLJ)

Description
Creates a SQLJ stored procedure by adding a SQL wrapper to a Java static method. Can accept user-supplied parameters and return result sets and output parameters.

Note For syntax and usage information about the Transact-SQL command for creating procedures, see create procedure on page 129.

Syntax
create procedure [owner.]sql_procedure_name
   ([in | out | inout] sql_parameter_name
    sql_datatype [(length) | (precision[, scale])]
    [=default]
   ...)
   [, [in | out | inout] sql_parameter_name
    sql_datatype [(length) | (precision[, scale])]
    [=default]
   ...])
   [modifies sql data]
   [dynamic result sets integer]
   [deterministic | not deterministic]
   language java
   parameter style java
   external name 'java_method_name
   [([java_datatype[, java_datatype
    ...])])'

Parameters

sql_procedure_name
is the Transact-SQL name of the procedure. It must conform to the rules for identifiers and cannot be a variable. Specify the owner’s name to create another procedure of the same name owned by a different user in the current database. The default value for owner is the current user.

in | out | inout
specifies the mode of the listed parameter. in indicates an input parameter; out indicates an output parameter; and inout indicates a parameter that is both an input and an output parameter. The default mode is in.
create procedure (SQLJ)

**sql_parameter_name**

is the name of an argument to the procedure. The value of each input parameter is supplied when the procedure is executed. Parameters are optional; a SQLJ stored procedure need not take arguments.

Parameter names must conform to the rules for identifiers. If the value of a parameter contains nonalphanumeric characters, it must be enclosed in quotes. This includes object names qualified by a database name or owner name, since they include a period. If the value of the parameter begins with a numeric character, it also must be enclosed in quotes.

**sql_datatype**

is the Transact-SQL datatype of the parameter.

**default**

defines a default value for the procedure’s parameter. If a default is defined, you can execute the procedure without a parameter value. The default must be a constant. It can include the wildcard characters (%, _, [ ], and ^) if the procedure uses the parameter name with the keyword **like**.

The default can be NULL. The procedure definition can specify that some action be taken if the parameter value is NULL.

**modifies sql data**

indicates that the Java method invokes SQL operations, reads, and modifies SQL data in the database. This is the default and only implementation. It is included for syntactic compatibility with the ANSI standard.

**dynamic result sets integer**

specifies that the Java method can return SQL result sets. **integer** specifies the maximum number of result sets the method can return. This value is implementation-defined.

**deterministic | not deterministic**

this syntax is supported for compatibility with other SQLJ-compliant vendors.

**language java**

specifies that the external routine is written in Java. This is a required clause for SQLJ stored procedures.

**parameter style java**

specifies that the parameters passed to the external routine at runtime are Java parameters. This is a required clause for SQLJ stored procedures.
external

indicates that create procedure defines a SQL name for an external routine
written in a programming language other than SQL.

name

defines the name of the external routine (Java method). The specified name
is a character-string literal and must be enclosed in single quotes:

'java_method_name [ java_datatype
 [{, java_datatype} ...]]'

java_method_name

defines the name of the external Java method.

java_datatype

defines a Java datatype that is mappable or result-set mappable. This is the
Java method signature.

Examples

Example 1 Creates the SQLJ procedure java_multiply, which multiplies two
integers and returns an integer.

create procedure java_multiply (param1 integer, param2 integer, out result integer)
language java parameter style java
external name 'MathProc.multiply'

Example 2 Returns values that are always larger than 10:

create procedure my_max (a int = 10, b int = 10)
language java parameter style java
external name 'java.lang.Math.max'

exec my_max
(return status = 10)

exec my_max 8
(return status = 10)

See also the examples for Transact-SQL create procedure.

Usage

• To avoid seeing unexpected results due to changes in settings, run set
rowcount 0 as your initial statement before executing create procedure. The
scope of set is limited to just your create procedure command, and resets
to your previous setting once the procedure exits.

• You can include a maximum of 31 in, inout, and out parameters in a create
procedure statement.
To comply with the ANSI standard, do not precede parameter names with the @ sign. When executing a SQLJ stored procedure from isql or other non-Java client, however, you must precede parameter names with the @ sign, which preserves the naming order.

The SQLJ create procedure syntax differs from the Transact-SQL create procedure syntax for compatibility with the SQLJ ANSI standard.

Adaptive Server executes each type of stored procedure in the same way.

See also

Permissions create procedure permission defaults to the Database Owner, who can transfer it to other users. Permission to use a procedure must be granted explicitly with the grant command and may be revoked with the revoke command.

System procedures sp_depends, sp_help, sp_helpjava, sp_helpprotect
create proxy_table

Description

Component Integration Services only  Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

Syntax

create proxy_table table_name
    [external [table | directory | file]]
    at pathname
    [column delimiter "<string>"]

Parameters

- **table_name**  specifies the local proxy table name to be used by subsequent statements.  
  `table_name` takes the form `dbname.owner.object`, where `dbname` and `owner` are optional and represent the local database and owner name. If `dbname` is not specified, the table is created in the current database; if `owner` is not specified, the table is owned by the current user. If either `dbname` or `owner` is specified, the entire `table_name` must be enclosed in quotes. If only `dbname` is present, a placeholder is required for `owner`.

- **external table**  specifies that the object is a remote table or view. external table is the default, so this clause is optional.

- **external directory**  specifies that the object is a directory with a path in the following format:  
  “/tmp/directory_name [:R]”. “R” indicates “recursive.”

- **external file**  specifies that the object is a file with a path in the following format:  
  “/tmp/filename”.

- **at pathname**  specifies the location of the remote object. `pathname` takes the form `server_name.dbname.owner.object`, where:
  
  - **server_name** (required) – is the name of the server that contains the remote object.
  - **dbname** (optional) – is the name of the database managed by the remote server that contains this object.
  - **owner** (optional) – is the name of the remote server user that owns the remote object.
  - **object** (required) – is the name of the remote table or view.
column delimiter

used to separate fields within each record when accessing flat files, column delimiters. The column delimiter can be up to 16 bytes long.

string

The column delimiter string can be any character sequence, but if the string is longer than 16 bytes, only the first 16 bytes are used. The use of column delimiter for proxy tables mapped to anything but files will result in a syntax error.

Examples

This example creates a proxy table named \texttt{t1} that is mapped to the remote table \texttt{t1}. Component Integration Services derives the column list from the remote table:

\begin{verbatim}
create proxy_table t1
at "SERVER_A.db1.joe.t1"
\end{verbatim}

Usage

- \texttt{create proxy_table} is a variant of the \texttt{create existing table} command. You use \texttt{create proxy_table} to create a proxy table, but (unlike \texttt{create existing table}) you do not specify a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

- The location information provided by the \texttt{at} keyword is the same information that is provided by \texttt{sp_addobjectdef}. The information is stored in the \texttt{sysattributes} table.

- If the remote server object does not exist, the command is rejected with an error message.

- If the object exists, the local system tables are updated. Every column is used. Columns and their attributes are obtained for the table or view.

- Component Integration Services automatically converts the datatype of the column into an Adaptive Server datatype. If the conversion cannot be made, the \texttt{create proxy_table} command does not allow the table to be defined.

- Index information from the remote server table is extracted and used to create rows for the system table \texttt{sysindexes}. This defines indexes and keys in Adaptive Server terms and enables the query optimizer to consider any indexes that may exist on the table.

- After defining the proxy table, issue an \texttt{update statistics} command for the table. This allows the query optimizer to make intelligent choices regarding join order.
When executing `create proxy_table table_name at pathname`, the table and column names assumes the same case as `table_name`, if the server identified by `pathname` is case-insensitive (such as DB2 and Oracle).

The columns returned by a case insensitive server (typically in uppercase), is stored in Adaptive Server as lower case, if `table_name` is lowercase. If `table_name` is uppercase, then the column names is stored as uppercase values. If `table_name` is in mixed case, all column names is stored as received from the remote site.

- `create proxy_table` is not supported with temp tables.
- You cannot combine `create proxy_table` statement with other statements in a single batch.
- A proxy table stores only metadata. As such, the only space used is the result of making entries in system catalogs. It is estimated that a hundred proxy tables consume about 1MB of space, assuming an average of two indexes per table.
- SQL user-defined functions are not currently supported with `create proxy table`, `create table at remote server`, or `alter table`.

**Note** The execution of SQL functions requires the syntax `username.functionname()`.

- If the remote Adaptive Server table has one or more encrypted columns, CIS updates the proxy table’s metadata in `syscolumns` to reflect the column’s encryption properties and its key ID.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`create proxy_table` permission defaults to the table owner and is not transferable.

**Auditing**

Values in event and extrainfo columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 11    | create       | create procedure         | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |
create proxy_table

See also  Commands create existing table, create table
create role

Description

Creates a user-defined role; specifies the password expiration interval, the minimum password length, and the maximum number of failed logins allowed for a specified role at creation. You can also associate a password with the role at the time that the role is created.

Syntax

create role role_name [with passwd "password"
  [, {passwd expiration | min passwd length | max failed_logins} option_value]
]

Parameters

role_name

is the name of the new role. It must be unique to the server and conform to the rules for identifiers. It cannot be a variable.

with passwd

attaches a password the user must enter to activate the role.

password

is the password to attach to the role. Passwords must be at least 6 characters in length and must conform to the rules for identifiers. You cannot use variables for passwords.

passwd expiration

password expiration interval specifies the password expiration interval in days. It can be any value between 0 and 32767, inclusive. For example, if you create a new login on August 1, 2007 at 10:30 AM, with a password expiration interval of 30 days, the password expires on August 31, 2007 at 10:30 AM

min passwd length

specifies the minimum password length required for the specified role.

max failed_logins

specifies the number of allowable failed login attempts for the specified login.

option_value

specifies the value for passwd expiration, min passwd length, or max failed_logins.

Examples

Example 1 Creates a role named doctor_role:

create role doctor_role

Example 2 Creates a role named doctor_role with the password “physician”:

create role doctor_role with passwd "physician"
Example 3 sets passwd expiration to 7 days. The password for the role expires at the time of day that the password was last changed after the specified period has passed (in this example, 7 days):

```
create role intern_role with passwd "temp244",
    passwd expiration 7
```

Example 4 Sets the maximum number of failed logins allowed for intern_role:

```
create role intern_role with passwd "temp244"
    max failed_logins 20
```

Example 5 Sets the minimum password length for intern_role:

```
create role intern_role with passwd "temp244",
    min passwd length 0
```

Usage

- Use create role from the master database.
- Use the with passwd password clause to attach a password to a role at creation. If you attach a password to the role, the user granted this role must specify the password to activate the role.

For information on adding a password to a role after creation, see the alter role command.

Note Passwords created in versions before 12.x that are attached to user-defined roles do not expire.

- Role names must be unique to the server.
- Role names cannot be the same as user names. You can create a role with the same name as a user, but when you grant privileges, Adaptive Server resolves naming conflicts by making the grant to the user instead of the role.

For more information on naming conflicts, see the grant role command.

Restrictions

- The maximum number of roles that can be created per server session is 1024. However, 32 roles are reserved for Sybase system roles, such as sa_role and sso_role. Therefore, the maximum number of user-defined roles that can be created per server session is 992.
- If you create a role with an attached password, a user cannot activate that role by default at login. Do not create a role with an attached password if the user to whom you grant that role needs to activate the role by default at login.
Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

You must be a System Security Officer to use `create role`. `create role` permission is not included in the `grant all` command.

Auditing

Values in event and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 85    | roles        | create role, drop role, alter role, grant role, or revoke role | • `Roles` – Current active roles  
• `Keywords or options` – NULL  
• `Previous value` – NULL  
• `Current value` – NULL  
• `Other information` – NULL  
• `Proxy information` – Original login name, if set proxy in effect |

See also **Commands**  alter role, drop role, grant, revoke, set

**System procedures**  `sp_activeroles`, `sp_displaylogin`, `sp_displayroles`, `sp_helprotect`, `sp_modifylogin`
create rule

Description
Specifies the domain of acceptable values for a particular column or for any column of a user-defined datatype, and creates access rules.

Syntax
create [[and | or] access] rule
  [owner.]rule_name
  as condition_expression

Parameters
access specifies that you are creating an access rule. For information on access rules, see Chapter 11, “Managing User Permissions” in the System Administration Guide.

rule_name is the name of the new rule. It must conform to the rules for identifiers and cannot be a variable. Specify the owner’s name to create another rule of the same name owned by a different user in the current database. The default value for owner is the current user.

condition_expression specifies the conditions that define the rule. It can be any expression that is valid in a where clause, and can include arithmetic operators, relational operators, in, like, between, and so on. However, it cannot reference a column or any other database object. Built-in functions that do not reference database objects can be included.

A condition_expression takes one argument. The argument is prefixed by the @ sign and refers to the value that is entered via the update or insert command. You can use any name or symbol to represent the value when you write the rule, but the first character must be the @ sign. Enclose character and date constants in quotes, and precede binary constants with “0x”.

Examples

Example 1 Creates a rule named limit, which limits the value of advance to less than $1000:

create rule limit
  as @advance < $1000

Example 2 Creates a rule named pubid_rule, which restricts the values of pub_id to 1389, 0736, or 0877:

create rule pubid_rule
  as @pub_id in ('1389', '0736', '0877')

Example 3 Creates a rule named picture, which restricts the value of value to always begin with the indicated characters:

create rule picture
as @value like ' _-%[0-9]'  

Usage

- To hide the text of a rule, use `sp_hidetext`.
- To rename a rule, use `sp_rename`.

Restrictions

- You can create a rule only in the current database.
- Rules do not apply to the data that already exists in the database at the time the rules are created.
- `create rule` statements cannot be combined with other statements in a single batch.
- You cannot bind a rule to an Adaptive-Server-supplied datatype or to a column of type `text`, `unitext`, `image`, or `timestamp`.
- You must drop a rule before you create a new one of the same name, and you must unbind a rule before you drop it. Use:
  ```
  sp_unbindrule objname [, futureonly]
  ```

Binding rules

- Use `sp_bindrule` to bind a rule to a column or user-defined datatype. Its syntax is:
  ```
  sp_bindrule rulename, objname [, futureonly]
  ```
- A rule that is bound to a user-defined datatype is activated when you insert a value into, or update, a column of that type. Rules do not test values inserted into variables of that type.
- The rule must be compatible with the datatype of the column. For example, you cannot use the following as a rule for an exact or approximate numeric column:
  ```
  @value like A%
  ```
  If the rule is not compatible with the column to which it is bound, Adaptive Server generates an error message when it tries to insert a value, not when you bind it.
- You can bind a rule to a column or datatype without unbinding an existing rule.
- Rules bound to columns always take precedence over rules bound to user-defined datatypes, regardless of which rule was most recently bound. Table 1-11 indicates the precedence when binding rules to columns and user-defined datatypes where rules already exist.
Table 1-11: Rule binding precedence

<table>
<thead>
<tr>
<th>New rule bound to</th>
<th>Old rule bound to user-defined datatype</th>
<th>Old rule bound to column</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined datatype</td>
<td>New rule replaces old</td>
<td>No change</td>
</tr>
<tr>
<td>Column</td>
<td>New rule replaces old</td>
<td>New rule replaces old</td>
</tr>
</tbody>
</table>

Rules and NULLs

- Rules do not override column definitions. If a rule is bound to a column that allows null values, you can insert NULL into the column, implicitly or explicitly, even though NULL is not included in the text of the rule. For example, if you create a rule specifying “@val in (1,2,3)” or “@amount > 10000”, and bind this rule to a table column that allows null values, you can still insert NULL into that column. The column definition overrides the rule.

Defaults and rules

- If a column has both a default and a rule associated with it, the default must fall within the domain defined by the rule. A default that conflicts with a rule is never inserted. Adaptive Server generates an error message each time it attempts to insert the default.

Using integrity constraints in place of rules

- You can define rules using check with the create table statement, which creates integrity constraints. However, these constraints are specific for that table; you cannot bind them to other tables. See create table and alter table for information about integrity constraints.

Getting information about rules

- To get a report on a rule, use sp_help.
- To display the text of a rule, which is stored in the syscomments system table, execute sp_helptext with the rule name as the parameter.
- After a rule is bound to a particular column or user-defined datatype, its ID is stored in the syscolumns or systypes system tables.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

To create rules using ANSI SQL-compliant syntax, use the check clause of the create table statement.

Permissions

create rule permission defaults to the Database Owner, who can transfer it to other users.

Auditing

Values in event and extrainfo columns of sysaudits are:
### Event Audit option | Command or access audited | Information in extrainfo
--- | --- | ---
13 | create | create rule

- **Roles** – Current active roles
- **Keywords or options** – NULL
- **Previous value** – NULL
- **Current value** – NULL
- **Other information** – NULL
- **Proxy information** – Original login name, if set proxy in effect

**See also**

- **Commands** alter table, create default, create table, drop rule, drop table
- **System procedures** sp_bindrule, sp_help, sp_helptext, sp_hidetext, sp_rename, sp_unbindrule
create schema

Description

Creates a new collection of tables, views, and permissions for a database user.

Syntax

```sql
create schema authorization authorization_name
create_object_statement
[create_object_statement ...]
[permission_statement ...]
```

Parameters

- `authorization_name` is the name of the current user in the database.
- `create_object_statement` is a `create table` or `create view` statement.
- `permission_statement` is a `grant` or `revoke` command.

Examples

Creates the `newtitles`, `newauthors`, `newtitleauthors` tables, the `tit_auth_view` view, and the corresponding permissions:

```sql
create schema authorization pogo
create table newtitles (
    title_id tid not null,
    title varchar (30) not null)
create table newauthors (
    au_id id not null,
    au_lname varchar (40) not null,
    au_fname varchar (20) not null)
create table newtitleauthors (
    au_id id not null,
    title_id tid not null)
create view tit_auth_view as
    select au_lname, au_fname
    from newtitles, newauthors, newtitleauthors
    where
    newtitleauthors.au_id = newauthors.au_id
    and
    newtitleauthors.title_id = newtitles.title_id
grant select on tit_auth_view to public
revoke select on tit_auth_view from churchy
```

Usage

- Schemas can be created only in the current database.
The authorization_name, also called the schema authorization identifier, must be the name of the current user.

The user must have the correct command permissions (create table and create view). If the user creates a view on tables owned by another database user, permissions on the view are checked when a user attempts to access data through the view, not when the view is created.

The create schema command is terminated by:

- The regular command terminator (“go” is the default in isql).
- Any statement other than create table, create view, grant, or revoke.

If any of the statements within a create schema statement fail, the entire command is rolled back as a unit, and none of the commands take effect.

create schema adds information about tables, views, and permissions to the system tables. Use the appropriate drop command (drop table or drop view) to drop objects created with create schema. You cannot change permissions granted or revoked in a schema with the standard grant and revoke commands outside the schema creation statement.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
create schema can be executed by any user of a database. The user must have permission to create the objects specified in the schema; that is, create table and create view permission.

See also
Commands create table, create view, grant, revoke
Utilities isql
create service

Description
Wraps the supplied SQL statement in a stored procedure with the specified name and parameters.

Syntax
```
create service service-name [secure security_options ] [, userpath path] [, alias alias-name] type { xml | raw | soap }
[[@parameter_name datatype [(length ) | (precision [, scale ])]
[= default][output]]
[, @parameter_name datatype [(length ) | (precision [, scale ])]
[= default][output]...])]
as SQL_statements
security_options ::= (security_option_item security_option_item)
```

Parameters

**service-name**
- is the name for the user-defined Web service. This name can be any name that is valid for a stored procedure. When the drop service command is invoked with this service name, the corresponding stored procedure is dropped. If you specify the name of an existing service, an exception results.

**security_option_item**
- is either clear or ssl:
  - clear – indicates that HTTP is used to access this Web service
  - ssl – indicates HTTPS is used to access this Web service

**path**
- is a character-string literal specifying the user-defined path to be appended to the URL accessing the Web service. This path is null by default.

**alias-name**
- is a character-string-literal specifying the user-defined Web service alias.

**parameter_name**
- is the name of an argument to the user-defined Web service. The value of this parameter is supplied when the Web service is executed. Parameter names must be preceded by the @ sign and conform to the rules for identifiers. These conditions are the same as for the parameter_name parameter of the create procedure command.

**SQL_statements**
- are the actions the user-defined Web service is to take. Any number and kind of SQL statements can be included, with the exception of create view, create default, create rule, create procedure, create trigger, and use. These conditions are the same as for the SQL_statements parameter of the create procedure command.
type can be soap, raw, or xml:

- soap implies an HTTP POST request and must be compliant with all the SOAP rules. The data is returned in SQL/XML format.
- raw indicates that the output is to be sent without any alteration or reformatting. This implies an HTTP GET request. The invoked stored procedure can specify the exact output.
- xml indicates that the result set output is returned in SQL/XML format. This implies an HTTP GET request.

**Note** For datatype mappings between ASE stored procedures and SOAP user-defined Web services, see the *Web Services User’s Guide*.

**Examples**

**Example 1** A user-defined Web service, rawservice, of type raw is created to return the version of the current database. The `create service` command is entered from the `isql` command line for the pubs2 database:

```
1> use pubs2
2> go
1> create service rawservice type raw as select
   '<html><h1>' + @@version + '</h1></html>'
2> go
```

The newly created user-defined Web service is then deployed:

```
1> sp_webservices 'deploy', 'all'
2> go
```

The WSDL for the newly created user-defined Web service is at http://myhost:8181/services/pubs2?wsdl.

The newly created user-defined Web service is available at the following URL, where bob and bob123 are the user ID and password of the creator of the user-defined Web service:

http://myhost:8181/services/pubs2?method=rawservice&username=bob&password=bob123

The output, an Adaptive Server Enterprise version string, is displayed in an HTML `<h1>` tag in the browser window.

**Example 2** A user-defined Web service, xmlservice, of type xml is created to return the version of the current database. The `create service` command is entered from the `isql` command line for the pubs2 database:

```
1> use pubs2
```
create service

2> go
1> create service xmlservice userpath "testing" type xml
   as select @@version
2> go

The newly created user-defined Web service is then deployed:

1> sp_webservices 'deploy', 'xmlservice'
2> go

Note For details on the deploy option, see sp_webservices on page 660.

The WSDL for user-defined Web service is at
http://myhost:8181/services/pubs2/testing?wsdl

You can invoke the user-defined Web service from a browser at the following
URL, where bob and bob123 are the user ID and password of the creator of the
user-defined Web service:

username=bob&password=bob123

The output displays as XML in the browser window.

Example 3  A user-defined Web service is made available to a SOAP client to
execute the stored procedure sp_who. One argument is supplied, and the
optional userpath token is specified:

create service sp_who_service userpath
   'myservices/args' type soap @loginname varchar(30) as
   exec sp_who @loginname

The Web service is created as sp_who_service in the pubs2 database and, after
being deployed, it is accessible at


The WSDL for the service is available at

The signature for the Web method, described in the WSDL file, is:

DataReturn[] sp_who_service (xsd:string username,
   xsd:string password, xsd:string loginname)

The new service is invoked by a SOAP client with one parameter, loginname,
of type varchar(30).
CHAPTER 1  Commands

Usage

Except for the following differences, the resulting stored procedure behaves the same as a stored procedure created with the `create procedure` command, follows existing stored procedure rules for execution, replication, `sp_helptext`, and recompilation, and is executable from `isql`:

- The resulting stored procedure can be dropped only with the `drop service` command, not the `drop procedure` command.
- The `syscomments` table is populated with DDL necessary to recreate the `create service` command.
- The specified service name may not create a stored procedure group.

**Note** To make a user-defined Web service available through the ASE Web Services Engine, you must use the `deploy` option of `sp_webservices`. However, the stored procedure for a user-defined Web service is accessible from `isql`, even if it has not been deployed.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

create service permission defaults to the Database Owner, who can transfer it to other users.

Permission to use a Web service must be granted explicitly with the `grant` command and may be revoked with the `revoke` command.

Permissions on objects at service creation  When you create a Web service, Adaptive Server makes no permission checks on objects, such as tables and views, that are referenced by the service. Therefore, you can create a Web service successfully even though you do not have access to its objects. All permission checks occur when a user executes the Web service.

Permissions on objects at Web service execution  When the Web service is executed, permission checks on objects depend upon whether the Web service and all referenced objects are owned by the same user.

- If the Web service’s objects are owned by different users, the invoker must have been granted direct access to the objects. For example, if the Web service performs a select from a table that the user cannot access, the Web service execution fails.
If a Web service and its objects are owned by the same user, however, special rules apply. The invoker automatically has “implicit permission” to access the Web service’s objects even though the invoker could not access them directly. Without having to grant users direct access to your tables and views, you can give them restricted access with a stored procedure. In this way, a stored procedure can be a security mechanism. For example, invokers of the Web service might be able to access only certain rows and columns of your table.

A detailed description of the rules for implicit permissions is discussed in the System Administration Guide.

### Auditing

Values in event and extrainfo columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 11    | create       | create services           | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

See also
create table

Description

- Creates new tables and optional integrity constraints.
- Defines computed columns when a table is created.
- Defines encrypted columns and decrypt defaults on encrypted columns when you create a table.
- Defines the table’s partition property when the table is created. Syntax for creating table partitions is listed separately. See “Syntax 2” on page 164.

Syntax

```sql
create table 
([database.]owner.]table_name [column_name datatype] 
[default {constant_expression }| user | null)]
[ (identity | null | not null)]
[off row | [in row [((size_in_bytes)]]]
[constraint constraint_name]
{ (unique | primary key) 
[clustered | nonclustered] [asc | desc] 
[with {fillfactor = pct, 
 max_rows_per_page = num_rows,} 
 reservepagegap = num_pages)]

[on segment_name]
| references [[database.]owner.]ref_table 
 { (ref_column)}
| (match full) 
| check (search_condition))}
| (match full)...

[encrypt [with key_name] [decrypt_default constant_expression | null]],
[constraint [[database.]owner.]key_name]
{ (unique | primary key) 
[clustered | nonclustered] 
(column_name [asc | desc] 
{[, column_name [asc | desc]]...}) 
[with {fillfactor = pct, 
 max_rows_per_page = num_rows,} 
 reservepagegap = num_pages)]

[on segment_name]
| foreign key (column_name [, column_name]...)
 references [[database.]owner.]ref_table 
 { (ref_column [, ref_column]...)}
| (match full) 
| check (search_condition) ...)
{[, (next_column | next_constraint) ...]}
| lock {datarows | datapages | allpages)]
[with {max_rows_per_page = num_rows, 
 exp_row_size = num_bytes, 
 reservepagegap = num_pages, 
 identity_gap = value}] 

[on segment_name]
```
create table

[[partition_clause]
[[external table] at pathname]

Syntax 2  Use this syntax for partitions

partition_clause::=  

definitions: 

| partition by range (column_name[, column_name]...)  
|   ([partition_name] values <= ([constant | MAX]  
|     [. (constant | MAX)] ... [on segment_name]  
|     [. partition_name] values <= ([constant | MAX]  
|       [. (constant | MAX)] ... [on segment_name]) ...  
|   | partition by hash (column_name[, column_name]...)  
|     { (partition_name [on segment_name]  
|        [, partition_name [on segment_name]]...)  
|     | number_of_partitions  
|        [on (segment_name[, segment_name] ...)])}  
| partition by list (column_name)  
|     { (partition_name) values (constant[, constant] ...)  
|        [on segment_name]  
|     | number_of_partitions  
|        [on (segment_name[, segment_name] ...)])}  
| partition by roundrobin  
|     { (partition_name [on segment_name]  
|        [, partition_name [on segment_name]]...)  
|     | number_of_partitions  
|        [on (segment_name[, segment_name] ...)])}

Syntax 3  Use this syntax for computed columns

create table [[database.[owner].] table_name  
(column_name {compute | as}  
    computed_column_expression [materialized | not materialized])
table_name

is the explicit name of the new table. Specify the database name if the table
is in another database, and specify the owner’s name if more than one table
of that name exists in the database. The default value for owner is the current
user, and the default value for database is the current database.

You cannot use a variable for the table name. The table name must be unique
within the database and to the owner. If you have set quoted_identifier on,
you can use a delimited identifier for the table name. Otherwise, it must
conform to the rules for identifiers. For more information about valid table
names, see “Identifiers” on page 345 in Chapter 4, “Expressions,
Identifiers, and Wildcard Characters,” of Reference Manual: Building
Blocks.

You can create a temporary table by preceding the table name with either a
 pound sign (#) or “tempdb.”. For more information, see “Tables beginning
with # (temporary tables)” on page 348 in Chapter 4, “Expressions,
Identifiers, and Wildcard Characters,” of Reference Manual: Building
Blocks.

You can create a table in a different database, as long as you are listed in the
sysusers table and have create table permission for that database. For
example, you can use either of the following to create a table called newtable
in the database otherdb:

create table otherdb..newtable
create table otherdb.yourname.newtable

column_name

is the name of the column in the table. It must be unique in the table. If you
have set quoted_identifier on, you can use a delimited identifier for the
column. Otherwise, it must conform to the rules for identifiers. For more
information about valid column names, see Chapter 4, “Expressions,
Identifiers, and Wildcard Characters,” of Reference Manual: Building
Blocks.

datatype

is the datatype of the column. System or user-defined datatypes are
acceptable. Certain datatypes expect a length, n, in parentheses:

datatype (n)

Others expect a precision, p, and scale, s:
create table

datatype (p, s)


If Java is enabled in the database, datatype can be the name of a Java class, either a system class or a user-defined class, that has been installed in the database. See Java in Adaptive Server Enterprise for more information.

default

specifies a default value for a column. If you specify a default, and the user does not provide a value for the column when inserting data, Adaptive Server inserts the default value. The default can be a constant expression or a built-in, to insert the name of the user who is performing the insert, or null, to insert the null value. Adaptive Server generates a name for the default in the form of `tabname_colname_objid`, where `tabname` is the first 10 characters of the table name, `colname` is the first 5 characters of the column name, and `objid` is the object ID number for the default. Defaults declared for columns with the IDENTITY property have no effect on column values.

You can reference global variables in the default section of `create table` statements that do not reference database objects. You cannot, however, use global variables in the check section of `create table`.

constant_expression

is a constant expression to use as a default value for the column. It cannot include global variables, the name of any columns, or other database objects, but can include built-in functions that do not reference database objects. This default value must be compatible with the datatype of the column, or Adaptive Server generates a datatype conversion error when attempting to insert the default.

user | null

specifies that Adaptive Server should insert the user name or the null value as the default if the user does not supply a value. For user, the datatype of the column must be either char (30) or varchar (30). For null, the column must allow null values.
encrypt [with key_name]
creates an encrypted column. Specify the database name if the key is in another database. Specify the owner’s name if key_name is not unique to the database. The default value for owner is the current user, and the default value for database is the current database.

The table creator must have select permission on the key. If you do not supply key_name, Adaptive Server looks for a default key in the database.

For more information, see Chapter 3, “Encrypted Data,” in User Guide for Encrypted Columns.

decrypt_default constant_expression
specifies that this column returns a default value for users who do not have decrypt permissions, and constant_expression is the constant value Adaptive Server returns on select statements instead of the decrypted value. The value can be NULL on nullable columns only. If the decrypt default value cannot be converted to the column’s data type, Adaptive Server catches the conversion error only when it executes the query.

identity
indicates that the column has the IDENTITY property. Each table in a database can have one IDENTITY column with a datatype of:

- exact numeric and a scale of 0; or
- Any of the integer datatypes, including signed or unsigned bigint, int, smallint, or tinyint.

IDENTITY columns are not updatable and do not allow nulls.

IDENTITY columns are used to store sequential numbers—such as invoice numbers or employee numbers—that are generated automatically by Adaptive Server. The value of the IDENTITY column uniquely identifies each row in a table.
null | not null
specifies Adaptive Server behavior during data insertion if no default exists.

null specifies that Adaptive Server assigns a null value if a user does not provide a value.

not null specifies that a user must provide a non-null value if no default exists.

The properties of a bit-type column must always be not null.

If you do not specify null or not null, Adaptive Server uses not null by default. However, you can switch this default using `sp_dboption` to make the default compatible with the SQL standards.

off row | in row
specifies whether a Java-SQL column is stored separate from the row (off row) or in storage allocated directly in the row (in row).

The default value is off row. For more information, see *Java in Adaptive Server Enterprise*.

size_in_bytes
specifies the maximum size of the in-row column. An object stored in-row can occupy up to approximately 16K bytes, depending on the page size of the database server and other variables. The default value is 255 bytes.

constraint
introduces the name of an integrity constraint.

constraint_name
is the name of the constraint. It must conform to the rules for identifiers and be unique in the database. If you do not specify the name for a referential or check constraint, Adaptive Server generates a name in the form `tabname_colname_objectid` where:

- `tabname` – is the first 10 characters of the table name
- `colname` – is the first 5 characters of the column name
- `objectid` – is the object ID number for the constraint

If you do not specify the name for a unique or primary key constraint, Adaptive Server generates a name in the format `tabname_colname_tabindid`, where `tabindid` is a string concatenation of the table ID and index ID.
unique  
constrains the values in the indicated column or columns so that no two rows have the same value. This constraint creates a unique index that can be dropped only if the constraint is dropped using alter table.

primary key  
constrains the values in the indicated column or columns so that no two rows have the same value, and so that the value cannot be NULL. This constraint creates a unique index that can be dropped only if the constraint is dropped using alter table.

clustered | nonclustered  
specifies that the index created by a unique or primary key constraint is a clustered or nonclustered index. clustered is the default for primary key constraints; nonclustered is the default for unique constraints. There can be only one clustered index per table. See create index for more information.

asc | desc  
specifies whether the index created for a constraint is to be created in ascending or descending order for each column. The default is ascending order.
fillfactor

specifies how full Adaptive Server makes each page when it creates a new index on existing data. The fillfactor percentage is relevant only when the index is created. As the data changes, the pages are not maintained at any particular level of fullness.

The default for fillfactor is 0; this is used when you do not include with fillfactor in the create index statement (unless the value has been changed with sp_configure). When specifying a fillfactor, use a value between 1 and 100.

A fillfactor of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes. There is seldom a reason to change the fillfactor.

If the fillfactor is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A fillfactor of 100 makes sense only for read-only tables—tables to which no data is ever added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A fillfactor of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small fillfactor values cause each index (or index and data) to take more storage space.

If Component Integration Services is enabled, you cannot use fillfactor for remote servers.

Warning! Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.
max_rows_per_page

limits the number of rows on data pages and the leaf-level pages of indexes. Unlike fillfactor, the max_rows_per_page value is maintained when data is inserted or deleted.

If you do not specify a value for max_rows_per_page, Adaptive Server uses a value of 0 when creating the table. Values for tables and clustered indexes are between 0 and 256. The maximum number of rows per page for nonclustered indexes depends on the size of the index key; Adaptive Server returns an error message if the specified value is too high.

A max_rows_per_page of 0 creates clustered indexes with full data pages and nonclustered indexes with full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes.

Using low values for max_rows_per_page reduces lock contention on frequently accessed data. However, using low values also causes Adaptive Server to create new indexes with pages that are not completely full, uses more storage space, and may cause more page splits.

If Component Integration Services is enabled, and you create a proxy table, then max_rows_per_page is ignored. Proxy tables do not contain any data. If max_rows_per_page is used to create a table, and later a proxy table is created to reference that table, then the max_rows_per_page limits apply when you insert or delete through the proxy table.

on segment_name

when used with the constraint option, specifies that the index is to be created on the named segment. Before the on segment_name option can be used, the device must be initialized with disk init, and the segment must be added to the database with sp_addsegment. See your System Administrator or use sp_helpsegment for a list of the segment names available in your database.

If you specify clustered and use the on segment_name option, the entire table migrates to the segment you specify, since the leaf level of the index contains the actual data pages.
references
specifies a column list for a referential integrity constraint. You can specify only one column value for a column constraint. By including this constraint with a table that references another table, any data inserted into the referencing table must already exist in the referenced table.

To use this constraint, you must have references permission on the referenced table. The specified columns in the referenced table must be constrained by a unique index (created by either a unique constraint or a create index statement). If no columns are specified, there must be a primary key constraint on the appropriate columns in the referenced table. Also, the datatypes of the referencing table columns must match the datatype of the referenced table columns.

foreign key
specifies that the listed columns are foreign keys in this table whose target keys are the columns listed in the following references clause. The foreign-key syntax is permitted only for table-level constraints, not for column-level constraints.

ref_table
is the name of the table that contains the referenced columns. You can reference tables in another database. Constraints can reference as many as 192 user tables and internally generated worktables.

ref_column
is the name of the column or columns in the referenced table.

match full
specifies that if all values in the referencing columns of a referencing row are:

• Null – the referential integrity condition is true.
• Non-null values – if there is a referenced row where each corresponding column is equal in the referenced table, then the referential integrity condition is true.

If they are neither, then the referential integrity condition is false when:

• All values are non-null and not equal, or
• Some of the values in the referencing columns of a referencing row are non-null values, while others are null.
check
specifies a search_condition constraint that Adaptive Server enforces for all
the rows in the table. You can specify check constraints as table or column
constraints; create table allows multiple check constraints in a column
definition.

Although you can reference global variables in the default section of create
table statements, you cannot use them in the check section.

search_condition
is the check constraint on the column values. These constraints can include:

- A list of constant expressions introduced with in
- A set of conditions introduced with like, which may contain wildcard
  characters

Column and table check constraints can reference any columns in the table.
An expression can include arithmetic operators and functions. The
search_condition cannot contain subqueries, aggregate functions, host
variables, or parameters.

next_column | next_constraint
indicates that you can include additional column definitions or table
constraints (separated by commas) using the same syntax described for a
column definition or table constraint definition.

lock datarows | datapages | allpages
specifies the locking scheme to be used for the table. The default is the
server-wide setting for the configuration parameter lock scheme.

exp_row_size = num_bytes
specifies the expected row size; applies only to datarows and datapages
locking schemes, and only to tables with variable-length rows. Valid values
are 0, 1, and any value between the minimum and maximum row length for
the table. The default value is 0, which means a server-wide setting is
applied.

reservepagegap = num_pages
specifies the ratio of filled pages to empty pages that are to be left during
extent I/O allocation operations. For each specified num_pages, an empty
page is left for future expansion of the table. Valid values are 0 – 255. The
default value is 0.

with identity_gap
specifies the identity gap for the table. This value overrides the system
identity gap setting for this table only.
value
is the identity gap amount. For more information about setting the identity
gap, see “IDENTITY columns” on page 197.

on segment_name
specifies the name of the segment on which to place the table. When using
on segment_name, the logical device must already have been assigned to the
database with create database or alter database, and the segment must have
been created in the database with sp_addsegment. See your System
Administrator or use sp_helpsegment for a list of the segment names
available in your database.

When used for partitions, specifies the segment on which to place the
partition.

external table
specifies that the object is a remote table or view. external table is the default,
so specifying this is optional.

partition by range
specifies records are to be partitioned according to specified ranges of
values in the partitioning column or columns.

column_name
when used in the partition_clause, specifies a partition key column.

partition_name
specifies the name of a new partition on which table records are stored.
Partition names must be unique within the set of partitions on a table or
index. Partition names can be delimited identifiers if set quoted_identifier is
on. Otherwise, they must be valid identifiers.

If partition_name is omitted, Adaptive Server creates a name in the form
table_name_partition_id. Adaptive Server truncates partition names that
exceed the allowed maximum length.

on segment_name
when used in the partition_clause, specifies the segment on which the
partition is to be placed. Before the on segment_name option can be used,
the device must be initialized with disk init, and the segment must be added
to the database using the sp_addsegment system procedure. See your
System Administrator or use sp_helpsegment for a list of the segment names
available in your database.
values <= constant | MAX
    specifies the inclusive upper bound of values for a named partition.
    Specifying a constant value for the highest partition bound imposes an
    implicit integrity constraint on the table. The keyword MAX specifies the
    maximum value in a given datatype.

partition by hash
    specifies records are to be partitioned by a system-supplied hash function.
    The function computes the hash value of the partition keys that specify the
    partition to which records are assigned.

partition by list
    specifies records are to be partitioned according to literal values specified in
    the named column. Only one column can partition a list-partitioned table.
    You can specify up to 250 distinct list values for each partition.

partition by round-robin
    specifies records are to be partitioned in a sequential manner. A round-robin
    partitioned table has no partitioning key. Neither the user nor the optimizer
    knows the partition of a particular record.

at pathname
    specifies the location of the remote object. Using the at pathname clause
    results in the creation of a proxy table.

    pathname takes the form server_name.dbname.owner.object;aux1.aux2,
    where:

    • server_name (required) – is the name of the server that contains the
      remote object.
    • dbname (optional) – is the name of the database managed by the remote
      server that contains this object.
    • owner (optional) – is the name of the remote server user that owns the
      remote object.
    • object (required) – is the name of the remote table or view.
    • aux1.aux2 (optional) – is a string of characters that is passed to the
      remote server during a create table or create index command. This string
      is used only if the server is class db2. aux1 is the DB2 database in which
      to place the table, and aux2 is the DB2 tablespace in which to place the
      table.

{compute | as}
    reserved keywords that you can use interchangeably to indicate that a
    column is a computed column.
create table

computed_column_expression

is any valid T-SQL expression that does not contain columns from other tables, local variables, aggregate functions, or subqueries. It can be one or a combination of column name, constant, function, global variable, or case expression, connected by one or more operators. You cannot cross-reference between computed columns except when virtual computed columns reference materialized computed columns.

materialized | not materialized

specifies whether or not the computed column is materialized and physically stored in the table. If neither keyword is specified, a computed column by default is not materialized, and thus not physically stored in the table.

using clustered

indicates you are creating a virtually-hashed table. The list of columns are treated as key columns for this table.

Examples

Example 1 Creates a virtually-hashed table named orders on the pubs2 database on the order_seg segment:

```sql
create table orders(
id int,
age int,
primary key using clustered (id, age) = (10, 1) with max 1000 key)
on order_seg
```

The layout for the data is:

- The order_seg segment starts on page ID 51200.
- The ID for the first data object allocation map (OAM) page is 51201.
- The maximum rows per page is 168.
- The row size is 10.
- The root index page of the overflow clustered region is 51217.
Example 2 Creates a virtually-hashed table named orders on the pubs2 database on the order_seg segment:

```sql
create table orders(
  id int default NULL,
  age int,
  primary key using clustered (id,age) = (10,1) with max
  100 key,
  name varchar(30)
)
```

```sql
on order_seg
```

The layout for the data is:

- The order_seg segment starts on page ID 51200.
- The ID for the first data OAM page is 51201.
- The maximum rows per page is 42.
- The row size is 45.
- The root index page of the overflow clustered region is 51217.
Example 3 Creates the foo table using the `@@spid` global variable with the default parameter:

```
create table foo (a int , b int default @@spid)
```

Example 4 Creates the titles table:

```
create table titles
(title_id tid not null,
title varchar (80) not null,
type char (12) not null,
pub_id char (4) null,
price money null,
advance money null,
total_sales int null,
notes varchar (200) null,
pubdate datetime not null,
contract bit not null)
```

Example 5 Creates the compute table. The table name and the column names, `max` and `min`, are enclosed in double quotes because they are reserved words. The `total score` column name is enclosed in double quotes because it contains an embedded blank. Before creating this table, you must set `quoted_identifier` on:

```
create table "compute"
("max" int, "min" int, "total score" int)
```
Example 6 Creates the sales table and a clustered index in one step with a unique constraint. (In the pubs2 database installation script, there are separate create table and create index statements):

```sql
create table sales
(stor_id char (4) not null,
ord_num varchar (20) not null,
date datetime not null,
unique clustered (stor_id, ord_num))
```

Example 7 Creates the salesdetail table with two referential integrity constraints and one default value. There is a table-level, referential integrity constraint named salesdet_constr and a column-level, referential integrity constraint on the title_id column without a specified name. Both constraints specify columns that have unique indexes in the referenced tables (titles and sales). The default clause with the qty column specifies 0 as its default value:

```sql
create table salesdetail
(stor_id char (4) not null,
ord_num varchar (20) not null,
title_id tid not null
    references titles (title_id),
qty smallint default 0 not null,
discount float not null,
constraint salesdet_constr
    foreign key (stor_id, ord_num)
    references sales (stor_id, ord_num))
```

Example 8 Creates the table publishers with a check constraint on the pub_id column. This column-level constraint can be used in place of the pub_idrule included in the pubs2 database:

```sql
create rule pub_idrule
    as @pub_id in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
create table publishers
(pub_id char (4) not null
    check (pub_id in ("1389", "0736", "0877", "1622", "1756")
    or pub_id like "99[0-9][0-9]"),
pub_name varchar (40) null,
city varchar (20) null,
state char (2) null)
```
Example 9  Specifies the ord_num column as the IDENTITY column for the sales_daily table. The first time you insert a row into the table, Adaptive Server assigns a value of 1 to the IDENTITY column. On each subsequent insert, the value of the column increments by 1:

```
create table sales_daily
    (stor_id   char (4) not null,
     ord_num   numeric (10,0) identity,
     ord_amt   money      null)
```

Example 10  Specifies the datapages locking scheme for the new_titles table and an expected row size of 200:

```
create table new_titles (
    title_id    tid,
    title       varchar (80) not null,
    type        char (12) ,
    pub_id      char (4) null,
    price       money null,
    advance     money null,
    total_sales int null,
    notes       varchar (200) null,
    pubdate     datetime,
    contract    bit )
lock datapages
    with exp_row_size = 200
```

Example 11  Specifies the datarows locking scheme and sets a reservepagegap value of 16 so that extent I/O operations leave 1 blank page for each 15 filled pages:

```
create table new_publishers ( 
    pub_id char (4) not null,
    pub_name varchar (40) null,
    city varchar (20) null,
    state char (2) null )
lock datarows
    with reservepagegap = 16
```

Example 12  Creates a constraint supported by a unique clustered index; the index order is ascending for stor_id and descending for ord_num:

```
create table sales_south
    (stor_id   char (4) not null,
     ord_num   varchar (20) not null,
     date      datetime not null,
     unique clustered (stor_id asc, ord_num desc))
```
Example 13  Creates a table named $t1$ at the remote server SERVER_A and creates a proxy table named $t1$ that is mapped to the remote table:

```sql
create table t1
(a int,
 b char (10))
at "SERVER_A.db1.joe.t1"
```

Example 14  Creates a table named employees. name is of type varchar, home_addr is a Java-SQL column of type Address, and mailing_addr is a Java-SQL column of type Address2Line. Both Address and Address2Line are Java classes installed in the database:

```sql
create table employees
(name varchar (30),
 home_addr Address,
 mailing_addr Address2Line)
```

Example 15  Creates a table named mytable with an identity column. The identity gap is set to 10, which means ID numbers are allocated in memory in blocks of ten. If the server fails or is shut down with no wait, the maximum gap between the last ID number assigned to a row and the next ID number assigned to a row is ten numbers:

```sql
create table mytable
(IdNum numeric (12,0) identity)
with identity_gap = 10
```

For more information about identity gaps, see “Managing Identity Gaps in Tables” in Chapter 7, “Creating Databases and Tables” in the *Transact-SQL User’s Guide*.

Example 16  Creates a table my_publishers, which is partitioned by list according to values in the state column. See the *Transact-SQL User’s Guide* for more information about creating table partitions.

```sql
create table my_publishers
(pub_id char (4) not null,
 pub_name varchar (40) null,
 city varchar (20) null,
 state char (2) null)
partition by list (state)
(west values ('CA', 'OR', 'WA') on seg1,
 east values ('NY', 'MA') on seg2)
```

Example 17  Creates the table fictionsales, which is partitioned by range according to values in the date column. See the *Transact-SQL User’s Guide* for more information about creating table partitions.
create table fictionsales
    (store_id int not null,
     order_num int not null,
     date datetime not null)
partition by range (date)
    (q1 values <= ('3/31/2005') on seg1,
     q2 values <= ('6/30/2005') on seg2,
     q3 values <= ('9/30/2005') on seg3,
     q4 values <= ('12/31/2005') on seg4)

Example 18 Creates the table currentpublishers, which is partitioned by round-robin. See the Transact-SQL User’s Guide for more information about creating table partitions.

    create table currentpublishers
        (pub_id char (4) not null,
         pub_name varchar (40) null,
         city varchar (20) null,
         state char (2) null)
partition by roundrobin 3 on (seg1)

Example 19 Creates the table mysalesdetail, which is partitioned by hash according to values in the ord_num column. See the Transact-SQL User’s Guide for more information.

    create table mysalesdetail
        (store_id char (4) not null,
         ord_num varchar (20) not null,
         title_id tid not null,
         qty smallint not null,
         discount float not null)
partition by hash (ord_num)
    (p1 on seg1, p2 on seg2, p3 on seg3)

Example 20 Creates a table called mytitles with one materialized computed column:

    create table mytitles
        (title_id tid not null,
         title varchar (80) not null,
         type char (12) not null,
         pub_id char (4) null,
         price money null,
         advance money null,
         total_sales int null,
         notes varchar (200) null,
         pubdate datetime not null,
         sum_sales compute price * total_sales materialized)
Example 21 Creates a table named t2 for which the ssnum column returns “??????????” when a user without decrypt permissions selects the column:

```sql
create table t2 (ssnum char(11)
  encrypt decrypt_default '??????????', ...)
```

Usage

- `create table` creates a table and optional integrity constraints. The table is created in the currently open database unless you specify a different database in the `create table` statement. You can create a table or index in another database, if you are listed in the `sysusers` table and have `create table` permission in the database.

- Space is allocated to tables and indexes in increments of one extent, or eight pages, at a time. Each time an extent is filled, another extent is allocated. To see the amount of space allocated and used by a table, use `sp_spaceused`.

- The maximum length for in-row Java columns is determined by the maximum size of a variable-length column for the table’s schema, locking style, and page size.

- `create table` performs error checking for check constraints before it creates the table.

- When using `create table` from Component Integration Services with a column defined as `char (n) NULL`, Component Integration Services creates the column as `varchar (n)` on the remote server.

Restrictions

- The maximum number of columns in a table depends on the width of the columns and the server’s logical page size:
  - The sum of the columns’ sizes cannot exceed the server’s logical page size.
  - The maximum number of columns per table cannot exceed 1024.
  - The maximum number of variable length columns for an APL table is 254.

For example, if your server uses a 2K logical page size and includes a table of integer columns, the maximum number of columns in the table is far fewer than 1024. (1024 * 4 bytes exceeds a 2K logical page size.)
You can mix variable- and fixed-length columns in a single table as long as the maximum number of columns does not exceed 1024. For example, if your server uses a 8K logical page size, a table configured for APL can have 254 nullable integer columns (these are variable length columns) and 770 non-nullable integers, for a total of 1024 columns.

- There can be as many as 2,000,000,000 tables per database and 1024 user-defined columns per table. The number of rows per table is limited only by available storage.

- Although Adaptive Server does create tables in the following circumstances, you will receive errors about size limitations when you perform DML operations:
  - If the total row size for rows with variable-length columns exceeds the maximum column size
  - If the length of a single variable-length column exceeds the maximum column size
  - For DOL tables, if the offset of any variable-length column other than the initial column exceeds the limit of 8191 bytes

- Adaptive Server reports an error if the total size of all fixed-length columns, plus the row overhead, is greater than the table’s locking scheme and page size allows. These limits are described in Table 1-12.

<table>
<thead>
<tr>
<th>Locking scheme</th>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL tables</td>
<td>2K (2048 bytes)</td>
<td>1962 bytes</td>
<td>1960 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4010 bytes</td>
<td>4008 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8106 bytes</td>
<td>8104 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16298 bytes</td>
<td>16296 bytes</td>
</tr>
</tbody>
</table>
The maximum number of bytes of variable length data per row depends on the locking scheme for the table. Table 1-13 describes the maximum size columns for an APL table:

<table>
<thead>
<tr>
<th>Locking scheme</th>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOL tables</td>
<td>2K (2048 bytes)</td>
<td>1964 bytes</td>
<td>1958 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4012 bytes</td>
<td>4006 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8108 bytes</td>
<td>8102 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300 bytes</td>
<td>16294 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If table does not include any variable length columns</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300 (subject to a max start offset of varlen = 8191)</td>
<td>8191-6-2 = 8183 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If table includes at least one variable length column.*</td>
</tr>
</tbody>
</table>

* This size includes six bytes for the row overhead and two bytes for the row length field

- Table 1-14 describes the maximum size of columns for a DOL table:

<table>
<thead>
<tr>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2K (2048 bytes)</td>
<td>1962</td>
<td>1960</td>
</tr>
<tr>
<td>4K (4096 bytes)</td>
<td>4010</td>
<td>4008</td>
</tr>
<tr>
<td>8K (8192 bytes)</td>
<td>8096</td>
<td>8104</td>
</tr>
<tr>
<td>16K (16384 bytes)</td>
<td>16298</td>
<td>16296</td>
</tr>
</tbody>
</table>

- If you create a DOL table with a variable-length column that exceeds a 8191-byte offset, you cannot add any rows to the column.
If you create tables with varchar, nvarchar, univarchar, or varbinary columns whose total defined width is greater than the maximum allowed row size, a warning message appears, but the table is created. If you try to insert more than the maximum number bytes into such a row, or to update a row so that its total row size is greater than the maximum length, Adaptive Server produces an error message, and the command fails.

**Note**  When a `create table` command occurs within an `if...else` block or a `while` loop, Adaptive Server creates the schema for the table before determining whether the condition is true. This may lead to errors if the table already exists. To avoid this situation, either make sure a view with the same name does not already exist in the database or use an `execute` statement, as follows:

```sql
if not exists (select * from sysobjects where name="my table")
begin
    execute "create table mytable (x int)"
end
```

You cannot issue `create table` with a declarative default or check constraint and then insert data into the table in the same batch or procedure. Either separate the create and insert statements into two different batches or procedures, or use `execute` to perform the actions separately.

You cannot use the following variable in `create table` statements that include defaults:

```sql
declare @p int
select @p = 2
create table t1 (c1 int default @p, c2 int)
```

Doing so results in error message 154, which says, “Variable is not allowed in default.”

SQL user-defined functions are not currently supported with `create proxy table`, `create table at remote server`, or `alter table`.

**Note** The execution of SQL functions requires the syntax `username.functionname()`.

### Column definitions

- When you create a column from a user-defined datatype:
You cannot change the length, precision, or scale.

You can use a NULL type to create a NOT NULL column, but not to create an IDENTITY column.

You can use a NOT NULL type to create a NULL column or an IDENTITY column.

You can use an IDENTITY type to create a NOT NULL column, but the column inherits the IDENTITY property. You cannot use an IDENTITY type to create a NULL column.

Only columns with variable-length datatypes can store null values. When you create a NULL column with a fixed-length datatype, Adaptive Server automatically converts it to the corresponding variable-length datatype. Adaptive Server does not inform the user of the type change.

Table 1-15 lists the fixed-length datatypes and the variable-length datatypes to which they are converted. Certain variable-length datatypes, such as moneyn, are reserved types that cannot be used to create columns, variables, or parameters:

<table>
<thead>
<tr>
<th>Original fixed-length datatype</th>
<th>Converted to</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>varchar</td>
</tr>
<tr>
<td>nchar</td>
<td>nvarchar</td>
</tr>
<tr>
<td>binary</td>
<td>varbinary</td>
</tr>
<tr>
<td>datetime</td>
<td>datetimn</td>
</tr>
<tr>
<td>float</td>
<td>floatn</td>
</tr>
<tr>
<td>bigint, int, smallint, tinyint</td>
<td>intn</td>
</tr>
<tr>
<td>unsigned bigint, unsigned int, unsigned smallint</td>
<td>uintn</td>
</tr>
<tr>
<td>decimal</td>
<td>decamin</td>
</tr>
<tr>
<td>numeric</td>
<td>numericn</td>
</tr>
<tr>
<td>money and smallmoney</td>
<td>moneyn</td>
</tr>
</tbody>
</table>

You can create column defaults in two ways: by declaring the default as a column constraint in the create table or alter table statement, or by creating the default using the create default statement and binding it to a column using sp_bindefault.

For a report on a table and its columns, execute the system procedure sp_help.

Temporary tables

Temporary tables are stored in the temporary database, tempdb.
create table

- The first 13 characters of a temporary table name must be unique per session. Such tables can be accessed only by the current Adaptive Server session. They are stored in tempdb..objects by their names plus a system-supplied numeric suffix, and they disappear at the end of the current session or when they are explicitly dropped.

- Temporary tables created with the “tempdb..” prefix are shareable among Adaptive Server user sessions. They exist until they are explicitly dropped by their owner or until Adaptive Server is restarted. Create temporary tables with the “tempdb..” prefix from inside a stored procedure only if you intend to share the table among users and sessions. To avoid inadvertent sharing of temporary tables, use the “#” prefix when creating and dropping temporary tables in stored procedures.

- Temporary tables can be used by multiple users during an Adaptive Server session. However, the specific user session usually cannot be identified because temporary tables are created with the “guest” user ID of 2. If more than one user runs the process that creates the temporary table, each user is a “guest” user so the uid values are all the same. Therefore, there is no way to know which user session in the temporary table is for a specific user. It is possible that the System Administrator can add the user to the temporary table using sp_addlogin, in which case the individual uid is available for that user’s session in the temporary table, but this circumstance is unlikely.

- You can associate rules, defaults, and indexes with temporary tables, but you cannot create views on temporary tables or associate triggers with them.

- When you create a temporary table, you can use a user-defined datatype only if the type is in tempdb..systypes. To add a user-defined datatype to tempdb for the current session only, execute sp_addtype while using tempdb. To add the datatype permanently, execute sp_addtype while using model, then restart Adaptive Server so that model is copied to tempdb.

Using indexes

- A table “follows” its clustered index. If you create a table on one segment, and then create its clustered index on another segment, the table migrates to the segment where the index is created.

- You can make inserts, updates, and selects faster by creating a table on one segment and its nonclustered indexes on another segment, if the segments are on separate physical devices. For more information, see Chapter 12, “Using clustered or nonclustered indexes,” in Transact-SQL User’s Guide.
Renaming a table or its columns

- Use sp_rename to rename a table or column.
- After renaming a table or any of its columns, use sp_depends to determine which procedures, triggers, and views depend on the table, and redefine these objects.

**Warning!** If you do not redefine these dependent objects, they will no longer work after Adaptive Server recompiles them.

Specifying ascending or descending ordering in indexes

- Use the asc and desc keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the order by clause of queries eliminates the sorting step during query processing.

Defining integrity constraints

- The create table statement helps control a database’s integrity through a series of integrity constraints as defined by the SQL standards. These integrity constraint clauses restrict the data that users can insert into a table. You can also use defaults, rules, indexes, and triggers to enforce database integrity.

  Integrity constraints offer the advantages of defining integrity controls in one step during the table creation process and of simplifying the process to create those integrity controls. However, integrity constraints are more limited in scope and less comprehensive than defaults, rules, indexes, and triggers.

- You must declare constraints that operate on more than one column as table-level constraints; declare constraints that operate on just one column as column-level constraints. Although the difference is rarely noticed by users, column-level constraints are checked only if a value in the column is being modified, while the table-level constraints are checked if there is any modification to a row, regardless of whether or not it changes the column in question.

  Place column-level constraints after the column name and datatype, before the delimiting comma (see Example 5). Enter table-level constraints as separate comma-delimited clauses (see Example 4). Adaptive Server treats table-level and column-level constraints the same way; neither way is more efficient than the other.
You can create the following types of constraints at the table level or the column level:

- A unique constraint requires that no two rows in a table have the same values in the specified columns. In addition, a primary key constraint requires that there be no null values in the column.

- A referential integrity constraint requires that the data being inserted or updated in specific columns has matching data in the specified table and columns.

- A check constraint limits the values of the data inserted into the columns.

You can also enforce data integrity by restricting the use of null values in a column (the null or not null keywords) and by providing default values for columns (the default clause).

- You can use `sp_primarykey`, `sp_foreignkey`, and `sp_commonkey` to save information in system tables, which can help clarify the relationships between tables in a database. These system procedures do not enforce the key relationships or duplicate the functions of the primary key and foreign key keywords in a create table statement. For a report on keys that have been defined, use `sp_helpkey`. For a report on frequently used joins, execute `sp_helpjoins`.

- Transact-SQL provides several mechanisms for integrity enforcement. In addition to the constraints you can declare as part of `create table`, you can create rules, defaults, indexes, and triggers. Table 1-16 summarizes the integrity constraints and describes the other methods of integrity enforcement:

<table>
<thead>
<tr>
<th>In create table</th>
<th>Other methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique constraint</td>
<td>create unique index (on a column that allows null values)</td>
</tr>
<tr>
<td>primary key constraint</td>
<td>create unique index (on a column that does not allow null values)</td>
</tr>
<tr>
<td>references constraint</td>
<td>create trigger</td>
</tr>
<tr>
<td>check constraint (table level)</td>
<td>create trigger or <code>create rule</code> and <code>sp_bindrule</code></td>
</tr>
<tr>
<td>check constraint (column level)</td>
<td>create default and <code>sp_bindefault</code></td>
</tr>
<tr>
<td>default clause</td>
<td>create default and <code>sp_bindefault</code></td>
</tr>
</tbody>
</table>
The method you choose depends on your requirements. For example, triggers provide more complex handling of referential integrity (such as referencing other columns or objects) than those declared in create table. Also, the constraints defined in a create table statement are specific for that table; unlike rules and defaults, you cannot bind them to other tables, and you can only drop or change them using alter table. Constraints cannot contain subqueries or aggregate functions, even on the same table.

- **create table** can include many constraints, with these limitations:
  - The number of unique constraints is limited by the number of indexes that a table can have.
  - A table can have only one primary key constraint.
  - You can include only one default clause per column in a table, but you can define different constraints on the same column.

For example:

```sql
create table discount_titles
    (title_id varchar (6) default "PS7777" not null
     unique clustered
     references titles (title_id)
     check (title_id like "PS%"),
     new_price money)
```

Column `title_id` of the new table `discount_titles` is defined with each integrity constraint.

- You can create error messages and bind them to referential integrity and check constraints. Create messages with `sp_addmessage` and bind them to the constraints with `sp_bindmsg`. For more information, see `sp_addmessage` and `sp_bindmsg`.

- Adaptive Server evaluates check constraints before enforcing the referential constraints, and evaluates triggers after enforcing all the integrity constraints. If any constraint fails, Adaptive Server cancels the data modification statement; any associated triggers do not execute. However, a constraint violation does not roll back the current transaction.

- In a referenced table, you cannot update column values or delete rows that match values in a referencing table. Update or delete from the referencing table first, then try updating or deleting from the referenced table.

- You must drop the referencing table before you drop the referenced table; otherwise, a constraint violation occurs.
create table

- For information about constraints defined for a table, use `sp_helpconstraint`.

**Unique and primary key constraints**

- You can declare unique constraints at the column level or the table level. Unique constraints require that all values in the specified columns be unique. No two rows in the table can have the same value in the specified column.

- A primary key constraint is a more restrictive form of unique constraint. Columns with primary key constraints cannot contain null values.

**Note** The create table statement’s unique and primary key constraints create indexes that define unique or primary key attributes of columns. `sp_primarykey`, `sp_foreignkey`, and `sp_commonkey` define logical relationships between columns. These relationships must be enforced using indexes and triggers.

- Table-level unique or primary key constraints appear in the create table statement as separate items and must include the names of one or more columns from the table being created.

- Unique or primary key constraints create a unique index on the specified columns. The unique constraint in Example 3 creates a unique, clustered index, as does the statement:

  ```sql
  create unique clustered index salesind
  on sales (stor_id, ord_num)
  ```

  The only difference is the index name, which you could set to `salesind` by naming the constraint.

- The definition of unique constraints in the SQL standard specifies that the column definition cannot allow null values. By default, Adaptive Server defines the column as not allowing null values (if you have not changed this using `sp_dboption`) when you omit null or not null in the column definition. In Transact-SQL, you can define the column to allow null values along with the unique constraint, since the unique index used to enforce the constraint allows you to insert a null value.

- Unique constraints create unique, nonclustered indexes by default; primary key constraints create unique, clustered indexes by default. There can be only one clustered index on a table, so you can specify only one unique clustered or primary key clustered constraint.
• The unique and primary key constraints of create table offer a simpler alternative to the create index statement. However, they have the following limitations:
  • You cannot create nonunique indexes.
  • You cannot use all the options provided by create index.
  • You must drop these indexes using alter table drop constraint.

Referential integrity constraints
• Referential integrity constraints require that data inserted into a referencing table that defines the constraint must have matching values in a referenced table. A referential integrity constraint is satisfied for either of the following conditions:
  • The data in the constrained columns of the referencing table contains a null value.
  • The data in the constrained columns of the referencing table matches data values in the corresponding columns of the referenced table.

Using the pubs2 database as an example, a row inserted into the salesdetail table (which records the sale of books) must have a valid title_id in the titles table. salesdetail is the referencing table and titles is the referenced table. Currently, pubs2 enforces this referential integrity using a trigger. However, the salesdetail table could include this column definition and referential integrity constraint to accomplish the same task:

```sql
title_id tid
  references titles (title_id)
```

• The maximum number of table references allowed for a query is 192. Use sp_helpconstraint to check a table’s referential constraints.

• A table can include a referential integrity constraint on itself. For example, the store_employees table in pubs3, which lists employees and their managers, has the following self-reference between the emp_id and mgr_id columns:

```sql
emp_id id primary key,
mgr_id id null
  references store_employees (emp_id),
```

This constraint ensures that all managers are also employees, and that all employees have been assigned a valid manager.
create table

- You cannot drop the referenced table until the referencing table is dropped or the referential integrity constraint is removed (unless it includes only a referential integrity constraint on itself).

- Adaptive Server does not enforce referential integrity constraints for temporary tables.

- To create a table that references another user’s table, you must have references permission on the referenced table. For information about assigning references permissions, see the grant command.

- Table-level, referential integrity constraints appear in the create table statement as separate items. They must include the foreign key clause and a list of one or more column names.

  Column names in the references clause are optional only if the columns in the referenced table are designated as a primary key through a primary key constraint.

  The referenced columns must be constrained by a unique index in that referenced table. You can create that unique index using either the unique constraint or the create index statement.

- The datatypes of the referencing table columns must match the datatypes of the referenced table columns. For example, the datatype of col1 in the referencing table (test_type) matches the datatype of pub_id in the referenced table (publishers):

  ```sql
  create table test_type
  (col1 char (4) not null
   references publishers (pub_id),
  col2 varchar (20) not null)
  ```

- The referenced table must exist at the time you define the referential integrity constraint. For tables that cross-reference one another, use the create schema statement to define both tables simultaneously. As an alternative, create one table without the constraint and add it later using alter table. See create schema or alter table for more information.

- The create table referential integrity constraints offer a simple way to enforce data integrity. Unlike triggers, they cannot:
  - Cascade changes through related tables in the database
  - Enforce complex restrictions by referencing other columns or database objects
  - Perform “what-if” analysis
Referential integrity constraints do not roll back transactions when a data modification violates the constraint. Triggers allow you to choose whether to roll back or continue the transaction depending on how you handle referential integrity.

**Note**  Adaptive Server checks referential integrity constraints before it checks any triggers, so a data modification statement that violates the constraint does not also fire the trigger.

Using cross-database referential integrity constraints

- When you create a cross-database constraint, Adaptive Server stores the following information in the `sysreferences` system table of each database:

<table>
<thead>
<tr>
<th>Information stored in <code>sysreferences</code></th>
<th>Columns with information about the referenced table</th>
<th>Columns with information about the referencing table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key column IDs</td>
<td>refkey1 through refkey16</td>
<td>fokey1 through fokey16</td>
</tr>
<tr>
<td>Table ID</td>
<td>reftabid</td>
<td>tableid</td>
</tr>
<tr>
<td>Database ID</td>
<td>pmrydbid</td>
<td>frgndbid</td>
</tr>
<tr>
<td>Database name</td>
<td>pmrydbname</td>
<td>frgndbname</td>
</tr>
</tbody>
</table>

- You can drop the referencing table or its database without problems. Adaptive Server automatically removes the foreign-key information from the referenced database.

- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains the referenced table, or
  - Rename either database with `sp_renamedb`.

You must remove the cross-database constraint with `alter table` before you can do any of these actions.

- Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump *both* of the affected databases.

**Warning!** Loading earlier dumps of databases containing cross-database constraints could cause database corruption.
create table

- The sysreferences system table stores the name and the ID number of the external database. Adaptive Server cannot guarantee referential integrity if you use load database to change the database name or to load it onto a different server.

**Warning!** Before dumping a database to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

**check constraints**

- A check constraint limits the values a user can insert into a column in a table. A check constraint specifies a search_condition that any non-null value must pass before it is inserted into the table. A search_condition can include:
  - A list of constant expressions introduced with in
  - A range of constant expressions introduced with between
  - A set of conditions introduced with like, which can contain wildcard characters

An expression can include arithmetic operators and Transact-SQL built-in functions. The search_condition cannot contain subqueries, aggregate functions, or a host variable or parameter. Adaptive Server does not enforce check constraints for temporary tables.

- If the check constraint is a column-level check constraint, it can reference only the column in which it is defined; it cannot reference other columns in the table. Table-level check constraints can reference any column in the table.

- create table allows multiple check constraints in a column definition.

- check integrity constraints offer an alternative to using rules and triggers. They are specific to the table in which they are created, and cannot be bound to columns in other tables or to user-defined datatypes.

- check constraints do not override column definitions. If you declare a check constraint on a column that allows null values, you can insert NULL into the column, implicitly or explicitly, even though NULL is not included in the search_condition. For example, if you create a check constraint specifying “pub_id in (‘1389’, ‘0736’, ‘0877’, ‘1622’, ‘1756’)” or “@amount > 10000” in a table column that allows null values, you can still insert NULL into that column. The column definition overrides the check constraint.
IDENTITY columns

- The first time you insert a row into the table, Adaptive Server assigns the IDENTITY column a value of 1. Each new row gets a column value that is 1 higher than the last value. This value takes precedence over any defaults declared for the column in the create table statement or bound to the column with sp_bindefault.

The maximum value that can be inserted into an IDENTITY column is $10^{\text{precision}} - 1$ for a numeric. For integer identities, it is the maximum permissible value of its type (such as 255 for tinyint, 32767 for smallint).


- Inserting a value into the IDENTITY column allows you to specify a seed value for the column or to restore a row that was deleted in error. The table owner, Database Owner, or System Administrator can explicitly insert a value into an IDENTITY column after using set identity_insert table_name on for the base table. Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the value. You can insert any positive integer.

- You can reference an IDENTITY column using the syb_identity keyword, qualified by the table name where necessary, in place of the actual column name.

- System Administrators can use the auto identity database option to automatically include a 10-digit IDENTITY column in new tables. To turn on this feature in a database, use:

  ```sql
  sp_dboption database_name, "auto identity", "true"
  ```

Each time a user creates a table in the database without specifying either a primary key, a unique constraint, or an IDENTITY column, Adaptive Server automatically defines an IDENTITY column. This column, SYB_IDENTITY_COL, is not visible when you retrieve columns with the select * statement. You must explicitly include the column name in the select list.
Server failures can create gaps in IDENTITY column values. Gaps can also occur due to transaction rollbacks, the deletion of rows, or the manual insertion of data into the IDENTITY column. The maximum size of the gap depends on the setting of the identity burning set factor and identity grab size configuration parameters, the identity_gap value given in the create table or select into statement. For details about using the different methods to set the identity gap, see “Managing Identity Gaps in Tables” in Chapter 7, “Creating Databases and Tables” in the Transact-SQL User’s Guide.

Specifying a locking scheme

- To specify the locking scheme for a table, use the keyword lock and one of the following locking schemes:
  - Allpages locking, which locks data pages and the indexes affected by queries
  - Datapages locking, which locks only data pages
  - Datarows locking, which locks only data rows

If you do not specify a locking scheme, the default locking scheme for the server is used. The server-wide default is set with the configuration parameter lock scheme.

- You can change the locking scheme for a table with the alter table command.

Space management properties

- The space management properties fillfactor, max_rows_per_page, exp_row_size, and reservepagegap help manage space usage for tables in the following ways:
  - fillfactor leaves extra space on pages when indexes are created, but the fillfactor is not maintained over time.
  - max_rows_per_page limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables, since reducing the number of rows can reduce lock contention. If you specify a max_rows_per_page value and datapages or datarows locking, a warning message is printed. The table is created, and the value is stored in sysindexes, but it is applied only if the locking scheme is changed later to allpages.
exp_row_size specifies the expected size of a data row. It applies only to data rows, not to indexes, and applies only to data-only-locked tables that have variable-length columns. It is used to reduce the number of forwarded rows in data-only-locked tables. It is needed mainly for tables where rows have null or short columns when first inserted, but increase in size as a result of subsequent updates.

exp_row_size reserves space on the data page for the row to grow to the specified size. If you specify exp_row_size when you create an allpages-locked table, a warning message is printed. The table is created, and the value is stored in sysindexes, but it is only applied if the locking scheme is changed later to datapages or datarows.

reservepagegap specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to both data and index pages, in all locking schemes.

Table 1-18 shows the valid combinations of space management properties and locking scheme. If a create table command includes incompatible combinations, a warning message is printed and the table is created. The values are stored in system tables, but are not applied. If the locking scheme for a table changes so that the properties become valid, then they are used.

<table>
<thead>
<tr>
<th>Property</th>
<th>allpages</th>
<th>datapages</th>
<th>datarows</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_rows_per_page</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>exp_row_size</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>reservepagegap</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fillfactor</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1-19 shows the default values and the effects of using default values for the space management properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Effect of using the default</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_rows_per_page</td>
<td>0</td>
<td>Fits as many rows as possible on the page, up to a maximum of 255</td>
</tr>
<tr>
<td>exp_row_size</td>
<td>0</td>
<td>Uses the server-wide default value, set with the configuration parameter default exp_row_size percent</td>
</tr>
<tr>
<td>reservepagegap</td>
<td>0</td>
<td>Leaves no empty pages during extent allocations</td>
</tr>
<tr>
<td>fillfactor</td>
<td>0</td>
<td>Fully packs leaf pages, with space left on index pages</td>
</tr>
</tbody>
</table>
Using `exp_row_size`

- If an application inserts short rows into a data-only-locked table and updates them later so that their length increases, use `exp_row_size` to reduce the number of times that rows in data-only-locked tables are forwarded to new locations.

Using `reservepagegap`

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The `reservepagegap` keyword causes these commands to leave empty pages so that subsequent page allocations happen close to the page being split or close to the page from which a row is being forwarded. Table 1-20 shows when `reservepagegap` is applied.

### Table 1-20: When `reservepagegap` is applied

<table>
<thead>
<tr>
<th>Command</th>
<th>Applies to data pages</th>
<th>Applies to index pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast <code>bcp</code></td>
<td>Yes</td>
<td>Fast <code>bcp</code> is not used if indexes exist</td>
</tr>
<tr>
<td>Slow <code>bcp</code></td>
<td>Only for heap tables, not for tables with a clustered index</td>
<td></td>
</tr>
<tr>
<td><code>select into</code></td>
<td>Yes</td>
<td>Extent allocation not performed</td>
</tr>
<tr>
<td><code>create index</code> or <code>alter table...constraint</code></td>
<td>Yes, for clustered indexes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>alter table...lock</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(For allpages-locking to data-only locking, or vice versa)</td>
</tr>
</tbody>
</table>

- The `reservepagegap` value for a table is stored in `sysindexes` and is applied when any of the above operations on a table are executed. To change the stored value, use `sp_chgattribute`.

- `reservepagegap` is not applied to worktables or sorts on worktables.

Using `at`

- The location information provided by the `at` keyword is the same information that is provided by `sp_addobjectdef`. The information is stored in the `sysattributes` table.

Java-SQL columns

- If Java is enabled in the database, you can create tables with Java-SQL columns. Refer to *Java in Adaptive Server Enterprise* for detailed information.
• The declared class (datatype) of the Java-SQL column must implement either the Serializable or Externalizable interface.

• When you create a table, a Java-SQL column cannot be specified:
  • As a foreign key
  • In a references clause
  • As having the UNIQUE property
  • As the primary key

• If in row is specified, the value stored cannot exceed 16K bytes, depending on the page size of the database server and other variables.

• If off row is specified:
  • The column cannot be referenced in a check constraint.
  • The column cannot be referenced in a select that specifies distinct.
  • The column cannot be specified in a comparison operator, in a predicate, or in a group by clause.

Getting information about tables

• `sp_help` displays information about tables, listing any attributes (such as cache bindings) assigned to the specified table and its indexes, giving the attribute’s class, name, integer value, character value, and comments.

• `sp_depends` displays information about the views, triggers, and procedures in the database that depend on a table.

• `sp_helpindex` reports information about the indexes created on a table.

• `sp_helppartition` reports information about the table’s partition properties.

Creating tables with partitions

• Before you create a table with partitions, you must prepare the disk devices and segments that you will use for the partitions.

• Range partitioning is dependent on sort order. If the sort order is changed, you must repartition the table for the new sort order.

• Range partition bounds must be in ascending order according to the order in which the partitions are created.

• A column of text, unitext, image, or bit, Java datatype, or computed column cannot be part of a partition key, but a partitioned table can include columns with these datatypes. A composite partition key can contain up to 31 columns.
For range and hash partitions, the partition key can be a composite key with as many as 31 columns. In general, however, a table with more than four partition columns becomes hard to manage and is not useful.

Bound values for range and list partitions must be compatible with the corresponding partition key datatype. If a bound value is specified in a compatible but different datatype, Adaptive Server converts the bound value to the partition key’s datatype. Adaptive Server does not support:

- Explicit conversions.
- Implicit conversions that result in data loss.
- NULL as a boundary in a range-partitioned table.
- Conversions from nonbinary datatypes to binary or varbinary datatypes.

You can use NULL in a value list for list-partitioned tables.

You can partition a table that contains text and image columns, but partitioning has no effect on the way Adaptive Server stores the text and image columns because they reside on their own partition.

Adaptive Server considers NULL to be lower than any other partition key value for a given partition key column.

Creating tables with computed columns

- computed_column_expression can reference only columns in the same table.
- The deterministic property of computed_column_expression significantly affects data operations. See “Deterministic property” in the Transact-SQL User’s Guide.
- Computed columns cannot have default values, and cannot be identity or timestamp columns.
- You can specify nullability only for materialized computed columns. If you do not specify nullability, all computed columns are nullable by default. Virtual computed columns are always nullable.
- Triggers and constraints, such as check, rule, unique, primary key, or foreign key) support only materialized computed columns. You cannot use them with virtual computed columns.
• If a user-defined function in a computed column definition is dropped or becomes invalid, any computed column operations that call that function fail.

Creating tables with encrypted columns

You can encrypt these datatypes:

• int, smallint, tinyint
• unsigned int, unsigned smallint, unsigned tinyint
• bigint, unsigned bigint
• decimal, numeric
• float4, float8
• money, smallmoney
• date, time, smalldatetime, datetime
• char, varchar
• unichar, univarchar
• binary, varbinary
• bit

The underlying datatype of encrypted data on disk is varbinary. Null values are not encrypted.

During create table, alter table, and select into operations, Adaptive Server calculates the maximum internal length of the encrypted column. Before he or she can make decisions about schema arrangements and page sizes, the database owner must know the maximum length of the encrypted columns. See Chapter 3, “Encrypted Data,” in the User Guide for Encrypted Columns.

You can create an index on an encrypted column if you specify the encryption key without any initialization vector or random padding. Adaptive Server issues an error if you execute create index on an encrypted column with an initialization vector or random padding.

You can define referential integrity constraints on encrypted columns when:

• Both referencing and referenced columns are encrypted.
• The key you use to encrypt the columns specifies init_vector null and you have not specified pad random.
You cannot encrypt a computed column, and an encrypted column cannot appear in the expression defining a computed column. You cannot specify an encrypted column in the partition_clause of create table.

For more information, see Chapter 3, “Encrypted Data,” in the User Guide for Encrypted Columns.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

The following are Transact-SQL extensions:

- Use of a database name to qualify a table or column name
- IDENTITY columns
- The not null column default
- The asc and desc options
- The reservepagegap option
- The lock clause
- The on segment_name clause


Permissions

create table permission defaults to the Database Owner, who can transfer it to other users. Any user can create temporary tables.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 10    | create       | create table              | • Roles – Current active roles  
|       |              |                           | • Keywords or options – NULL  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – NULL  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect  

See also

Commands alter table, create existing table, create index, create rule, create schema, create view, drop index, drop rule, drop table
System procedures  sp_addmessage, sp_addsegment, sp_addtype, sp_bindmsg, sp_chgattribute, sp_commonkey, sp_depends, sp_foreignkey, sp_help, sp_helpjoins, sp_helpsegment, sp_primarykey, sp_rename, sp_spaceused
create trigger

Description
Creates a trigger, a type of stored procedure that is often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.

Syntax
create trigger [owner.]trigger_name
on [owner.]table_name
{for | instead of} {insert , update , delete}
    as SQL_statements

Or, using the if update clause:
create trigger [owner.]trigger_name
on [owner.]table_name
{for {insert , update} | instead of {insert, update, delete}}
    as
        [if update (column_name)
            [(and | or) update (column_name)]...]
            SQL_statements
        [if update (column_name)
            [(and | or) update (column_name)]...]
            SQL_statements

Parameters

trigger_name
is the name of the trigger. It must conform to the rules for identifiers and be unique in the database. Specify the owner’s name to create another trigger of the same name owned by a different user in the current database. The default value for owner is the current user. If you use an owner name to qualify a trigger, you must explicitly qualify the table name the same way.

You cannot use a variable for a trigger name.

table_name
is the name of the table on which to create the trigger. If more than one table of the same name exists in the database, specify the owner’s name. The default value for owner is the current user.

for | instead of
for – used before insert, delete, or update to indicate what you are creating the trigger for.

instead of – creates and fills the inserted and deleted pseudo tables that are used in the trigger to examine the rows that would have been modified by the original insert, delete, or update query.

insert, update, delete
can be included in any combination. delete cannot be used with the if update clause.
**SQL statements** specify trigger conditions and trigger actions. Trigger conditions determine whether the attempted insert, update, or delete causes the trigger actions to be carried out. The SQL statements often include a subquery preceded by the keyword if. In Example 2, below, the subquery that follows the keyword if is the trigger condition.

Trigger actions take effect when the user action (insert, update, or delete) is attempted. If multiple trigger actions are specified, they are grouped with begin and end.

See “Triggers and transactions” on page 213 for a list of statements that are not allowed in a trigger definition. See “The deleted and inserted logical tables” on page 211 for information about the deleted and inserted logical tables that can be included in trigger definitions.

**Examples**

**Example 1** Prints a message when anyone tries to add data or change data in the titles table:

```sql
create trigger reminder
on titles
for insert, update as
print "Don't forget to print a report for accounting."
```

**Example 2** Prevents insertion of a new row into titleauthor if there is no corresponding title_id in the titles table:

```sql
create trigger t1
on titleauthor
for insert as
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) = 0
begin
    print "Please put the book's title_id in the titles table first."
    rollback transaction
end
```

**Example 3** If the pub_id column of the publishers table is changed, make the corresponding change in the titles table:
create trigger t2
on publishers
for update as
if update (pub_id) and @@rowcount = 1
begin
    update titles
    set titles.pub_id = inserted.pub_id
    from titles, deleted, inserted
    where deleted.pub_id = titles.pub_id
end

Example 4 Deletes title from the titles table if any row is deleted from titleauthor. If the book was written by more than one author, other references to it in titleauthor are also deleted:

create trigger t3
on titleauthor
for delete as
begin
    delete titles
    from titles, deleted
    where deleted.title_id = titles.title_id
    delete titleauthor
    from titleauthor, deleted
    where deleted.title_id = titleauthor.title_id
    print "All references to this title have been deleted from titles and titleauthor."
end

Example 5 Prevents updates to the primary key on weekends. Prevents updates to the price or advance of a title unless the total revenue amount for that title surpasses its advance amount:

create trigger stopupdatetrig
on titles
for update
as
if update (title_id)
    and datename (dw, getdate ())
in ("Saturday", "Sunday")
begin
    rollback transaction
    print "We don't allow changes to"
    print "primary keys on the weekend!"
end
if update (price) or update (advance)
if (select count (*) from inserted
    where (inserted.price * inserted.total_sales)
create table EmployeeWest (  
    empid int primary key,
    empname varchar(30),
    empdob datetime,
    region char(5)
    
    constraint region_chk
    check (region='West'))

create table EmployeeEast (  
    empid int primary key,
    empname varchar(30),
    empdob datetime,
    region char(5)
    
    constraint region_chk
    check (region='East'))

create view Employees as  
    select * from EmployeeEast
    union all
    select * from EmployeeWest

create trigger EmployeesInsertTrig on Employees
    instead of insert as
begin

    insert into EmployeeEast select * from inserted where region = "East"

    insert into EmployeeWest select * from inserted where region = "West"
end

--will insert the data into the EmployeeEast table
insert into Employees values (10, 'Jane Doe', '11/11/1967', 'East')

--will insert the data into the EmployeeWest table
insert into Employees values (11, 'John Smith', '01/12/1977', 'West')
Example 7 Uses instead of triggers to implement encrypted column support, storing data in the database in encrypted form without changing applications (the user-defined functions, my_encrypt and my_decrypt, perform the encryption and decryption operations on the data):

```sql
CREATE TABLE Employee_t (id int PRIMARY KEY, name varchar(20),
                          salary binary (64))
--where the id and name columns are stored unencrypted, salary is
--encrypted and id is a primary key.
create view employee_v as select id, name, my_decrypt (salary)
                          from employee_t
CREATE TRIGGER EmployeeInsert
ON employee_v
INSTEAD OF INSERT
AS
BEGIN
    INSERT employee_t SELECT id, name, my_encrypt (salary)
    FROM inserted
END
CREATE TRIGGER employeeUpdate
ON employee_v
INSTEAD OF UPDATE
AS
BEGIN
    DELETE FROM employee_t WHERE id IN (SELECT id FROM deleted)
    INSERT employee_t SELECT id, name, my_encrypt (salary)
    FROM inserted
END
CREATE TRIGGER employeeDelete
ON employee_v
INSTEAD OF DELETE
AS
BEGIN
    DELETE FROM employee_t WHERE id IN (SELECT id FROM deleted)
END
```
Usage

- To avoid seeing unexpected results due to changes in settings, run `set rowcount 0` as your initial statement before executing `create trigger`. The scope of `set` is limited to just your `create trigger` command, and resets to your previous setting once the procedure exits.

- A trigger fires only once per data modification statement. A complex query containing a `while` loop may repeat an update or insert many times, and the trigger is fired each time.

Triggers and referential integrity

- Triggers are commonly used to enforce referential integrity (integrity rules about relationships between the primary and foreign keys of tables or views), to supply cascading deletes, and to supply cascading updates (see Examples 2, 3, and 4, respectively).

- A trigger fires only after the data modification statement has completed and Adaptive Server has checked for any datatype, rule, or integrity constraint violations. The trigger and the statement that fires it are treated as a single transaction that can be rolled back from within the trigger. If a severe error is detected, the entire transaction is rolled back.

- You can also enforce referential integrity using constraints defined with the `create table` statement as an alternative to using `create trigger`. See `create table` and `alter table` for information about integrity constraints.

The deleted and inserted logical tables

- `deleted` and `inserted` are logical (conceptual) tables. They are structurally identical to the table for which the trigger is defined—that is, the table on which the user action is attempted—and hold the old values or new values of the rows that would be changed by the user action.

  **Note** Both inserted and deleted tables appear as views on the transaction log, but they are fake tables on syslogs.

- `deleted` and `inserted` tables can be examined by the trigger to determine whether or how the trigger action should be carried out, but the tables themselves cannot be altered by the trigger’s actions.

- `deleted` tables are used with `delete` and `update`; `inserted` tables, with `insert` and `update`. An update is a delete followed by an insert: it affects the `deleted` table first, and then the `inserted` table.
Trigger restrictions

- You can create a trigger only in the current database. If you use an owner name to qualify a trigger, you must explicitly qualify the table name the same way. A trigger can reference objects outside the current database.

- A trigger cannot apply to more than one table. However, the same trigger action can be defined for more than one user action (for example, insert and update) in the same `create trigger` statement. A table can have a maximum of three triggers—one each for insert, update, and delete.

- Each new trigger in a table or column for the same operation (insert, update, or delete) overwrites the previous one. No warning message displays before the overwrite occurs.

- You cannot create a trigger on a session-specific temporary table.

- You cannot create a trigger on a view.

- You cannot create a trigger on a system table.

- You cannot use triggers that select from a text, uni text, or image column of the inserted or deleted table.

- Sybase recommends that triggers not include select statements that return results to the user, since special handling for these returned results must be written into every application program that allows modifications to the trigger table.

- If a trigger references table names, column names, or view names that are not valid identifiers, you must set `quoted_identifier` on before the `create trigger` command and enclose each such name in double quotes. The `quoted_identifier` option does not need to be on when the trigger fires.

Triggers and performance

- In performance terms, trigger overhead is usually very low. The time involved in running a trigger is spent mostly in referencing other tables, which are either in memory or on the database device.

- The deleted and inserted tables often referenced by triggers are always in memory rather than on the database device, because they are logical tables. The location of other tables referenced by the trigger determines the amount of time the operation takes.
CHAPTER 1 Commands

Setting options within triggers

- You can use the set command inside a trigger. The set option you invoke remains in effect during the execution of the trigger, then reverts to its former setting. In particular, the self_recursion option can be used inside a trigger so that data modifications by the trigger itself can cause the trigger to fire again.

Dropping a trigger

- You must drop and re-create the trigger if you rename any of the objects referenced by the trigger. You can rename a trigger with sp_rename.

- When you drop a table, any triggers associated with it are also dropped.

Actions that do not cause triggers to fire

- A truncate table command is not caught by a delete trigger. Although a truncate table statement is, in effect, like a delete without a where clause (it removes all rows), changes to the data rows are not logged, and so cannot fire a trigger.

  Since permission for the truncate table command defaults to the table owner and is not transferable, only the table owner need worry about inadvertently circumventing a delete trigger with a truncate table statement.

- The writetext command, whether logged or unlogged, does not cause a trigger to fire.

Triggers and transactions

- When a trigger is defined, the action it specifies on the table to which it applies is always implicitly part of a transaction, along with the trigger itself. Triggers are often used to roll back an entire transaction if an error is detected, or they can be used roll back the effects of a specific data modification:

  - When the trigger contains the rollback transaction command, the rollback aborts the entire batch, and any subsequent statements in the batch are not executed.

  - When the trigger contains the rollback trigger, the rollback affects only the data modification that caused the trigger to fire. The rollback trigger command can include a raiserror statement. Subsequent statements in the batch are executed.

- Since triggers execute as part of a transaction, the following statements and system procedures are not allowed in a trigger:
create trigger

- All create commands, including create database, create default, create index, create procedure, create rule, create table, create trigger, and create view
- All drop commands
- alter database and alter table
- truncate table
- grant and revoke
- update statistics
- sp_configure
- load database and load transaction
- disk init, disk refit, disk reinit, disk remirror, disk unmirror
- select into

- If a desired result (such as a summary value) depends on the number of rows affected by a data modification, use @@rowcount to test for multirow data modifications (an insert, delete, or update based on a select statement), and take appropriate actions. Any Transact-SQL statement that does not return rows (such as an if statement) sets @@rowcount to 0, so the test of @@rowcount should occur at the beginning of the trigger.

Inserting and updating triggers

- When an insert or update command executes, Adaptive Server adds rows to both the trigger table and the inserted table at the same time. The rows in the inserted table are always duplicates of one or more rows in the trigger table.

- An update or insert trigger can use the if update command to determine whether the update or insert changed a particular column. if update (column_name) is true for an insert statement whenever the column is assigned a value in the select list or in the values clause. An explicit NULL or a default assigns a value to a column and thus activates the trigger. An implicit NULL, however, does not.

For example, if you create the following table and trigger:

```sql
create table junk
    (aaa int null,
     bbb int not null)
create trigger trigtest on junk
for insert as
if update (aaa)
```
print "aaa updated"
if update (bbb)
  print "bbb updated"

Inserting values into either column or into both columns fires the trigger for both column aaa and column bbb:

```
insert junk (aaa, bbb)
values (1, 2)
aaa updated
bbb updated
```

Inserting an explicit NULL into column aaa also fires the trigger:

```
insert junk
values (NULL, 2)
aaa updated
bbb updated
```

If there was a default for column aaa, the trigger would also fire.

However, with no default for column aaa and no value explicitly inserted, Adaptive Server generates an implicit NULL and the trigger does not fire:

```
insert junk (bbb)
values (2)
bbb updated
```

if update is never true for a delete statement.

Nesting triggers and trigger recursion

- Adaptive Server allows nested triggers by default. To prevent triggers from nesting, use `sp_configure` to set the `allow nested triggers` option to 0 (off), as follows:

```
sp_configure "allow nested triggers", 0
```
- Triggers can be nested to a depth of 16 levels. If a trigger changes a table on which there is another trigger, the second trigger fires and can then call a third trigger, and so forth. If any trigger in the chain sets off an infinite loop, the nesting level is exceeded and the trigger aborts, rolling back the transaction that contains the trigger query.

**Note** Since triggers are put into a transaction, a failure at any level of a set of nested triggers cancels the entire transaction: all data modifications are rolled back. Supply your triggers with messages and other error handling and debugging aids to determine where the failure occurred.

- The global variable `@@nestlevel` contains the nesting level of the current execution. Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. The nesting level is also incremented by one when a cached statement is created. If the maximum of 16 is exceeded, the transaction aborts.

- If a trigger calls a stored procedure that performs actions that would cause the trigger to fire again, the trigger is reactivated only if nested triggers are enabled. Unless there are conditions within the trigger that limit the number of recursions, this causes a nesting-level overflow.

For example, if an update trigger calls a stored procedure that performs an update, the trigger and stored procedure execute once if `allow nested triggers` is off. If `allow nested triggers` is on, and the number of updates is not limited by a condition in the trigger or procedure, the procedure or trigger loop continues until it exceeds the 16-level maximum nesting value.

- By default, a trigger does not call itself in response to a second data modification to the same table within the trigger, regardless of the setting of the `allow nested triggers` configuration parameter. A set option, `self_recursion`, enables a trigger to fire again as a result of a data modification within the trigger. For example, if an update trigger on one column of a table results in an update to another column, the update trigger fires only once when `self_recursion` is disabled, but it can fire up to 16 times if `self_recursion` is set on. The `allow nested triggers` configuration parameter must also be enabled in order for self-recursion to take place.
instead of and for triggers

- You can interleave nesting instead of and for triggers. For example, an update statement on a view with an instead of update trigger causes the trigger to execute. If the trigger contains a SQL statement updating a table with a for trigger defined on it, that trigger fires. The for trigger may contain a SQL statement that updates another view with an instead of trigger that then executes, and so forth.

- instead of and for triggers have different recursive behaviors, however. for triggers support recursion, while instead of triggers do not. If an instead of trigger references the same view on which the trigger was fired, the trigger is not called recursively. Rather, the triggering statement applies directly to the view; in other words, the statement is resolved as modifications against the base tables underlying the view. In this case, the view definition must meet all restrictions for an updatable view. If the view is not updatable, an error is raised.

- For example, if a trigger is defined as an instead of update trigger for a view, the update statement executed against the same view within the instead of trigger does not cause the trigger to execute again. The update exercised by the trigger is processed against the view, as though the view did not have an instead of trigger. The columns changed by the update must be resolved to a single base table.

Restrictions for instead of

- You cannot create instead of triggers on a table, although triggers can reference tables. You can only create instead of triggers on views.

- You can define only one instead of trigger for each insert, update, or delete statement on a view. A view can have one trigger defined for multiple operations, or separate triggers for each operation.

- Each new trigger created on a view for the same operation overwrites the previous trigger.

- A trigger cannot apply to more than one view.

- instead of triggers are not allowed on updatable views using the with check option.

- The create trigger statement must be the first statement in the batch. All other statements that follow in that batch are treated as part of the definition of the create trigger statement.

- Permission to create triggers defaults to the view owner, who cannot transfer it to other users.
• Triggers are database objects, and their names must follow the rules for identifiers.

• You can create a trigger only in the current database, although a trigger can reference objects outside of the current database.

• Enabling and disabling the instead of trigger is not supported.

• If a trigger references table names, column names, or view names that are not valid identifiers, you must set quoted_identifier on before the create trigger command, and enclose each such name in double quotation marks. The quoted_identifier option does not need to be on when the trigger fires; bracketed identifiers work as well.

• Using the set cursor rows command with client cursors, cursors declared through Open Client calls or embedded SQL, may prevent positioned delete and update from firing an instead of trigger. A positioned update statement is a SQL update statement that contains the where current of <cursorname> clause to update only the row upon which the cursor, <cursorname>, is currently positioned.

• Joins are not allowed in searched delete and update statements that would fire an instead of trigger.

• positioned delete and update on cursors defined with joins does not fire an instead of trigger.

A positioned delete is a SQL delete statement containing a where current of <cursorname> clause to delete only the row upon which the cursor, <cursorname>, is currently positioned.

A positioned update is a SQL update statement that contains the where current of <cursorname> clause to update only the row upon which the cursor, <cursorname>, is currently positioned.

• For positioned delete and update statements that fire an instead of trigger, the instead of trigger must exist when the cursor is declared.

Getting information about triggers

• The execution plan for a trigger is stored in sysprocedures.

• Each trigger is assigned an identification number, which is stored as a new row in sysobjects with the object ID for the table to which it applies in the deltrig column, and also as an entry in the deltrig, instrig, and updtrig columns of the sysobjects row for the table to which it applies.

• To display the text of a trigger, which is stored in syscomments, use sp_helptext.
If the System Security Officer has reset the allow select on syscomments.text column parameter with sp_configure (as required to run Adaptive Server in the evaluated configuration), you must be the creator of the trigger or a System Administrator to view the text of the trigger through sp_helptext.

- For a report on a trigger, use sp_help.
- For a report on the tables and views that are referenced by a trigger, use sp_depends.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Only a System Security Officer can grant or revoke permissions to create triggers. The Database Owner has implicit permission to create a trigger on any user table. Users can create triggers only on tables that they own.

The System Security Officer may revoke a user’s permission to create triggers. Revoking permission to create triggers affects only the database in which the SSO issues the revoke command. Permission to run the create trigger command is restored to the user whose permission was revoked when the System Security Officer explicitly grants them create trigger permission.

Permissions on objects at trigger creation  When you create a trigger, Adaptive Server makes no permission checks on objects such as tables or views that the trigger references. Therefore, you can create a trigger successfully, even though you do not have access to its objects. All permission checks occur when the trigger fires.

Permissions on objects at trigger execution  When the trigger executes, permission checks on its objects depend on whether the trigger and its objects are owned by the same user.

- If the trigger and its objects are not owned by the same user, the user who caused the trigger to fire must have been granted direct access to the objects. For example, if the trigger performs a select from a table the user cannot access, the trigger execution fails. In addition, the data modification that caused the trigger to fire is rolled back.
- If a trigger and its objects are owned by the same user, special rules apply. The user automatically has implicit permission to access the trigger’s objects, even though the user cannot access them directly. A detailed description of the rules for implicit permissions is discussed in the System Administration Guide.
Permissions for instead of and for triggers instead of triggers have the same permission requirements as for triggers: to create a view with instead of triggers, permission for insert/update/delete for the view, not the underlying tables, must be granted to the user.

Auditing Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>create</td>
<td>create trigger</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also Commands alter table, create procedure, drop trigger, rollback trigger, set

System procedures sp_commonkey, sp_configure, sp_depends, sp_foreignkey, sp_help, sp_helptext, sp_primarykey, sp_rename, sp_spaceused
create view

Description

Creates a view, which is an alternative way of looking at the data in one or more tables.

Syntax

create view [owner.]view_name
    [(column_name[, column_name]...)]
    as
    select [distinct] select_statement
    [with check option]

Parameters

`view_name`

is the name of the view. The name cannot include the database name. If you have set `quoted_identifier` on, you can use a delimited identifier. Otherwise, the view name cannot be a variable and must conform to the rules for identifiers. For more information about valid view names, see “Identifiers” in Chapter 4, “Expressions, Identifiers, and Wildcard Characters,” of Reference Manual: Building Blocks. Specify the owner’s name to create another view of the same name owned by a different user in the current database. The default value for `owner` is the current user.

`column_name`

specifies names to be used as headings for the columns in the view. If you have set `quoted_identifier` on, you can use a delimited identifier. Otherwise, the column name must conform to the rules for identifiers. For more information about valid column names, see “Identifiers” in Chapter 4, “Expressions, Identifiers, and Wildcard Characters,” of Reference Manual: Building Blocks.

You can always supply column names, but they are required only:

- A column is derived from an arithmetic expression, function, string concatenation, or constant
- Two or more columns have the same name (usually because of a join)
- You want to give a column in a view a different name than the column from which it is derived (see Example 3)

Column names can also be assigned in the `select` statement (see Example 4). If no column names are specified, the view columns acquire the same names as the columns in the `select` statement.

`select`

begins the `select` statement that defines the view.

`distinct`

specifies that the view cannot contain duplicate rows.
The `CREATE VIEW` statement is used to define a new view that is derived from a base table or another view. The view is a virtual table that contains a set of rows resulting from the `SELECT` statement that defines the view. Views can be used to simplify complex queries, provide a named reference to a set of rows, and enforce data consistency through the use of the WITH CHECK OPTION clause.

The `SELECT` statement that defines a view can use more than one table and other views. It can also include FROM clauses that reference one or more views. The WITH CHECK OPTION clause is important as it indicates that all data modification statements are validated against the view selection criteria. All rows inserted or updated through the view must remain visible through the view.

### Examples

**Example 1** Creates a view derived from the `title`, `type`, `price`, and `pubdate` columns of the base table `titles`:

```sql
CREATE VIEW titles_view
AS SELECT title, type, price, pubdate
FROM titles
```

**Example 2** Creates a view that contains the `title`, `advance`, and `amt_due` columns of books with a price less than $5.00:

```sql
CREATE VIEW accounts (title, advance, amt_due)
AS SELECT title, advance, price * total_sales
FROM titles
WHERE price > $5
```

**Example 3** Creates a view with the same definition as in the previous example, but with column headings included in the `SELECT` statement:

```sql
CREATE VIEW cities2
AS SELECT authorname = au_lname, acity = authors.city, publishername = pub_name, pcity = publishers.city
FROM authors, publishers
WHERE authors.city = publishers.city
```

**Example 4** Creates a view derived from two base tables, `authors` and `publishers`. The view contains the names and cities of authors who live in a city in which there is a publisher:

```sql
CREATE VIEW cities
(authorname, acity, publishername, pcity)
AS SELECT au_lname, authors.city, pub_name, publishers.city
FROM authors, publishers
WHERE authors.city = publishers.city
```

**Example 5** Creates a view with the same definition as in the previous example, but with column headings included in the `SELECT` statement:

```sql
CREATE VIEW cities2
AS SELECT authorname = au_lname, acity = authors.city, publishername = pub_name, pcity = publishers.city
FROM authors, publishers
WHERE authors.city = publishers.city
```
CHAPTER 1 Commands

Example 6 Creates a view, author_codes, derived from titleauthor that lists the unique author identification codes:

```sql
create view author_codes
as select distinct au_id
from titleauthor
```

Example 7 Creates a view, price_list, derived from title that lists the unique book prices:

```sql
create view price_list (price)
as select distinct price
from titles
```

Example 8 Creates a view of the stores table that excludes information about stores outside of California. The with check option clause validates each inserted or updated row against the view’s selection criteria. Rows for which state has a value other than “CA” are rejected:

```sql
create view stores_cal
as select * from stores
where state = "CA"
with check option
```

Example 9 Creates a view, stores_cal30, which is derived from stores_cal. The new view inherits the check option from stores_cal. All rows inserted or updated through stores_cal30 must have a state value of “CA”. Because stores_cal30 has no with check option clause, you can insert or update rows through stores_cal30 for which payterms has a value other than “Net 30”:

```sql
create view stores_cal30
as select * from stores_cal
where payterms = "Net 30"
```

Example 10 Creates a view, stores_cal30_check, derived from stores_cal. The new view inherits the check option from stores_cal. It also has a with check option clause of its own. Each row that is inserted or updated through stores_cal30_check is validated against the selection criteria of both stores_cal and stores_cal30_check. Rows with a state value other than “CA” or a payterms value other than “Net 30” are rejected:

```sql
create view stores_cal30_check
as select * from stores_cal
where payterms = "Net 30"
with check option
```

Example 11 Uses a SQL-derived table in creating a view:

```sql
from authors, publishers
where authors.city = publishers.city
```
create view psych_titles as
    select *
    from (select * from titles
           where type = "psychology") dt_psych

Usage

- You can use views as security mechanisms by granting permission on a view, but not on its underlying tables.
- You can rename a view using sp_rename.
- When you query through a view, Adaptive Server checks to make sure that all the database objects referenced anywhere in the statement exist, that they are valid in the context of the statement, and that data update commands do not violate data integrity rules. If any of these checks fail, you see an error message. If the checks are successful, create view “translates” the view into an action on the underlying tables.
- For more information about views, see the Transact-SQL User’s Guide.

Restrictions on views

- You can create a view only in the current database.
- The number of columns referenced by a view cannot exceed 1024.
- You cannot create a view on a temporary table.
- You cannot create a trigger or build an index on a view.
- You cannot use readtext or writetext on text or unitext, image columns in views.
- You cannot include order by, compute clauses or the keyword into in the select statements that define views.
- You cannot update, insert, or delete from a view with select statements that include the union operator.
- If you create a view using a local or a global variable, Adaptive Server issues error message 7351: “Local or global variables not allowed in view definition.”
You can combine create view statements with other SQL statements in a single batch.

**Warning!** When a create view command occurs within an if...else block or a while loop, Adaptive Server creates the schema for the view before determining whether the condition is true. This may lead to errors if the view already exists. To avoid this, verify that a view with the same name does not already exist in the database or use an execute statement, as follows:

```sql
if not exists
    (select * from sysobjects where name="mytable")
begin
    execute "create table mytable (x int)"
end
```

View resolution
- If you alter the structure of a view’s underlying tables by adding or deleting columns, the new columns do not appear in a view defined with a select * clause unless the view is dropped and redefined. The asterisk shorthand is interpreted and expanded when the view is first created.

- If a view depends on a table or view that has been dropped, Adaptive Server produces an error message when anyone tries to use the view. If a new table or view with the same name and schema is created to replace the one that has been dropped, the view again becomes usable.

- You can redefine a view without redefining other views that depend on it, unless the redefinition makes it impossible for Adaptive Server to translate any dependent views.

Modifying data through views
- `delete` statements are not allowed on multitable views.

- `insert` statements are not allowed unless all not null columns in the underlying table or view are included in the view through which you are inserting new rows. Adaptive Server cannot supply values for not null columns in the underlying table or view.

- You cannot insert directly to a computed column through a view. The value of computed columns can only be generated internally by Adaptive Server.
• insert statements are not allowed on join views created with distinct or with check option.

• update statements are allowed on join views with check option. The update fails if any of the affected columns appear in the where clause, in an expression that includes columns from more than one table.

• If you insert or update a row through a join view, all affected columns must belong to the same base table.

• You cannot update or insert into a view defined with the distinct clause.

• Data update statements cannot change any column in a view that is a computation, and cannot change a view that includes aggregates.

IDENTITY columns and views
• You cannot add a new IDENTITY column to a view with the column_name = identity (precision) syntax.

• To insert an explicit value into an IDENTITY column, the table owner, Database Owner, or System Administrator must set identity_insert table_name on for the column’s base table, not through the view through which it is being inserted.

group by clauses and views
• When creating a view for security reasons, be careful when using aggregate functions and the group by clause. A Transact-SQL extension allows you to name columns that do not appear in the group by clause. If you name a column that is not in the group by clause, Adaptive Server returns detailed data rows for the column. For example, this Transact-SQL extended column query returns a row for every 18 rows—more data than you might intend:

```sql
select title_id, type, sum (total_sales)
from titles
group by type
```

While this ANSI-compliant query returns one row for each type (6 rows):

```sql
select type, sum (total_sales)
from titles
group by type
```

For more information about group by, see “group by and having clauses on page 390.”
distinct clauses and views

- The distinct clause defines a view as a database object that contains no duplicate rows. A row is defined to be a duplicate of another row if all of its column values match the same column values in another row. Null values are considered to be duplicates of other null values.

Querying a subset of a view’s columns can result in what appear to be duplicate rows. If you select a subset of columns, some of which contain the same values, the results appear to contain duplicate rows. However, the underlying rows in the view are still unique. Adaptive Server applies the distinct requirement to the view’s definition when it accesses the view for the first time (before it does any projection and selection) so that all the view’s rows are distinct from each other.

You can specify distinct more than once in the view definition’s select statement to eliminate duplicate rows, as part of an aggregate function or a group by clause. For example:

```
select distinct count (distinct title_id), price
from titles
```

- The scope of the distinct applies only for that view; it does not cover any new views derived from the distinct view.

with check option clauses and views

- If a view is created with check option, each row that is inserted or updated through the view must meet the selection criteria of the view.

- If a view is created with check option, all views derived from the “base” view must satisfy its check option. Each row inserted or updated through the derived view must remain visible through the base view.

Getting information about views

- To create a report of the tables or views on which a view depends, and of objects that depend on a view, execute `sp_depends`.

- To display the text of a view, which is stored in `syscomments`, execute `sp_helptext` with the view name as the parameter.

Creating views from SQL derived tables

- To create a view using a SQL-derived table, add the derived table expression in the from clause of the select part of the `create view` statement (see Example 11).
A view created using a SQL-derived table can be updated if the derived table expression can be updated. The update rules for the derived table expression follow the update rules for the `select` part of the `create view` statement.

Data can be inserted through a view that contains a SQL-derived table if the `insert` rules and permission settings for the derived table expression follow the `insert` rules and permission settings for the `select` part of the `create view` statement.

Temporary tables and local variables are not permitted in a derived table expression that is part of a `create view` statement.

SQL-derived tables cannot have unnamed columns.

For more information about derived table expressions, see the `Transact-SQL Guide`.

**Standards**

ANSI SQL – Compliance level: Entry-level compliant.

The use of more than one distinct keyword and the use of “`column_heading = column_name`” in the `select` list are `Transact-SQL` extensions.

**Permissions**

`create view` permission defaults to the Database Owner, who can transfer it to other users.

**Permissions on objects at view creation**  When you create a view, Adaptive Server makes no permission checks on objects, such as tables and views, that are referenced by the view. Therefore, you can create a view successfully even if you do not have access to its objects. All permission checks occur when a user invokes the view.

**Permissions on objects at view execution**  When a view is invoked, permission checks on its objects depend on whether the view and all referenced objects are owned by the same user.

- If the view and its objects are not owned by the same user, the invoker must have been granted direct access to the objects. For example, if the view performs a `select` from a table the invoker cannot access, the `select` statement fails.
• If the view and its objects are owned by the same user, special rules apply. The invoker automatically has implicit permission to access the view’s objects even though the invoker could not access them directly. Without having to grant users direct access to your tables, you can give them restricted access with a view. In this way, a view can be a security mechanism. For example, invokers of the view might be able to access only certain rows and columns of your table. A detailed description of the rules for implicit permissions is discussed in the System Administration Guide.

• If a column in the table is encrypted, you must have decrypt permission to run create view. If the view and its objects are not owned by the same user, you must have decrypt permission on the encrypted column in the table to run create view. If the view and its objects are owned by the same user, it is sufficient to grant user running create view decrypt permission on the view column that corresponds to the encrypted column in the table.

Auditing

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>create</td>
<td>create view</td>
<td>Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also

**Commands** create schema, drop view, select, update

**System procedures** sp_depends, sp_help, sp_helptext, sp_rename
**dbcc**

**Description**

Database consistency checker (dbcc) checks the logical and physical consistency of a database and provides statistics, planning, and repair functionality.

**Syntax**

```sql
dbcc addtempdb (dbid | database_name)
dbcc checkalloc ([database_name[, fix | nofix]])
dbcc checkcatalog ([database_name[, fix]])
dbcc checkdb ([database_name[, skip_ncindex]])
dbcc checkindex ([table_name | table_id], index_id
        [, bottom_up[, partition_name | partition_id]])
dbcc checkstorage ([database_name])
dbcc checktable (table_name | table_id
        [, skip_ncindex | fix_spacebits | "check spacebits" | bottom_up | NULL[, partition_name | partition_id])
dbcc checkverify (dbname[, tblname[, ignore_exclusions]])
dbcc complete_xact (xid, ["commit", "1pc" | "rollback")
dbcc forget_xact (xid)
dbcc dbrepair (database_name, dropdb)
dbcc engine ([offline, [enginenum] | "online")
dbcc fix_text ([table_name | table_id])
dbcc indexalloc (table_name | table_id, index_id
        [, optimized | fast | NULL [, fix | nofix | NULL
        [, partition_name | partition_id]])
```

**dbcc**

```sql
dbcc monitor (increment, <group name>)
dbcc monitor (decrement, <group name>)
dbcc monitor (reset, <group name>)
dbcc pravailabletempdbs
dbcc rebuild_text (table_name | table_id [, all][, column[, text_page
        [, data_partition_name | data_partition_id]])
dbcc reindex ([table_name | table_id])
dbcc serverlimits
dbcc stackused
dbcc tablealloc (table_name | table_id [, full | optimized | fast | NULL
        [, fix | nofix | NULL [, data_partition_name | data_partition_id]])
dbcc textalloc (table_name | table_id [, full | optimized | fast | NULL
        [, fix | nofix | NULL [, data_partition_name | data_partition_id]])
```
dbcc {traceon | traceoff} (flag [, flag ...])
dbcc tune (|ascinserts, {0 | 1}, table_name |
cleanup, {0 | 1} | cpuaffinity, start_cpu [{on | off}| 
des_greedyalloc, dbid, object_name, 
"{on | off}" | deviochar vdevno, "batch_size" |
doneinproc {0 | 1}))
dbcc upgrade_object (|dbid | dbname |
|object_name | object_type| [force | check]))

Parameters

addtempdb
adds a temporary database to the global list of available temporary

databases. If the database does not exist or is not a temporary database, an
error is generated. If the database is already a member of the list, an
informational message prints.

dbid
is the database ID.

database_name
is the name of the database to check. If no database name is given, dbcc uses
the current database.

checkalloc
checks the specified database to see that all pages are correctly allocated and
that no page that is allocated is not used. If no database name is given,
checkalloc checks the current database. It always uses the optimized report
option (see tablealloc).

checkalloc reports on the amount of space allocated and used.

fix | nofix
determines whether dbcc fixes the allocation errors found. The default mode
for checkalloc is nofix. You must put the database into single-user mode to
use the fix option.

For a discussion of page allocation in Adaptive Server, see the System
Administration Guide.
checkcatalog

checks for consistency in and between system tables. For example, checkcatalog makes sure that every type in syscolumns has a matching entry in systypes, that every table and view in sysobjects has at least one column in syscolumns, and that the last checkpoint in syslogs is valid. For a complete list of the checking that checkcatalog performs, see “Checking performed by dbcc checkcatalog” on page 246. You can use checkcatalog in an archive database, but not the fix version of checkcatalog.

checkcatalog also reports on any segments that have been defined. If no database name is given, checkcatalog checks the current database.

fix
determines whether dbcc fixes the sysindexes errors it finds. The default mode for checkcatalog is to not fix the errors. You must put the database into singleuser mode to use the fix option. The new sysindexes checks may result in new errors, not raised by dbcc checkcatalog in a Adaptive Servers earlier than version 12.5.2.

checkdb

runs the same checks as checktable, but on each table, including syslogs, in the specified database. If no database name is given, checkdb checks the current database. You can use checkdb in an archive database.

skip_ncindex

causes dbcc checktable or dbcc checkdb to skip checking the nonclustered indexes on user tables. The default is to check all indexes.

checkindex

runs the same checks as checktable, but only on the specified index. You can use checkindex in an archive database.

bottom_up

DOL tables only – checks indexes in a bottom-up order when specifying this option with checkindex. The bottom_up check involves verifying whether each datarow has a corresponding index row.

checkstorage

checks the specified database for allocation, object allocation map (OAM) page entries, page consistency, text valued columns, allocation of text valued columns, and text column chains. The results of each dbcc checkstorage operation are stored in the dbccdb database. For details on using dbcc checkstorage, and on creating, maintaining, and generating reports from dbccdb, see the System Administration Guide.
checktable
  checks the specified table to see that index and data pages are correctly
  linked, that indexes are in properly sorted order, that all pointers are
  consistent, that the data information on each page is reasonable, and that
  page offsets are reasonable. You can use checktable in an archive database.

  \text{table\_name} | \text{table\_id}
  is the name or object ID of the table to check.

fix_spacebits
  is for tables of type \text{datapages} or \text{datarows} lockscheme, and checks for the
  validity of space bits and fixes any invalid space bits. Space bits are stored
  per page and indicate the room available in a page for new inserts.

check spacebits
  checks space bits for tables of type \text{datapages} or \text{datarows} lockscheme. If
  you specify \text{check spacebits}, \text{dbcc} does not check nonclustered indexes.

bottom_up
  \text{DOL tables only} – checkes indexes in a bottom-up order when specifying
  this option with checktable.

partition_name | partition_id
  is the name or ID of the data partition to check. If you specify a partition,
  \text{dbcc} skips global indices.

checkverify
  verifies the results of the most recent run of \text{dbcc checkstorage} for the
  specified database. For details on using \text{dbcc checkverify}, see the \text{System
  Administration Guide}.

ignore_exclusions
  enables or disables the exclusion list. Value is either 0, the default (enables
  the exclusion list) or 1 (disables the exclusion list)
complete_xact
heuristically completes a transaction by either committing or rolling back its work. Adaptive Server retains information about all heuristically completed transactions in the master.dbo.systransactions table, so that the external transaction coordinator may have some knowledge of how the transaction was completed.

**Warning!** Heuristically completing a transaction in the prepared state can cause inconsistent results for an entire distributed transaction. The System Administrator’s decision to heuristically commit or roll back a transaction may contradict the decision made by the coordinating Adaptive Server or protocol.

xid
is a transaction name from the systransactions.xactname column. You can also determine valid xid values using sp_transactions.

1pc
heuristically completes a transaction that was subject to a one-phase commit protocol optimization—instead of the regular two-phase commit protocol—by the external transaction manager that was coordinating its completion. This option allows the heuristic commit of a transaction that was not in the prepared state.

forget_xact
removes the completion status of a heuristically completed transaction from master.dbo.systransactions. forget_xact can be used when the System Administrator does not want the coordinating service to have knowledge that a transaction was heuristically completed, or when an external coordinator is not available to clear commit status in systransactions.

**Warning!** Do not use dbcc forget_xact in a normal DTP environment, since the external transaction coordinator should be permitted to detect heuristically-completed transactions. X/Open XA-compliant transaction managers and Adaptive Server transaction coordination services automatically clear the commit status in systransactions.

dbrepair (database_name, dropdb)
drops a damaged database. drop database does not work on a damaged database.

No one can use the database being dropped when this dbcc statement is issued (including the user issuing the statement).
CHAPTER 1  Commands

engine
takes Adaptive Server engines offline or brings them online. If enginenum is not specified, dbcc engine (offline) takes the highest-numbered engine offline. For more information, see Chapter 8, “Managing Multiprocessor Servers,” in the System Administration Guide.

fix_text
upgrades text values after an Adaptive Server character set has been changed from any character set to a new multibyte character set.

Changing to a multibyte character set makes the internal management of text data more complicated. Since a text value can be large enough to cover several pages, Adaptive Server must be able to handle characters that span page boundaries. To do so, the server requires additional information on each of the text pages. The System Administrator or table owner must run dbcc fix_text on each table that has text data to calculate the new values needed. For more information, see the System Administration Guide.

indexalloc
checks the specified index to see that all pages are correctly allocated and that no page that is allocated is not used. This is a smaller version of checkalloc, providing the same integrity checks on an individual index. You can use indexalloc in an archive database.

indexalloc produces the same three types of reports as tablealloc: full, optimized, and fast. If no type is indicated, or if you use null, Adaptive Server uses optimized. The fix | nofix option functions the same with indexalloc as with tablealloc.

Note  You can specify fix or nofix only if you include a value for the type of report (full, optimized, fast, or null).

table_name | table_id
is the table’s name or the table’s object ID.

indid
is the ID of the index that is checked during dbcc indexalloc.

fix_spacebits
is for tables of type datapages or datarows lockscheme, and checks for the validity of space bits and fixes any invalid space bits. Space bits are stored per page and indicate the room available in a page for new inserts.
check spacebits
checks space bits for tables of type datapages or datarows lockscheme. If
you specify check spacebits, dbcc does not check nonclustered indexes.

full
reports all types of allocation errors.

optimized
produces a report based on the allocation pages listed in the object allocation
map (OAM) pages for the index. It does not report and cannot fix
unreferenced extents on allocation pages that are not listed in the OAM
pages. The optimized option is the default.

fast
does not produce an allocation report, but produces an exception report of
pages that are referenced but not allocated in the extent (2521-level errors).

fix | nofix
determines whether indexalloc fixes the allocation errors found in the table.
The default is fix for all indexes except indexes on system tables, for which
the default is nofix. To use the fix option with system tables, you must first
put the database in single-user mode.

You can specify fix or nofix only if you include a value for the type of report
(full, optimized, fast, or null).

partition_name | partition_id
if you specify a partition ID, allocation checks are performed on the partition
identified by (indid, partition id).

pravailabletempdbs
prints the global list of available temporary databases.

rebuild_text
rebuilds or creates an internal Adaptive Server 12.x data structure for text,
or unitext, image data. This data structure enables Adaptive Server to
perform random access and asynchronous prefetch during data queries. You
can run rebuild_text on all tables in a database, a single table, or a data
partition.

table_name | table_id | "all"
is the table’s name or the table’s object ID, or all the objects in the database
column
is the ID or name of the column of the text column. dbcc rebuild_text rebuilds
the internal data structure of each text value of this column.
text_page
is the logical page number of the first text page. dbcc rebuild_text rebuilds
the internal data structure of this text page.

data_partition_name | data_partition_id
is name or ID of the data partition. If you specify text_page,
data_partition_name (or data_partition_id) is ignored.

monitor increment, group name
The increment and decrement commands increase and decrease the usage
counts for the monitor counters in the specified group by 1. The reset
command sets the usage count for the monitor counters in the specified
group to zero. This will turn off collection of monitoring data for this group.

group name can be one of the following:

• 'all' – determine usage count for the all group, which comprises most of
  the monitor counters, by selecting the @@monitors_active global
  variable.
• spinlock_s – usage counts for spinlock_s reported by the dbcc resource
  command
• appl – usage counts for appl reported by the dbcc resource command

reindex
checks the integrity of indexes on user tables by running a fast version of
dbcc checktable. It can be used with the table name or the table’s object ID
(the id column from sysobjects). reindex prints a message when it discovers
the first index-related error, then drops and re-creates the suspect indexes.
The System Administrator or table owner must run dbcc reindex after the
Adaptive Server sort order has been changed and indexes have been marked
“suspect” by Adaptive Server.

When dbcc finds corrupt indexes, it drops and re-creates the appropriate
indexes. If the indexes for a table are already correct, or if the table has no
indexes, dbcc reindex does not rebuild the index, but prints an informational
message instead.

dbcc reindex aborts if a table is suspected of containing corrupt data. When
that happens, an error message instructs the user to run dbcc checktable. dbcc
reindex does not allow reindexing of system tables. System indexes are
checked and rebuilt, if necessary, as an automatic part of recovery after
Adaptive Server is restarted following a sort order change.
stackused
reports the maximum amount of stack memory used since the server first started.

serverlimits
display the limits Adaptive Server enforces on various entities. These include the lengths of identifiers and the maximum number of different objects such as number of columns in a table, number of indexes on a table, page sizes, row-overheads, and so on. Use the information to determine the various sizing characteristics of the Adaptive Server process.

tablealloc
checks the specified table or data partition to see that all pages are correctly allocated and that no page that is allocated is not used. This is a smaller version of checkalloc, providing the same integrity checks on an individual table. It can be used with the table name or the table’s object ID (the id column from sysobjects). You can use tablealloc in an archive database. For an example of tablealloc output, see the System Administration Guide.

Three types of reports can be generated with tablealloc: full, optimized, and fast. If no type is indicated, or if you use null, Adaptive Server uses optimized.

textalloc
checks the allocation integrity of text or image pages in a database. You can use dbcc textalloc with an archive database.

full
is equivalent to checkalloc at a table level; it reports all types of allocation errors.

optimized
produces a report based on the allocation pages listed in the object allocation map (OAM) pages for the table. It does not report and cannot fix unreferenced extents on allocation pages that are not listed in the OAM pages. The optimized option is the default.

fast
does not produce an allocation report, but produces an exception report of pages that are referenced but not allocated in the extent (2521-level errors).
fix | nofix

determines whether or not tablealloc fixes the allocation errors found in the
table. The default is fix for all tables except system tables, for which the
default is nofix. To use the fix option with system tables, you must first put
the database in single-user mode.

You can specify fix or nofix only if you include a value for the type of report
(full, optimized, fast, or null).

data_partition_name | data_partition_id

is name or ID of the data partition to check. If you specify a partition, dbcc
tablealloc skips global indexes.

traceon | traceoff

toggles the printing of diagnostics during query optimization. Values 3604
and 3605 toggle, sending trace output to the user session and to the error log,
respectively.

tune

enables or disables tuning flags for special performance situations. You must
reissue dbcc tune each time you restart Adaptive Server. For more
information on the individual options, see Performance and Tuning Guide:
Basics.

upgrade_object

upgrades a compiled object from the text stored in the syscomments table.

check

checks syntax for the specified compiled object in syscomments in the
specified database. Does not raise errors on occurrences of select.

force

forces an upgrade of the object from syscomments even if an upgrade is not
required.

object_name

is the name of the compiled object.

object_type

is one of the following object types that Adaptive Server compiles:
procedure, function, view, trigger, default, rule, condition.

Examples

Example 1 Checks pubs2 for page allocation errors:

```
dbcc checkalloc (pubs2)
```

Example 2 Checks database consistency for pubs2 and places the information
in the dbccdb database:
Example 3  Checks the salesdetail table:

```
dbcc checktable (salesdetail)
```

Checking salesdetail
The total number of pages in partition 1 is 3.
The total number of pages in partition 2 is 1.
The total number of pages in partition 3 is 1.
The total number of pages in partition 4 is 1.
The total number of data pages in this table is 10.
Table has 116 data rows.
DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator (SA)
role.

Example 4  Heuristically aborts the transaction, “distributedxact1:”

```
dbcc complete_xact (distributedxact1, "rollback")
```

Example 5  Upgrades text values for blurbs after a character set change:

```
dbcc fix_text (blurbs)
```

Example 6  Removes information for the transaction “distributedxact1” from
master.dbo.systransactions:

```
dbcc forget_xact (distributedxact1)
```

Example 7  Adaptive Server returns a full report of allocation for the index
with an indid of 2 on the titleauthor table and fixes any allocation errors:

```
dbcc indexalloc ("pubs..titleauthor", 2, full)
```

Example 8  Prints the global list of available temporary databases:

```
dbcc pravailabletempdbs
```

Available temporary databases are:
Dbid: 2
Dbid: 4
Dbid: 5
Dbid: 6
Dbid: 7
DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator (SA)
role.

Example 9  Rebuilds or creates an internal Adaptive Server data structure for
all text and image columns in the blurbs table:

```
dbcc rebuild_text (blurbs)
```
Example 10 Checks part of the titles table that resides on the smallsales partition (which contains all booksales less than 5000)

dbcc checktable (titles, NULL, "smallsales")

Example 11 dbcc reindex has discovered one or more corrupt indexes in the titles table:

dbcc reindex (titles)

One or more indexes are corrupt. They will be rebuilt.

Example 12 Checks the logical and physical consistency of a database and provides statistics, planning, and repair functionality:

dbcc stackused

Example 13 Displays an abridged output showing various forms of limits in your Adaptive Server:

dbcc serverlimits

Limits independent of page size:
=================================

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max engines per server</td>
<td>128</td>
</tr>
<tr>
<td>Max number of logins per server</td>
<td>2147516416</td>
</tr>
<tr>
<td>Max number of users per database</td>
<td>2146484223</td>
</tr>
<tr>
<td>Max number of groups per database</td>
<td>1032193</td>
</tr>
<tr>
<td>Max number of user-defined roles per server</td>
<td>1024</td>
</tr>
<tr>
<td>Max number of user-defined roles per (user)session</td>
<td>127</td>
</tr>
<tr>
<td>Min database page size</td>
<td>2048</td>
</tr>
<tr>
<td>Max database page size</td>
<td>16384</td>
</tr>
</tbody>
</table>

Database page-specific limits

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>APL page header size</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>DOL page header size</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Max reserved page gap</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Max fill factor</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Table, Index related limits

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max number of columns in a table/view</td>
<td>1024</td>
</tr>
<tr>
<td>Max number of indexes on a table</td>
<td>250</td>
</tr>
<tr>
<td>Max number of user-keys in a single index on an unpartitioned table</td>
<td>31</td>
</tr>
<tr>
<td>Max number of user-keys in a single local index on a partitioned table</td>
<td>31</td>
</tr>
</tbody>
</table>
General SQL related

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max size of character literals, sproc parameters</td>
<td>16384</td>
</tr>
<tr>
<td>Max size of local &amp;variables in T-SQL</td>
<td>16384</td>
</tr>
<tr>
<td>Max number of arguments to stored procedures</td>
<td>2048</td>
</tr>
<tr>
<td>Max number of arguments to dynamic SQL</td>
<td>2048</td>
</tr>
<tr>
<td>Max number of aggregates in a COMPUTE clause</td>
<td>254</td>
</tr>
</tbody>
</table>

Maximum lengths of different Identifiers

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Max length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max length of server name</td>
<td>30</td>
</tr>
<tr>
<td>Max length of host name</td>
<td>30</td>
</tr>
<tr>
<td>Max length of login name</td>
<td>30</td>
</tr>
<tr>
<td>Max length of user name</td>
<td>30</td>
</tr>
</tbody>
</table>

Limits as a function of the page size:

<table>
<thead>
<tr>
<th>Page Size</th>
<th>2048</th>
<th>4096</th>
<th>8192</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-wide, Database-specific limits and sizes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Min number of virtual pages in master device | 11780 22532 45060 90116 |
| Default number of virtual pages in master device | 23556 45060 90116 180228 |
| Min number of logical pages in master device | 11776 11264 11264 11264 |
| Min number of logical pages in tempdb | 2048 1536 1536 1536 |

Table-specific row-size limits

| Physical Max size of an APL data row, incl row-overheads | 1962 4010 8106 16298 |
| Physical Max size of a DOL data row, incl row-overheads | 1964 4012 8108 16300 |

| Max user-visible size of an APL data row | 1960 4008 8104 16296 |
| Max user-visible size of a DOL data row | 1958 4006 8102 16294 |
| Max user-visible size of a fixed-length column in an APL table | 1960 4008 8104 16296 |
| Max user-visible size of a fixed-length column in a DOL table | 1958 4006 8102 16294 |
Note To show a complete listing of limits in the server, execute dbcc traceon (3604) to get the output to the client session.

Example 14 Adaptive Server returns an optimized report of allocation for this table, but does not fix any allocation errors:

   dbcc tablealloc (publishers, null, nofix)

Example 15 Performs allocation checks on the smallsales partition. All the local indexes on smallsales are included in the check, while the global indexes are excluded:

   dbcc tablealloc (titles, null, null, smallsales)

Example 16 Uses sp_transactions to determine the name of a one-phase commit transaction that did not heuristically commit because it was not in a "prepared" state. The example then explains how to use the 1pc parameter to successfully commit the transaction:

   sp_transactions
   xactkey type coordinator starttime
   state connection dbid spid loid failover srvnname namelen
   xactname
   ------------------------- ------ ------- ---------------
   state connection dbid spid loid failover srvnname namelen
   xactname
   ------------------------- ------ ------- ---------------

   0xbc0500000b00000030c316480100 External XA Feb 2 2004 1:07PM
   Done-Detached Detached  1  0 2099 Resident Tx NULL  88
   28_u7dAc31Wc380000000000000000000000000000001HfpfSxkDM0000FU_00003M00
   00Y_:SYBBEV0A_LRM
   (1 row affected)

   (return status = 0)

   If you try to commit this transaction, Adaptive Server issues an error message:

   dbcc complete_xact
   (*28_u7dAc31Wc380000000000000000000000000000001HfpfSxkDM0000FU_00003M00
   00Y_:SYBBEV0A_LRM", "commit")

   The error message Adaptive Server issues:

   Msg 3947, Level 16, State 1:
   Server 'PISSARRO_1251_P', Line 1:
   A heuristic completion related operation failed. Please 
   see errorlog for more details.
   DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator (SA) role.

Because the transaction is in a “done” state, you can use a one-phase commit protocol optimization to heuristically complete the transaction after verifying the transaction was committed. You can commit this transaction using the `dbcc complete_xact` parameter:

```
DBCC execution completed. If DBCC printed error messages, contact a user with System Administrator (SA) role.
```

You can remove the transaction from systranactions with the `dbcc forget_xact` command:

```
DBCC execution completed. If DBCC printed error messages, contact a user with System Administrator (SA) role.
```

If you run `sp_transactions` again, the previous transaction does not appear:

```
Usage

- `dbcc checkstorage` reports a soft fault if any data page that is not the first data page is empty for non-hashed tables. However, `dbcc checkstorage` does not report this soft fault for the hashed region of a virtually-hashed table. Any data page in the hashed region of a virtually-hashed table can be empty.

- You can run `dbcc` while the database is active, except for the `dbrepair (database_name, dropdb)` option and `dbcc checkalloc` with the `fix` option.

- `dbcc` locks database objects as it checks them. For information on minimizing performance problems while using `dbcc`, see the `dbcc` discussion in the `System Administration Guide`.

- When `dbcc` commands are executing, others cannot access an archive database. If you attempt to access an archive database while `dbcc` commands are being performed, you receive a message saying that the database is in single-user mode.
You can use variants of the `dbcc` commands on an archive database that is online or offline. However, you can use a `dbcc` command with a `fix` option only on an archive database that is online.

To qualify a table or an index name with a user name or database name, enclose the qualified name in single or double quotation marks. For example:

```
dbcc tablealloc ("pubs2.pogo.testtable")
```

You cannot run `dbcc reindex` within a user-defined transaction.

`dbcc fix_text` can generate a large number of log records, which may fill up the transaction log. `dbcc fix_text` is designed so that updates are performed in a series of small transactions: in case of a log space failure, only a small amount of work is lost. If you run out of log space, clear your log and restart `dbcc fix_text` using the same table that was being upgraded when the original `dbcc fix_text` failed.

If you are using a replicated database, use `dbcc dbrepair` to load a dump from a previous release of Adaptive Server to a more current version. For example:

- Loading a dump from a production system of an earlier release of Adaptive Server into a test system of the current release Adaptive Server, or
- In a warm standby application, initializing a standby database of the current release of Adaptive Server with a database dump from an active database of an earlier release of Adaptive Server.

If you attempt to use `select`, `readtext`, or `writetext` on text values after changing to a multibyte character set, and you have not run `dbcc fix_text`, the command fails, and an error message instructs you to run `dbcc fix_text` on the table. However, you can delete text rows after changing character sets without running `dbcc fix_text`.

`dbcc` output is sent as messages or errors, rather than as result rows. Client programs and scripts should check the appropriate error handlers.

If a table is partitioned, `dbcc checktable` returns information about each partition.

Text and image data that has been upgraded to Adaptive Server version 12.x is not automatically upgraded to the new storage format. To improve query performance and enable prefetch for this data, use the `rebuild_text` keyword against the upgraded text and image columns.
The amount of stack memory used in the past is only an indication of possible needs in the future. Adaptive Server may require more stack memory than it used in the past. Run \texttt{dbcc stackused} periodically to find your current stack memory usage.

\texttt{dbcc upgrade\_object check} is used to detect \texttt{syscomments} text corruption caused by Adaptive Server defects that occurred before the upgrade. This \texttt{syscomments} text corruption is serious because it causes upgrade to fail.

If any error is reported by \texttt{dbcc upgrade\_object check}, you must drop and re-create the \texttt{compiled\_object}.

**Additions to \texttt{dbcc complete\_xact}**

\texttt{dbcc complete\_xact} enables a System Administrator to commit or roll back a distributed transaction in circumstances where the external transaction coordinator cannot. In versions of Adaptive Server earlier than 15.0, a transaction could not heuristically committed unless it was in the “prepare” state, and the transaction coordinator used a two-phase commit protocol to commit the transaction. However, in some cases, a transaction coordinator may want to use a one-phase commit protocol as an optimization to commit the transaction.

1pc heuristically completes a transaction that was subject to a one-phase commit protocol optimization—instead of the regular two-phase commit protocol—by the external transaction manager that was coordinating its completion. Heuristically committing such a transaction requires that the transaction be in a “done” state (as reported by \texttt{sp\_transactions}).

**Note** Before heuristically completing the transaction, the System Administrator should make every effort to determine whether the coordinating transaction manager committed or rolled back the distributed transaction.

**Checking performed by \texttt{dbcc checkcatalog}**

\texttt{dbcc checkcatalog} checks that:

- For each row in \texttt{sysindexes} that maps to a range, hash, or list partitioned table, there exists one row in \texttt{sysobjects} where \texttt{sysindexes.conditionid equals sysobjects.id}. \texttt{dbcc checkcatalog} also performs this check for each row in \texttt{sysindexes} that maps to a round-robin partitioned table that has a partition condition.
• For each row in sysindexes that maps to a range, hash, or list partitioned table, there exists one or more rows in sysprocedures where sysindexes.conditionid equals sysprocedures.id. dbcc checkcatalog also performs this check for each row in sysindexes that maps to a round-robin partitioned table that has a partition condition.

• For each row in sysindexes that maps to a range, hash, or list partitioned table, there exists one row in syspartitionkeys where the following conditions are true: sysindexes.id equals syspartitionkeys.id and sysindexes.indid equals syspartitionkeys.indid. dbcc checkcatalog also performs this check for each row in sysindexes that maps to a round-robin partitioned table that has a partition condition.

• For each row in sysindexes, there exists one or more rows in syspartitionkeys where both of the following conditions are true: sysindexes.id equals syspartitionkeys.id and sysindexes.indid equals syspartitionkeys.indid.

• For each row in sysobjects where type is N, there exists one row in sysindexes where sysindexes.conditionid equals sysobjects.id.

• For each row in syspartitionkeys, there exists a row in sysindexes where the following conditions are true: syspartitionkeys.id equals sysindexes.id and syspartitionkeys.indid equals sysindexes.indid.

• For each row in syspartitions, there exists one row in sysindexes where the following condition is true: syspartitions.segments equals sysindexes.segment.

• For each row in systabstats, there exists a row in syspartitions where the following conditions are true: systabstats.id equals syspartitions.id, systabstats.indid equals syspartitions.partitionid, and systabstats.partitionid equals syspartitions.segments.

Text indices (indid=255) do not have entries in systabstats.

• For each row in sysstatistics, there exists a row in sysobjects where the following condition is true: sysstatistics.id equals sysobjects.id.

• For each encryption key row in sysobjects, Adaptive Server checks sysencryptkeys for a row defining that key.

• For each column in syscolumns marked for encryption, Adaptive Server verifies that a key in sysobjects and sysencryptkeys exists.
The corresponding base key is present in sysencryptkeys for every key copy in sysencryptkeys. If the corresponding base key is not present, Adaptive Server issues an error.

For every key copy, the corresponding uid is present in sysusers. If the corresponding uid is not present, Adaptive Server issues an error.

For every decrypt default defined on a column, that the corresponding decrypt default is present in sysobjects and sysattributes. If the corresponding decrypt default is not present, Adaptive Server issues an error.

Using dbcc checktable

If the log segment is on its own device, running dbcc checktable on the syslogs table reports the logs used and free space. For example:

Checking syslogs
The total number of data pages in this table is 1.
*** NOTICE: Space used on the log segment is 0.20 Mbytes, 0.13%.
*** NOTICE: Space free on the log segment is 153.4 Mbytes, 99.87%.
DBCC execution completed. If dbcc printed error messages, see your System Administrator.

If the log segment is not on its own device, the following message appears:

*** NOTICE: Notification of log space used/free cannot be reported because the log segment is not on its own device.

In addition to the regular checks it performs, checktable verifies that the preallocation performed during table creation is correct:

- The number of pages preallocated matches the total number of data pages that must be allocated for the specified max hash key value.
- The data pages are not preallocated in an extent where the preallocation scheme specifies that only OAM pages are allowed.
- The OAM pages are allocated only in the first extent of an allocation unit (AU).

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
**Table owner**  
Only the table owner can execute dbcc with checktable, fix_text, rebuild_text, or reindex.

**Database Owner**  
Only the Database Owner can use checkstorage, checkdb, checkcatalog, checkalloc, indexalloc, and tablealloc.

**System Administrator**  
Only a System Administrator can use dbrepair, complete_xact, engine, and forget_xact, stackused, traceoff, and traceon.
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>dbcc</td>
<td>dbcc</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – Any of the <code>dbcc</code> keywords such as <code>checkstorage</code> and the options for that keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also

**Commands** drop database

**System procedures** `sp_configure`, `sp_helpdb`
deallocate cursor

Description
Makes a cursor inaccessible and releases all memory resources committed to that cursor.

Syntax
deallocate [cursor] cursor_name

description
cursor_name
is the name of the cursor to deallocate.

Parameters

description

Examples

Example 1 Deallocates the cursor named “authors_crsr”:

deallocate cursor authors_crsr

Example 2 Also deallocates the cursor named “authors_crsr”, but omits cursor from the syntax:

deallocate authors_crsr

Usage

• You can use deallocate cursor with an archive database.

• Adaptive Server returns an error message if the cursor does not exist.

• You must deallocate a cursor before you can use its cursor name as part of another declare cursor statement.

• deallocate cursor has no effect on memory resource usage when specified in a stored procedure or trigger.

• You can deallocate a cursor whether it is open or closed.

Permissions
deallocate cursor permission defaults to all users. No permission is required to use it.

See also

Commands close, declare cursor
**declare**

**Description**
Declares the name and type of local variables for a batch or procedure.

**Syntax**
Variable declaration:

```
declare @variable_name datatype [, @variable_name datatype]...
```

Variable assignment:

```
select @variable = {expression | select_statement} [, @variable = {expression | select_statement} ...]
[from table_list]
[where search_conditions]
[group by group_by_list]
[having search_conditions]
[order by order_by_list]
[compute function_list [by by_list]]
```

**Parameters**
 @variable_name must begin with @ and must conform to the rules for identifiers.

datatype can be either a system datatype or a user-defined datatype.

**Examples**

**Example 1** Declares two variables and prints strings according to the values in the variables:

```
declare @one varchar (18), @two varchar (18)
select @one = "this is one", @two = "this is two"
if @one = "this is one"
    print "you got one"
if @two = "this is two"
    print "you got two"
else print "nope"
```

```
you got one
you got two
```

**Example 2** Prints “Ouch!” if the maximum book price in the titles table is more than $20.00:

```
declare @veryhigh money
select @veryhigh = max (price)
    from titles
if @veryhigh > $20
    print "Ouch!"
```

**Usage**

- Assign values to local variables with a select statement.
• The maximum number of parameters in a procedure is 2048. The number of local or global variables is limited only by available memory. The @ sign denotes a variable name.

• Local variables are often used as counters for while loops or if...else blocks. In stored procedures, they are declared for automatic, noninteractive use by the procedure when it executes. Local variables must be used in the batch or procedure in which they are declared.

• The select statement that assigns a value to the local variable usually returns a single value. If there is more than one value to return, the variable is assigned the last one. The select statement that assigns values to variables cannot be used to retrieve data in the same statement.

• The print and raiserror commands can take local variables as arguments.

• Users cannot create global variables and cannot update the value of global variables directly in a select statement.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions declare permission defaults to all users. No permission is required to use it.

See also Commands print, raiserror, select, while
**declare cursor**

**Description**

Defines a cursor, by associating a select statement with a cursor name. You can use `declare cursor` with an archive database.

**Syntax**

```
declare cursor_name
    [semi_sensitive | insensitive] [scroll | no scroll]
    cursor for select_statement
    [for {read only | update [of column_name_list]}]
```

**Parameters**

- `cursor_name`
  - is the name of the cursor being defined.

- `select_statement`
  - is the query that defines the cursor result set. See `select` for more information.

- `semi_sensitive`
  - specifies the data changes made independently of the cursor, and may be visible to the cursor result set. The visibility of the dependent data changes depends on the query plan chosen by the optimizer. If there is no worktable created in the plan, the data changes are visible to the result set. The default is `semi_sensitive`. You cannot update an insensitive cursor.

- `insensitive`
  - specifies the data changes made independently of the cursor are not visible to the cursor result set. If you do not specify this argument, the default is `semi_sensitive`. You cannot update an insensitive cursor.

- `scroll | no scroll`
  - specifies whether the declared cursor is scrollable. Scrollable cursors allowing you fetch the cursor result set non-sequentially, allowing you to scan the cursor back and forth. You cannot update an scrollable cursor.

- `for read only`
  - specifies that the cursor result set cannot be updated.

- `for update`
  - specifies that the cursor result set is updatable.

- `of column_name_list`
  - is the list of columns from the cursor result set (specified by the `select_statement`) defined as updatable. Adaptive Server also allows you to include columns that are not specified in the list of columns of the cursor’s `select_statement` (and excluded from the result set), but that are part of the tables specified in the `select_statement`. 
Examples

**Example 1** Defines a result set for the authors_crsr cursor that contains all authors from the authors table who do not reside in California:

```sql
declare authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state != 'CA'
```

**Example 2** Defines a read-only result set for the titles_crsr cursor that contains the business-type books from the titles table:

```sql
declare titles_crsr cursor
for select title, title_id from titles
where title_id like 'BU%'
for read only
```

**Example 3** Defines an updatable result set for the pubs_crsr cursor that contains all of the rows from the publishers table. It defines the address of each publisher (city and state columns) for update:

```sql
declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state
```

**Example 4** Defines an insensitive scrollable result set for the stores_scrollcrsr that contains the book stores in California:

```sql
declare stores_scrollcrsr insensitive scroll cursor
for select stor_id, stor_name
from stores where state = 'CA'
```

**Example 5** Defines an insensitive non-scrollable result set for the stores_scrollcrsr that contains the book stores in California:

```sql
declare stores_scrollcrsr insensitive no scroll cursor
for select stor_id, stor_name
from stores where state = 'CA'
```

Usage

Restrictions on cursors

- A declare cursor statement must precede any open statement for that cursor.
- You cannot include other statements with declare cursor in the same Transact-SQL batch.
- You can include up to 1024 columns in an update clause of a client’s declare cursor statement.
cursor_name must be a valid Adaptive Server identifier containing no more than 30 characters.

- You cannot update a scrollable cursor.
- You cannot update an insensitive cursor.

Cursor select statements

- select_statement can use the full syntax and semantics of a Transact-SQL select statement, with these restrictions:
  - Must contain a from clause
  - Cannot contain a compute, for browse, or into clause
  - Can contain the holdlock keyword

- The select_statement can contain references to Transact-SQL parameter names or Transact-SQL local variables (for all cursor types except language). The names must reference the Transact-SQL parameters and local variables defined in the procedure, trigger, or statement batch that contains the declare cursor statement.

  The parameters and local variables referenced in the declare cursor statement do not have to contain valid values until the cursor is opened.

- The select_statement can contain references to the inserted and deleted temporary tables that are used in triggers.

Cursor scope

- A cursor’s existence depends on its scope. The scope refers to the context in which the cursor is used, that is, within a user session, within a stored procedure, or within a trigger.

  Within a user session, the cursor exists only until the user ends the session. The cursor does not exist for any additional sessions started by other users. After the user logs off, Adaptive Server deallocates the cursors created in that session.

  If a declare cursor statement is part of a stored procedure or trigger, the cursor created within it applies to stored procedure or trigger scope and to the scope that launched the stored procedure or trigger. Cursors declared inside a trigger on an inserted or a deleted table are not accessible to any nested stored procedures or triggers. However, cursors declared inside a trigger on an inserted or a deleted table are accessible within the scope of the trigger. Once the stored procedure or trigger completes, Adaptive Server deallocates the cursors created within it.
A cursor name must be unique within a given scope. Adaptive Server detects name conflicts within a particular scope only during runtime. A stored procedure or trigger can define two cursors with the same name if only one is executed. For example, the following stored procedure works because only one `names_crsr` cursor is defined in its scope:

```sql
create procedure proc2 @flag int
as
if @flag > 0
    declare names_crsr cursor
    for select au_fname from authors
else
    declare names_crsr cursor
    for select au_lname from authors
return
```

---

**Figure 1-3: How cursors operate within scopes**

1 – User Session
- declare cursor c1
  - go
  - exec sp_proc1

2 – Within proc1
- Can access cursor c1 from User Session
- declare cursor c2
  - go
  - update command fires trigger1

3 – Within trigger1
- Can access cursor c2 from proc1 and cursor c1 from User Session
- declare cursor c3
  - go
  - exit trigger1 back to User Session (Scope 1)

4 – User Session
- Can access cursor c1, but not cursor c2 or c3, since they no longer exist
Result set

- Cursor result set rows may not reflect the values in the actual base table rows. For example, a cursor declared with an order by clause usually requires the creation of an internal table to order the rows for the cursor result set. Adaptive Server does not lock the rows in the base table that correspond to the rows in the internal table, which permits other clients to update these base table rows. In that case, the rows returned to the client from the cursor result set would not be in sync with the base table rows.

- A cursor result set is generated as the rows are returned through a fetch of that cursor. This means that a cursor select query is processed like a normal select query. This process, known as a cursor scan, provides a faster turnaround time and eliminates the need to read rows that are not required by the application.

A restriction of cursor scans is that they can only use the unique indexes of a table. However, if none of the base tables referenced by the cursor result set are updated by another process in the same lock space as the cursor, the restriction is unnecessary. Adaptive Server allows the declaration of cursors on tables without unique indexes, but any attempt to update those tables in the same lock space closes all cursors on the tables.

Updatable cursors

- After defining a cursor using declare cursor, Adaptive Server determines whether the cursor is updatable or read-only. If:
  - A cursor is updatable – you can update or delete rows through the cursor; that is, using cursor_name to do a position update or delete.
  - A cursor is read-only – you cannot use cursor_name to do a position update or delete.

- Use the for update or for read only clause to explicitly define a cursor as updatable or read-only. You cannot define an updatable cursor if its select_statement contains one of the following constructs:
  - distinct option
  - group by clause
  - Aggregate function
  - Subquery
  - union operator
  - at isolation read uncommitted clause
If you do not specify either the for update or the read only clause, Adaptive Server checks to see whether the cursor is updatable.

Adaptive Server also defines a cursor as read-only if you declare a language- or server-type cursor that includes an order by clause as part of its select statement. Adaptive Server handles updates differently for client- and execute-type cursors, thereby eliminating this restriction.

**Updatable cursors and allpages locking**

- If you do not specify a column_name_list with the for update clause, all the specified columns in the query are updatable. Adaptive Server attempts to use unique indexes for updatable cursors when scanning the base table. For cursors, Adaptive Server considers an index containing an IDENTITY column to be unique, even if it is not so declared.

- If you do not specify the for update clause, Adaptive Server chooses any unique index, although it can also use other indexes or table scans if no unique index exists for the specified table columns. However, when you specify the for update clause, Adaptive Server must use a unique index defined for one or more of the columns to scan the base table. If none exists, it returns an error.

- In most cases, include only columns to be updated in the column_name_list of the for update clause. If the table has only one unique index, you do not need to include its column in the for update column_name_list. Adaptive Server will find it when it performs the cursor scan. If the table has more than one unique index, do not include any of them in the for update column_name_list.

This allows Adaptive Server to use that unique index for its cursor scan, which helps prevent an update anomaly called the Halloween problem. Another way to prevent the Halloween problem is to create tables with the unique auto_identity index database option. For more information, see the System Administration Guide.

The Halloween problem occurs when a client updates a column of a cursor result set row that defines the order in which the rows are returned from the base tables. For example, if Adaptive Server accesses a base table using an index, and the index key is updated by the client, the updated index row can move within the index and be read again by the cursor. This is a result of an updatable cursor only logically creating a cursor result set. The cursor result set is actually the base tables that derive the cursor.

If you specify the read only option, you cannot update the cursor result set using the cursor name to perform update or delete.
Using scrollable cursors

- If you do specify insensitive or semi_sensitive when you declare cursor, the default sensitivity is implicit, so that the cursor’s sensitivity depends on the query plan chosen by the optimizer. If the query plan has any worktable created, the cursor becomes insensitive.

- If you specify the cursor’s sensitivity to be semi-sensitive, sensitivity also depends on the query plan.

- If you specify insensitive, the cursor is read_only. You cannot use a for update clause in a cursor declaration.

- If you do not specify the cursor’s scrollability, no scroll is implied.

- All scrollable cursors are read-only. You cannot use a for update clause in a cursor declaration.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

Permissions

declare cursor permission defaults to all users. No permission is required to use it.

See also

Commands open
delete

Description
Removes rows from a table.

Syntax
```
delete
database, owner, table_name
from (after delete)

Parameters

(view_name | table_name)
is the name of the view or table from which to remove rows. Specify the
database name if the view or table is in another database, and specify the
owner’s name if more than one view or table of that name exists in the
database. The default value for owner is the current user, and the default
value for database is the current database.

where
is a standard where clause. See where clause for more information.

from (after table_name or view_name)
lets you name more than one table or view to use with a where clause when
specifying which rows to delete. This from clause allows you to delete rows
from one table based on data stored in other tables, giving you much of the
power of an embedded select statement.

top unsigned_integer
is used to limit the number of rows to the number of rows specified by the
integer.
CHAPTER 1    Commands

readpast
specifies that the delete command skip all pages or rows on which incompatible locks are held, without waiting for locks or timing out. For datapages-locked tables, readpast skips all rows on pages on which incompatible locks are held; for datarows-locked tables, it skips all rows on which incompatible locks are held.

index index_name
specifies an index to use for accessing table_name. You cannot use this option when you delete from a view.

prefetch size
specifies the I/O size, in kilobytes, for tables that are bound to caches with large I/Os configured. You cannot use this option when you delete from a view. sp_helpcache shows the valid sizes for the cache an object is bound to or for the default cache.

When using prefetch and designating the prefetch size (size), the minimum is 2K and any power of two on the logical page size up to 16K. prefetch size options in kilobytes are:

<table>
<thead>
<tr>
<th>Logical page size</th>
<th>Prefetch size options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2, 4, 8, 16</td>
</tr>
<tr>
<td>4</td>
<td>4, 8, 16, 32</td>
</tr>
<tr>
<td>8</td>
<td>8, 16, 32, 64</td>
</tr>
<tr>
<td>16</td>
<td>16, 32, 64, 128</td>
</tr>
</tbody>
</table>

The prefetch size specified in the query is only a suggestion. To allow the size specification, configure the data cache at that size. If you do not configure the data cache to a specific size, the default prefetch size is used.

To configure the data cache size, use sp_cacheconfigure.

Note If Component Integration Services is enabled, you cannot use the prefetch keyword for remote servers.

lru | mru
specifies the buffer replacement strategy to use for the table. Use lru to force the optimizer to read the table into the cache on the MRU/LRU (most recently used/least recently used) chain. Use mru to discard the buffer from cache, and replace it with the next buffer for the table. You cannot use this option when you delete from a view.
plan "abstract plan"
  specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. See Chapter 16, “Creating and Using Abstract Plans,” in the Performance and Tuning Guide: Optimizer and Abstract Plans for more information.

where current of cursor_name
  causes Adaptive Server to delete the row of the table or view indicated by the current cursor position for cursor_name.

Examples

Example 1 Deletes all rows from the authors table:

```sql
delete authors
```

Example 2 Deletes a row or rows from the authors table:

```sql
delete from authors
where au_lname = 'McBadden'
```

Example 3 Deletes rows for books written by Bennet from the titles table.

```sql
delete titles
from titles, authors, titleauthor
where authors.au_lname = 'Bennet'
  and authors.au_id = titleauthor.au_id
  and titleauthor.title_id = titles.title_id
```

The pubs2 database includes a trigger (deltitle) that prevents the deletion of the titles recorded in the sales table; drop this trigger for this example to work.

Example 4 Deletes a row from the titles table currently indicated by the cursor title_crsr:

```sql
delete titles where current of title_crsr
```

Example 5 Determines which row has a value of 4 for the IDENTITY column and deletes it from the authors table. Note the use of the syb_identity keyword instead of the actual name of the IDENTITY column:

```sql
delete authors
where syb_identity = 4
```

Example 6 Deletes rows from authors, skipping any locked rows:

```sql
delete from authors from authors readpast
where state = 'CA'
```

Example 7 Deletes rows from stores, skipping any locked rows. If any rows in authors are locked, the query blocks on these rows, waiting for the locks to be released:

```sql
delete stores from stores readpast, authors
```
where stores.city = authors.city

Usage

- `delete` removes rows from the specified table.
- You can refer to as many as 15 tables in a `delete` statement.
- In pre-12.5.2 versions of Adaptive Server, queries that used `update` and `delete` on views with a union all clause were sometimes resolved without using worktables, which occasionally lead to incorrect results. In Adaptive Server 12.5.2 and later, queries that use `update` and `delete` on views with a union all clause are always resolved using worktables in tempdb.

Restrictions

- You cannot use `delete` with a multitable view (one whose `from` clause names more than one table), even though you may be able to use `update` or `insert` on that same view. Deleting a row through a multitable view changes multiple tables, which is not permitted. `insert` and `update` statements that affect only one base table of the view are permitted.
- Adaptive Server treats two different designations for the same table in a `delete` as two tables. For example, the following `delete` issued in pubs2 specifies discounts as two tables (discounts and pubs2..discounts):

  ```sql
  delete discounts
  from pubs2..discounts, pubs2..stores
  where pubs2..discounts.stor_id = pubs2..stores.stor_id
  ```

  In this case, the join does not include discounts, so the `where` condition remains true for every row; Adaptive Server deletes all rows in discounts (which is not the desired result). To avoid this problem, use the same designation for a table throughout the statement.

- If you are deleting a row from a table that is referenced from other tables via referential constraints, Adaptive Server checks all the referencing tables before permitting the delete. If the row you are attempting to delete contains a primary key that is being used as a foreign key by one of the referencing tables, the delete is not allowed.

Deleting all rows from a table

- If you do not use a `where` clause, all rows in the table named after `delete` [from] are removed. The table, though empty of data, continues to exist until you issue a `drop table` command.
truncate table and delete without a row specification are functionally equivalent, but truncate table is faster. delete removes rows one at a time and logs these transactions. truncate table removes whole data pages, and the rows are not logged.

Both delete and truncate table reclaim the space occupied by the data and its associated indexes.

You cannot use the truncate table command on a partitioned table. To remove all rows from a partitioned table, either use the delete command without a where clause, or unpartition the table before issuing the truncate table command.

delete and transactions

In chained transaction mode, each delete statement implicitly begins a new transaction if no transaction is currently active. Use commit to complete any deletes, or use rollback to undo the changes. For example:

```sql
delete from sales where date < '01/01/89'
if exists (select stor_id
         from stores
         where stor_id not in
         (select stor_id from sales))
   rollback transaction
else
   commit transaction
```

This batch begins a transaction (using the chained transaction mode) and deletes rows with dates earlier than Jan. 1, 1989 from the sales table. If it deletes all sales entries associated with a store, it rolls back all the changes to sales and ends the transaction. Otherwise, it commits the deletions and ends the transaction. For more information about the chained mode, see the Transact-SQL User’s Guide.

delete triggers

You can define a trigger to take a specified action when a delete command is issued on a specified table.
Using `delete where current of`

- Use the clause `where current of` with cursors. Before deleting rows using the clause `where current of`, you must first define the cursor with `declare cursor` and open it using the `open` statement. Position the cursor on the row to delete using one or more `fetch` statements. The cursor name cannot be a Transact-SQL parameter or local variable. The cursor must be an updatable cursor or Adaptive Server returns an error. Any deletion to the cursor result set also affects the base table row from which the cursor row is derived. You can delete only one row at a time using the cursor.

- You cannot delete rows in a cursor result set if the cursor’s `select` statement contains a join clause, even though the cursor is considered updatable. The `table_name` or `view_name` specified with a `delete...where current of` must be the table or view specified in the first from clause of the `select` statement that defines the cursor.

- After the deletion of a row from the cursor’s result set, the cursor is positioned before the next row in the cursor’s result set. You must issue a `fetch` to access the next row. If the deleted row is the last row of the cursor result set, the cursor is positioned after the last row of the result set. The following describes the position and behavior of open cursors affected by a `delete`:
  - If a client deletes a row (using another cursor or a regular `delete`) and that row represents the current cursor position of other opened cursors owned by the same client, the position of each affected cursor is implicitly set to precede the next available row. However, one client cannot delete a row representing the current cursor position of another client’s cursor.
  - If a client deletes a row that represents the current cursor position of another cursor defined by a join operation and owned by the same client, Adaptive Server accepts the `delete` statement. However, it implicitly closes the cursor defined by the join.

Using `readpast`

- The `readpast` option allows `delete` commands on data-only-locked tables to proceed without being blocked by incompatible locks held by other tasks.
  - On datarows-locked tables, `readpast` skips all rows on which shared, update, or exclusive locks are held by another task.
  - On datapages-locked tables, `readpast` skips all pages on which shared, update, or exclusive locks are held by another task.
Commands specifying readpast block if there is an exclusive table lock.

- If the readpast option is specified for an allpages-locked table, the readpast option is ignored. The command blocks as soon as it finds an incompatible lock.

- If the session-wide isolation level is 3, the readpast option is silently ignored. The command executes at level 3. The command blocks on any rows or pages with incompatible locks.

- If the transaction isolation level for a session is 0, a delete command using readpast does not issue warning messages. For datapages-locked tables, delete with readpast modifies all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, it affects all rows that are not locked with incompatible locks.

- If the delete command applies to a row with two or more text columns, and any text column has an incompatible lock on it, readpast locking skips the row.

Using index, prefetch, or lru | mru

- The index, prefetch, and lru | mru options override the choices made by the Adaptive Server optimizer. Use these options with caution, and always check the performance impact with set statistics io on. For more information about using these options, see Chapter 4, “Using the set statistics command,” in Performance and Tuning Guide: Monitoring and Analyzing.

Standards
ANSI SQL – Compliance level: Entry-level compliant.

The use of more than one table in the from clause and qualification of table name with database name are Transact-SQL extensions.

readpast is a Transact-SQL extension.

Permissions
delete permission defaults to the table or view owner, who can transfer it to other users.

If set ansi_permissions is on, you must have select permission on all columns appearing in the where clause, in addition to the regular permissions required for delete statements. By default, ansi_permissions is off.

Auditing
Values in event and extrainfo columns of sysaudits are:
<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 18    | delete       | delete from a table       | • Roles – Current active roles  
• Keywords or options – delete  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |
| 19    | delete       | delete from a view        | • Roles – Current active roles  
• Keywords or options – delete  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

See also \textbf{Commands} \begin{itemize} \item create trigger, drop table, drop trigger, truncate table, where clause \end{itemize}
delete statistics

Description
Removes statistics from the sysstatistics system table.

Syntax
delete [shared] statistics table_name
    [partition data_partition_name]
    [(column_name, column_name) ...]

Parameters

shared
removes simulated statistics information from sysstatistics in the master database.

table_name
removes statistics for all columns in the table.

data_partition_name
deletes all statistics for the data partition. Global statistics are not deleted.

column_name
removes statistics for the specified column.

Examples

Example 1 Deletes the densities, selectivities, and histograms for all columns in the titles table:

delete statistics titles

Example 2 Deletes densities, selectivities, and histograms for the pub_id column in the titles table:

delete statistics titles (pub_id)

Example 3 Deletes densities, selectivities, and histograms for the smallsales partition of the titles table:

delete statistics titles partition smallsales

Example 4 Deletes densities, selectivities, and histograms for pub_id, pubdate, without affecting statistics on the single-column pub_id or the single-column pubdate:

delete statistics titles partition smallsales (pub_id, pubdate)

Example 5 Deletes densities, selectivities, and histograms for the column pub_id and for the data partition smallsales:

delete statistics titles partition smallsales (pub_id)

Usage

• delete statistics removes statistics for the specified columns or table from the sysstatistics table. It does not affect statistics in the systabstats table.

• delete statistics on a data partition does not delete global statistics.
• When you issue the drop table command, the corresponding rows in sysstatistics are dropped. When you use the drop index command, the rows in sysstatistics are not deleted. This allows the query optimizer to continue to use index statistics without incurring the overhead of maintaining the index on the table.

Warning! Densities, selectivities, and histograms are essential to good query optimization. The delete statistics command is provided as a tool to remove statistics not used by the optimizer. If you inadvertently delete statistics needed for query optimization, run update statistics on the table, index, or column.

• Loading simulated statistics with the optdiag utility command adds a small number of rows to master..sysstatistics table. If the simulated statistics are no longer in use, the information in master..sysstatistics can be dropped with the delete shared statistics command.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Permission to use delete statistics can be granted or transferred to anyone by table owner or System Administrator.

See also
Commands create index, grant, revoke, update
Utilities optdiag
**disk init**

**Description**

Makes a physical device or file usable by Adaptive Server.

**Syntax**

```plaintext
disk init
    name = "device_name",
    physname = "physicalname",
    skip_alloc={true | false},
    [vdevno = virtual_device_number,]
    size = number_of_blocks
    [, vstart = virtual_address
      , cntrltyp = controller_number]
    [, dsync = {true | false}]
    [, directio = {true | false}]
```

**Parameters**

- **name**
  
  is the name of the database device or file. The name must conform to the rules for identifiers and must be enclosed in single or double quotes. This name is used in the create database and alter database commands.

- **physname**
  
  is the full specification of the database device. This name must be enclosed in single or double quotes. When the physical device path is relative, `disk init` returns a warning.

- **skip_alloc**
  
  is a boolean parameter for the `disk init` command. It is supported for devices created on non-Windows file systems and on Windows raw systems. When `skip_alloc` is set to be “true,” it allows user to avoid initialization of pages with zeros. The default of `skip_alloc` is false.

- **vdevno**
  
  is the virtual device number, which must be unique among the database devices associated with Adaptive Server. The device number 0 is reserved for the master device. Otherwise, valid device numbers must be between 1 and 2,147,483,647.

  To determine the virtual device number, look at the `device_number` column of the `sp_helpdevice` report, and use the next unused integer.
size

is the amount of space to allocate to the new device. The following are example unit specifiers, using uppercase, lowercase, single and double quotes interchangeably: ‘k’ or “K” (kilobytes), ‘m’ or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier. Acceptable values are:

- \( 5120 \) = 10 MB
- \( "5120" \) = 10 MB
- \( "10M" \) = 10 MB

vstart

is the starting virtual address, or the offset, for Adaptive Server to begin using the database device. The following are example unit specifiers, using uppercase, lowercase, single and double quotes interchangeably: ‘k’ or “K” (kilobytes), ‘m’ or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.

The size of the offset depends on how you enter the value for vstart.

- If you do not specify a unit size, vstart uses 2K pages for its starting address. For example, if you specify vstart = 13, Adaptive Server uses 13 * 2K pages as the offset for the starting address.
- If you specify a unit value, vstart uses this as the starting address. For example, if you specify vstart = "13M", Adaptive Server sets the starting address offset at 13 megabytes.

The default value (and usually the preferred value) of vstart is 0. If the specified device does not have the sum of vstart + size blocks available, the disk init command fails. If you are running the Logical Volume Manager on an AIX operating system, vstart should be 2. Specify vstart only if instructed to do so by Sybase Technical Support.

cntrltype

specifies the disk controller. Its default value is 0. Reset cntrltype only if instructed to do so by Sybase Technical Support.
disk init

**dsync**
specifies whether writes to the database device take place directly to the storage media, or are buffered when using operating system files. This option is meaningful only when you are initializing an operating system file; it has no effect when initializing devices on a raw partition. By default, all operating system files are initialized with dsync set to true.

directio
allows you to configure Adaptive Server to transfer data directly to disk, bypassing the operating system buffer cache. directio is a static parameter that requires a restart of Adaptive Server to take effect.

**Examples**

**Example 1** Does not initialize pages with zeroes:

```sql
disk init name="d2",
physname="/usr/sybase/devices/d3.dat",
skip_alloc="true",
size="10G"
```

Adaptive Server does not allocate space during disk initialization if skip_alloc is set to true.

**Example 2** Initializes 10MB of a disk on a UNIX system:

```sql
disk init
name = "user_disk",
physname = "/dev/rxyl1a",
vdevno = 2, size = 5120
```

**Example 3** Initializes 10MB of a disk on a UNIX operating system file. Adaptive Server opens the device file with the dsync setting, and writes to the file are guaranteed to take place directly on the storage media:

```sql
disk init
name = "user_file",
physname = "/usr/u/sybase/data/userfile1.dat",
vdevno = 2, size = 5120, dsync = true
```

**Example 4** Creates a device named “user_disk” that uses directio to write data directly to disk:

```sql
disk init
name = "user_disk",
physname = "/usr/u/sybase/data/userfile1.dat",
size = 5120,
directio= true
```
CHAPTER 1  Commands

Usage

- disk init and disk reinit display a warning message if you attempt to create a block device on a platform that Sybase recommends that you not use block device.

Sybase recommends that you use block devices as a database device only on the HP-UX, Windows, and Linux platforms.

- Use skip_alloc to expedite crash recovery on non-NT file systems and on NT raw systems. Also, using skip_alloc with the directio feature creates device faster and improves durability of updates. Regardless of space availability, skip_alloc always prints a warning message to users saying they need to make sure Adaptive Server has the required space for future use.

- The master device is initialized by the installation program; you need not initialize this device with disk init.

- To successfully complete disk initialization, the “sybase” user must have the appropriate operating system permissions on the device that is being initialized.

- You can specify the size as a float, but the size is rounded down to the nearest multiple of 2K.

- If you do not use a unit specifier for size, disk init uses the virtual page size of 2K.

- The minimum size of a disk piece that you can initialize using disk init is the larger of:
  - One megabyte
  - One allocation unit of the server’s logical page size

- directio and dsync are mutually exclusive. If a device has dsync set to “true,” you cannot set directio to “true” for this device. To enable directio for a device, you must first reset dsync to “false.”

- directio is not available on all platforms. If you issue disk init with the directio parameter on a platform on which it is not supported, Adaptive Server issues the message No such parameter: 'directio'.

- Use disk init for each new database device. Each time disk init is issued, a row is added to master..sysdevices. A new database device does not automatically become part of the pool of default database storage. Assign default status to a database device with sp_diskdefault.
• Back up the master database with the `dump database` or `dump transaction` command after each use of `disk init`. This makes recovery easier and safer in case master is damaged. If you add a device with `disk init` and fail to back up master, you may be able to recover the changes by using `disk reinit`, then stopping and restarting Adaptive Server.

• Assign user databases to database devices with the `on` clause of the `create database` or `alter database` command.

• The preferred method for placing a database’s transaction log (the system table `syslogs`) on a different device than the one on which the rest of the database is stored, is to use the `log on` extension to `create database`. Alternatively, you can name at least two devices when you create the database, then execute `sp_logdevice`. You can also use `alter database` to extend the database onto a second device, then run `sp_logdevice`. The `log on` extension immediately moves the entire log to a separate device. The `sp_logdevice` method retains part of the system log on the original database device until transaction activity causes the migration to become complete.

• For a report on all Adaptive Server devices on your system (both database and dump devices), execute `sp_helpdevice`.

• Remove a database device with `sp_dropdevice`. You must first drop all existing databases on that device.

Using `dsync`

**Note** Do not set `dsync` to `false` for any device that stores critical data. The only exception is `tempdb`, which can safely be stored on devices for which `dsync` is set to `false`.

• When `dsync` is on, writes to the database device are guaranteed to take place on the physical storage media, and Adaptive Server can recover data on the device in the event of a system failure.

• When `dsync` is off, writes to the database device may be buffered by the UNIX file system. The UNIX file system may mark an update as being completed, even though the physical media has not yet been modified. In the event of a system failure, there is no guarantee that data updates have ever taken place on the physical media, and Adaptive Server may be unable to recover the database.

• `dsync` is always on for the master device file.
The dsync value should be turned off only when databases on the device need not be recovered after a system failure. For example, you may consider turning dsync off for a device that stores only the tempdb database.

Adaptive Server ignores the dsync setting for devices stored on raw partitions—writes to those device are guaranteed to take place on the physical storage media, regardless of the dsync setting.

disk reinit ensures that master..sysdevices is correct if the master database has been damaged or if devices have been added since the last dump of master.

Adaptive Server ignores the dsync setting for devices stored on raw partitions—writes to those device are guaranteed to take place on the physical storage media, regardless of the dsync setting.

disk reinit ensures that master..sysdevices is correct if the master database has been damaged or if devices have been added since the last dump of master.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
disk init permission defaults to System Administrators and is not transferable. You must be using the master database to use disk init.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 20    | disk         | disk init                 | • Roles – Current active roles  
|       |              |                           | • Keywords or options – disk init  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – Name of the disk  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect |

See also
Commands alter database, create database, disk refit, disk reinit, dump database, dump transaction, load database, load transaction

System procedures sp_diskdefault, sp_dropdevice, sp_helpdevice, sp_logdevice
**disk mirror**

**Description**

Creates a software mirror that immediately takes over when the primary device fails.

**Syntax**

```sql
disk mirror
    name = "device_name",
    mirror = "physicalname"
    [, writes = {serial | noserial}]
    [clear = {TRUE | FALSE}]
```

**Parameters**

- `name` is the name of the database device to mirror. This is recorded in the `name` column of the `sysdevices` table. The name must be enclosed in single or double quotes.

- `mirror` is the full path name of the database mirror device that is to be your secondary device. It must be enclosed in single or double quotes. If the secondary device is a file, `physicalname` should be a path specification that clearly identifies the file, which Adaptive Server creates. The value of `physicalname` cannot be an existing file.

- `writes` allows you to choose whether to enforce serial writes to the devices. In the default case (`serial`), the write to the primary database device is guaranteed to finish before the write to the secondary device begins. If the primary and secondary devices are on different physical devices, serial writes can ensure that at least one of the disks is unaffected in the event of a power failure.

- `clear` initializes the mirror device with zeros to guarantee that the underlying filesystem has reserved space for the mirror device. The default value, `FALSE`, does not clear the mirror, and executing a write to the device might fail through lack of space on the file system. If you specify `TRUE`, the mirror is cleared, forcing the file system to reserve space for the device.

**Examples**

Creates a software mirror for the database device `user_disk` on the file `mirror.dat`:

```sql
disk mirror
    name = "user_disk",
    mirror = "/server/data/mirror.dat"
```

**Usage**

- Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.
Disk mirroring does not interfere with ongoing activities in the database. You can mirror or unmirror database devices without shutting down Adaptive Server.

- Back up the master database with the `dump database` command after each use of `disk mirror`. This makes recovery easier and safer in case master is damaged.

- When a read or write to a mirrored device is unsuccessful, Adaptive Server unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. The System Administrator must use the `disk remirror` command to restart mirroring.

- The `clear` option in this command has no effect when used on the NT platform.

- You can mirror the master device, devices that store data, and devices that store transaction logs. However, you cannot mirror dump devices.

- Devices are mirrored; databases are not.

- A device and its mirror constitute one logical device. Adaptive Server stores the physical name of the mirror device in the `mirrorname` column of the `sysdevices` table. It does not require a separate entry in `sysdevices` and should not be initialized with `disk init`.

- To retain use of asynchronous I/O, always mirror devices that are capable of asynchronous I/O to other devices capable of asynchronous I/O. In most cases, this means mirroring raw devices to raw devices and operating system files to operating system files.

  If the operating system cannot perform asynchronous I/O on files, mirroring a raw device to a regular file produces an error message. Mirroring a regular file to a raw device works, but does not use asynchronous I/O.

- Mirror all default database devices so that you are still protected if a `create database` or `alter database` command affects a database device in the default list.

- For greater protection, mirror the database device used for transaction logs.
disk mirror

- Always put user database transaction logs on a separate database device. To put a database’s transaction log (that is, the system table syslogs) on a device other than the one on which the rest of the database is stored, name the database device and the log device when you create the database. Alternatively, use alter database to extend the database onto a second device, then run sp_logdevice.

- If you mirror the database device for the master database, you can use the -r option and the name of the mirror for UNIX, when you restart Adaptive Server with the dataserver utility program. Add this to the RUN_servername file for that server so that the startserver utility program knows about it. For example, to start a master device named master.dat and its mirror, mirror.dat enter:

  dataserver -dmaster.dat -rmirror.dat

  For more information, see dataserver and startserver in the Utility Guide.

- If you mirror a database device that has unallocated space (room for additional create database and alter database statements to allocate part of the device), disk mirror begins mirroring these allocations when they are made, not when the disk mirror command is issued.

- For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute sp_helpdevice.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
disk mirror permission defaults to the System Administrator and is not transferable. You must be using the master database to use disk mirror.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 23    | disk         | disk mirror               | • Roles – Current active roles  
|       |              |                           | • Keywords or options – disk mirror  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – Name of the disk  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect  

See also

Commands alter database, create database, disk init, disk refit, disk reinit, disk remirror, disk unmirror, dump database, dump transaction, load database, load transaction
CHAPTER 1 Commands

System procedures  sp_diskdefault, sp_helpdevice, sp_logdevice
Utilities  dataserver, startserver
disk refit

Description
Rebuilds the master database’s sysusages and sysdatabases system tables from information contained in sysdevices.

Syntax
disk refit

Examples
disk refit

Usage
• Adaptive Server automatically shuts down after disk refit rebuilds the system tables.
• Use disk refit after disk reinit as part of the procedure to restore the master database.

Note You must start Adaptive Server with trace flag 3608 before you run disk refit. However, make sure you read the information in the Troubleshooting and Error Messages Guide before you start Adaptive Server with any trace flag.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
disk refit permission defaults to System Administrators and is not transferable.
You must be in the master database to use disk refit.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 21    | disk         | disk refit                | • Roles – Current active roles  
|       |              |                           | • Keywords or options – disk refit  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – Name of the disk  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect |

See also
Documents For more information, see the System Administration Guide.

Commands disk init, disk reinit

System procedures sp_addumpdevice, sp_helpdevice
disk reinit

Description
Rebuilds the master database’s sysdevices system table. Use disk reinit as part of the procedure to restore the master database.

Syntax
disk reinit
   name = "device_name",
   physname = "physicalname",
   [vdevno = virtual_device_number,]
   size = number_of_blocks
   [, vstart = virtual_address
      , cntrtype = controller_number]
   [, dsync = {true | false}]
   [, directio = {true | false}]

Parameters
name
   is the name of the database device. It must conform to the rules for identifiers, and it must be enclosed in single or double quotes. This name is used in the create database and alter database commands.

physname
   is the name of the database device. The physical name must be enclosed in single or double quotes.

vdevno
   is the virtual device number, which must be unique among the database devices associated with Adaptive Server. The device number 0 is reserved for the master device. Otherwise, valid device numbers must be between 1 and 2,147,483,647.

To determine the virtual device number, look at the device_number column of the sp_helpdevice report, and use the next unused integer.

size
   is the current size of the device being reinitialized. The following are example unit specifiers, using uppercase, lowercase, single and double quotes interchangeably: `k` or “K” (kilobytes), “m” or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.
vstart

is the starting virtual address, or the offset, for Adaptive Server to begin using the database device. The following are example unit specifiers, using uppercase, lowercase, single and double quotes interchangeably: ‘k’ or “K” (kilobytes), “m” or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier. If you do not provide a unit specifier, the value provided is presumed to be in megabytes. The size of the offset depends on how you enter the value for vstart.

- If you do not specify a unit size, vstart uses 2K pages for its starting address. For example, if you specify $vstart = 13$, Adaptive Server uses $13 \times 2K$ pages as the offset for the starting address.
- If you specify a unit value, vstart uses this as the starting address. For example, if you specify $vstart = "13M"$, Adaptive Server sets the starting address offset at 13 megabytes.

The default value (and usually the preferred value) of vstart is 0. If the specified device does not have the sum of vstart + size blocks available, the disk reinit command fails.

**Note** If you are running the Logical Volume Manager on an AIX operating system, vstart should be 2.

cntrtype

specifies the disk controller. Its default value is 0. Reset it only if instructed to do so by Sybase Technical Support.

dsync

specifies whether writes to the database device take place directly to the storage media, or are buffered when using operating system files. This option is meaningful only when you are initializing an operating system file; it has no effect when initializing devices on a raw partition. By default, all operating system files are initialized with dsync set to true.

directio

allows you to configure Adaptive Server to transfer data directly to disk, bypassing the operating system buffer cache. directio is a static parameter that requires a restart of Adaptive Server to take effect.
Examples

**Example 1** Adds a new row to the `sysdevices` table. This new row contains the characteristics of the existing device currently being reinitialized:

```
disk reinit
name = "user_file",
physname = "/usr/u/sybase/data/userfile1.dat",
vdevno = 2, size = 5120, dsync = true
```

**Example 2** Adds a new row to the `sysdevices` table. This new row contains the characteristics of the existing device currently being reinitialized:

```
disk reinit
name = "user_disk",
physname = "/usr/u/sybase/data/userfile1.dat",
size = 5120, directio= true
```

Usage

- `disk init` and `disk reinit` display a warning message if you attempt to create a block device on a platform that Sybase recommends that you not use block device.

  Sybase recommends that you use block devices as a database device only on the HP-UX, Windows, and Linux platforms.

- `disk reinit` ensures that `master..sysdevices` is correct if the master database has been damaged or if devices have been added since the last dump of master.

- `disk reinit` is similar to `disk init`, but does not initialize the database device.

- You can specify the `size` as a float, but the size is rounded down to the nearest multiple of 2K.

- If you do not use a unit specifier for `size`, `disk reinit` uses the virtual page size of 2K.

- By default, the `directio` option is set to “false” (off) for all platforms.

- For complete information on restoring the master database, see the *System Administration Guide*.

Using `dsync`

**Note** Do not set `dsync` to `false` for any device that stores critical data. The only exception is `tempdb`, which can safely be stored on devices for which `dsync` is set to `false`.

- When `dsync` is on, writes to the database device are guaranteed to take place on the physical storage media, and Adaptive Server can recover data on the device in the event of a system failure.
disk reinit

- directio and dsync are mutually exclusive. If a device has dsync set to “true,” you cannot set directio to “true” for this device. To enable directio for a device, you must first reset dsync to “false.”

- When dsync is off, writes to the database device may be buffered by the UNIX file system. The UNIX file system may mark an update as being completed, even though the physical media has not yet been modified. In the event of a system failure, there is no guarantee that data updates have ever taken place on the physical media, and Adaptive Server may be unable to recover the database.

- dsync is always on for the master device file.

- Turn off the dsync value only when databases on the device need not be recovered after a system failure. For example, you may consider turning dsync off for a device that stores only the tempdb database.

- Adaptive Server ignores the dsync setting for devices stored on raw partitions—writes to those device are guaranteed to take place on the physical storage media, regardless of the dsync setting.

- The dsync setting is not used on the Windows NT platform.

- disk reinit ensures that master..sysdevices is correct if the master database has been damaged or if devices have been added since the last dump of master.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
disk reinit permission defaults to System Administrators and is not transferable. You must be in the master database to use disk reinit.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>disk</td>
<td>disk reinit</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – disk reinit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – Name of the disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also Commands alter database, create database, dbcc, disk init, disk refit

System procedures sp_addumpdevice, sp_helpdevice
disk remirror

Description
Restarts disk mirroring after it is stopped by failure of a mirrored device or temporarily disabled by the disk unmirror command.

Syntax
disk remirror  
name = "device_name"

Parameters
name is the name of the database device that you want to remirror. The name is recorded in the name column of the sysdevices table, and must be enclosed in single or double quotes.

Examples
Resumes software mirroring on the database device user_disk:

disk remirror  
name = "user_disk"

Usage
• Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

  Use the disk remirror command to reestablish mirroring after it has been temporarily stopped by failure of a mirrored device or temporarily disabled with the mode = retain option of the disk unmirror command. The disk remirror command copies data on the retained disk to the mirror.

• Back up the master database with the dump database command after each use of disk remirror. This makes recovery easier and safer in case master is damaged.

• If mirroring was permanently disabled with the mode = remove option, you must remove the operating system file that contains the mirror before using disk remirror.

• Database devices, not databases, are mirrored.

• You can mirror, remirror, or unmirror database devices without shutting down Adaptive Server. Disk mirroring does not interfere with ongoing activities in the database.

• When a read or write to a mirrored device is unsuccessful, Adaptive Server unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. The System Administrator must use disk remirror to restart mirroring.
In addition to mirroring user database devices, always put user database transaction logs on a separate database device. The database device used for transaction logs can also be mirrored for even greater protection. To put a database’s transaction log (that is, the system table syslogs) on a different device than the one on which the rest of the database is stored, name the database device and the log device when you create the database.

Alternatively, alter database to a second device, then run sp_logdevice.

If you mirror the database device for the master database, you can use the -r option and the name of the mirror for UNIX, when you restart Adaptive Server with the dataserver utility program. Add this option to the RUN_servername file for that server so that the startserver utility program knows about it. For example, the following command starts a master device named master.dat and its mirror, mirror.dat:

```
    dataserver -dmaster.dat -rmirror.dat
```

For more information, see dataserver and startserver in the Utility Guide.

For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute sp_helpdevice.

### Standards

ANSI SQL – Compliance level: Transact-SQL extension.

### Permissions

disk remirror permission defaults to the System Administrator and is not transferable. You must be using the master database to use disk remirror.

### Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 25    | disk         | disk remirror             | • Roles – Current active roles  
|       |              |                           | • Keywords or options – disk remirror  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – Name of the disk  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect  |

### See also

Commands: alter database, create database, disk init, disk mirror, disk refit, disk reinit, disk unmirror, dump database, dump transaction, load database, load transaction

Utilities: dataserver, startserver

---

286 Adaptive Server Enterprise
disk resize

Description
Dynamically increases the size of the device used by Adaptive Server.

Syntax
disk resize
   name = "device_name",
   size = additional_space

Parameters
- name
  Is the name of the device whose size you are increasing.
- additional_space
  Is the additional space you are adding to the device.

Examples
To increase the size of testdev by 4MB, enter:

disk resize
   name = "test_dev",
   size = "4M"

Usage
- The disk resize command allows you to dynamically increase the size of your disks.
- After you resize a device, dump the master device, which maintains the size of the device in the sysdevices table. If you attempt a recovery from an old dump of the master device, the information stored in sysdevices will not be current.
- Any properties that are set on the device continue to be set after you increase its size.
- During the physical initialization of the disk, if any error occurs due to insufficient disk space, disk resize extends the database device to the point before the error occurs.

For example, on a server that uses 4K logical pages, if you try to increase the size of the device by 40MB, but only 39.5MB is available, then the device is extended only by 39.5MB. From the extended size (39.5MB), only 39MB is used by Adaptive Server. The last 0.5MB is allocated but not used, as 4K servers configure devices in one MB minimums.

To utilize the last 0.5MB, make sure that there is at least another 1.5MB available for the device, then re-run disk resize, specifying 1.5MB as the incremental size.
- You cannot use disk resize to decrease the size of the device.
- device_name must have a valid identifier. The device is initialized using the disk init command and, it must refer to a valid Adaptive Server device, not a dump or load device.
Use the following are example unit specifiers, using uppercase, lowercase, single and double quotes interchangeably: ‘k’ or “K” (kilobytes), “m” or ‘M’ (megabytes), “g” or “G” (gigabytes), and ‘t’ or ‘T’ (terabytes). Sybase recommends that you always include a unit specifier. Although it is optional, Sybase recommends that you always include the unit specifier with the disk resize command to avoid confusion in the actual number of pages allocated.

You must enclose the unit specifier in single or double quotes. If you do not use a unit specifier, the size defaults to the number of disk pages.

Permanently disable mirroring while the resize operation is in progress. You can reestablish mirroring when the resize operation is completed.

Standards
ANSI SQL – compliance level: Transact-SQL extension

Permissions
Only a user with the sa_role can execute the disk resize command.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>disk</td>
<td>disk resize</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – Index name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if a set proxy is in effect</td>
</tr>
</tbody>
</table>

See also

**Commands**
create database, disk init, drop database, load database

**System procedures**
sp_addsegment, sp_dropsegment, sp_helpdb, sp_helpsegment, sp_logdevice, sp_renamedb, sp_spaceused
disk unmirror

Description
Suspends disk mirroring initiated with the disk mirror command to allow hardware maintenance or the changing of a hardware device.

Syntax
disk unmirror
  name = "device_name"
  [, side = {"primary" | secondary}]
  [, mode = {retain | remove}]

Parameters
name
  is the name of the database device that you want to unmirror. The name must be enclosed in single or double quotes.

side
  specifies whether to disable the primary device or the secondary device (the mirror). By default, the secondary device is unmirrored.

mode
  determines whether the unmirroring is temporary (retain) or permanent (remove). By default, unmirroring is temporary.

Specify retain when you plan to remirror the database device later in the same configuration. This option mimics what happens when the primary device fails:

- I/O is directed only at the device not being unmirrored.
- The status column of sysdevices indicates that mirroring is deactivated.
- remove eliminates all sysdevices references to a mirror device.
- The status column indicates that the mirroring feature is ignored.
- The phyname column is replaced by the name of the secondary device in the mirrorname column if the primary device is the one being deactivated.
- The mirrorname column is set to NULL.

Examples

Example 1  Suspends software mirroring for the database device user_disk:
  disk unmirror
  name = "user_disk"

Example 2  Suspends software mirroring for the database device user_disk on the secondary side:
  disk unmirror name = "user_disk", side = secondary

Example 3  Suspends software mirroring for the database device user_disk and removes all device references to the mirror device:


**Usage**

- Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

  disk unmirror name = "user_disk", mode = remove

  disk unmirror disables either the original database device or the mirror, either permanently or temporarily, so that the device is no longer available to Adaptive Server for reads or writes. It does not remove the associated file from the operating system.

- Disk unmirroring alters the sysdevices table in the master database. Back up the master database with the dump database command after each use of disk unmirror. This makes recovery easier and safer in case master is damaged.

- You can unmirror a database device while it is in use.

- You cannot unmirror any of a database’s devices while a dump database, load database, or load transaction is in progress. Adaptive Server displays a message asking whether to abort the dump or load or to defer the disk unmirror until after the dump or load completes.

- You cannot unmirror a database’s log device while a dump transaction is in progress. Adaptive Server displays a message asking whether to abort the dump or defer the disk unmirror until after the dump completes.

  **Note**  dump transaction with truncate_only and dump transaction with no_log are not affected when a log device is unmirrored.

- Mirror all the default database devices so that you are still protected if a create or alter database command affects a database device in the default list.

- When a read or write to a mirrored device is unsuccessful, Adaptive Server automatically unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. A System Administrator must restart mirroring with the disk remirror command.

- For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute sp_helpdevice.
• Use `disk remirror` to reestablish mirroring after it is temporarily stopped with the `mode = retain` option of the `disk unmirror` command. If mirroring is permanently disabled with the `mode = remove` option, you must remove the operating system file that contains the mirror before using `disk remirror`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
disk unmirror permission defaults to the System Administrator, and is not transferable. You must be using the master database to use `disk unmirror`.

**Auditing**
Values in `event` and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 24    | disk         | disk unmirror             | • Roles – Current active roles  
• Keywords or options – disk unmirror  
• Previous value – NULL  
• Current value – NULL  
• Other information – Name of the disk  
• Proxy information – Original login name, if set proxy in effect |

See also
- **Commands**  
  alter database, create database, disk init, disk mirror, disk refit, disk reinit, disk remirror, dump database, dump transaction, load database, load transaction
- **System procedures**  
  `sp_diskdefault`, `sp_helpdevice`, `sp_logdevice`
- **Utilities**  
  `dataserver`, `startserver`
**drop database**

**Description**  
Removes one or more databases from Adaptive Server, including archive databases.

**Syntax**  
\`drop database database_name [ , database_name ] ...\`

**Parameters**  
`database_name`  
is the name of a database to remove. Use `sp_helpdb` to get a list of databases.

**Examples**  
Removes the publishing database and all its contents:

\`drop database publishing\`

**Usage**  
- When dropping an archive database, all the rows for that database are deleted from the `sysaltusages` table in the scratch database. This requires log space in the scratch database.
- Removing a database deletes the database and all its objects, frees its storage allocation, and erases its entries from the `sysdatabases` and `sysusages` system tables in the `master` database.
- `drop database` clears the suspect page entries pertaining to the dropped database from `master..sysattributes`.

**Encrypted columns and drop database**  
To prevent accidental loss of keys, Adaptive Server fails `drop database` if it contains keys currently used to encrypt columns in other databases. Before dropping the database containing the encryption keys you must first remove the encryption or drop the database containing the encrypted columns.

In the following example `key_db` is the database where the encryption key resides and `col_db` is the database containing the encrypted columns:

\`drop database key_db, col_db\`

Adaptive Server raises an error and fails to drop `key_db`. The drop of `col_db` succeeds. To drop both databases, drop `col_db` first:

\`drop database col_db, key_db\`

**Restrictions**  
- You must be using the `master` database to drop a database.
- You cannot drop a database that is in use (open for reading or writing by any user).
You cannot use `drop database` to remove a database that is referenced by a table in another database. Execute the following query to determine which tables and external databases have foreign-key constraints on primary key tables in the current database:

```sql
select object_name (tableid), frgndbname
from sysreferences
where frgndbname is not null
```

Use `alter table` to drop these cross-database constraints, then reissue the `drop database` command.

- You can use `drop database` to remove a damaged database. If `drop database` does not run because the database is damaged, use the `dbcc dbrepair` to fix the database:

  ```sql
dbcc dbrepair (database_name, dropdb)
  ```

- You cannot drop the `sybsecurity` database if auditing is enabled. When auditing is disabled, only the System Security Officer can drop `sybsecurity`.

### Standards
ANSI SQL – Compliance level: Transact-SQL extension.

### Permissions
Only the Database Owner can execute `drop database`, except for the `sybsecurity` database, which can be dropped only by the System Security Officer.

### Auditing
Values in event and extrainfo columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 26    | `drop`       | `drop database`           | - `Roles` – Current active roles
|       |              |                           | - `Keywords or options` – NULL
|       |              |                           | - `Previous value` – NULL
|       |              |                           | - `Current value` – NULL
|       |              |                           | - `Other information` – NULL
|       |              |                           | - `Proxy information` – Original login name, if set proxy in effect

See also

- **Commands** `alter database`, `create database`, `dbcc`, `use`
- **Procedures** `sp_changedbowner`, `sp_helpdb`, `sp_renamedb`, `sp_spaceused`
drop default

Description
Removes a user-defined default.

Syntax
drop default [owner.]default_name
    [, [owner.]default_name] ...

Parameters
default_name
is the name of an existing default. Execute sp_help to display a list of
existing defaults. Specify the owner’s name to drop a default of the same
name owned by a different user in the current database. The default value for
owner is the current user.

Examples
Removes the user-defined default datedefault from the database:
    drop default datedefault

Usage
- You cannot drop a default that is currently bound to a column or to a
user-defined datatype. Use sp_unbindefault to unbind the default before
you drop it.
- You can bind a new default to a column or user-defined datatype without
unbinding its current default. The new default overrides the old one.
- When you drop a default for a NULL column, NULL becomes the
column’s default value. When you drop a default for a NOT NULL
column, an error message appears if users do not explicitly enter a value
for that column when inserting data.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
drop default permission defaults to the owner of the default and is not
transferable.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 31    | drop         | drop default              | • Roles – Current active roles
        |               |                           | • Keywords or options – NULL
        |               |                           | • Previous value – NULL
        |               |                           | • Current value – NULL
        |               |                           | • Other information – NULL
        |               |                           | • Proxy information – Original login name, if set proxy in effect

See also
Commands create default
System procedures sp_help, sp_helptext, sp_unbindefault
drop encryption key

Description
Allows key owners to drop the named encryption key.

Syntax
`drop encryption key [[database.[owner].]keyname`

Parameters
- `database`
  is the name of the database
- `owner`
  is the owner
- `keyname`
  is the name of the key

Examples
Drops the encryption key `cc_key`:

```
drop encryption key cust.dbo.cc_key
```

Usage
- If the key has key copies, the copies are dropped along with the base key.
- The command fails if any column in any database is encrypted using the key.
- `drop encryption key` cannot check databases that are archived, suspect, offline, unrecovered, and currently being loaded for columns encrypted by the key. The command issues a warning message naming the unavailable database, but does not fail. When the database is brought online, any tables with columns that were encrypted with the dropped key are not usable. To restore the key, the System Administrator must load a dump of the dropped key’s database from a time that precedes when the key was dropped.

Permissions
The key owner and the System Security Officer can drop encryption keys.

Auditing
Values in `event` and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td></td>
<td>drop encryption key</td>
<td></td>
</tr>
</tbody>
</table>

See also
`create encryption key`, `alter encryption key`, `sp_encryption`, `sp_help`. 
**drop function**

Description  
Removes one or more user-defined functions from the current database.

Syntax  
drop function{ [ owner_name . ] function_name } [ ,...n ]

Parameters  
- **owner_name**  
is the name of the user ID that owns the user-defined function. Must be an existing user ID.

- **function_name**  
is name of the user-defined function to be removed. Specifying the owner name is optional; the server name and database name cannot be specified.

Examples  
Drops the \textit{bonus} function:

```
drop function bonus
```

Usage  
drop function drops scalar SQL user-defined functions from your current database.

Permissions  
Permissions default to the function owner, and are not transferable. Users with the sa and the dbo roles can drop any object by specifying the owner.
drop function (SQLJ)

Description
Removes a SQLJ function.

Syntax
drop func[tion] [owner.]function_name
[. [owner.]function_name ] ...

Parameters
[owner.]function_name
is the SQL name of a SQLJ function.

Examples
Removes the SQLJ function square_root:

drop function square_root

Usage
drop function removes only user-created functions from the current database. It
does not remove system functions.

Permissions
Only the Database Owner or user with the sa_role can execute drop function.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 98    | drop         | drop function             | • Roles – Current active roles
|       |              |                           | • Keywords or options – NULL
|       |              |                           | • Previous value – NULL
|       |              |                           | • Current value – NULL
|       |              |                           | • Other information – NULL
|       |              |                           | • Proxy information – Original login name, if a set proxy is in effect

See also
Documents See *Java in Adaptive Server Enterprise* for more information about SQLJ functions.

Commands create function (SQLJ)
**drop index**

**Description**
Removes an index from a table in the current database.

**Syntax**
```
drop index table_name.index_name
   [, table_name.index_name] ...
```

**Parameters**
- `table_name`
  is the table in which the indexed column is located. The table must be in the current database.
- `index_name`
  is the index to drop. In Transact-SQL, index names need not be unique in a database, though they must be unique within a table.

**Examples**
Removes `au_id_ind` from the `authors` table:
```
drop index authors.au_id_ind
```

**Usage**
- Once the `drop index` command is issued, you regain all the space that was previously occupied by the index. This space can be used for any database objects.
- You cannot use `drop index` on system tables.
- `drop index` cannot remove indexes that support unique constraints. To drop such indexes, drop the constraints through `alter table` or drop the table. See `create table` for more information about unique constraint indexes.
- You cannot drop indexes that are currently used by any open cursor. For information about which cursors are open and what indexes they use, use `sp_cursorinfo`.
- To get information about what indexes exist on a table, use the following, where `objname` is the name of the table:
  ```
  sp_helpindex objname
  ```

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
drop index permission defaults to the index owner and is not transferable.

**Auditing**
Values in `event` and `extrainfo` columns of `sysaudits` are:
### Reference Manual: Commands

#### CHAPTER 1    Commands

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>drop index</td>
<td></td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if a set proxy is in effect</td>
</tr>
</tbody>
</table>

**See also**

- **Commands**: `create index`

- **System procedures**: `sp_cursorinfo`, `sp_helpindex`, `sp_spaceused`
drop procedure

Description
Removes a procedure.

Syntax
drop proc[edure] [owner.]procedure_name
[., [owner.]procedure_name] ...

Parameters

procedure_name
is the name of the Transact-SQL or SQLJ procedure to drop. Specify the
owner’s name to drop a procedure of the same name owned by a different
user in the current database. The default value for owner is the current user.

Examples

Example 1 Deletes the stored procedure showind:

drop procedure showind

Example 2 Unregisters the extended stored procedure xp_echo:

drop procedure xp_echo

Usage

• drop procedure drops user-defined stored procedures, system procedures,
and extended stored procedures (ESPs).

• Adaptive Server checks the existence of a procedure each time a user or a
program executes that procedure.

• A procedure group (more than one procedure with the same name but with
different number suffixes) can be dropped with a single drop procedure
statement. For example, if the procedures used with the application named
orders were named orderproc;1, orderproc;2, and so on, the following
statement drops the entire group:

drop proc orderproc

Once procedures have been grouped, individual procedures within the
group cannot be dropped. For example, the following statement is not
allowed:

drop procedure orderproc;2

You cannot drop extended stored procedures as a procedure group.

• sp_helptext displays the procedure’s text, which is stored in syscomments.

• sp_helpextendedproc displays ESPs and their corresponding DLLs.

• Dropping an ESP unregisters the procedure by removing it from the
system tables. It has no effect on the underlying DLL.

• drop procedure drops only user-created procedures from your current
database.
Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
drop procedure permission defaults to the procedure owner and is not transferable.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>drop</td>
<td>drop procedure</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also

**Commands** create procedure, create procedure (SQLJ)

**System procedures** sp_depends, sp_dropextendedproc, sp_helpextendedproc, sp_helptext, sp_rename
**drop role**

**Description**
Drops a user-defined role.

**Syntax**
drop role *role_name* [with override]

**Parameters**
- *role_name*
  - is the name of the role you want to drop.
- with override
  - overrides any restrictions on dropping a role. When you use the with override option, you can drop any role without having to check whether the role permissions have been dropped in each database.

**Examples**

**Example 1** Drops the named role only if all permissions in all databases have been revoked. The System Administrator or object owner must revoke permissions granted in each database before dropping a role, or the command fails:

```
drop role doctor_role
```

**Example 2** Drops the named role and removes permission information and any other reference to the role from all databases:

```
drop role doctor_role with override
```

**Usage**
- You need not drop memberships before dropping a role. Dropping a role automatically removes any user’s membership in that role, regardless of whether you use the with override option.
- Use drop role from the master database.

**Restrictions**
- You cannot use drop role to drop system roles.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
You must be a System Security Officer to use drop role. drop role permission is not included in the grant all command.

**Auditing**
Values in event and extrainfo columns of sysaudits are:
### CHAPTER 1 Commands

#### Reference Manual: Commands

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 85    | roles        | create role, drop role, alter role, grant role, or revoke role | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

See also **Commands** alter role, create role, grant, revoke, set

**System procedures** sp_activeroles, sp_displaylogin, sp_displayroles, sp_helprotect, sp_modifylogin
**drop rule**

**Description**  
Removes a user-defined rule.

**Syntax**  
drop rule [owner.]rule_name[, [owner.]rule_name] ...

**Parameters**  
*rule_name*  
is the name of the rule to drop. Specify the owner’s name to drop a rule of  
the same name owned by a different user in the current database. The default  
value for *owner* is the current user.

**Examples**  
Removes the rule *pubid_rule* from the current database:

```
drop rule pubid_rule
```

**Usage**  
- Before dropping a rule, you must unbind it using *sp_unbindrule*. If the rule  
  has not been unbound, an error message appears, and the *drop rule*  
  command fails.

- You can bind a new rule to a column or user-defined datatype without  
  unbinding its current rule. The new rule overrides the old one.

- After you drop a rule, Adaptive Server enters new data into the columns  
  that were previously governed by the rule without constraints. Existing  
  data is not affected in any way.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
drop rule permission defaults to the rule owner and is not transferable.

**Auditing**  
Values in event and extrainfo columns of *sysaudits* are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 30    | drop         | drop rule                  | • *Roles* – Current active roles  

• *Keywords or options* – NULL  

• *Previous value* – NULL  

• *Current value* – NULL  

• *Other information* – NULL  

• *Proxy information* – Original login name, if set  
  proxy in effect

**See also**  
Commands  
create rule

**System procedures**  
*sp_bindrule*, *sp_help*, *sp_helptext*, *sp_unbindrule*

---

Adaptive Server Enterprise
drop service

Description
The drop service command removes a user-defined Web service from the current database. Both the metadata and the corresponding stored procedure are removed.

Syntax
drop service service-name

Parameters
service-name
the name for the user-defined Web service. This name can be any name that is valid for a stored procedure. If you specify the name of an service that does not exist, an exception results. Also, you cannot drop a service that is currently in use by another session.

Examples
This example drops the user-defined Web service named sp_who_service:

```
drop service sp_who_service
```

Usage
You must undeploy a user-defined Web service before you can drop it. For details on the undeploy option, see sp_webservices on page 660.

See also
Commands  create service
Stored procedures  sp_webservices
Documentaiton  Web Services User’s Guide
**drop table**

**Description**
Removes a table definition and all of its data, indexes, partition properties, triggers, encryption properties, and permissions from the database.

**Syntax**
```
drop table [[database.]owner.]table_name [, [database.]owner.]table_name] ...
```

**Parameters**
- `table_name`
  is the name of the table to drop. Specify the database name if the table is in another database, and specify the owner’s name if more than one table of that name exists in the database. The default value for `owner` is the current user, and the default value for `database` is the current database.

**Examples**
Removes the table `roysched` and its data and indexes from the current database:

```
drop table roysched
```

**Usage**
- When you use `drop table`, any rules or defaults on the table lose their binding, and any triggers associated with it are automatically dropped. If you re-create a table, you must rebind the appropriate rules and defaults and re-create any triggers.
- When you drop a table, any partition condition associated with the table is also dropped.
- Dropping a table drops any decrypt default associated with the table’s columns, and drops the columns’ encryption properties.
- The system tables affected when a table is dropped are `sysobjects`, `syscolumns`, `sysindexes`, `sysprotects`, `syscomments`, `syspartitions`, `syspartitionkeys`, and `sysprocedures`.
- If Component Integration Services is enabled, and if the table being dropped was created with `create existing table`, the table is not dropped from the remote server. Instead, Adaptive Server removes references to the table from the system tables.

**Restrictions**
- You cannot use the `drop table` command on system tables.
- You can drop a table in any database, as long as you are the table owner. For example, use either of the following to drop a table called `newtable` in the database `otherdb`:

  ```
drop table otherdb..newtable
  drop table otherdb.yourname.newtable
  ```
- If you delete all the rows in a table or use the `truncate table` command, the table still exists until you drop it.
Dropping tables with cross-database referential integrity constraints

- When you create a cross-database constraint, Adaptive Server stores the following information in the sysreferences system table of each database:

### Table 1-21: Information stored about referential integrity constraints

<table>
<thead>
<tr>
<th>Information stored in sysreferences</th>
<th>Columns with information about referenced table</th>
<th>Columns with information about referencing table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key column IDs</td>
<td>refkey1 through refkey16</td>
<td>fokey1 through fokey16</td>
</tr>
<tr>
<td>Table ID</td>
<td>reftabid</td>
<td>tableid</td>
</tr>
<tr>
<td>Database name</td>
<td>pmrydbname</td>
<td>frgndbname</td>
</tr>
</tbody>
</table>

- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains it, or
  - Rename either database with `sp_renamedb`.

Use `sp_helpconstraint` to determine which tables reference the table you want to drop. Use `alter table` to drop the constraints before reissuing `drop table`.

- You can drop a referencing table or its database. Adaptive Server automatically removes the foreign-key information from the referenced database.

- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases can cause database corruption. For more information about loading databases with cross-database referential integrity constraints, see the *System Administration Guide*.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`drop table` permission defaults to the table owner and is not transferable.

**Auditing**

Values in event and extrainfo columns of sysaudits are:
drop table

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 27    | drop         | drop table                | • *Roles* – Current active roles  
• *Keywords or options* – NULL  
• *Previous value* – NULL  
• *Current value* – NULL  
• *Other information* – NULL  
• *Proxy information* – Original login name, if set proxy in effect |

See also  

**Commands**  
alter table, create table, delete, truncate table  

**System procedures**  
sp_depends, sp_help, sp_spaceused
drop trigger

Description
Removes a trigger.

Syntax
\[
drop \text{ trigger} \ [\text{owner}.] \text{trigger\_name} \\
\quad [, [\text{owner}.] \text{trigger\_name}] \ldots
\]

Parameters
trigger\_name
is the name of the trigger to drop. Specify the owner’s name to drop a trigger of the same name owned by a different user in the current database. The default value for owner is the current user.

Examples
Removes trigger1 from the current database:

```
drop trigger trigger1
```

Usage
- drop trigger drops a trigger in the current database.
- You need not explicitly drop a trigger from a table to create a new trigger for the same operation (insert, update, or delete). In a table or column each new trigger for the same operation overwrites the previous one.
- When a table is dropped, Adaptive Server automatically drops any triggers associated with it.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
drop trigger permission defaults to the trigger owner and is not transferable.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 29    | drop         | drop trigger              | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

See also
Commands create trigger

System procedures sp_depends, sp_help, sp_helptext
**drop view**

**Description**  
Removes one or more views from the current database.

**Syntax**  
drop view [owner.]view_name [, [owner.]view_name] ...

**Parameters**  
view_name  
is the name of the view to drop. Specify the owner’s name to drop a view of  
the same name owned by a different user in the current database. The default  
value for owner is the current user.

**Examples**  
Removes the view new_price from the current database:

drop view new_price

**Usage**  
- When you use drop view, the definition of the view and other information  
  about it, including privileges, is deleted from the system tables sysobjects,  
syscolumns, syscomments, sysdepends, sysprocedures, and sysprotects.
- Existence of a view is checked each time the view is referenced, for  
  example, by another view or by a stored procedure.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
drop view permission defaults to the view owner and is not transferable.

**Auditing**  
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 33    | drop         | drop view                 | • Roles – Current active roles  
        |               |                            | • Keywords or options – NULL  
        |               |                            | • Previous value – NULL  
        |               |                            | • Current value – NULL  
        |               |                            | • Other information – NULL |

**See also**  
Commands  
create view

System procedures  
sp_depends, sp_help, sp_helptext
dump database

Description

Makes a backup copy of the entire database, including the transaction log, in a form that can be read in with load database. Dumps and loads are performed through Backup Server.

The target platform of a load database operation need not be the same platform as the source platform where the dump database operation occurred. dump database and load database are performed from either a big endian platform to a little endian platform, or from a little endian platform to a big endian platform.

Syntax

dump database database_name
to [compress::[compression_level::]]stripe_device
[at backup_server_name]
density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name
with verify[= header | full]
[stripe on [compress::[compression_level::]]stripe_device
[at backup_server_name]
density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]]
[[stripe on [compress::[compression_level::]]stripe_device
[at backup_server_name]
density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name][...]

[with {
density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
compression = compress_level
dumpvolume = volume_name,
file = file_name,
[dismount | nodismount],
[nounload | unload],
passwd = password,
retaindays = number_days,
[noinit | init],
notify = {client | operator_console} }]

Reference Manual: Commands
dump database

Parameters

**database_name**

is the name of the database from which you are copying data. The database name can be specified as a literal, a local variable, or a stored procedure parameter.

**compress::compression_level**

is a number between 0 and 9, with 0 indicating no compression, and 9 providing the highest level of compression. If you do not specify `compression_level`, the default is 1. See Chapter 28, “Backing Up and Restoring User Databases” in the *System Administration Guide* for more information about the `compress` option.

**Note** Sybase recommends the native "compression = compression_level" option as preferred over the older "compress::compression_level" option. The native option allows compression of both local and remote dumps, and the dumps that it creates will describe their own compression level during a load. The older option is retained for compatibility with older applications.

to **stripe_device**

is the device to which to copy the data. See “Specifying dump devices” on page 320 in this section for information about what form to use when specifying a dump device.

at **backup_server_name**

is the name of the Backup Server. Do not specify this parameter when dumping to the default Backup Server. Specify this parameter only when dumping over the network to a remote Backup Server. You can specify as many as 32 remote Backup Servers with this option. When dumping across the network, specify the `network name` of a remote Backup Server running on the machine to which the dump device is attached. For platforms that use interfaces files, the `backup_server_name` must appear in the interfaces file.

density = **density_value**

overrides the default density for a tape device. Valid densities are 800, 1600, 6250, 6666, 10000, and 38000. Not all values are valid for every tape drive; use the correct density for your tape drive.

**blocksize = number_bytes**

overrides the default block size for a dump device. The block size must be at least one database page (2048 bytes for most systems) and must be an exact multiple of the database page size. For optimal performance, specify the blocksize as a power of 2, for example, 65536, 131072, or 262144.
capacity = number_kilobytes
is the maximum amount of data that the device can write to a single tape volume. The capacity must be at least five database pages and should be less than the recommended capacity for your device.

A general rule for calculating capacity is to use 70 percent of the manufacturer’s maximum capacity for the device, allowing 30 percent for overhead such as record gaps and tape marks. The maximum capacity is the capacity of the device on the drive, not the drive itself. This rule works in most cases, but may not work in all cases due to differences in overhead across vendors and across devices.

On UNIX platforms that cannot reliably detect the end-of-tape marker, indicate how many kilobytes can be dumped to the tape. You must supply a capacity for dump devices specified as a physical path name. If a dump device is specified as a logical device name, the Backup Server uses the size parameter stored in the sysdevices system table unless you specify a capacity.

compression = compress_level
is a number between 1 and 9, with 9 providing the highest level of compression. There is no default compression level; if you do not specify compression = compress_level, Adaptive Server does not compress the dump.

### Note
Sybase recommends the native "compression = compress_level" option as preferred over the older "compress::compression_level" option. The native option allows compression of both local and remote dumps, and the dumps that it creates will describe their own compression level during a load. The older option is retained for compatibility with older applications.

dumpvolume = volume_name
establishes the name that is assigned to the volume. The maximum length of volume_name is 6 characters. Backup Server writes the volume_name in the ANSI tape label when overwriting an existing dump, dumping to a new tape, or dumping to a tape whose contents are not recognizable. The load database command checks the label and generates an error message if the wrong volume is loaded.

### Warning!
Label each tape volume as you create it so that the operator can load the correct tape.
dump database

with verify[= header | full]  
allows the backup server to perform a minimal header or structural row  
check on the data pages as they are being copied to the archives. There are  
no structural checks done at this time to gam, oam, allocation pages, indexes,  
text, or log pages. The only other check is done on pages where the page  
number matches to the page header.

stripe on stripe_device

is an additional dump device. You can use as many as 32 devices, including  
the device named in the to stripe_device clause. The Backup Server splits the  
database into approximately equal portions, and sends each portion to a  
different device. Dumps are made concurrently on all devices, reducing the  
time required to make a dump and requiring fewer volume changes during  
during the dump. See “Specifying dump devices” on page 320 for information  
about how to specify a dump device.

dismount | nodismount

on platforms that support logical dismount, determines whether tapes  
remain mounted. By default, all tapes used for a dump are dismounted when  
the dump completes. Use nodismount to keep tapes available for additional  
dumps or loads.

nounload | unload

determines whether tapes rewind after the dump completes. By default,  
tapes do not rewind, allowing you to make additional dumps to the same  
tape volume. Specify unload for the last dump file to be added to a  
multidump volume. This rewinds and unloads the tape when the dump  
completes.

passwd = password

is the password you provide to protect the dump file from unauthorized  
users. The password must be between 6 and 30 characters long. You cannot  
use variables for passwords. For rules on passwords, see Chapter 14,  
“Managing Adaptive Server Logins, Database Users, and Client  
retaindays = number_days

*on UNIX systems* – when dumping to disk, specifies the number of days that Backup Server protects you from overwriting the dump. If you try to overwrite the dump before it expires, Backup Server requests confirmation before overwriting the unexpired volume.

**Note** This option is meaningful only when dumping to a disk. It is not meaningful for tape dumps.

The *number_days* must be a positive integer or 0, for dumps that you can overwrite immediately. If you do not specify a retaindays value, Backup Server uses the *tape retention in days* value set by `sp_configure`.

noinit | init
determines whether to append the dump to existing dump files or reinitialize (overwrite) the tape volume. By default, Adaptive Server appends dumps following the last end-of-tape mark, allowing you to dump additional databases to the same volume. New dumps can be appended only to the last volume of a multivolume dump. Use `init` for the first database you dump to a tape to overwrite its contents.

Use `init` when you want Backup Server to store or update tape device characteristics in the tape configuration file. For more information, see the *System Administration Guide*.

file = file_name

is the name of the dump file. The name cannot exceed 17 characters and must conform to operating system conventions for file names. For more information, see “Dump files” on page 321.

notify = {client | operator_console}

overrides the default message destination.

On operating systems that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which Backup Server is running. Use `client` to route other Backup Server messages to the terminal session that initiated the dump database.

On operating systems that do not offer an operator terminal feature, such as UNIX, messages are sent to the client that initiated the dump database. Use `operator_console` to route messages to the terminal on which Backup Server is running.
dump database

Examples

**Example 1** Dumps the database pubs2 to a tape device. If the tape has an ANSI tape label, this command appends this dump to the files already on the tape, since the init option is not specified:

```
dump database pubs2
to "/dev/nrmt0"
```

**Example 2** *For UNIX* – dumps the pubs2 database, using the REMOTE_BKP_SERVER Backup Server. The command names three dump devices, so the Backup Server dumps approximately one-third of the database to each device. This command appends the dump to existing files on the tapes. On UNIX systems, the retaindays option specifies that the tapes cannot be overwritten for 14 days:

```
dump database pubs2
to "/dev/rmt4" at REMOTE_BKP_SERVER
stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
with retaindays = 14
```

**Example 3** The init option initializes the tape volume, overwriting any existing files:

```
dump database pubs2
to "/dev/nrmt0"
with init
```

**Example 4** Rewinds the dump volumes upon completion of the dump:

```
dump database pubs2
to "/dev/nrmt0"
with unload
```

**Example 5** *For UNIX* – the notify clause sends Backup Server messages requesting volume changes to the client which initiated the dump request, rather than sending them to the default location, the console of the Backup Server machine:

```
dump database pubs2
to "/dev/nrmt0"
with notify = client
```

**Example 6** Creates a compressed dump of the pubs2 database into a local file called *dmp090100.dmp* using a compression level of 4:

```
dump database pubs2 to
"compress::4::/opt/bin/Sybase/dumps/dmp090100.dmp"
```
Alternatively, you can create a compressed dump of the pubs2 database into a local file called \texttt{dmp090100.dmp} using a compression level of 4 using \texttt{compression = compression\_level} syntax:

\begin{verbatim}
dump database pubs2 to "/opt/bin/Sybase/dumps/dmp090100.dmp"
with compression = 4
\end{verbatim}

\textbf{Example 7} Dumps the pubs2 database to the remote machine called “remotemachine” and uses a compression level of 4:

\begin{verbatim}
dump database pubs2 to "/Syb\_backup/mydb.db" at remotemachine
with compression = "4"
\end{verbatim}

\textbf{Usage}
\begin{itemize}
  \item If you use \texttt{sp\_hidetext} followed by a cross-platform dump and load, you must manually drop and re-create all hidden objects.
  \item Table 1-22 describes the commands and system procedures used to back up databases:
\end{itemize}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{To do this} & \textbf{Use this command} \\
\hline
Make routine dumps of the entire database, including the transaction log. & \texttt{dump database} \\
\hline
Make routine dumps of the transaction log, then truncate the inactive portion. & \texttt{dump transaction} \\
\hline
Dump the transaction log after failure of a database device. & \texttt{dump transaction with no\_truncate} \\
\hline
Truncate the log without making a backup, then copy the entire database. & \texttt{dump transaction with truncate\_only}
\texttt{dump database} \\
\hline
Truncate the log after your usual method fails due to insufficient log space, then copy the entire database. & \texttt{dump transaction with no\_log}
\texttt{dump database} \\
\hline
Respond to the Backup Server volume change messages. & \texttt{sp\_volchanged} \\
\hline
\end{tabular}
\caption{Commands used to back up databases and logs}
\end{table}

\textbf{Restrictions}
\begin{itemize}
  \item If a database has proxy tables, the proxy tables are a part of the database save set. The content data of proxy tables is not included in the save; only the pointer is saved and restored.
  \item You cannot dump from an 11.x Adaptive Server to a 10.x Backup Server.
  \item You cannot mix Sybase dumps and non-Sybase data (for example, UNIX archives) on the same tape.
\end{itemize}
• If a database has cross-database referential integrity constraints, the sysreferences system table stores the name—not the ID number—of the external database. Adaptive Server cannot guarantee referential integrity if you use load database to change the database name or to load it onto a different server.

**Warning!** Before dumping a database to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

• You cannot use dump database in a user-defined transaction.

• If you issue dump database on a database where a dump transaction is already in progress, dump database sleeps until the transaction dump completes.

• When using 1/4-inch cartridge tape, you can dump only one database or transaction log per tape.

• You cannot dump a database if it has offline pages. To force offline pages online, use sp_forceonline_db or sp_forceonline_page.

• Before you run dump database, for a cross platform dump and load, use the following procedures to move the database to a transactional quiescent status:
  a. Verify the database runs cleanly by executing dbcc checkdb and dbcc checkalloc.
  b. To prevent concurrent updates from open transactions by other processes during dump database, use sp_dboption to place the database in a single-user mode.
  c. Flush statistics to systabstats using sp_flushstats.
  d. Wait for 10 to 30 seconds, depending on the database size and activity.
  e. Run checkpoint against the database to flush updated pages.
  f. Run dump database.

• dump transaction and load transaction are not allowed across platforms.

• dump database and load database to or from a remote backupserver are not supported across platforms.

• You cannot load a password-protected dump file across platforms.
If you perform `dump database` and `load database` for a parsed XML object, you must parse the text again after the `load database` is completed.

You cannot perform `dump database` and `load database` across platforms on Adaptive Servers versions earlier than 11.9.

Adaptive Server cannot translate embedded data structures stored as binary, varbinary, or image columns.

`load database` is not allowed on the master database across platforms.

Stored procedures and other compiled objects are recompiled from the SQL text in `syscomments` at the first execution after the `load database`.

If you do not have permission to recompile from text, then the person who does has to recompile from text using `dbcc upgrade_object` to upgrade objects.

**Note** If you migrate login records in `syslogins` system table in the master database from Solaris to Linux, you can use `bcp` with character format. The login password from the Solaris platform is compatible on Linux without a trace flag from this release. For all other combinations and platforms, login records need to be re-created because the passwords are not compatible.

**Scheduling dumps**

- Adaptive Server database dumps are *dynamic*—they can take place while the database is active. However, they may slow the system down slightly, so you may want to run `dump database` when the database is not being heavily updated.

- Back up the master database *regularly and frequently*. In addition to your regular backups, dump master after each `create database`, `alter database`, and `disk init` command is issued.

- Back up the model database each time you make a change to the database.

- Use `dump database` immediately after creating a database, to make a copy of the entire database. You cannot run `dump transaction` on a new database until you have run `dump database`. 
dump database

- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases can cause database corruption.

- Develop a regular schedule for backing up user databases and their transaction logs.
- Use thresholds to automate backup procedures. To take advantage of Adaptive Server last-chance threshold, create user databases with log segments on a device that is separate from data segments. For more information about thresholds, see the *System Administration Guide*.

**Dumping the system databases**

- The master, model, and sybsystemprocs databases do not have separate segments for their transaction logs. Use `dump transaction` with `truncate_only` to purge the log, then use `dump database` to back up the database.
- Backups of the master database are needed for recovery procedures in case of a failure that affects the master database. See the *System Administration Guide* for step-by-step instructions for backing up and restoring the master database.
- If you are using removable media for backups, the entire master database must fit on a single volume unless you have another Adaptive Server that can respond to volume change messages.

**Specifying dump devices**

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You cannot dump to the null device (on UNIX, `/dev/null`).
- Dumping to multiple stripes is supported for tape and disk devices. Placing multiple dumps on a device is supported only for tape devices.
- You can specify a local dump device as:
  - A logical device name from the `sysdevices` system table
  - An absolute path name
  - A relative path name
Backup Server resolves relative path names using the current working directory in Adaptive Server.

- When dumping across the network, you must specify the absolute path name of the dump device. The path name must be valid on the machine on which Backup Server is running. If the name includes any characters except letters, numbers, or the underscore (_), you must enclose it in quotes.

- Ownership and permissions problems on the dump device may interfere with the use of dump commands. `sp_addumpdevice` adds the device to the system tables, but does not guarantee that you can dump to that device or create a file as a dump device.

- You can run more than one dump (or load) at the same time, as long as each uses different dump devices.

- If the device file already exists, Backup Server overwrites it; it does not truncate it. For example, suppose you dump a database to a device file and the device file becomes 10MB. If the next dump of the database to that device is smaller, the device file is still 10MB.

Determining tape device characteristics

- If you issue a dump command without the `init` qualifier and Backup Server cannot determine the device type, the dump command fails. For more information, see the *System Administration Guide*.

Backup servers

- You must have a Backup Server running on the same machine as Adaptive Server. The Backup Server must be listed in the `master..sys.servers` table. This entry is created during installation or upgrade; do not delete it.

- If your backup devices are located on another machine so that you dump across a network, you must also have a Backup Server installed on the remote machine.

Dump files

- Dumping a database with the `init` option overwrites any existing files on the tape or disk.
If you perform two or more dumps to a tape device and use the same file name for both dumps (specified with the FILENAME parameter), Adaptive Server appends the second dump to the archive device. You will not be able to restore the second dump because Adaptive Server locates the first instance of the dump image with the specified file name and restores this image instead. Adaptive Server does not search for subsequent dump images with the same file name.

Backup Server sends the dump file name to the location specified by the with notify clause. Before storing a backup tape, the operator should label it with the database name, file name, date, and other pertinent information. When loading a tape without an identifying label, use the with headeronly and with listonly options to determine the contents.

File names and archive names

The name of a dump file identifies the database that was dumped and when the dump was made. However, in the syntax, file_name has different meanings depending on whether you are dumping to disk or to a UNIX tape:

\[
file = file\_name
\]

In a dump to disk, the path name of a disk file is also its file name.

In a dump to a UNIX tape, the path name is not the file name. The ANSI Standard Format for File Interchange contains a file name field in the HDR1 label. For tapes conforming to the ANSI specification, this field in the label identifies the file name. The ANSI specification applies these labels only to tape; it does not apply to disk files.

This creates two problems:

- UNIX does not follow the ANSI convention for tape file names. UNIX considers the tape’s data to be unlabeled. Although it can be divided into files, those files have no name.
- In Backup Server, the ANSI tape labels are used to store information about the archive, negating the ANSI meanings. Therefore, disk files also have ANSI labels, because the archive name is stored there.

The meaning of filename changes depending on the kind of dump you are performing. For example, in the following syntax:

\[
dump\ \text{database} \ \text{database\_name} \ \text{to} \ 'filename' \ \text{with} \ \text{file='filename'}
\]

- The first filename refers to the path name you enter to display the file.
The second *filename* is actually the archive name, the name stored in the HDR1 label in the archive, which the user can specify with the *file=* *filename* parameter of the dump or load command.

When the archive name is specified, the server uses that name during a database load to locate the selected archive.

If the archive name is not specified, the server loads the first archive it encounters.

In both cases, *file=* *archivename* establishes the name that is stored in the HDR1 label, and which the subsequent load uses to validate that it is looking at the correct data.

If the archive name is not specified, a dump creates one; a load uses the first name it encounters.

The meaning of *filename* in the to *filename* clause changes according to whether this is a disk or tape dump:

- If the dump is to tape, *filename* is the name of the tape device,
- If the dump is to disk, it is the name of a disk file.

If this is a disk dump and *filename* is not a complete path, it is modified by prepending the server’s current working directory.

- If you are dumping to tape and you do not specify a file name, Backup Server creates a default file name by concatenating the following:
  - Last seven characters of the database name
  - Two-digit year number
  - Three-digit day of the year (1–366)
  - Hexadecimal-encoded time at which the dump file was created

For example, the file *cations980590E100* contains a copy of the publications database made on the 59th day of 1998:
Volume names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.

- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

Note When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing dump volumes

- *On UNIX systems* – Backup Server requests a volume change when the tape capacity has been reached. After mounting another volume, the operator notifies Backup Server by executing `sp_volchanged` on any Adaptive Server that can communicate with Backup Server.

- If Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. The operator responds to these messages with the `sp_volchanged` system procedure.

Appending to or overwriting a volume

- By default (noinit), Backup Server writes successive dumps to the same tape volume, making efficient use of high-capacity tape media. Data is added following the last end-of-tape mark. New dumps can be appended only to the last volume of a multivolume dump. Before writing to the tape, Backup Server verifies that the first file has not yet expired. If the tape contains non-Sybase data, Backup Server rejects it to avoid destroying potentially valuable information.
Use the init option to reinitialize a volume. If you specify init, Backup Server overwrites any existing contents, even if the tape contains non-Sybase data, the first file has not yet expired, or the tape has ANSI access restrictions.

Figure 1-5 illustrates how to dump three databases to a single volume using:

- init to initialize the tape for the first dump
- noinit (the default) to append subsequent dumps
- unload to rewind and unload the tape after the last dump

**Figure 1-5: Dumping several databases to the same volume**

Dumping from a 32-bit OS to a 64-bit OS

Database dumps from a 32-bit version of Adaptive Server are fully compatible with a 64-bit version of Adaptive Server of the same platform, and vice-versa.

Dumping databases whose devices are mirrored

- At the beginning of a dump database, Adaptive Server passes Backup Server the primary device name of all database and log devices. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before the Backup Server completes its data transfer, Adaptive Server aborts the dump.
If a user attempts to unmirror any of the named database devices while a *dump database* is in progress, Adaptive Server displays a message. The user executing the *disk unmirror* command can abort the dump or defer the *disk unmirror* until after the dump is complete.

**Performance Notes**

Due to the design of indexes within a dataserver that provides an optimum search path, index rows are ordered for fast access to the table's data row. Index rows which contain row identifiers (RIDs), are treated as binary to achieve a fast access to the user table.

Within the same architecture platform, the order of index rows remains valid and search order for a selection criteria takes its normal path. However, when index rows are translated across different architectures, this invalidates the order by which optimization was done. This results in an invalid index on user tables when the cross platform dump and load feature is performed.

A database dump from a different architecture, such as big endian to little endian, is loaded, certain indexes are marked as suspect:

- Non-clustered index on APL table.
- Clustered index on DOL table.
- Non-clustered index on DOL table.

To fix indexes on the target system, after load from a different architecture dump, you could use one of two methods:

1. Drop and re-create all of the indexes.

Since the data point and information varies from usage on indexes, the schema, user data, number of indexes, index key length, and number of index rows, in general, it requires planning to recreate indexes on large tables as it can be a lengthy process. `sp_post_xpload` validates indexes, drops invalid indexes, and recreates dropped indexes, in a single command on databases.

Since `sp_post_xpload` performs many operations it can take longer than drop and recreate indexes. Sybase recommends that you use the drop and recreate indexes on those databases larger that 10G.

**Compressed dumps for an archive database**

To use a compressed dump for an archive database, you must:
• Create the compressed dump with the with compression = <compression level> option of the dump database or dump tran command.

• Create a memory pool for accessing the archive database.

**Note** Dumps generated with “compress::” cannot be loaded into an archive database. Therefore, any references to compression in this chapter refer to dumps generated using the with compression = <compression level> option.

**Compatibility issues for a compressed dump** • You cannot load dumps generated with “compress::” into an archive database. There are no compatibility issues with dumps using this compression option on traditional databases.

• The format of a compressed dump generated with the with compression = compression_level option has changed. Backup Server versions 15.0 ESD #2 and later is the component that writes the new compression format. Therefore:
  
  • A compressed dump made using a Backup Server version 15.0 ESD #2 and later can be loaded only into a pre-15.0 ESD #2 installation using a Backup Server version 15.0 ESD #2 or later.
  
  • If you are using a pre-15.0 ESD #2 installation and want to use your dumps for an archive database, use Backup Server version 15.0 ESD #2 or higher to create compressed database dumps.

**Note** A Backup Server version 15.0 ESD #2 and later understands both 15.0 ESD #2 and earlier compression formats; therefore, you can use a 15.0 ESD #2 Backup Server for both dump and loads.

### Encrypted columns and dump database

dump and load work on the ciphertext of encrypted columns. This behavior ensures that the data for encrypted columns remains encrypted while on disk. 
dump and load pertain to the whole database.

If your keys are in a separate database from the columns they encrypt, Sybase recommends that:

• When you dump the database containing encrypted columns, you also dump the database where the key was created. This is necessary if new keys have been added since the last dump.
dump database

- When you dump the database containing an encryption key, dump all databases containing columns encrypted with that key. This keeps the encrypted data in sync with the available keys.

- After loading the database containing the encryption keys and the database containing the encrypted columns, bring both databases on line at the same time.

Because of metadata dependencies of encrypted columns on the key’s database, follow the steps below if you intend to load the key database into a database with a different name (if your data is stored in the same database as your keys, you need not follow these steps):

1. Before dumping the database containing the encrypted columns, use `alter table` to decrypt the data.

2. Dump the databases containing keys and encrypted columns.

3. After loading the databases, use `alter table` to re-encrypt the data with the keys in the newly-named database.

The consistency issues between encryption keys and encrypted columns are similar to those for cross-database referential integrity. See “Cross-database constraints and loading databases” in the System Administration Guide.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Only the System Administrator, the Database Owner, and users with the Operator role can execute `dump database`.

Auditing

Values in `event` and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>dump</td>
<td>dump database</td>
<td></td>
</tr>
</tbody>
</table>

• Roles – Current active roles
• Keywords or options – NULL
• Previous value – NULL
• Current value – NULL
• Other information – NULL
• Proxy information – Original login name, if `set proxy` in effect

See also


Commands `dump transaction`, `load database`, `load transaction`
CHAPTER 1 Commands

System procedures: sp_addthreshold, sp_addumpdevice, sp_dropdevice, sp_dropthreshold, sp_helpdb, sp_helpdevice, sp_helpthreshold, sp_hidetext, sp_logdevice, sp_spaceused, sp_volchanged
dump transaction

Description
Makes a copy of a transaction log and removes the inactive portion.

Syntax
To make a routine log dump:

```
dump transaction database_name
to [compress::[compression_level::]]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [capacity = number_kilobytes,]
   [dumpvolume = volume_name,]
   [file = file_name]
[stripe on [compress::[compression_level::]]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [capacity = number_kilobytes,]
   [dumpvolume = volume_name,]
   [file = file_name]]...
[with {
   density = density_value,]
   [blocksize = number_bytes,]
   [capacity = number_kilobytes,]
   [compression = compress_level,]
   [dumpvolume = volume_name,]
   [file = file_name,]
   [dismount | nodismount],
   [nounload | unload],
   [retaindays = number_days,]
   [noinit | init],
   notify = {client | operator_console},
   standby_access}]
```

To truncate the log without making a backup copy:

```
dump transaction database_name
   with truncate_only
```

To truncate a log that is filled to capacity. Use only as a last resort:

```
dump transaction database_name
   with no_log
```

To back up the log after a database device fails:
dump transaction database_name
to [compress::compression_level::]stripe_device
  [at backup_server_name]
    [density = density_value,
     blocksize = number_bytes,
     capacity = number_kilobytes,
     dumpvolume = volume_name,
     file = file_name]
[stripe on [compress::compression_level::]stripe_device
  [at backup_server_name]
    [density = density_value,
     blocksize = number_bytes,
     capacity = number_kilobytes,
     dumpvolume = volume_name,
     file = file_name]...[
with {
    density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    compression = compress_level
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    retaindays = number_days,
    [noinit | init],
    no_truncate,
    notify = {client | operator_console}}]

Parameters

database_name

is the name of the database from which you are copying data. The name can be given as a literal, a local variable, or a parameter to a stored procedure.
dump transaction

compress::compression_level

is a number between 0 and 9, with 0 indicating no compression, and 9
providing the highest level of compression. If you do not specify
compression_level, the default is 1. For more information about the
compress option, see Chapter 27, “Backing Up and Restoring User
Databases” in the System Administration Guide.

Note: The compression = compression_level option allows you to compress a
dump file on both local and remote machines, and differs from the
compress::compression_level option, which you can use only to compress a
dump file on local machine.

Beginning with Adaptive Server version 15.0, Sybase supports—and
recommends—the native compression = compression_level syntax.

trace_only

removes the inactive part of the log without making a backup copy. Use on
databases without log segments on a separate device from data segments. Do
not specify a dump device or Backup Server name.

no_log

removes the inactive part of the log without making a backup copy and
without recording the procedure in the transaction log. Use no_log only
when you are completely out of log space and cannot run the usual dump
transaction command. Use no_log as a last resort and use it only once after
dump transaction with truncate_only fails. For additional information, see the
System Administration Guide.

to stripe_device

is the device to which data is being dumped. See “Specifying dump devices”
on page 320 for information about what form to use when specifying a dump
device.

at backup_server_name

is the name of the Backup Server. Do not specify this parameter if you are
dumping to the default Backup Server. Specify this parameter only if you are
dumping over the network to a remote Backup Server. You can specify as
many as 32 different remote Backup Servers using this option. When
dumping across the network, specify the network name of a remote Backup
Server running on the machine to which the dump device is attached. For
platforms that use interfaces files, backup_server_name must appear in the
interfaces file.
density = density_value
overrides the default density for a tape device. Valid densities are 800, 1600,
6250, 6666, 10000, and 38000. Not all values are valid for every tape drive;
use the correct density for your tape drive.

blocksize = number_bytes
overrides the default block size for a dump device. The block size must be
at least one database page (2048 bytes for most systems) and must be an
exact multiple of the database page size.

Note Whenever possible, use the default block size; it is the best block size for
your system.

capacity = number_kilobytes
is the maximum amount of data that the device can write to a single tape
volume. The capacity must be at least five database pages, and should be
slightly less than the recommended capacity for your device.

A general rule for calculating capacity is to use 70 percent of the
manufacturer’s maximum capacity for the device, leaving 30 percent for
overhead such as record gaps and tape marks. This rule works in most cases,
but may not work in all cases because of differences in overhead across
vendors and devices.

On UNIX platforms that cannot reliably detect the end-of-tape marker, you
must indicate how many kilobytes can be dumped to the tape. You must
supply a capacity for dump devices specified as a physical path name. If a
dump device is specified as a logical device name, the Backup Server uses the
size parameter stored in the sysdevices system table, unless you specify
a capacity.

compression = compress_level
is a number between 1 and 9, with 9 providing the highest level of
compression. There is no default compression level; if you do not specify
compression = compress_level, Adaptive Server does not compress the
dump.

Note Sybase recommends the native "compression = compress_level" option as
preferred over the older "compress::compression_level" option. The native
option allows compression of both local and remote dumps, and the dumps that
it creates will describe their own compression level during a load. The older
option is retained for compatibility with older applications.
dumpvolume = \textit{volume\_name}  

establishes the name that is assigned to the volume. The maximum length of \textit{volume\_name} is 6 characters. The Backup Server writes the \textit{volume\_name} in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. The load transaction command checks the label and generates an error message if the wrong volume is loaded.

\textbf{stripe on stripe\_device}  

is an additional dump device. You can use up to 32 devices, including the device named in the to \textit{stripe\_device} clause. The Backup Server splits the log into approximately equal portions and sends each portion to a different device. Dumps are made concurrently on all devices, reducing the time and the number of volume changes required. See “Specifying dump devices” on page 320 for information about how to specify a dump device.

\textbf{dismount} | \textbf{nodismount}  

\textit{on platforms that support logical dismount} – determines whether tapes remain mounted. By default, all tapes used for a dump are dismounted when the dump completes. Use \textbf{nodismount} to keep tapes available for additional dumps or loads.

\textbf{noulload} | \textbf{unload}  

determines whether tapes rewind after the dump completes. By default, tapes do not rewind, allowing you to make additional dumps to the same tape volume. Specify \textbf{unload} for the last dump file to be added to a multidump volume. This rewinds and unloads the tape when the dump completes.

\textbf{retaindays = number\_days}  

\textit{on UNIX platforms} – specifies the number of days that Backup Server protects you from overwriting a dump. If you try to overwrite a dump before it expires, Backup Server requests confirmation before overwriting the unexpired volume.

\textbf{Note} This option is meaningful for disk, 1/4-inch cartridge, and single-file media. On multifile media, this option is meaningful for all volumes except the first.

The \textit{number\_days} must be a positive integer or 0, for dumps you can overwrite immediately. If you do not specify a \textbf{retaindays} value, Backup Server uses the server-wide tape retention in days value, set by \texttt{sp\_configure}.
noinit | init
determines whether to append the dump to existing dump files or reinitialize (overwrite) the tape volume. By default, Adaptive Server appends dumps following the last end-of-tape mark, allowing you to dump additional databases to the same volume. New dumps can be appended only to the last volume of a multivolume dump. Use init for the first database you dump to a tape, to overwrite its contents.

Use init when you want Backup Server to store or update tape device characteristics in the tape configuration file. For more information, see the System Administration Guide.

file = file_name
is the name of the dump file. The name cannot exceed 17 characters and must conform to operating system conventions for file names. If you do not specify a file name, Backup Server creates a default file name. For more information, see “Dump files” on page 321.

no_truncate
dumps a transaction log, even if the disk containing the data segments for a database is inaccessible, using a pointer to the transaction log in the master database. The with no_truncate option provides up-to-the-minute log recovery when the transaction log resides on an undamaged device, and the master database and user databases reside on different physical devices.

If you use dump tran with no_truncate you must follow it with dump database, not with another dump tran. If you load a dump generated using the no_truncate option, Adaptive Server prevents you from loading any subsequent dump.

notify = {client | operator_console}
overrides the default message destination.

- On operating systems that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use client to route other Backup Server messages to the terminal session that initiated the dump database.

- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that initiated the dump database. Use operator_console to route messages to the terminal on which the Backup Server is running.
with standby_access
specifies that only completed transactions are to be dumped. The dump
continues to the furthest point it can find at which a transaction has just
completed and there are no other active transactions.

**Examples**

**Example 1** Dumps the transaction log to a tape, appending it to the files on the
tape, since the init option is not specified:

```
dump transaction pubs2
to "/dev/nrmt0"
```

**Example 2** Dumps the transaction log for the mydb database, using the Backup
Server REMOTE_BKP_SERVER. The Backup Server dumps approximately
half the log to each of the two devices. The init option overwrites any existing
files on the tape. The retaindays option specifies that the tapes cannot be
overwritten for 14 days:

```
dump transaction mydb
to "/dev/nrmt4" at REMOTE_BKP_SERVER
stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
with init, retaindays = 14
```

**Example 3** Dumps completed transactions from the inventory_db transaction
log file to device dev1:

```
dump tran inventory_db to dev1 with standby_access
```

**Usage**

- If you use sp_hidetext followed by a cross-platform dump and load, you
  must manually drop and re-create all hidden objects.
- Table 1-23 describes the commands and system procedures used to back up
databases and logs.

<table>
<thead>
<tr>
<th>To do this</th>
<th>Use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make routine dumps of the entire database, including the transaction log.</td>
<td>dump database</td>
</tr>
<tr>
<td>Make routine dumps of the transaction log, then truncate the inactive portion.</td>
<td>dump transaction</td>
</tr>
<tr>
<td>Dump the transaction log after failure of a database device.</td>
<td>dump transaction with no_truncate</td>
</tr>
<tr>
<td>Truncate the log without making a backup. Then copy the entire database.</td>
<td>dump transaction with truncate_only</td>
</tr>
<tr>
<td>Truncate the log after your usual method fails due to insufficient log space. Then copy the entire database.</td>
<td>dump transaction with no_log</td>
</tr>
<tr>
<td>Respond to the Backup Server volume change messages.</td>
<td>sp_volchanged</td>
</tr>
</tbody>
</table>

**Restrictions**

- You cannot dump to the null device (on UNIX, /dev/null).
• You cannot use the `dump transaction` command in a transaction.

• When using 1/4-inch cartridge tape, you can dump only one database or transaction log per tape.

• You cannot run `dump transaction database_name` to before fully dumping a newly created databases.

• You cannot use `dump transaction database_name` to once a non-logged operation has been performed in the database.

• You cannot issue `dump the transaction log` while the trunc log on chkpt database option is enabled or after enabling `select into/bulk copy/plisort` and making minimally logged changes to the database with `select into`, fast bulk copy operations, default unlogged `writetext` operations, or a parallel sort. Use `dump database` instead.

**Warning!** Do not modify the log table `syslogs` with a delete, update, or insert command.

• If a database does not have a log segment on a separate device from data segments, you cannot use `dump transaction` to copy the log and truncate it.

• If a user or threshold procedure issues a `dump transaction` command on a database where a `dump database` or another `dump transaction` is in progress, the second command sleeps until the first completes.

• To restore a database, use `load database` to load the most recent database dump; then use `load transaction` to load each subsequent transaction log dump `in the order in which it was made`.

• Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases can cause database corruption.

• You cannot dump from an 11.x Adaptive Server to a 10.x Backup Server.

• You cannot mix Sybase dumps and non-Sybase data (for example, UNIX archives) on the same tape.

• You cannot dump a transaction with `no_log` or with `truncate_only` if the database has offline pages.
Restrictions on using the \texttt{with no\_truncate} option

Under normal circumstances, Adaptive Server returns an error message when:

- Running \texttt{dump transaction database\_name} to before fully dumping a newly created databases. This causes Adaptive Server to return an error message such as:

  This database has not been dumped since it was created or upgraded or a transaction dump may have been loaded using the UNTIL\_TIME clause. You must perform a DUMP DATABASE before you can dump its transaction log.

- Using \texttt{dump transaction database\_name} to once you have performed a non-logged operation in the database. This causes Adaptive Server to return an error message such as:

  Dump transaction is not allowed because a non-logged operation was performed on the database. Dump your database or use dump transaction with truncate\_only until you can dump your database.

- Using \texttt{dump transaction database\_name} to after you have performed dump transaction with truncate\_only. This causes Adaptive Server to return an error message such as:

  DUMP TRANsaction to a dump device is not allowed where a truncate-only transaction dump has been performed after the last DUMP DATABASE. Use DUMP DATABASE instead.

When you use the with \texttt{no\_truncate} option in your dump transaction \texttt{database\_name to dump\_file} command, however, Adaptive Server does not perform a check of the database and thus does not return any of these error messages. This is because Adaptive Server assumes that your database has some lost data (for example, from a crashed disk) and is therefore inaccessible.

You do, however, get an error message when you then try to load your transaction. Your load transaction process may fail, with the following error message:

  Specified file 'dump device' is out of sequence. Current timestamp is \texttt{<X>} while dump was from \texttt{<Y>}.

Copying the log after device failure

- After device failure, use \texttt{dump transaction with no\_truncate} to copy the log without truncating it. You can use this option only if your log is on a separate segment and your master database is accessible.
The backup created by `dump transaction with no_truncate` is the most recent dump for your log. When restoring the database, load this dump last.

### Dumping databases without separate log segments

- When a database does not have a log segment on a separate device from data segments, use `dump transaction with truncate_only` to remove committed transactions from the log without making a backup copy.

**Warning!** `dump transaction with truncate_only` provides no means to recover your databases. Run `dump database` at the earliest opportunity to ensure recoverability.

- You can also use `with truncate_only` on very small databases that store the transaction log and data on the same device.

- Mission-critical user databases should have log segments on a separate device from data segments. Use the `log` clause of `create database` to create a database with a separate log segment, or use `alter database` and `sp_logdevice` to transfer the log to a separate device.

### Dumping only complete transactions

- Use the `with standby_access` option to dump transaction logs for loading into a server that acts as a warm standby server for the database.

- When you use `with standby_access` to dump the transaction log, the dump proceeds to the furthest point in the log at which all earlier transactions have completed and there are no records belonging to open transactions.

- You must use `dump transaction...with standby_access` in all situations where you load two or more transaction logs in sequence and you want the database to be online between loads.
dump transaction

- After loading a dump made with the with standby_access option, use the online database command with the for standby_access option to make the database accessible.

**Warning!** If a transaction log contains open transactions and you dump it without the with standby_access option, Adaptive Server does not allow you to load the log, bring the database online, then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after a load that was originally dumped with standby_access or after loading the entire series.

Dumping without the log

**Warning!** Use dump transaction with no_log only as a last resort, after your usual method of dumping the transaction log (dump transaction or dump transaction with truncate_only) fails because of insufficient log space. dump transaction with no_log provides no means to recover your databases. Run dump database at the earliest opportunity to ensure recoverability.

- dump transaction...with no_log truncates the log without logging the dump transaction event. Because it copies no data, it requires only the name of the database.
- Every use of dump transaction...with no_log is considered an error and is recorded in the Adaptive Server error log.
- If you have created your databases with log segments on a separate device from data segments, written a last-chance threshold procedure that dumps your transaction log often enough, and allocated enough space to your log and database, you should not have to use with no_log. If you must use with no_log, increase the frequency of your dumps and the amount of log space.

Scheduling dumps

- Transaction log dumps are dynamic—they can take place while the database is active. They may slow the system slightly, so run dumps when the database is not being heavily updated.
- Develop a regular schedule for backing up user databases and their transaction logs.
- dump transaction uses less storage space and takes less time than dump database. Typically, transaction log dumps are made more frequently than database dumps.
Using thresholds to automate *dump transaction*

- Use thresholds to automate backup procedures. To take advantage of the Adaptive Server last-chance threshold, create user databases with log segments on a separate device from data segments.

- When space on the log segment falls below the last-chance threshold, Adaptive Server executes the last-chance threshold procedure. Including a dump transaction command in your last-chance threshold procedure helps protect you from running out of log space. For more information, see sp_thresholdaction.

- You can use `sp_addthreshold` to add a second threshold to monitor log space. For more information about thresholds, see the *System Administration Guide*.

Specifying dump devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.

- You can specify a local dump device as:
  - A logical device name from the `sysdevices` system table
  - An absolute path name
  - A relative path name

  The Backup Server resolves relative path names using the current working directory in Adaptive Server.

- Dumping to multiple stripes is supported for tape and disk devices. Placing multiple dumps on a device is supported only for tape devices.

- When dumping across the network, specify the absolute path name of the dump device. The path name must be valid on the machine on which the Backup Server is running. If the name includes any characters except letters, numbers, or the underscore (_), enclose it in quotes.

- Ownership and permissions problems on the dump device may interfere with use of dump commands. `sp_addumpdevice` adds the device to the system tables, but does not guarantee that you can dump to that device or create a file as a dump device.

- You can run more than one dump (or load) at the same time, as long as they use different dump devices.
Determining tape device characteristics

- If you issue a `dump transaction` command without the `init` qualifier and Backup Server cannot determine the device type, the `dump transaction` command fails. For more information, see the *System Administration Guide*.

Backup servers

- You must have a Backup Server running on the same machine as your Adaptive Server. The Backup Server must be listed in the `master.sysservers` table. This entry is created during installation or upgrade and should not be deleted.

- If your backup devices are located on another machine so that you dump across a network, you must also have a Backup Server installed on the remote machine.

Dump files

- Dumping a log with the `init` option overwrites any existing files on the tape or disk.

- Dump file names identify which database was dumped and when the dump was made. If you do not specify a file name, Backup Server creates a default file name by concatenating the following:
  - Last seven characters of the database name
  - Two-digit year number
  - Three-digit day of the year (1–366)
  - Hexadecimal-encoded time at which the dump file was created

For example, the file `cations930590E100` contains a copy of the `publications` database made on the 59th day of 1993:
The Backup Server sends the dump file name to the location specified by the with notify clause. Before storing a backup tape, the operator should label it with the database name, file name, date, and other pertinent information. When loading a tape without an identifying label, use the with headeronly and with listonly options to determine the contents.

Volume names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.

- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

Note When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing dump volumes

- On UNIX systems – the Backup Server requests a volume change when the tape capacity has been reached. After mounting another volume, the operator notifies the Backup Server by executing the $sp\_volchanged$ system procedure on any Adaptive Server that can communicate with the Backup Server.

- If the Backup Server detects a problem with the currently mounted volume (for example, if the wrong volume is mounted), it requests a volume change by sending messages to either the client or its operator console. The operator responds to these messages with the $sp\_volchanged$ system procedure.
Appending to or overwriting a volume

- By default (noinit), Backup Server writes successive dumps to the same tape volume, making efficient use of high-capacity tape media. Data is added following the last end-of-tape mark. New dumps can be appended only to the last volume of a multivolume dump. Before writing to the tape, Backup Server verifies that the first file has not yet expired. If the tape contains non-Sybase data, Backup Server rejects it to avoid destroying potentially valuable information.

- Use the init option to reinitialize a volume. If you specify init, Backup Server overwrites any existing contents, even if the tape contains non-Sybase data, the first file has not yet expired, or the tape has ANSI access restrictions.

- Figure 1-7 illustrates how to dump three transaction logs to a single volume. Use:
  - init to initialize the tape for the first dump
  - noinit (the default) to append subsequent dumps
  - unload to rewind and unload the tape after the last dump

Figure 1-7: Dumping three transaction logs to a single volume

dump tran mydb to /dev/nrmt4 with init
dump tran yourdb to /dev/nrmt4
dump tran pubs2 to /dev/nrmt4 with unload
Dumping logs stored on mirrored devices

- At the beginning of a dump transaction, Adaptive Server passes the primary device name of each logical log device to the Backup Server. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If the named device fails before Backup Server completes its data transfer, Adaptive Server aborts the dump.

- If you attempt to unmirror a named log device while a dump transaction is in progress, Adaptive Server displays a message. The user executing the disk unmirror command can abort the dump or defer the disk unmirror until after the dump completes.

- dump transaction with truncate_only and dump transaction with no_log do not use the Backup Server. These commands are not affected when a log device is unmirrored, either by a device failure or by a disk unmirror command.

- dump transaction copies only the log segment. It is not affected when a data-only device is unmirrored, either by a device failure or by a disk unmirror command.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Only System Administrators, users who have been granted the Operator role, and the Database Owner can execute dump transaction.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>dump</td>
<td>dump transaction</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
</tbody>
</table>

See also

Commands dump database, load database, load transaction, online database

System procedures sp_addumpdevice, sp_dboption, sp_dropdevice, sp_helpdevice, sp_hidetextsp_logdevice, sp_volchanged
execute

Description

Runs a procedure or dynamically executes Transact-SQL commands.

Syntax

```
[exec[ute]] [@return_status =]
[[[server].[database].[owner].]procedure_name[.number]
[[[@parameter_name =] value ]
   [@parameter_name =] @variable [output]
[, [@parameter_name =] value ]
   [@parameter_name =] @variable [output]...]]
[ [with recompile]]
```

or

```
exec[ute] ("string" | char_variable
[+ "string" | char_variable]...)
```

Parameters

**execute** | **exec**

is used to execute a stored procedure or an extended stored procedure (ESP). This keyword is necessary if there are multiple statements in the batch.

**execute** is also used to execute a string containing Transact-SQL.

**@return_status**

is an optional integer variable that stores the return status of a stored procedure. **@return_status** must be declared in the batch or stored procedure before it is used in an **execute** statement.

**server**

is the name of a remote server. You can execute a procedure on another Adaptive Server as long as you have permission to use that server and to execute the procedure in that database. If you specify a server name, but do not specify a database name, Adaptive Server looks for the procedure in your default database.

**database**

is the database name. Specify the database name if the procedure is in another database. The default value for **database** is the current database. You can execute a procedure in another database as long as you are its owner or have permission to execute it in that database.

**owner**

is the procedure owner’s name. Specify the owner’s name if more than one procedure of that name exists in the database. The default value for **owner** is the current user. The owner name is optional only if the Database Owner owns the procedure or if you own it.

**procedure_name**

is the name of a procedure defined with **create procedure**.
number

is an optional integer used to group procedures of the same name so that they can be dropped together with a single drop procedure statement. Procedures used in the same application are often grouped this way. For example, if the procedures used with an application named orders are named orderproc;1, orderproc;2, and so on, the following statement drops the entire group:

```
drop proc orderproc
```

After procedures have been grouped, individual procedures within the group cannot be dropped. For example, you cannot execute the statement:

```
drop procedure orderproc;2
```

parameter_name

is the name of an argument to the procedure, as defined in create procedure. Parameter names must be preceded by the @ sign.

If the “@parameter_name = value” form is used, parameter names and constants need not be supplied in the order defined in create procedure. However, if this form is used for any parameter, it must be used for all subsequent parameters.

value

is the value of the parameter or argument to the procedure. If you do not use the “@parameter_name = value” form, you must supply parameter values in the order defined in create procedure.

@variable

is the name of a variable used to store a return parameter.

output

indicates that the stored procedure is to return a return parameter. The matching parameter in the stored procedure must also have been created with the keyword output.

The output keyword can be abbreviated to out.
with recompile

forces compilation of a new plan. Use this option if the parameter you are
supplying is atypical or if the data has significantly changed. The changed
plan is used on subsequent executions. Adaptive Server ignores this option
when executing an extended system procedure.

**Note** Using execute procedure with recompile many times can adversely affect
the procedure cache performance. Since a new plan is generated every time you
use with recompile, a useful performance plan may be pushed out of the cache
if there is insufficient space for new plans.

**string**

is a literal string containing part of a Transact-SQL command to execute.
There are no restrictions to the number of characters supplied with the literal
string.

**char_variable**

is the name of a variable that supplies the text of a Transact-SQL command.

### Examples

**Example 1** All three statements execute showind with a parameter value titles:

```
execute showind titles
exec showind @tabname = titles
```

If this is the only statement in a batch or file:

```
showind titles
```

**Example 2** Executes checkcontract on the remote server GATEWAY. Stores
the return status indicating success or failure in @retstat:

```
declare @retstat int
execute @retstat = GATEWAY.pubs.dbo.checkcontract
"409-56-4008"
```

**Example 3** Executes roy_check, passing three parameters. The third
parameter, @pc, is an output parameter. After execution of the procedure, the
return value is available in the variable @percent:

```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent output
select Percent = @percent
```

**Example 4** This procedure displays information about the system tables if you
do not supply a parameter:

```
create procedure
showsysind @table varchar (30) = "sys%"

as

select sysobjects.name, sysindexes.name, indid
from sysindexes, sysobjects
where sysobjects.name like @table
and sysobjects.id = sysindexes.id

Example 5  Executes xp_echo, passing in a value of “Hello World!” The returned value of the extended stored procedure is stored in a variable named result:

declare @input varchar (12), @in varchar (12),
@out varchar (255), @result varchar (255)
select @input="Hello World!"
execute xp_echo @in = @input, @out = @result output

Example 6  The final execute command concatenates string values and character variables to issue the Transact-SQL command:

select name from sysobjects where id=3
declare @tablename char (20)
declare @columname char (20)
select @tablename="sysobjects"
select @columname="name"
execute ('select ' + @columname + ' from ' + @tablename
+ ' where id=3')

Example 7  Executes sp_who:

declare @sproc varchar (255)
select @sproc = "sp_who"
execute @sproc

Usage

• You can use execute with an archive database as long as any statements that reference the archive database are allowed within the archive database. A transaction inside or outside a stored procedure is not permitted with an execute command.

• Procedure results may vary, depending on the database in which they are executed. For example, the user-defined system procedure sp_foo, which executes the db_name() system function, returns the name of the database from which it is executed. When executed from the pubs2 database, it returns the value “pubs2”:

exec pubs2..sp_foo
-------------------------------
pubs2
(1 row affected, return status = 0)
When executed from sybsystemprocs, it returns the value “sybsystemprocs”:

```sql
exec sybsystemprocs..sp_foo

-------------
systemprocs  
(1 row affected, return status = 0)
```

- There are two ways to supply parameters—by position, or by using:
  ```
  @parameter_name = value
  ```
  If you use the second form, you need not supply the parameters in the order defined in `create procedure`.

If you are using the output keyword and intend to use the return parameters in additional statements in your batch or procedure, the value of the parameter must be passed as a variable. For example:

```sql
parameter_name = @variable_name
```

- The dynamic SQL syntax of `exec (@parameter_name)` is also valid; however, it may take more keystrokes. For example, the dynamic SQL command `exec (@sproc ="7")` passes the integer value 7 to the procedure, but this can also be accomplished as `exec @sproc 7`.

- You cannot use text, unitext, and image columns as parameters to stored procedures or as values passed to parameters.

- Executing a procedure specifying output for a parameter that is not defined as a return parameter in `create procedure` causes an error.

- You cannot pass constants to stored procedures using `output`; the return parameter requires a variable name. You must declare the variable’s datatype and assign it a value before executing the procedure. Return parameters cannot have a datatype of text, unitext, and image.

- You need not use the keyword `execute` if the statement is the first one in a batch. A batch is a segment of an input file terminated by the word “go” on a line by itself.

- Since the execution plan for a procedure is stored the first time it is run, subsequent run time is much shorter than for the equivalent set of standalone statements.
Nesting occurs when one stored procedure calls another. The nesting level is incremented when the called procedure begins execution and it is decremented when the called procedure completes execution. The nesting level is also incremented by one when a cached statement is created. Exceeding the maximum of 16 levels of nesting causes the transaction to fail. The current nesting level is stored in the @@nestlevel global variable.

Return values 0 and -1 through -14 are currently used by Adaptive Server to indicate the execution status of stored procedures. Values from -15 through -99 are reserved for future use. See return for a list of values.

Parameters are not part of transactions, so if a parameter is changed in a transaction that is later rolled back, its value does not revert to its previous value. The value that is returned to the caller is always the value at the time the procedure returns.

If you use select * in a stored procedure, the procedure does not pick up any new columns you might have added to the table using alter table, even if you use the with recompile option. To do so, you must drop and re-create the stored procedure, or else an insert based on a select * can cause erroneous results. Even if the newly added column has a default bound to it, the result of the insert is NULL for the newly added column.

When you drop and re-create the stored procedure or reload the database, you see an error message if the column definitions of the target table do not match the select * result.

Commands executed via remote procedure calls cannot be rolled back.

The with recompile option is ignored when Adaptive Server executes an extended stored procedure.

Dynamically executing Transact-SQL

When used with the string or char_variable options, execute concatenates the supplied strings and variables to execute the resulting Transact-SQL command. This form of the execute command may be used in SQL batches, procedures, and triggers.

You cannot supply string and char_variable options to execute the following commands: begin transaction, commit, connect to, declare cursor, rollback, dump transaction, dbcc, set, use, or nested execute commands.

The contents of the string or char_variable options cannot reference local variables declared in the SQL batch or procedure.
- `string` and `char_variable` options can be concatenated to create new tables. Within the same SQL batch or procedure, however, the table created with `execute` is visible only to other `execute` commands. After the SQL batch or procedure has completed, the dynamically-created table is persistent and visible to other commands.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
`execute` permission defaults to the owner of the procedure, who can transfer it to other users.

The permission to execute Transact-SQL commands defined with the `string` or `char_variable` options is checked against the user executing the command, unless the procedure was set up using the execution mode “dynamic ownership chain”. See `sp_procxmode` for more information.

**Auditing**
Values in event and extrainfo columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 38    | exec_procedure | Execution of a procedure | • `Roles` – Current active roles  
• `Keywords or options` – NULL  
• `Previous value` – NULL  
• `Current value` – NULL  
• `Other information` – All input parameters  
• `Proxy information` – Original login name, if `set proxy` in effect |
| 39    | exec_trigger  | Execution of a trigger    | • `Roles` – Current active roles  
• `Keywords or options` – NULL  
• `Previous value` – NULL  
• `Current value` – NULL  
• `Other information` – NULL  
• `Proxy information` – Original login name, if `set proxy` in effect |

**See also**
**Commands** create procedure, drop procedure, return

**System procedures** `sp_addextendedproc`, `sp_depends`, `sp_dropextendedproc`, `sp_helpext`, `sp_procxmode`
**fetch**

Returns a row or a set of rows from a cursor result set.

**Syntax**

```
fetch [next | prior | first | last | absolute
    fetch_offset] [relative fetch_offset]
    [from] cursor_name
    [into fetch_target_list]
```

**Parameters**

- `next | prior | first | last | absolute`
  - are keywords that specify the fetch direction. You do not need to specify the fetch direction for non-scrollable cursors. If you specify the fetch direction, you can use any of the other options to access the rows from a scrollable cursor. You must specify the `fetch_offset` when you use `absolute` or `relative`.

- `[from] cursor_name`
  - is the name of the cursor. `from` is optional.

- `fetch_offset`
  - specifies the offset value from a specific position. `fetch_offset` is required when you specify `absolute` or `relative`. `fetch_offset` can be either signed numeral literal with scale of zero, or Transact-SQL variable with a type of integer or numeric with a zero-scale numeral. See “Rules for positioning the scrollable cursor” on page 357 for more details.

- `fetch_target_list`
  - is a comma-separated list of parameters or local variables into which cursor results are placed. The parameters and variables must be declared prior to the fetch.

**Examples**

- **Example 1** Returns a row of information from the cursor result set defined by the `authors_crsr` cursor:

  ```
  fetch authors_crsr
  ```

- **Example 2** Returns a row of information from the cursor result set defined by the `pubs_crsr` cursor into the variables `@name`, `@city`, and `@state`:

  ```
  fetch pubs_crsr into @name, @city, @state
  ```

- **Example 3** With scrollable cursors, you can use numeric literal offset with orientation keyword `absolute`. In this example, the 25th row is specified. Enter:

  ```
  fetch absolute 25 from pubs_crsr
  into @name, @city, @state
  ```

- **Example 4** To use a Transact-SQL variable representing the 25th row, enter:

  ```
  declare @offset int
  select @offset = 25
  ```
fetch absolute @offset from c1

Usage

- Before you can use fetch, you must declare the cursor and open it.
- You can use fetch with an archive database.
- The cursor_name cannot be a Transact-SQL parameter or local variable.
- For nonscrollable cursors, you cannot fetch a row that has already been fetched. There is no way to backtrack through the result set, but you can close and reopen the cursor to create the cursor result set again and start from the beginning.
- Adaptive Server expects a one-to-one correspondence between the variables in the fetch_target_list and the target list expressions specified by the select statement that defines the cursor. The datatypes of the variables or parameters must be compatible with the datatypes of the columns in the cursor result set.
- When you set chained transaction mode, Adaptive Server implicitly begins a transaction with the fetch statement if no transaction is currently active. However, this situation occurs only when you set the close on endtran option and the cursor remains open after the end of the transaction that initially opened it, since the open statement also automatically begins a transaction.

Cursor position

- For nonscrollable cursors, after you fetch all the rows, the cursor points to the last row of the result set. If you fetch again, Adaptive Server returns a warning through the @@sqlstatus and @@fetch_status global variables, with value that indicate there is no more data, and the cursor position moves beyond the end of the result set. You can no longer update or delete from that current cursor position.
- With fetch into, Adaptive Server does not advance the cursor position when an error occurs because the number of variables in the fetch_target_list does not equal the number of target list expressions specified by the query that defines the cursor. However, it does advance the cursor position, even if a compatibility error occurs between the datatypes of the variables and the datatypes of the columns in the cursor result set.

Determining the number of rows fetched

- You can fetch one or more rows at a time. Use the cursor rows option of the set command to specify the number of rows to fetch.
Getting information about fetches

- The `@@sqlstatus` global variable holds status information (warning exceptions) resulting from the execution of a fetch statement. Its value reflects the last cursor fetched. The value of `@@sqlstatus` is:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates successful completion of the fetch statement.</td>
</tr>
<tr>
<td>1</td>
<td>Indicates that the fetch statement resulted in an error.</td>
</tr>
<tr>
<td>2</td>
<td>Indicates that there is no more data in the result set. This warning can occur if the current cursor position is on the last row in the result set and the client submits a fetch statement for that cursor.</td>
</tr>
</tbody>
</table>

- The `@@fetch_status` global variable provides information about whether fetch is executed successfully in a scrollable cursor. The value of `@@fetch_status` is:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates successful completion of the fetch statement.</td>
</tr>
<tr>
<td>-1</td>
<td>Indicates that the fetch operation failed, or the row fetched was beyond the result set.</td>
</tr>
<tr>
<td>-2</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

- Only a fetch statement can set `@@sqlstatus` and `@@fetch_status`. Other statements have no effect on `@@sqlstatus` or `@@fetch_status`.

The value of `@@rowcount` is affected by whether the specified cursor is forward-only or scrollable. If the cursor is the default, non-scrollable cursor, the value of `@@rowcount` increments one by one, in the forward direction only, until the total number of rows in the result set are fetched.

Once all the rows have been read from the cursor result set, `@@rowcount` represents the total number of rows in the cursor results set. `@@rowcount` after a fetch to get the number of rows read for the cursor specified in that fetch.

If the cursor is scrollable, there is no maximum value for `@@rowcount`. For more information on `@@rowcount`, see Reference Manual: Building Blocks.

Using scrollable cursors

```
fetched_direction:
```

- If not specified, the default value is `next`.
- If not `next`, the cursor must be declared scrollable.
• *fetch_offset* must be an exact, signed numeric, with a scale of zero.

• Positions the cursor beyond the last row or before the first row, no data is returned and no error is raised.

• Is absolute, when *fetch_offset*>0, the offset is calculated from the position before the first row of the result set. If *fetch_offset*<0, the offset is calculated from the position after the last row of the result set.

• Is relative, when *fetch_offset* n>0, the cursor is placed n rows after the current position; if *fetch_offset* n<0, the cursor is placed abs(n) rows before the current position.

The row number specified in the result set is counted from 1; the first row is number 1.

**Multiple rows per fetch**

Each fetch returns one row to the client in default behavior. The returned rows per fetch can be changed to another number by entering:

```
set cursor rows number for cursor_name
```

*number* specifies the number of rows per fetch the cursor can execute. This number can be a numeric literal with no decimal point, or a local variable of type *integer*. If *cursor rows* is greater than one, multiple rows return to the client after fetch. In some cases, the rows returned by fetch may be less than the number of rows specified, depending on the cursor’s position. The current cursor position is always one row.

**Terms used in rules for positioning the scrollable cursor**

These terms are used in “Rules for positioning the scrollable cursor” next.

• *curRowsetStart* – the cursor’s current position.

• *new_CurRowsetStart* – the new current position of the cursor.

• *total_rows* – the total number of rows in the cursor result set.

• *before_first* – the row position before the first row of the cursor result set. This variable has a value of 0.

• *after_last* – the row position after the last row of the cursor result set. This variable has a value of *total_rows* + 1.

• *first_row* – the position at the first row of the cursor result set. This variable has value of 1.

• *last_row* – the position at the last row of the cursor result set. This variable has the same value as *total_rows*.
• *fetchSize* – the number of rows requested for each fetch operation.

Rules for positioning the scrollable cursor

These rules govern the position of the cursor in `fetch_orientation` options when you are fetching the cursor rows, where `cursPos` is the cursor position. See the `fetch_orientation` options syntax:

**Fetch first** The `new_CurRowsetStart` always moves to `first_row`, regardless of the position of `CurRowsetStart` and the value of `fetchSize`.

**Fetch last**

- If `total_rows >= fetchSize`, then
  \[ new_{\text{CurRowsetStart}} = total_{\text{rows}} - \text{fetchSize} + 1. \]
- If `total_rows < fetchSize`, then `new_CurRowsetStart` is on `first_row`.

**Fetch next**

- If `CurRowsetStart` is `before_first`, then `new_CurRowsetStart` is on `first_row`
- Let `curPos = (CurRowsetStart + \text{fetchSize})`,
  - `curPos <= total_rows`, then `new_CurRowsetStart = curPos`
  - `curPos > total_rows`, `new_CurRowsetStart` is `after_last`
- If `CurRowsetStart` is `after_last row`, then `new_CurRowsetStart` remains on `after_last`

**Fetch prior**

- `new_CurRowsetStart` is `before_first` when one of these conditions is true:
  - \((\text{CurRowsetStart} >= 1) \&\& (\text{CurRowsetStart} - \text{fetchSize} <= 0)\)
  - `CurRowsetStart` is `before_first`
- Let `curPos = CurRowsetStart - \text{fetchSize}`; if `1 <= curPos <= \text{total_rows}`, then `new_CurRowsetStart = curPos`.
- If `CurRowsetStart` is `after_last`, let `curPos = total_rows - \text{fetchSize} + 1` `new_CurRowsetStart = curPos` if `curPos > 0`
  \[ new_{\text{CurRowsetStart}} = \text{before_first} \text{ if } curPos <= 0 \]

**Fetch relative**

- If `(\text{CurRowsetStart is before_first}) \&\& (\text{fetch_offset > 0})`, then `new_CurRowsetStart = \text{fetch_offset}`.
- `new_CurRowsetStart` is `before_first` if one of these conditions is true:
  - \((\text{CurRowsetStart is before_first}) \text{ and } (\text{fetch_offset < 0})\)
• (CurRowsetStart is on first_row) and (fetch_offset < 0)

• (CurRowsetStart is after_last)
  and ((CurRowsetStart + fetch_offset + 1) <= 0)

• If (1 < CurRowsetStart <= total_rows),
  let curPos = CurRowsetStart + fetch_offset, then:
  • new_CurRowsetStart is on first_row iff
    (curPos < 1) and abs(fetch_offset) <= fetchSize
  • new_CurRowsetStart is before first_row iff
    (curPos < 1) && (abs(fetch_offset) > fetchSize)
  • new_CurRowsetStart = curPos iff (0 < curPos <= total_rows)
  • new_CurRowsetStart is after_last iff curPos > total_rows

• If (CurRowsetStart is after_last),
  let curPos = CurRowsetStart + fetch_offset +1, then:
  • new_CurRowsetStart = curPos iff 1 <= curPos <= total_rows
  • new_CurRowsetStart is before_first iff curPos <= 0
  • new_CurRowsetStart is after_last iff curPos > total_rows

Fetch absolute

• If fetch_offset = 0, new_CurRowsetStart is before_first

• If fetch_offset > total_rows, new_CurRowsetStart is after_last

• If 0 < fetch_offset <= total_rows, new_CurRowsetStart = fetch_offset

• If (fetch_offset < 0) && (abs (fetch_offset) > total_rows),
  let abs_offset = abs (fetch_offset)
  new_CurRowsetStart is on first_row iff abs_offset > fetchSize
  new_CurRowsetStart is before_first iff abs_offset <= fetchSize

• If (fetch_offset < 0) && (abs (fetch_offset) <= total_rows)
  new_CurRowsetStart = total_rows + fetch_offset + 1

Standards
ANSI SQL – Compliance level: Entry-level compliant.

The fetch of multiple rows is a Transact-SQL extension.

Permissions
fetch permission defaults to all users.

See also
 Commands declare cursor, open, set
**goto label**

**Description**
Branches to a user-defined label.

**Syntax**

```
label:  
goto label
```

**Examples**
Shows the use of a label called `restart`:

```
declare @count smallint
select @count = 1
restart:
    print "yes"
select @count = @count + 1
while @count <= 4
    goto restart
```

**Usage**
- The label name must conform to the rules for identifiers and must be followed by a colon (:) when it is declared. It is not followed by a colon when it is used with `goto`.
- Make the `goto` dependent on an `if` or `while` test, or some other condition, to avoid an endless loop between `goto` and the label.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
goto permission defaults to all users. No permission is required to use it.

**See also**
*Commands* if...else, while
grant

Description
Assigns permissions to individual users, groups of users, and roles. Assigns roles to users or system or user-defined roles.

Syntax
Grants permission to access database objects:

```sql
grant {all [privileges] | permission_list}
on {table_name [(column_list)]
    | view_name[(column_list)]
    | stored_procedure_name
    | function_name
    | keyname}
to {public | name_list | role_list}
[with grant option]
```

Grants permission to use built-in functions:

```sql
grant select
on [builtin] builtin
to {name_list | role_list}
```

Grants permission to execute certain commands:

```sql
grant {all [privileges] | command_list}
to {public | name_list | role_list}
```

Grants access on certain dbcc commands:

```sql
grant dbcc {dbcc_command [on {all | database}]
    [, dbcc_command [on {all | database}], ...]}
to {user_list | role_list}
```

Grants decrypt permission on a table or a list of columns in a table:

```sql
grant decrypt on [ owner. ]tablename[(columnname [{, columnname}])]
to {user | group | role}
```

**Note** grant all on a table or column does not grant decrypt permission.

Grants the default permissions for specific system tables:

```sql
grant default permissions on system tables
```

Grants a role to a user or a role:

```sql
grant {role role_granted [, role_granted ...]}
to grantee [, grantee...]
```

Switches your server user identity to any other server login and limit its use based on the target login roles:

```sql
grant set proxy to role_list
    [restrict role role_list | all | system]
```
Parameters

all

when used to assign permission to access database objects (the first syntax format), all specifies that all permissions except decrypt applicable to the specified object are granted. All object owners can use grant all with an object name to grant permissions on their own objects. You must grant decrypt permissions separately.

Only a System Administrator or the Database Owner can assign permission to create database objects (the third syntax format). When used by a System Administrator, grant all assigns all create permissions (create database, create default, create procedure, create rule, create table, and create view). When the database owner uses grant all, or executes the grant all outside the master database, Adaptive Server grants all create permissions except create database and connect and prints an informational message.

Specifying all does not include permission to execute set proxy or set session authorization, create trigger, and create encryption key. The system security officer must grant these explicitly.

When used to grant set proxy to role_list, restricts the grantee from being granted any new roles when switching identities.

permission_list

is a list of object access permissions granted. If more than one permission is listed, separate them with commas. The following table illustrates the access permissions that can be granted on each type of object:

<table>
<thead>
<tr>
<th>Object</th>
<th>permission_list can include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>select, insert, delete, update, references, update statistics, delete statistics, truncate table, decrypt</td>
</tr>
<tr>
<td>View</td>
<td>select, insert, delete, update, decrypt</td>
</tr>
<tr>
<td>Column</td>
<td>select, update, references, decrypt</td>
</tr>
<tr>
<td></td>
<td>Column names can be specified in either permission_list or column_list.</td>
</tr>
<tr>
<td>Stored procedure</td>
<td>execute</td>
</tr>
<tr>
<td>Encryption key</td>
<td>select</td>
</tr>
</tbody>
</table>

*table_name*

is the name of the table on which you are granting permissions. The table must be in your current database. Only one object can be listed for each grant statement.
Grant:

- **column_list**: a list of columns, separated by commas, to which the permissions apply. If columns are specified, only select, references, and update permissions can be granted.

- **view_name**: is the name of the view on which you are granting permissions. The view must be in your current database. Only one object can be listed for each grant statement.

- **key_name**: is the name of an encryption key on which you are granting access. The table must be in your current database. Only one object can be listed for each grant statement.

- **stored_procedure_name**: is the name of the stored procedure on which you are granting permissions. The stored procedure must be in your current database. Only one object can be listed for each grant statement.

- **public**: is all users. For object access permissions, public excludes the object owner. For object creation permissions or set proxy authorizations, public excludes the Database Owner. You cannot grant permissions with grant option to “public” or to other groups or roles.

- **name_list**: is a list of users’ database names and group names, separated by commas.

- **role_list**: is a list of roles—either system-defined or user-defined—to which you are granting the permission. If grantees do not have the roles in the role_list already granted to them, set proxy to the target login fails if the target login has any roles in the role_list granted.

- **role_list** cannot be a variable.

**Note**: You cannot grant or revoke dbcc commands to public or groups.

**with grant option**: allows the users specified in name_list to grant object access permissions to other users. You can grant permissions with grant option only to individual users, not to “public” or to a group or role.
**builtin**

is a built-in function. Specifying built-in functions allows you to
differentiate between a table and a grantable built-in function with the same
name. The functions are set_appcontext, get_appcontext, list_appcontext, and
rm_appcontext.

**command_list**

is a list of commands that the user can execute. If more than one command
is listed, separate them with commas. The command list can include create
database, create default, create procedure, create rule, create table, create
view, set proxy, set session authorization, create encryption key, and connect.

create database permission can be granted only by a system administrator,
and only from within the master database. Only a system security officer can
grant permission to execute create encryption key and create trigger.

Only a system security officer can grant users permission to execute set
proxy or set session authorization. Granting permission to execute set proxy
or set session authorization allows the grantee to impersonate another login
in the server. set proxy and set session authorization are identical, except that
set session authorization follows the ANSI92 standard, and set proxy is a
Transact-SQL extension.

**dbcc_command**

is the name of the dbcc command you are granting. It cannot be a variable.
Table 1-25 on page 381 lists the valid grant dbcc commands.

**database**

is the name of the database on which you are granting permissions. It is used
with database-specific dbcc commands to grant permission only on the
target database. The grantee must be a valid user in the target database.
*database* conforms to the rules for identifiers and cannot be a variable.

If there are multiple granted actions in the same command, *database* must
be unique.

See “on all | database parameter and server-level commands” on page 383
for more information.

**role**

grants a role to a user or to a system or user-defined role.

**role_granted**

is the name of a system or user-defined role that the System Security Officer
is granting to a user or a role.
grant

grantee

is the name of a system role, user-defined role, or a user, to whom you are
granting a role.

role_list

is a list of system-defined or user-defined roles to which you are granting the
permission.

default permissions on system tables

specifies that you grant the default permissions for the system tables listed
in “granting default permissions on system tables” on page 383.

system

ensures that the grantee has the same set of system roles as the target login.

Examples

Example 1 Grants Mary and the “sales” group permission to use the insert and
delete commands on the titles table:

grant insert, delete
on titles
to mary, sales

Example 2 Grants select permission on the get_appcontext function to
“public” (which includes all users):

grant select on builtin get_appcontext to public

Compare this to the following, which grants select permission on a table called
get_appcontext, if a table with that name exists:

grant select on get_appcontext to public

Specifically including the builtin argument in your grant statement ensures that
you do not mistakenly select a table that has the same name as a function—in
this example, the get_appcontext function versus a table called get_appcontext.

Example 3 Two ways to grant update permission on the price and advance
columns of the titles table to “public” (which includes all users):

grant update
on titles (price, advance)
to public

or:

grant update (price, advance)
on titles
to public

Example 4 Grants Harry and Billy permission to execute either set proxy or
set session authorization to impersonate another user in the server:
grant set proxy to harry, billy

**Example 5** Grants users with sso_role permission to execute either set proxy or set session authorization to impersonate another user in the server:

grant set session authorization to sso_role

**Example 6** Grants users with vip_role the ability to impersonate another user in the server. vip_role must be a role defined by a System Security Officer with the `create role` command:

grant set proxy to vip_role

**Example 7** Grants Mary and John permission to use the create database and create table commands. Because create database permission is being granted, this command can be executed only by a System Administrator within the master database. Mary and John’s create table permission applies only to the master database:

grant create database, create table to mary, john

**Example 8** Grants complete access permissions, except decrypt permission, on the titles table to all users:

grant all on titles to public

**Example 9** Grants all object creation permissions, except create encryption key, in the current database to all users. If this command is executed by a System Administrator from the master database, it includes create database permission:

grant all to public

**Example 10** Gives Mary permission to use the update command on the authors table and to grant that permission to others:

grant update on authors to mary with grant option

**Example 11** Gives Bob permission to use the select and update commands on the price column of the titles table and to grant that permission to others:

grant select, update on titles (price) to bob with grant option
Example 12 Grants permission to execute the new_sproc stored procedure to all System Security Officers:

```sql
grant execute on new_sproc
to sso_role
```

Example 13 Grants James permission to create a referential integrity constraint on another table that refers to the price column of the titles table:

```sql
grant references on titles (price)
to james
```

Note Before you create a table that includes a referential integrity constraint to reference another user’s table, you must be granted references permission on that referenced table. The table must also include a unique constraint or unique index on the referenced columns. See create table for more information about referential integrity constraints.

Example 14 Grants the database owner permission to specify column encryption using the ssn_key, when executed by the key owner. The database owner requires select permission on ssn_key to reference it on create table, alter table or select into:

```sql
grant select on ssn_key to dbo
```

Example 15 Grants decrypt permission on all encrypted columns in the customer table:

```sql
grant decrypt on customer to accounts_role
```

Example 16 Creates key k1, which uses non-default values for the initialization vector and random pad. The employee salary column is padded with random data before encryption:

```sql
create encryption key k1 init_vector null pad random
create table employee (eid int, esalary money encrypt with k1, ...)
```

Example 17 Grants the role “specialist”, with all its permissions and privileges, to the role “doctor”:

```sql
grant role specialist_role to doctor_role
```

Example 18 Grants the role “doctor” to Mary:

```sql
grant role doctor_role to mary
```

Example 19 On a user database called pubs2 owned by Jane, only Jane or the System Administrator can execute the dbcc checkdb command. Others encounter the following error:

```sql
grant
```
CHAPTER 1  Commands

1> dbcc checkdb (pubs2)
2> go

Msg 10302, Level 14, State 1:
Line 1:
Only the DBO of database 'test' or a user with System
Administrator (SA) role can run this command. DBCC
execution completed. If DBCC printed error messages,
contact a user with System Administrator (SA) role.

Example 20 If Walter needs to be a maintenance user for pubs2 but the System
Administrator does not want to grant him administrator-level privileges
elsewhere, the System Administrator executes the following:

1> use pubs2
2> go
1> grant dbcc checkdb on pubs2 to walter
2> go

Note The System Administrator must be in the target database—in this case
pubs2—and Walter must be a valid user in this target database.

Example 21 Walter (from the previous example) can now execute the dbcc
checkdb command on the customers database without encountering an error:

%isql -Uwalter -Pwalterpassword -SSERVER
1> use pubs2
2> go
1> dbcc checkdb (pubs2)
2> go

Checking sysobjects: Logical pagesize is 2048 bytes
The total number of data pages in this table is 2.
Table has 27 data rows.
...
Table has 1 data rows.
DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator (SA) role.

Example 22 Grants the use of dbcc to a role instead of a user. This lets System
Administrators assign the ability to execute dbcc to individual users based on
their role:

1> use master
2> go
1> create role checkdb_role
2> go
grant

1> use pubs2
2> go
1> grant dbcc checkdb on pubs2 to checkdb_role
2> go

Next, the System Administrator grants the role to Joe:

1> sp_addlogin joe, joepassword
2> go
Password correctly set.
Account unlocked.
New login created.
(return status = 0)
1> use pubs2
2> sp_adduser joe
3> go
1> grant role checkdb_role to joe
2> go

Joe can now execute the dbcc checkdb command on the pubs2 database when activating checkdb_role. Joe must be a valid user in pubs2:

% isql -Ujoe -Pjoepassword -SSERVER
1> use pubs2
2> go
1> dbcc checkdb (pubs2)
2> go

Msg 10302, Level 14, State 1:
Line 1:
Only the DBO of database 'pubs2' or a user with System Administrator (SA) role can run this command. DBCC execution completed. If DBCC printed error messages, contact a user with System Administrator (SA) role.
1> set role checkdb_role on
2> go
1> dbcc checkdb (pubs2)
2> go

Checking sysobjects: Logical pagesize is 2048 bytes
The total number of data pages in this table is 2.
...
The total number of data pages in this table is 1.
Table has 1 data rows. DBCC execution completed. If DBCC printed error messages, contact a user with System Administrator (SA) role.
Example 23  Through the use of a role, the System Administrator allows Carlos to run `dbcc checkalloc` on any database where he is a valid user, or where a database allows a “guest” user.

**Note**  You do not need to add Carlos as an actual user in the `master` database if the user “guest” already exists in `master`.

```
1> use master
2> go
1> create role checkalloc_role
2> go
1> grant dbcc checkalloc on all to checkalloc_role
2> go
1> sp_addlogin carlos, carlospassword
2> go
1> grant role checkalloc_role to carlos
2> go
```

Example 24  Gives Frank, a valid user in the `master` database, the ability to execute `dbcc checkdb` for all databases in the server:

```
1> use master
2> go
1> sp_addlogin frank, frankpassword
2> go

Password correctly set.
Account unlocked.
New login created.
(return status = 0)
1> sp_adduser frank
2> go

New user added.
(return status = 0)
1> grant dbcc checkdb on all to frank
2> go
```

Now Frank can execute the `dbcc checkdb` command on each database in the server where he is a valid user:

```
% isql -Ufrank -Pfrankpassword -SSERVER
1> dbcc checkdb (tempdb)
2> go
Checking tempdb: Logical pagesize is 2048 bytes
Checking sysobjects: Logical pagesize is 2048 bytes
```
The total number of data pages in this table is 1. DBCC execution completed. If DBCC printed error messages, contact a user with System Administrator (SA) role.

Note You cannot grant or revoke dbcc commands to public or groups.

Example 25 Grants Alex permission to use the dbcc tune command on pubs2. This example returns an error because you cannot grant server-level dbcc commands at the database level:

```sql
grant dbcc tune on pubs2 to alex
```

Msg 4626, Level 16, State 1:
Line 1:
DBCC command 'tune' cannot be assigned at database-level.

Example 26 Grants dbcc tune on the master database to Alex. This returns an error because even if the current database is master, you cannot grant a server-level command at the database level. The on database parameter shows the intention to restrict the access to the current database scope, and this is not possible for server-level commands:

```sql
grant dbcc tune on master to alex
```

Msg 4626, Level 16, State 1:
Line 1:
DBCC command 'tune' cannot be assigned at the database-level.

Example 27 Grants dbcc tune to Alex. This returns an error because server-level commands require that master be the current database:

```sql
use pubs2
grant dbcc tune to alex
```

Msg 4627, Level 16, State 1:
Line 1:
The user must be in the master database to GRANT/REVOKE this command.

Example 28 Grants dbcc checkalloc on the pubs2 database to “nonuser.” This returns an error because a user must be a valid user in the database to be granted database-level access:

```sql
grant dbcc checkalloc on pubs2 to nonuser
```

Msg 11105, Level 11, State 1:
Line 1:
No such user/role 'nonuser' exists.

**Example 29** Grants dbcc tune on all to Alex:

```
grant dbcc tune on all to alex
```

The `on all` parameter is ignored because server-wide commands are always granted in the master database, and any access granted in the master database is granted for any database by default. Although the `on all` clause is not designed for server-wide commands, its use does not cause any errors because it is a default behavior.

**Example 30** Grants dbcc checkalloc on all and dbcc checkdb on pubs2 to Alex. Although several commands can be granted under the same statement, they must all affect the same database, so you must be in master if one of them is on all:

```
grant dbcc checkalloc on all,
    dbcc checkdb on pubs2 to alex
```

Msg 4627, Level 16, State 1:
Line 1:
The user must be in the master database in order to grant/revoke server-wide DBCC access.

**Example 31** Erroneously applies grant dbcc and revoke dbcc to groups or public:

```
1> grant dbcc tablealloc on pubs2 to public
Msg 4629, Level 16, State 1:
Line 1:
GRANT/REVOKE DBCC does not apply to groups or PUBLIC.
1> sp_addgroup gr
New group added.
(return status = 0)
1> grant dbcc tablealloc on pubs2 to gr
Msg 4629, Level 16, State 1:
Line 1:
GRANT/REVOKE DBCC does not apply to groups or PUBLIC.
```

**Example 32** Granting a database-level command at the database level has no effect if a server-wide permission exists:

```
1> grant dbcc checkalloc on all to alex
1> use pubs2
1> grant dbcc checkalloc on pubs2, dbcc tablealloc on pubs2 to alex
```
Example 33 Only the System Administrator can grant the privilege:

```
set role sa_role off
grant dbcc tablealloc on all to alex
```

Msg 10353, Level 14, State 1:
Line 1:
You must have the following roles to execute this command/procedure: 'sa_role'. Please contact a user with the appropriate role for help.

Example 34 Granting a dbcc traceon results in an error message because dbcc traceon is not a grantable command:

```
grant dbcc traceon to joe
go
```

Msg 4607, Level 16, State 2:
Line 12:
Privilege DBCC traceon may not be GRANTed or REVOKEd.

Example 35 The col_name function displays only the dbcc commands that can be granted, and returns the string dbcc internal for all the dbcc commands that cannot be granted.

```
1> declare @a int
2> select @a=1
3> while (@a<200)
4> begin
5> insert #t values (@a, col_name (-317, @a))
6> select @a=@a+1
7> end
8> select dbcc_id=a, dbcc_command=b from #t where b!="dbcc internal"
```

<table>
<thead>
<tr>
<th>dbcc_id</th>
<th>dbcc_command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dbcc catalogcheck</td>
</tr>
<tr>
<td>2</td>
<td>dbcc checktable</td>
</tr>
<tr>
<td>3</td>
<td>dbcc checkalloc</td>
</tr>
<tr>
<td>4</td>
<td>dbcc checkdb</td>
</tr>
<tr>
<td>5</td>
<td>dbcc checkindex</td>
</tr>
<tr>
<td>6</td>
<td>dbcc reindex</td>
</tr>
</tbody>
</table>
Example 36 You cannot use the grant dbcc command using the grant option:

```
grant dbcc tune to alex with grant option
Msg 156, Level 15, State 1:
Line 1:
Incorrect syntax near the keyword 'with'.
```

Example 37 Allows user “harry” to use truncate table and updates statistics on the authors table:

```
grant truncate table on authors to harry
grant update statistics on authors to harry
```

Example 38 Allows user “billy” to use the delete statistics command on the authors table:

```
grant delete statistics on authors to billy
```

Example 39 Grants truncate table, update, and delete statistics privileges to all users with the oper_role (if users “billy” and “harry” possess the oper_role, they can now execute these commands on authors):

```
grant truncate table on authors to oper_role
grant update statistics on authors to oper_role
grant delete statistics on authors to oper_role
```

Example 40 Implicitly grants permissions for truncate table, delete statistics, and update statistics through a stored procedure. For example, assuming “billy” owns the authors table, he can execute the following to grant “harry” privileges to run truncate table and update statistics on authors:

```
create procedure sproc1
as
truncate table authors
update statistics authors
go
grant execute on sproc1 to harry
go
```

You can also implicitly grant permissions at the column level for update statistics and delete statistics through stored procedures.
Example 41 Grants set proxy to user “joe” but restricts him from switching identities to any user with the sa, sso, or admin roles (however, if he already has these roles, he can set proxy for any user with these roles):

```
grant set proxy to joe
restrict role sa_role, sso_role, admin_role
```

When “joe” tries to switch his identity to a user with admin_role (in this example, Our_admin_role), the command fails unless he already has admin_role:

```
set proxy Our_admin_role
```

Msg 10368, Level 14, State 1:
Server 's', Line 2: Set session authorization permission denied because the target login has a role that you do not have and you have been restricted from using.

After “joe” is granted the admin_role and retries the command, it succeeds:

```
grant role admin_role to joe
set proxy Our_admin_role
```

Example 42 Restricts “joe” from being granted any new roles when switching identities:

```
grant set proxy to joe
restrict role all
```

“joe” can set proxy only to those users who have the same (or lesser) roles than he has.

Example 43 Restricts Joe from acquiring any new system roles when using set proxy:

```
grant set proxy to joe
restrict role system
```

set proxy fails if the target login has system roles that Joe lacks.

Usage

- grant dbcc issues the following warning when you execute it while set fipsflagger option is enabled:

```
SQL statement on line number 1 contains Non-ANSI text. The error is caused due to the use of DBCC.
```

- You can substitute the word from for to in the grant syntax.
Table 1-24 summarizes default permissions on Transact-SQL commands in Adaptive Server. The user listed under the “Defaults to” heading is the lowest level of user that is automatically granted permission to execute a command. This user can grant or revoke the permission if it is transferable. Users at higher levels than the default are either automatically assigned permission or (in the case of Database Owners) can get permission by using the setuser command.

For example, the owner of a database does not automatically receive permission on objects owned by other users. A Database Owner can gain such permission by assuming the identity of the object owner with the setuser command, and then issuing the appropriate grant or revoke statement. System Administrators have permission to access all commands and objects at any time.

The Adaptive Server installation script assigns a set of permissions to the default group “public.” grant and revoke statements need not be written for these permissions.

Table 1-24 does not include the System Security Officer, who does not have any special permissions on commands and objects, but only on certain system procedures.

Table 1-24: Command and object permissions

<table>
<thead>
<tr>
<th>Statement</th>
<th>Defaults to</th>
<th>Can be granted/revoked</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System Admin Operator</td>
<td>Database Owner</td>
</tr>
<tr>
<td>alter database</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>alter role</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>alter table</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>begin transaction</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>checkpoint</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>compute clause</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>connect to</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create database</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create default</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create encryption key</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create index</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- alter database: Defaults to the system security officer and the key custodian. The system security officer can grant this permission to other users.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Defaults to</th>
<th>Can be granted/revoked</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System Admin</td>
<td>Operator</td>
</tr>
<tr>
<td>create procedure</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create role</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create table</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create trigger</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>create view</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dbcc</td>
<td>Varies depending upon options. See dbcc in this manual.</td>
<td>X</td>
</tr>
<tr>
<td>delete</td>
<td>X (3)</td>
<td></td>
</tr>
<tr>
<td>delete statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decrypt permission</td>
<td>Defaults to the object owner and the system security officer. The object owner and the system security officer can grant decrypt permission to other users.</td>
<td>X</td>
</tr>
<tr>
<td>disk init</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disk mirror</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disk refit</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disk reinit</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disk remirror</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>disk unmirror</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>drop any object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dump database</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>dump transaction</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>execute</td>
<td>X (4)</td>
<td></td>
</tr>
<tr>
<td>grant on object</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>grant command</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>X (3)</td>
<td></td>
</tr>
<tr>
<td>kill</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>load database</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>load transaction</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>print</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raiserror</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>readtext</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>revoke on object</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>revoke command</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>rollback</td>
<td></td>
<td></td>
</tr>
<tr>
<td>save transaction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If the object to which you are granting permission is an encryption key, select permission defaults to the system security officer and the key owner.

You can grant permissions only on objects in your current database.

grant and revoke commands are order-sensitive. The command that takes effect when there is a conflict is the one issued most recently.

A user can be granted permission on a view or stored procedure even if he or she has no permissions on objects referenced by the procedure or view. For more information, see Chapter 16, “Managing User Permissions,” in the System Administration Guide.

Adaptive Server grants all users permission to declare cursors, regardless of the permissions defined for the base tables or views referenced in the declare cursor statement. Cursors are not defined as Adaptive Server objects (such as tables), so no permissions can be applied against a cursor. When a user opens a cursor, Adaptive Server determines whether the user has select permissions on the objects that define that cursor’s result set. It checks permissions each time a cursor is opened.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Defaults to</th>
<th>System Admin</th>
<th>Operator</th>
<th>Database Owner</th>
<th>Object owner</th>
<th>Public</th>
<th>Can be granted/revoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>X (3)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setuser</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shutdown</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>truncate table</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>X (3)</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update all statistics</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update index statistics</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update partition statistics</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update statistics</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writetext</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(6)</td>
</tr>
</tbody>
</table>

1. Transferred with database ownership
2. Public can create temporary tables, no permission required
3. If a view, permission defaults to view owner
4. Defaults to stored procedure owner
5. Transferred with select permission
6. Transferred with update permission

“No” means use of the command is never restricted “N/A” means use of the command is always restricted
If the user has permission to access the objects defined by the cursor, Adaptive Server opens the cursor and allows the user to fetch row data through the cursor. Adaptive Server does not apply permission checking for each fetch. However, if the user performs a delete or an update through that cursor, the regular permission checking applies for deleting and updating the data of objects referenced in the cursor result set.

- A grant statement adds one row to the sysprotects system table for each user, group, or role that receives the permission. If you subsequently revoke the permission from the user or group, Adaptive Server removes the row from sysprotects. If you revoke the permission from selected group members only, but not from the entire group to which it was granted, Adaptive Server retains the original row and adds a new row for the revoke.

- If a user inherits a particular permission by virtue of being a member of a group, and the same permission is explicitly granted to the user, no row is added to sysprotects. For example, if “public” has been granted select permission on the phone column in the authors table, then John, a member of “public,” is granted select permission on all columns of authors. The row added to sysprotects as a result of the grant to John contains references to all columns in the authors table except for the phone column, on which he already had permission.

- Permission to issue the create trigger command is granted to users by default. When you revoke permission for a user to create triggers, a revoke row is added in the sysprotects table for that user. To grant permission to that user to issue create trigger, you must issue two grant commands. The first command removes the revoke row from sysprotects; the second inserts a grant row. If you revoke permission to create triggers, the user cannot create triggers even on tables that the user owns. Revoking permission to create triggers from a user affects only the database where the revoke command was issued.

- You can display information about permissions with these system procedures:
  - sp_helpprotect reports permissions information for database objects, users, groups, and roles.
  - sp_column_privileges reports permissions information for one or more columns in a table or view.
  - sp_table_privileges reports permissions information for all columns in a table or view.
• sp_activeroles displays all active roles—and all roles contained by those roles—for the current login session of Adaptive Server.

• sp_displayroles displays all roles granted to another role or user, or displays the entire hierarchy tree of roles in table format.

• You can view permissions using sp_helprotect:

  1> use pubs2
  2> go
  1> sp_helprotect
  2> go

<table>
<thead>
<tr>
<th>grantor</th>
<th>grantee</th>
<th>type</th>
<th>action</th>
<th>object</th>
<th>column</th>
<th>grantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo</td>
<td>public</td>
<td>Grant</td>
<td>Select</td>
<td>sysalternates</td>
<td>All</td>
<td>FALSE</td>
</tr>
<tr>
<td>1...</td>
<td>1...</td>
<td>1...</td>
<td>1...</td>
<td>1...</td>
<td>1...</td>
<td>1...</td>
</tr>
<tr>
<td>dbo</td>
<td>Walter</td>
<td>Grant</td>
<td>DBCC</td>
<td>DBCC</td>
<td>dbcc</td>
<td>checkdb</td>
</tr>
</tbody>
</table>

(1 row affected)
(return status = 0)

• You cannot use the grant with grant option with grant dbcc.

grant all object creation permissions

• When used with only user or group names (no object names), grant all assigns these permissions: create database, create default, create procedure, create rule, create table, and create view. create database permission can be granted only by a System Administrator and only from within the master database.

• Only the Database Owner and a System Administrator can use the grant all syntax without an object name to grant create command permissions to users or groups. When the grant all command is used by the Database Owner, an informational message is printed, stating that only a System Administrator can grant create database permission. All other permissions noted above are granted.

• All object owners can use grant all with an object name to grant permissions on their own objects. When used with a table or view name plus user or group names, grant all enables delete, delete statistics, insert, select, truncate table, update, and update statistics permissions on the table.

grant with grant option rules

• You cannot grant permissions with grant option to “public” or to a group or role.
In granting permissions, a System Administrator is treated as the object owner. If a System Administrator grants permission on another user’s object, the owner’s name appears as the grantor in sysprotects and in sp_helprotect output.

You cannot grant create encryption key permission with the grant option parameter.

Information for each grant is kept in the system table sysprotects with the following exceptions:

- Adaptive Server displays an informational message if a specific permission is granted to a user more than once by the same grantor. Only the first grant is kept.
- If two grants are exactly same except that one of them is granted with grant option, the grant with grant option is kept.
- If two grant statements grant the same permissions on a particular table to a specific user, but the columns specified in the grants are different, Adaptive Server treats the grants as if they were one statement. For example, the following grant statements are equivalent:

  ```sql
  grant select on titles (price, contract) to keiko
  grant select on titles (advance) to keiko
  grant select on titles (price, contract, advance) to keiko
  ```

Granting permission to roles

- You can use the grant command to grant permissions to all users who have been granted a specified role. The role can be either a system role, like sso_role or sa_role, or a user-defined role. For a user-defined role, the System Security Officer must create the role with a create role command.

However, grant execute permission does not prevent users who do not have a specified role from being individually granted permission to execute a stored procedure. If you want to ensure, for example, that only System Security Officers can ever be granted permission to execute a stored procedure, use the proc_role system function within the stored procedure itself. It checks to see whether the invoking user has the correct role to execute the procedure. For more information, see proc_role.
Permissions that are granted to roles override permissions that are granted to users or groups. For example, say John has been granted the System Security Officer role, and $sso\_role$ has been granted permission on the sales table. If John’s individual permission on sales is revoked, he can still access sales because his role permissions override his individual permissions.

Users and user groups

- User groups allow you to grant or revoke permissions to more than one user with a single statement. Each user can be a member of one other group and is always a member of “public.”

- The Database Owner or System Administrator can add new users with sp_adduser and create groups with sp_addgroup. To allow users with logins on Adaptive Server to use the database with limited privileges, you can add a “guest” user with sp_adduser and assign limited permissions to “guest.” All users with logins can access the database as “guest.”

- To remove a user, use sp_dropuser. To remove a group, use sp_dropgroup.

To add a new user to a group other than “public,” use sp_adduser. To change an established user’s group, use sp_changegroup.

To display the members of a group, use sp_helpgroup.

- When sp_changegroup is executed to change group membership, it clears the in-memory protection cache by executing:

  grant all to null

so that the cache can be refreshed with updated information from the sysprotects table. To modify sysprotects directly, contact Sybase Technical Support.

grant dbcc command options

Table 1-25 lists the valid grant dbcc commands.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkalloc</td>
<td>Checks the specified database to make sure all of its pages are correctly allocated, and that there are no unused allocated pages.</td>
</tr>
<tr>
<td>checkcatalog</td>
<td>Checks for consistency in and between system tables.</td>
</tr>
<tr>
<td>checkdb</td>
<td>Runs the same checks as checktable, but on each table in the specified database, including syslogs.</td>
</tr>
</tbody>
</table>
All of the options in Table 1-25 are database-level commands except for `tune`, which is a server-level command.

See Chapter 25, “Checking Database Consistency” in the *System Administration Guide* for more information on these `dbcc` commands.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `checkindex` | Checks the specified index to make sure that:  
  - Index and data pages are correctly linked.  
  - Indexes are correctly sorted.  
  - All pointers are consistent.  
  - Data information on each page is reasonable.  
  - Page offsets are reasonable. |
| `checkstorage` | Checks the specified database for:  
  - Allocation  
  - OAM page entries  
  - Page consistency  
  - Text-valued columns  
  - Allocation of text-valued columns  
  - Text-column chains |
| `checktable` | Checks the specified table to make sure that:  
  - Index and data pages are correctly linked.  
  - Indexes are correctly sorted.  
  - All pointers are consistent.  
  - Data information on each page is reasonable.  
  - Page offsets are reasonable. |
| `checkverify` | Verifies the results of the most recent run of `dbcc checkstorage` for the specified database. |
| `fix_text` | Upgrades text values after any Adaptive Server character set is converted to a new multibyte character set. |
| `indexalloc` | Checks the specified index to make sure all pages are correctly allocated, and that there are no unused allocated pages. |
| `reindex` | Checks the integrity of indexes on user tables by running a fast version of `dbcc checktable`. |
| `tablealloc` | Checks the specified table to make sure that all pages are correctly allocated, and that there are no unused allocated pages. |
| `textalloc` | Checks for a violation of the format of the root page of a `text`, `unitext`, or `image` index. |
| `tune` | Enables or disables tuning flags for special performance situations. |
on all | database parameter and server-level commands

The on database parameter specifies the database on which to invoke the database-level grant dbcc command. Because on master grants the ability to use dbcc commands on all databases, on master is the same as on all. You must be in the master database to use either the on all or on master parameters.

Neither the on database nor on all parameters work when invoking a server-level grant dbcc command such as dbcc tune, because by doing so, you are forcing a server-level command to restrict itself to individual databases. For this reason, using the server-level grant dbcc tune on master command raises an error.

on all and guest

Before you grant dbcc permission for a database to a user, that user must first be a valid user in the database, and cannot be a “guest” user. However, if you grant dbcc through roles, the users can then execute that dbcc command in any database where they are a valid user, including the user “guest.”

granting default permissions on system tables

Table 1-26 lists the system tables you can grant and revoke the default permissions for when you issue the command from any database.

| Table 1-26: System tables with grantable permissions issued from any database |
|---------------------------------|----------------|----------------|----------------|----------------|
| sysalternates                   | sysjars        | sysqueryplans  | sysusermessages|
| sysattributes                   | syskeys        | sysreferences  | sysusers       |
| syscolumns                      | syslogs        | sysroles       | sysroles       |
| syscomments                     | sysobjects     | syssegments    | sysxtypes      |
| syscolumns                      | sysalternates  | sysobjects     | sysxtypes      |
| sysconstraints                  | syspartitions  | sysstatistics  |                |
| sysdepends                      | sysprocedures  | systabstats    |                |
| sysindexes                      | sysprotects    | sysreferences  |                |
|                                 |                | systhresholds  |                |

The command also makes the following changes:

- Revokes syscolumns (encrkyid) and syscolumns (encrkydb) permissions from public.
- Revokes syscolumns (encrkydb) and syscolumns (encrkyid) permissions from public.
- Revokes sysobjects(audflags) permissions from public
- Grants permissions for sysobjects to sso_role
- Revokes select on all columns of sysencryptkeys from public
Grants select on all sysencryptkeys columns to sso_role

Grants permissions for syscolumns to sso_role

Table 1-27 lists the system tables you can grant and revoke the default permissions for when you issue the command from the master database.

**Table 1-27: System tables with grantable permissions issued from the master database**

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysdatabases</td>
<td>sysusages</td>
<td>sysresourcelimits</td>
</tr>
<tr>
<td>sysdevices</td>
<td>sysconfigures</td>
<td>syslogins</td>
</tr>
<tr>
<td>syslocks</td>
<td>syscurconfig</td>
<td>sysremotelogins</td>
</tr>
<tr>
<td>sysmessages</td>
<td>syslanguages</td>
<td>syssessions</td>
</tr>
<tr>
<td>sysprocesses</td>
<td>syscharsets</td>
<td></td>
</tr>
<tr>
<td>systransactions</td>
<td>sysservers</td>
<td></td>
</tr>
</tbody>
</table>

The command also makes the following changes:

- Revokes select on sysdatabases(deftabaud) from public
- Revokes select on sysdatabases(defvwaud) from public
- Revokes select on sysdatabases(defpraud) from public
- Revokes select on sysdatabases(audflags2) from public
- Grants select on sysdatabases to sso_role
- Revokes select on syslogins(password) to public
- Revokes select on syslogins(audflags) from public
- Grants select on syslogins to sso_role
- Revokes select on syslisteners(net_type) from public
- Revokes select on syslisteners(address_info) from public
- Grants select on syslisteners to sso_role
- Revokes select on syssrvroles(srid) from public
- Revokes select on syssrvroles(name) from public
- Revokes select on syssrvroles(password) from public
- Revokes select on syssrvroles(pwdate) from public
- Revokes select on syssrvroles(status) from public
- Revokes select on syssrvroles(logincount) from public
- Grants select on syssrvroles to public
• Revokes select on sysloginroles(suid) from public
• Revokes select on sysloginroles(srid) from public
• Revokes select on sysloginroles(status) from public
• Grants select on sysloginroles to sso_role

Grants permissions for **update statistics**, **delete statistics**, and **truncate table**

Adaptive Server allows you to grant permissions for users, roles, and groups for the update statistics, delete statistics, and truncate table commands. Table owners can also provide permissions through an implicit grant by adding update statistics, delete statistics, and truncate table to a stored procedure and then granting execute permissions on that procedure to a user or role.

You cannot grant permissions for update statistics at the column level. You must have the **sso_role** to run update statistics or delete statistics on sysroles, syssrvroles, and sysloginroles security tables.

By default, users with the **sa_role** have permission to run update statistics and delete statistics on system tables other than sysroles, syssrvroles and sysloginroles, and can transfer this privilege to other users.

You can also issue **grant all** to grant permissions on update statistics, delete statistics, and truncate table.

**Note** Once you grant permission to execute update statistics to a user, they also have permission to execute variations of this command, such as **update all statistics**, **update partition statistics**, **update index statistics**, **update statistics table**, and so on. For example, the following grants “billy” permission to run all variations of update statistics on the authors table:

```
grant update statistics on authors to billy
```

If you revoke a user’s permission to execute update statistics, you also revoke their ability to execute the variations of this command.

You cannot grant variants of update statistics (for example, update index statistics) separately. That is, you cannot issue:

```
grant update all statistics to harry
```

However, you can write stored procedures that control who executes these commands. For example, the following grants “billy” execute permission for update index statistics on the authors table:

```
cREATE PROC sp_ups AS
UPDATE INDEX STATISTICS ON AUTHORS
```

Reference Manual: Commands

385
You cannot grant and revoke delete statistics permissions at the column level.

Although Adaptive Server audits truncate table as a global, miscellaneous audit, it does not audit update statistics. To retain clear audit trails for both truncate table and update statistics, Sybase recommends that you include both commands in a stored procedure to which you grant users execute permission, as described above.

The command fails and generates an error message if a user issues update statistics, delete statistics or truncate table and they:

- Do not own the table.
- Do not have the sa_role.
- Are not a database owner who has successfully used setuser to become the user who is the owner of the table.
- Have not been granted update statistics, delete statistics, or truncate table privileges.

Granting proxies and session authorizations

- Granting permission to execute set proxy or set session authorization allows the grantees to impersonate another login in Adaptive Server. set proxy and set session authorization are identical with one exception: set session authorization follows the SQL standard, and set proxy is a Transact-SQL extension.
- To grant set proxy or set session authorization permission, you must be a System Security Officer, and you must be in the master database.
- The name you specify in the grant set proxy command must be a valid user in the database; that is, the name must be in the sysusers table in the database.
- grant all does not include the set proxy or set session authorization permissions.
- You can restrict roles incrementally with grant set proxy. For example, you can first restrict the sa_role, then the sso_role:

```sql
grant set proxy to joe
restrict role sa_role
grant set proxy to joe
```
restrict role sso_role

- You cannot unrestrict individual roles. You must revoke set proxy to
  revoke permissions from all roles, as illustrated in this query:

```
select distinct user_name (p.uid), b.name, p.grantor,
    Restricted_role=case
        when 1 then
            "None"
        else
            isnull (role_name (c.number - 1), "System"
            "+convert (char,c.number))
    end
from sysprotects p, master.dbo.spt_values b, master.dbo.spt_values c
    where
        convert (tinyint,substring (isnull (p.columns,0x1),c.low,1)) &
        c.high = 0
        and c.type = "P" and c.number <= 1024 and c.number >0 and
        p.action = 167
        and b.type = "T"
        and b.number = (p.protecttype + 204)
        and role_name (c.number - 1) is not null
```

- Granting permission to execute set proxy or set session authorization
  allows the grantee to impersonate another login in Adaptive Server. set
  proxy and set session authorization are identical with one exception: set
  session authorization follows the SQL standard, and set proxy is a
  Transact-SQL extension.

- To grant set proxy or set session authorization permission, you must be a
  System Security Officer, and you must be in the master database.

- The name you specify in the grant set proxy command must be a valid user
  in the database; that is, the name must be in the sysusers table in the
  database.

- grant all does not include the set proxy or set session authorization
  permissions.

Standards

- ANSI SQL – Compliance level: Entry-level compliant. grant dbcc is also
  Transact-SQL extension.

- grant dbcc, and granting permissions to groups and granting set proxy are
  Transact-SQL extensions. Granting set session authorization (identical in
  function to set proxy) follows the ANSI standard. grant dbcc is also
  Transact-SQL extension.
Permissions

**Command execution** Only System Administrators can grant create database permission, and only from the master database. Only System Security Officers can grant create trigger permission. Only system security officers can grant create trigger and create encryption key permissions.

**Database consistency checking** Only System Administrators can run grant dbcc commands. Database Owners cannot run grant dbcc.

**Database object access** grant permission for database objects defaults to object owners. Object owners can grant permission to other users on their own database objects.

**Functions** Only System Administrators can grant permissions on built-in functions.

**Proxy and session authorization** Only System Security Officers can grant set proxy or set session authorization, and only from the master database. Granting permission to execute set proxy or set session authorization allows the grantee to impersonate another login in the server. set proxy and set session authorization are identical, except that set session authorization follows the ANSI92 standard, and set proxy is a Transact-SQL extension.

**Roles** You can grant roles only from the master database. Only System Security Officers can grant sso_role, oper_role or a user-defined role to a user or a role. Only System Administrators can grant sa_role to a user or a role. Only users who have both sa_role and sso_role can grant a role that includes sa_role.

**System tables** Database Owners can grant default permissions on system tables.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 40    | grant        | grant                     | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |
See also

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 85    | roles        | create role, drop role, alter role, grant role, or revoke role | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

**Catalog stored procedures**  
sp_column_privileges, sp_table_privileges

**Commands**  
create role, revoke, setuser, set

**Functions**  
proc_role, show_role

**System procedures**  
sp_addgroup, sp_adduser, sp_changdbowner, sp_changeuser, sp_dropgroup, sp_dropuser, sp_helpgroup, sp_helpprotect, sp_helpuser, sp_role
group by and having clauses

Description
Used in select statements to divide a table into groups and to return only groups that match conditions in the having clause. group by is typically used in conjunction with aggregates to specify how to group the unaggregated columns of a select query. having clauses are applied to these groups.

Syntax
Start of select statement
[group by [all] aggregate_free_expression [, aggregate_free_expression]...]
[having search_conditions]
End of select statement

Parameters
group by
specifies the groups into which the table is divided, and if aggregate functions are included in the select list, finds a summary value for each group. These summary values appear as columns in the results, one for each group. You can refer to these summary columns in the having clause.

You can use the avg, count, count_big, max, min, and sum aggregate functions in the select list before group by (the expression is usually a column name). For more information, see “Aggregate functions” in Chapter 2, “Transact-SQL Functions” of Reference Manual: Building Blocks.

A table can be grouped by any combination of columns—that is, groups can be nested within each other, as in Example 2.

all
is a Transact-SQL extension that includes all groups in the results, even those excluded by a where clause. For example:

```
select type, avg (price)
from titles
where advance > 7000
group by all type
```

type

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
</tr>
<tr>
<td>business</td>
<td>2.99</td>
</tr>
<tr>
<td>mod_cook</td>
<td>2.99</td>
</tr>
<tr>
<td>popular_comp</td>
<td>20.00</td>
</tr>
<tr>
<td>psychology</td>
<td>NULL</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
</tr>
</tbody>
</table>
“NULL” in the aggregate column indicates groups that would be excluded by the where clause. A having clause negates the meaning of all.

**aggregate_free_expression**

is an expression that includes no aggregates. A Transact-SQL extension allows grouping by an aggregate-free expression as well as by a column name.

You cannot group by column heading or alias. This example is correct:

```sql
select Price=avg (price), Pay=avg (advance),
      Total=price * $1.15
from titles
group by price * $1.15
```

having

sets conditions for the group by clause, similar to the way in which where sets conditions for the select clause.

having search conditions can include aggregate expressions; otherwise, having search conditions are identical to where search conditions. This is an example of a having clause with aggregates:

```sql
select pub_id, total = sum (total_sales)
from titles
where total_sales is not null
group by pub_id
having count (*)>5
```

When Adaptive Server optimizes queries, it evaluates the search conditions in where and having clauses, and determines which conditions are search arguments (SARGs) that can be used to choose the best indexes and query plan. All of the search conditions are used to qualify the rows. For more information on search arguments, see the *Performance and Tuning Guide: Optimizer and Abstract Plans*.

**Examples**

**Example 1** Calculates the average advance and the sum of the sales for each type of book:

```sql
select type, avg (advance), sum (total_sales)
from titles
group by type
```

**Example 2** Groups the results by type, then by pub_id within each type:

```sql
select type, pub_id, avg (advance), sum (total_sales)
from titles
group by type, pub_id
```
Example 3  Calculates results for all groups, but displays only groups whose type begins with “p”:

```
select type, avg (price)
from titles
group by type
having type like 'p%'
```

Example 4  Calculates results for all groups, but displays results for groups matching the multiple conditions in the having clause:

```
select pub_id, sum (advance), avg (price)
from titles
group by pub_id
having sum (advance) > $15000
and avg (price) < $10
and pub_id > "0700"
```

Example 5  Calculates the total sales for each group (publisher) after joining the titles and publishers tables:

```
select p.pub_id, sum (t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id
```

Example 6  Displays the titles that have an advance of more than $1000 and a price that is more than the average price of all titles:

```
select title_id, advance, price
from titles
where advance > 1000
having price > avg (price)
```

Usage

- You can use a column name or any expression (except a column heading or alias) after group by. You can use group by to calculate results or display a column or an expression that does not appear in the select list (a Transact-SQL extension described in “Transact-SQL extensions to group by and having” on page 395).
- The maximum number of group by columns (or expressions) is not explicitly limited. The only limit of group by results is that the width of the group by columns plus the aggregate results be no greater than 64K.
- Null values in the group by column are put into a single group.
- You cannot name text, unitext, or image columns in group by and having clauses.
You cannot use a group by clause in the select statement of an updatable cursor.

Aggregate functions can be used only in the select list or in a having clause. They cannot be used in a where or group by clause.

Aggregate functions are of two types. Aggregates applied to all the qualifying rows in a table (producing a single value for the whole table per function) are called scalar aggregates. An aggregate function in the select list with no group by clause applies to the whole table; it is one example of a scalar aggregate.

Aggregates applied to a group of rows in a specified column or expression (producing a single value for each group per function) are called vector aggregates. For either aggregate type, the results of the aggregate operations are shown as new columns that the having clause can refer to.


How group by works with the optimizer

In Adaptive Server version 15.0, there are two possible algorithms (implemented as operators) for doing group by: GroupHashing and GroupSorted. The optimizer chooses which operator to use based on factors such as the requirements these operators place on the input data streams.

The GroupSorted operator requires that the input rows to be aggregated are already sorted on the group by columns. Since the input rows must be sorted, the optimizer uses either of the following:

• An index on the order by columns to read the rows from the source table, and the maximum width of the group by columns is limited by the maximum width of an index key, which depends upon the database page size.

• An sort operator to order the rows on the group by columns before they are processed by the GroupSorted operator. The group by columns and the columns to be aggregated must fit into a worktable, so the maximum width of the group by columns is limited to the maximum row size on a database page, minus the width of the columns to be aggregated. The maximum group by column width is limited by the database page size.
The optimizer uses the GroupHashing operator if ordering on the group by columns is not available or the row size limitations of the GroupSorted operator are exceeded. The GroupHashing operator applies a hashing function to the values of the group by columns to be able to put rows with the same group by column values into the same hash bucket. Once the input rows have all been hashed into buckets, the rows in the buckets are aggregated to generate the group by results. The only limitation of the GroupHashing operator is that the total row size of group by columns and aggregate results cannot be larger than 64K. There is no limitation on the number of group by columns or the number of aggregation operations, just the total row width.

**How group by and having queries with aggregates work**

- The where clause excludes rows that do not meet its search conditions; its function remains the same for grouped or nongrouped queries.
- The group by clause collects the remaining rows into one group for each unique value in the group by expression. Omitting group by creates a single group for the whole table.
- Aggregate functions specified in the select list calculate summary values for each group. For scalar aggregates, there is only one value for the table. Vector aggregates calculate values for the distinct groups.
- The having clause excludes groups from the results that do not meet its search conditions. Even though the having clause tests only rows, the presence or absence of a group by clause may make it appear to be operating on groups:
  - When the query includes group by, having excludes result group rows. This is why having seems to operate on groups.
  - When the query has no group by, having excludes result rows from the (single-group) table. This is why having seems to operate on rows (the results are similar to where clause results).

**Standard group by and having queries**

- All group by and having queries in the Examples section adhere to the SQL standard, which dictates that queries using group by, having, and vector aggregate functions produce one row and one summary value per group, using these guidelines:
  - Columns in a select list must also be in the group by expression, or they must be arguments of aggregate functions.
A group by expression can contain only column names that are in the select list. However, columns used only as arguments of aggregate functions in the select list do not qualify.

Columns in a having expression must be single-valued—arguments of aggregates, for instance—and they must be in the select list or group by clause. Queries with a select list aggregate and a having clause must have a group by clause. If you omit the group by for a query without a select list aggregate, all the rows not excluded by the where clause are considered to be a single group.

In nongrouped queries, the principle that “where excludes rows” seems straightforward. In grouped queries, the principle expands to “where excludes rows before group by, and having excludes rows from the display of results.”

The SQL standard allows queries that join two or more tables to use group by and having, if they also adhere to the above guidelines. When specifying joins or other complex queries, use the standard syntax of group by and having until you fully comprehend the effect of the Transact-SQL extensions to both clauses.

To help you avoid problems with extensions, Adaptive Server provides the fipsflagger option to the set command that issues a nonfatal warning for each occurrence of a Transact-SQL extension in a query. See set for more information.

Transact-SQL extensions to group by and having

- A select list that includes aggregates can include extended columns that are not arguments of aggregate functions and are not included in the group by clause. An extended column affects the display of final results, since additional rows are displayed.

- The group by clause can include columns or expressions that are not in the select list.

- The group by all clause displays all groups, even those excluded from calculations by a where clause. See the example for the keyword all in the “Parameters” section.

- The having clause can include columns or expressions that are not in the select list and not in the group by clause.
When the Transact-SQL extensions add rows and columns to a display, or if group by is omitted, query results may be hard to interpret. The examples that follow can help you understand how Transact-SQL extensions can affect query results.

- The following examples illustrate the differences between queries that use standard group by and having clauses and queries that use the Transact-SQL extensions:

a  An example of a standard grouping query:

```sql
select type, avg (price)
from titles
group by type
```

<table>
<thead>
<tr>
<th>type</th>
<th>avg (price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
</tr>
<tr>
<td>business</td>
<td>13.73</td>
</tr>
<tr>
<td>mod_cook</td>
<td>11.49</td>
</tr>
<tr>
<td>popular_comp</td>
<td>21.48</td>
</tr>
<tr>
<td>psychology</td>
<td>13.50</td>
</tr>
<tr>
<td>trad_cook</td>
<td>15.96</td>
</tr>
</tbody>
</table>

(6 rows affected)

b  The Transact-SQL extended column, price (in the select list, but not an aggregate and not in the group by clause), causes all qualified rows to display in each qualified group, even though a standard group by clause produces a single row per group. The group by still affects the vector aggregate, which computes the average price per group displayed on each row of each group (they are the same values that were computed for example a):

```sql
select type, price, avg (price)
from titles
group by type
```

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>avg (price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>business</td>
<td>19.99</td>
<td>13.73</td>
</tr>
<tr>
<td>business</td>
<td>11.95</td>
<td>13.73</td>
</tr>
<tr>
<td>business</td>
<td>2.99</td>
<td>13.73</td>
</tr>
<tr>
<td>business</td>
<td>19.99</td>
<td>13.73</td>
</tr>
<tr>
<td>mod_cook</td>
<td>19.99</td>
<td>11.49</td>
</tr>
<tr>
<td>mod_cook</td>
<td>2.99</td>
<td>11.49</td>
</tr>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>popular_comp</td>
<td>22.95</td>
<td>21.48</td>
</tr>
</tbody>
</table>
The way Transact-SQL extended columns are handled can make it look as if a query is ignoring a where clause. This query computes the average prices using only those rows that satisfy the where clause, but it also displays rows that do not match the where clause.

Adaptive Server first builds a worktable containing only the type and aggregate values using the where clause. This worktable is joined back to the titles table in the grouping column type to include the price column in the results, but the where clause is not used in the join.

The only row in titles that is not in the results is the lone row with type = “UNDECIDED” and a NULL price, that is, a row for which there were no results in the worktable. If you also want to eliminate the rows from the displayed results that have prices of less than $10.00, you must add a having clause that repeats the where clause, as shown in Example 4:

```
SELECT type, price, AVG(price) FROM titles
WHERE price > 10.00
GROUP BY type
```

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>avg_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>business</td>
<td>19.99</td>
<td>17.31</td>
</tr>
<tr>
<td>business</td>
<td>11.95</td>
<td>17.31</td>
</tr>
<tr>
<td>business</td>
<td>2.99</td>
<td>17.31</td>
</tr>
<tr>
<td>business</td>
<td>19.99</td>
<td>17.31</td>
</tr>
<tr>
<td>mod_cook</td>
<td>19.99</td>
<td>19.99</td>
</tr>
<tr>
<td>mod_cook</td>
<td>2.99</td>
<td>19.99</td>
</tr>
<tr>
<td>popular_comp</td>
<td>22.95</td>
<td>21.48</td>
</tr>
<tr>
<td>popular_comp</td>
<td>20.00</td>
<td>21.48</td>
</tr>
<tr>
<td>popular_comp</td>
<td>NULL</td>
<td>21.48</td>
</tr>
<tr>
<td>psychology</td>
<td>21.59</td>
<td>17.51</td>
</tr>
</tbody>
</table>

(18 rows affected)
**group by and having clauses**

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>avg (price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>psychology</td>
<td>10.95</td>
<td>17.51</td>
</tr>
<tr>
<td>psychology</td>
<td>7.00</td>
<td>17.51</td>
</tr>
<tr>
<td>psychology</td>
<td>19.99</td>
<td>17.51</td>
</tr>
<tr>
<td>psychology</td>
<td>7.99</td>
<td>17.51</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>15.96</td>
</tr>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>15.96</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>15.96</td>
</tr>
</tbody>
</table>

(17 rows affected)

d If you are specifying additional conditions, such as aggregates, in the having clause, also include all conditions specified in the where clause. Adaptive Server appears to ignore any where clause conditions that are missing from the having clause:

```sql
select type, price, avg (price)
from titles
where price > 10.00
group by type
having price > 10.00
```

type | price
-----|-------
business | 19.99 | 17.31
business | 11.95 | 17.31
business | 19.99 | 17.31
mod_cook  | 19.99 | 19.99
popular_comp | 22.95 | 21.48
popular_comp | 20.00 | 21.48
psychology | 21.59 | 17.51
psychology | 10.95 | 17.51
psychology | 19.99 | 17.51
trad_cook  | 20.95 | 15.96
trad_cook  | 11.95 | 15.96
trad_cook  | 14.99 | 15.96

(12 rows affected)
e This is an example of a standard grouping query using a join between two tables. It groups by pub_id, then by type within each publisher ID, to calculate the vector aggregate for each row:

```sql
select p.pub_id, t.type, sum (t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id, t.type
```

pub_id | type
-------|-------

Adaptive Server Enterprise
It may seem that it is only necessary to specify `group by` for the `pub_id` and `type` columns to produce the results, and add extended columns as follows:

```sql
select p.pub_id, p.pub_name, t.type, 
    sum (t.total_sales) 
from publishers p, titles t 
where p.pub_id = t.pub_id 
group by p.pub_id, t.type
```

However, the results for the above query are much different from the results for the first query in this example. After joining the two tables to determine the vector aggregate in a worktable, Adaptive Server joins the worktable to the table (publishers) of the extended column for the final results. Each extended column from a different table invokes an additional join.

As you can see, using the extended column extension in queries that join tables can easily produce results that are difficult to comprehend. In most cases, use the standard `group by` to join tables in your queries.

This example uses the Transact-SQL extension to `group by` to include columns that are not in the select list. Both the `pub_id` and `type` columns are used to group the results for the vector aggregate. However, the final results do not include the type within each publisher. In this case, you may only want to know how many distinct title types are sold for each publisher:

```sql
select p.pub_id, sum (t.total_sales) 
from publishers p, titles t 
where p.pub_id = t.pub_id 
group by p.pub_id, t.type
```

```plaintext
-------- -------- --------
0736  18722
0736  9564
0877  NULL
0877  24278
0877  375
0877  19566
1389  12066
1389  12875
(8 rows affected)
```
This example combines two Transact-SQL extension effects. First, it omits the group by clause while including an aggregate in the select list. Second, it includes an extended column. By omitting the group by clause:

- The table becomes a single group. The scalar aggregate counts three qualified rows.
- `pub_id` becomes a Transact-SQL extended column because it does not appear in a group by clause. No having clause is present, so all rows in the group are qualified to be displayed.

```sql
select pub_id, count (pub_id)
from publishers
```

<table>
<thead>
<tr>
<th>pub_id</th>
<th>count (pub_id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>3</td>
</tr>
<tr>
<td>0877</td>
<td>3</td>
</tr>
<tr>
<td>1389</td>
<td>3</td>
</tr>
</tbody>
</table>

(3 rows affected)

The `where` clause excludes publishers with a `pub_id` of 1000 or more from the single group, so the scalar aggregate counts two qualified rows. The extended column `pub_id` displays all qualified rows from the `publishers` table:

```sql
select pub_id, count (pub_id)
from publishers
where pub_id < "1000"
```

<table>
<thead>
<tr>
<th>pub_id</th>
<th>count (pub_id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>2</td>
</tr>
<tr>
<td>0877</td>
<td>2</td>
</tr>
<tr>
<td>1389</td>
<td>2</td>
</tr>
</tbody>
</table>

(3 rows affected)
This example illustrates an effect of a having clause used without a group by clause.

- The table is considered a single group. No where clause excludes rows, so all the rows in the group (table) are qualified to be counted.
- The rows in this single-group table are tested by the having clause.
- These combined effects display the two qualified rows.

```sql
select pub_id, count (pub_id)
from publishers
having pub_id < "1000"

pub_id
----------
0736  3
0877  3
(2 rows affected)
```

This example uses the extension to having that allows columns or expressions not in the select list and not in the group by clause. It determines the average price for each title type, but it excludes those types that do not have more than $10,000 in total sales, even though the sum aggregate does not appear in the results:

```sql
select type, avg (price)
from titles
group by type
having sum (total_sales) > 10000

type
-------
business 13.73
mod_cook  11.49
popular_comp 21.48
trad_cook  15.96
(4 rows affected)
```

**group by and having and sort orders**

- If your server has a case-insensitive sort order, group by ignores the case of the grouping columns. For example, given this data on a case-insensitive server:

```sql
select lname, amount
from groupdemo
```
group by and having clauses

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>10.00</td>
</tr>
<tr>
<td>smith</td>
<td>5.00</td>
</tr>
<tr>
<td>SMITH</td>
<td>7.00</td>
</tr>
<tr>
<td>Levi</td>
<td>9.00</td>
</tr>
<tr>
<td>Lévi</td>
<td>20.00</td>
</tr>
</tbody>
</table>

grouping by lname produces these results:

```
select lname, sum (amount)
from groupdemo
```

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levi</td>
<td>9.00</td>
</tr>
<tr>
<td>Lévi</td>
<td>20.00</td>
</tr>
<tr>
<td>Smith</td>
<td>22.00</td>
</tr>
</tbody>
</table>

The same query on a case- and accent-insensitive server produces these results:

<table>
<thead>
<tr>
<th>lname</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levi</td>
<td>29.00</td>
</tr>
<tr>
<td>Smith</td>
<td>22.00</td>
</tr>
</tbody>
</table>

Standards

ANSI SQL – Compliance level: Entry-level compliant.

The use of columns within the select list that are not in the group by list and have no aggregate functions is a Transact-SQL extension.

The use of the all keyword is a Transact-SQL extension.

See also

Commands  compute clause, declare, select, where clause

Functions  Aggregate functions
**if...else**

**Description**
Imposes conditions on the execution of a SQL statement.

**Syntax**

```
if logical_expression [plan "abstract plan"]
  statements
[else]
  [if logical_expression] [plan "abstract plan"]
  statement
```

**Parameters**

- `logical_expression` is an expression (a column name, a constant, any combination of column names and constants connected by arithmetic or bitwise operators, or a subquery) that returns TRUE, FALSE, or NULL. If the expression contains a `select` statement, you must enclose the `select` statement in parentheses.
- `plan "abstract plan"` specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can be specified only for optimizable SQL statements, that is, `select` queries that access tables. For more information, see Chapter 16, “Creating and Using Abstract Plans,” in the *Performance and Tuning Guide: Optimizer and Abstract Plans*.
- `statements` is either a single SQL statement or a block of statements delimited by `begin` and `end`.

**Examples**

**Example 1**
Prints “yes” if 3 is larger than 2:

```
if 3 > 2
  print "yes"
```

**Example 2**
The `if...else` condition tests for the presence of authors whose postal codes are 94705, then prints “Berkeley author” for the resulting set:

```
if exists (select postalcode from authors
  where postalcode = "94705")
  print "Berkeley author"
```

**Example 3**
The `if...else` condition tests for the presence of user-created objects (all of which have ID numbers greater than 100) in a database. Where user tables exist, the `else` clause prints a message and selects their names, types, and ID numbers:

```
if (select max (id) from sysobjects) < 100
  print "No user-created objects in this database"
else
  begin
```
print "These are the user-created objects"
select name, type, id
from sysobjects
where id > 100
end

Example 4  Since the value for total sales for PC9999 in the titles table is
NULL, this query returns FALSE. The else portion of the query is performed
when the if portion returns FALSE or NULL. For more information on truth
values and logical expressions, see “Expressions” in Chapter 4, “Expressions,

if (select total_sales
  from titles
  where title_id = "PC9999") > 100
  select "true"
else
  select "false"

Usage

•  The statement following an if keyword and its condition is executed if the
  condition is satisfied (when the logical expression returns TRUE). The
  optional else keyword introduces an alternate SQL statement that executes
  when the if condition is not satisfied (when the logical expression returns
  FALSE).

•  The if or else condition affects the performance of only a single SQL
  statement, unless statements are grouped into a block between the
  keywords begin and end (see Example 3).

  The statement clause can be an execute command or any other legal SQL
  statement or statement block.

•  If a select statement is used as part of the Boolean expression, it must
  return a single value.

•  if...else constructs can be used either in a stored procedure (where they are
  often used to test for the existence of some parameter) or in ad hoc queries
  (see Examples 1 and 2).
• if tests can be nested either within another if or following an else. The maximum number of if tests you can nest varies with the complexity of any select statements (or other language constructs) that you include with each if...else construct.

**Note** When an alter table, create table, or create view command occurs within an if...else block, Adaptive Server creates the schema for the table or view before determining whether the condition is true. This may lead to errors if the table or view already exists.

• If you create tables with varchar, nvarchar, univarchar, or varbinary columns whose total defined width is greater than the maximum allowed row size, a warning message appears, but the table is created. If you try to insert more than the maximum number bytes into such a row, or to update a row so that its total row size is greater than the maximum length, Adaptive Server produces an error message, and the command fails.

**Note** When a create table command occurs within an if...else block or a while loop, Adaptive Server creates the schema for the table before determining whether the condition is true. This may lead to errors if the table already exists. To avoid this situation, either make sure a view with the same name does not already exist in the database or use an execute statement, as follows:

```sql
    if not exists
        (select * from sysobjects where name="my table")
    begin
        execute "create table mytable (x int)"
    end
```

**Standards** ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions** if...else permission defaults to all users. No permission is required to use it.

**See also** Commands begin...end, create procedure
insert

Description
Adds new rows to a table or view.

Syntax
insert [into] [database.[owner.]][table_name|view_name]
[(column_list)]
{values (expression [, expression]...)
 |select_statement [plan "abstract plan"]}

Parameters
into
is optional.

table_name | view_name
is the name of the table or view from which you want to remove rows.
Specify the database name if the table or view is in another database, and
specify the owner’s name if more than one table or view of that name exists
in the database. The default value for owner is the current user, and the
default value for database is the current database.

column_list
is a list of one or more columns to which data is to be added. Enclose the list
in parentheses. The columns can be listed in any order, but the incoming data
(whether in a values clause or a select clause) must be in the same order. If
a column has the IDENTITY property, you can substitute the syb_identity
keyword for the actual column name.

The column list is necessary when some, but not all, of the columns in the
table are to receive data. If no column list is given, Adaptive Server assumes
that the insert affects all columns in the receiving table (in create table order).

See “The column list” on page 408 for more information.

values
introduces a list of expressions.
expression

specifies constant expressions, variables, parameters, or null values for the indicated columns. Enclose character and datetime constants in single or double quotes.

You cannot use a subquery as an expression.

The values list:

- Must be enclosed in parentheses
- Must match the explicit or implicit column list
- Can use “default” as a value


select statement

is a standard select statement used to retrieve the values to be inserted.

plan "abstract plan"

specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for insert...select statements. See Chapter 16, “Creating and Using Abstract Plans,” in the Performance and Tuning Guide: Optimizer and Abstract Plans for more information.

Examples

Example 1

insert titles
values ("BU2222", "Faster!", "business", "1389",
null, null, null, "ok", "06/17/87", 0)

Example 2

insert titles
(select * from authors
where city = "San Francisco"
values ('BU1237', 'Get Going!', 'business', '1389', 'great', '06/18/86', 1)

Example 3

insert newauthors
select * from authors
where city = "San Francisco"

Example 4

insert test
insert *
from test
where city = "San Francisco"

Example 5

insert table1 (col1, col2, col3, col4)
values (10, 4, default, 34)

Usage

- Use `insert` only to add new rows. Use `update` to modify column values in a row you have already inserted.

The column list

- The column list determines the order in which values are entered. For example, suppose that you have a table called `newpublishers` that is identical in structure and content to the `publishers` table in `pubs2`. In the example below, the columns in the column list of the `newpublishers` table match the columns of the select list in the `publishers` table.

  ```sql
  insert newpublishers (pub_id, pub_name)
  select pub_id, pub_name
  from publishers
  where pub_name="New Age Data"
  ```

  The `pub_id` and `pub_name` for “New Age Data” are stored in the `pub_id` and `pub_name` columns of `newpublishers`.

In the next example, the order of the columns in the column list of the `newpublishers` table does not match the order of the columns of the select list of the `publishers` table.

  ```sql
  insert newpublishers (pub_id, pub_name)
  select pub_name, pub_id
  from publishers
  where pub_name="New Age Data"
  ```

  The result is that the `pub_id` for “New Age Data” is stored in the `pub_name` column of the `newpublishers` table, and the `pub_name` for “New Age Data” is stored in the `pub_id` column of the `newpublishers` table.

- You can omit items from the column and values lists as long as the omitted columns allow null values (see Example 2).

Validating column values

- `insert` interacts with the `ignore_dup_key`, `ignore_dup_row`, and `allow_dup_row` options, which are set with the `create index` command. See `create index` for more information.
A rule or check constraint can restrict the domain of legal values that can be entered into a column. Rules are created with the `create rule` command and bound with `sp_bindrule`. Check constraints are declared with `create table`.

A default can supply a value if you do not explicitly enter one. Defaults are created with the `create default` command and bound with `sp_bindefault`, or they are declared with `create table`.

If an insert statement violates domain or integrity rules (see `create rule` and `create trigger`), or if it is the wrong datatype (see `create table` and Chapter 1, “System and User-Defined Datatypes” in Reference Manual: Building Blocks), the statement fails, and Adaptive Server displays an error message.

Treatment of blanks

- Inserting an empty string (""") into a variable character type or text column inserts a single space. Char columns are padded to the defined length.

- All trailing spaces are removed from data that is inserted into varchar and univarchar columns, except in the case of a string that contains only spaces. Strings that contain only spaces are truncated to a single space. Strings that are longer than the specified length of a char, nchar, unichar, univarchar, varchar, or nvarchar column are silently truncated unless the `string_rtruncation` option is set to on.

Inserting into `text`, `unitext`, and `image` columns

- An insert of a NULL into a text, ortext, or an image column simply allocates space for a text pointer. Use `update` to get a valid text pointer for that column.

`insert` triggers

- You can define a trigger that takes a specified action when an insert command is issued on a specified table.

Using `insert` when Component Integration Services is enabled

- You can send an `insert` as a language event or as a parameterized dynamic statement to remote servers.

Inserting rows selected from another table

- You can select rows from a table and insert them into the same table in a single statement (see Example 4).
To insert data with `select` from a table that has null values in some fields into a table that does not allow null values, provide a substitute value for any NULL entries in the original table. For example, to insert data into an `advances` table that does not allow null values, substitute 0 for the NULL fields:

```
insert advances
    select pub_id, isnull (advance, 0) from titles
```

Without the `isnull` function, this command inserts all the rows with non-null values into the `advances` table, which produces error messages for all the rows where the `advance` column in the `titles` table contained NULL.

If you cannot make this kind of substitution for your data, you cannot insert data containing null values into the columns that have a `not null` specification.

Two tables can be identically structured, and yet be different as to whether null values are permitted in some fields. Use `sp_help` to see the null types of the columns in your table.

**Transactions and `insert`**

- When you set chained transaction mode, Adaptive Server implicitly begins a transaction with the `insert` statement if no transaction is currently active. To complete any inserts, you must commit the transaction, or roll back the changes. For example:

  ```
  insert stores (stor_id, stor_name, city, state)
  values ('999', 'Books-R-Us', 'Fremont', 'AZ')
  if exists (select t1.city
    from stores t1, stores t2
    where t1.city = t2.city
    and t1.state = t2.state
    and t1.stor_id < t2.stor_id)
    rollback transaction
  else
    commit transaction
  ```

In chained transaction mode, this batch begins a transaction and inserts a new row into the `stores` table. If it inserts a row containing the same city and state information as another store in the table, it rolls back the changes to `stores` and ends the transaction. Otherwise, it commits the insertions and ends the transaction. For more information about chained transaction mode, see the *Transact-SQL User’s Guide*. 
Inserting values into IDENTITY columns

- When inserting a row into a table, do not include the name of the IDENTITY column in the column list or its value in the values list. If the table consists of only one column, an IDENTITY column, omit the column list and leave the values list empty as follows:

  ```
  insert id_table values ()
  ```

- The first time you insert a row into a table, Adaptive Server assigns the IDENTITY column a value of 1. Each new row gets a column value that is one higher than the last. This value takes precedence over any defaults declared for the column in the `create table` or `alter table` statement or defaults bound to the column with `sp_bindefault`.

  Server failures can create gaps in IDENTITY column values. The maximum size of the gap depends on the setting of the `identity_burning_set factor` configuration parameter. Gaps can also result from manual insertion of data into the IDENTITY column, deletion of rows, and transaction rollbacks.

- Only the table owner, Database Owner, or System Administrator can explicitly insert a value into an IDENTITY column after setting `identity_insert table_name` on for the column’s base table. A user can set `identity_insert table_name` on for one table at a time in a database. When `identity_insert` is on, each insert statement must include a column list and must specify an explicit value for the IDENTITY column.

  Inserting a value into the IDENTITY column allows you to specify a seed value for the column or to restore a row that was deleted in error. Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the value; you can insert any positive integer.

  To insert an explicit value into an IDENTITY column, the table owner, Database Owner, or System Administrator must set `identity_insert table_name` on for the column’s base table, not for the view through which it is being inserted.

- The maximum value that can be inserted into an IDENTITY column is `10^precision - 1` for a numeric. For integer identities, it is the maximum permissible value of its type (such as 255 for `tinyint`, 32767 for `smallint`). Once an IDENTITY column reaches this value, any additional `insert` statements return an error that aborts the current transaction.
When this happens, use the `create table` statement to create a new table that is identical to the old one, but that has a larger precision for the IDENTITY column. Once you have created the new table, use either the `insert` statement or the `bcp` utility to copy the data from the old table to the new one.

- Use the `@@identity` global variable to retrieve the last value that you inserted into an IDENTITY column. If the last `insert` or `select into` statement affected a table with no IDENTITY column, `@@identity` returns the value 0.

- An IDENTITY column selected into a result table observes the following rules with regard to inheritance of the IDENTITY property:
  
  - If an IDENTITY column is selected more than once, it is defined as `not null` in the new table. It does not inherit the IDENTITY property.
  
  - If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as `null` if any column in the expression allows nulls; otherwise, it is created as `not null`.

  - If the select statement contains a group by clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created `null`; others are created `not null`.

  - An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a `null` column, the new column is defined as `null`; otherwise, it is defined as `not null`.

Inserting data through views

- If a view is created with check option, each row that is inserted through the view must meet the selection criteria of the view.

  For example, the `stores_cal` view includes all rows of the `stores` table for which `state` has a value of “CA”:

  ```sql
  create view stores_cal
  as select * from stores
  where state = "CA"
  with check option
  ```

  The `with check option` clause checks each `insert` statement against the view’s selection criteria. Rows for which `state` has a value other than “CA” are rejected.
If a view is created with check option, all views derived from the base view must satisfy the view’s selection criteria. Each new row inserted through a derived view must be visible through the base view.

Consider the view stores_cal30, which is derived from stores_cal. The new view includes information about stores in California with payment terms of “Net 30:"

```sql
create view stores_cal30
as select * from stores_cal
where payterms = "Net 30"
```

Because stores_cal was created with check option, all rows inserted or updated through stores_cal30 must be visible through stores_cal. Any row with a state value other than “CA” is rejected.

stores_cal30 does not have a with check option clause of its own. This means you can insert or update a row with a payterms value other than “Net 30” through stores_cal30. The following update statement would be successful, even though the row would no longer be visible through stores_cal30:

```sql
update stores_cal30
set payterms = "Net 60"
where stor_id = "7067"
```

- insert statements are not allowed on join views created with check option.
- If you insert or update a row through a join view, all affected columns must belong to the same base table.

Partitioning tables for improved insert performance

- An unpartitioned table with no clustered index consists of a single doubly linked chain of database pages, so each insertion into the table uses the last page of the chain. Adaptive Server holds an exclusive lock on the last page while it inserts the rows, blocking other concurrent transactions from inserting data into the table.

Partitioning a table with the partition clause of the alter table command creates additional page chains. Each chain has its own last page, which can be used for concurrent insert operations. This improves insert performance by reducing page contention. If the table is spread over multiple physical devices, partitioning also improves insert performance by reducing I/O contention while the server flushes data from cache to disk. For more information about partitioning tables for insert performance, see Chapter 6, “Controlling Physical Data Placement,” in Performance and Tuning Guide: Basics.
Standards

ANSI SQL – Compliance level: Entry-level compliant.

The following are Transact-SQL extensions:

- A union operator in the select portion of an insert statement.
- Qualification of a table or column name by a database name.
- Insertion through a view that contains a join.

Note The FIPS flagger does not detect insertions through a view that contains a join.

Permissions

- insert permission defaults to the table or view owner, who can transfer it to other users.
- insert permission for a table’s IDENTITY column is limited to the table owner, Database Owner, and System Administrator.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 41    | insert       | insert into a table        | • Roles – Current active roles  
|       |              |                             | • Keywords or options  
|       |              |                             |   • If insert – INSERT  
|       |              |                             |   • If select into – INSERT INTO followed by the fully qualified object name  
|       |              |                             |   • Previous value – NULL  
|       |              |                             |   • Current value – NULL  
|       |              |                             |   • Other information – NULL  
|       |              |                             |   • Proxy information – Original login name, if set proxy in effect  

42 insert insert into a view | • Roles – Current active roles  
| • Keywords or options – INSERT  
| • Previous value – NULL  
| • Current value – NULL  
| • Other information – NULL  
| • Proxy information – Original login name, if a set proxy is in effect  

See also Commands alter table, create default, create index, create rule, create table, create trigger, dbcc, delete, select, update  

414 Adaptive Server Enterprise
Datatypes  Chapter 1, “System and User-Defined Datatypes” of *Reference Manual: Building Blocks*

System procedures  `sp_bindefault`, `sp_bindrule`, `sp_help`, `sp_helppartition`, `sp_unbindefault`, `sp_unbindrule`

Utilities  `bcp`
**kill**

**Description**
Kills a process.

**Syntax**
```
kill spid with statusonly
```

**Parameters**
- `spid` is the identification number of the process you want to kill. `spid` must be a constant; it cannot be passed as a parameter to a stored procedure or used as a local variable. Use `sp_who` to see a list of processes and other information.

- `with statusonly` reports on the progress of a server process ID (`spid`) in rollback status. It does not terminate the `spid`. The `statusonly` report displays the percent of rollback completed and the estimated length of time in seconds before the rollback completes.

**Examples**

**Example 1** Kills process number 1378:
```
kill 1378
```

**Example 2** Reports on the process of the rollback of `spid` number 13:
```
kill 13 with statusonly
```

```
spid: 13 Transaction rollback in progress. Estimated rollback completion: 17%
Estimated time left: 13 seconds
```

To track the progress of a rollback, you must run `kill...with statusonly` multiple times. If the rollback of the `spid` has completed when you issue `kill...statusonly` or if Adaptive Server is not rolling back the specified `spid`, `kill...statusonly` returns the following message:

```
Status report cannot be obtained. KILL spid:nn is not in progress.
```

**Usage**
To get a report on the current processes, execute `sp_who`. Following is a typical report:

<table>
<thead>
<tr>
<th>fid</th>
<th>spid</th>
<th>status</th>
<th>loginame</th>
<th>origname</th>
<th>hostname</th>
<th>blk</th>
<th>dbname</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>recv</td>
<td>bird</td>
<td>bird</td>
<td>jazzy</td>
<td>0</td>
<td>master</td>
<td>AWAITING COMMAND</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>sleeping</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>master</td>
<td>NETWORK HANDLER</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>sleeping</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>master</td>
<td>MIRROR HANDLER</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>sleeping</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>master</td>
<td>AUDIT PROCESS</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>sleeping</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>master</td>
<td>CHECKPOINT SLEEP</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>recv</td>
<td>rose</td>
<td>rose</td>
<td>petal</td>
<td>0</td>
<td>master</td>
<td>AWAITING COMMAND</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>running</td>
<td>robert</td>
<td>sa</td>
<td>helos</td>
<td>0</td>
<td>master</td>
<td>SELECT</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>send</td>
<td>daisy</td>
<td>daisy</td>
<td>chain</td>
<td>0</td>
<td>pubs2</td>
<td>SELECT</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>alarm</td>
<td>lily</td>
<td>lily</td>
<td>pond</td>
<td>0</td>
<td>master</td>
<td>WAITFOR</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>lock</td>
<td>viola</td>
<td>viola</td>
<td>cello</td>
<td>7</td>
<td>pubs2</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

Adaptive Server Enterprise
The `spid` column contains the process identification numbers used in the Transact-SQL `kill` command. The `blk` column contains the process ID of a blocking process, if there is one. A blocking process (which may have an exclusive lock) is one that is holding resources that are needed by another process. In this example, process 10 (a `select` on a table) is blocked by process 7 (a `begin transaction` followed by an `insert` on the same table).

The `status` column reports the state of the command. Table 1-28 shows the status values and the effects of `sp_who`:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
<th>Effect of <code>kill</code> command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>recv sleep</code></td>
<td>Waiting on a network read.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>send sleep</code></td>
<td>Waiting on a network send.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>alarm sleep</code></td>
<td>Waiting on an alarm, such as <code>waitfor delay &quot;10:00&quot;</code>.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>lock sleep</code></td>
<td>Waiting on a lock acquisition.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>sleeping</code></td>
<td>Waiting on disk I/O or some other resource.</td>
<td>Process is killed when it &quot;wakes up,&quot; usually immediately. A few sleeping processes do not wake up, and require an Adaptive Server restart to clear.</td>
</tr>
<tr>
<td><code>runnable</code></td>
<td>In the queue of runnable processes.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>running</code></td>
<td>Actively running on one of the server engines.</td>
<td>Immediate</td>
</tr>
<tr>
<td><code>infected</code></td>
<td>Adaptive Server has detected a serious error condition; extremely rare.</td>
<td><code>kill</code> command not recommended. Adaptive Server restart probably required to clear process.</td>
</tr>
<tr>
<td><code>background</code></td>
<td>A process, such as a threshold procedure, run by Adaptive Server rather than by a user process.</td>
<td>Immediate; use <code>kill</code> with extreme care. Recommend a careful check of <code>sysprocesses</code> before killing a background process.</td>
</tr>
<tr>
<td><code>log suspend</code></td>
<td>Processes suspended by reaching the last-chance threshold on the log.</td>
<td>Immediate</td>
</tr>
</tbody>
</table>

To get a report on the current locks and the `spids` of the processes holding them, use `sp_lock`.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`kill` permission defaults to System Administrators and is not transferable.

**See also**

- `Commands`: `shutdown`
- `System procedures`: `sp_lock`, `sp_who`
load database

Description

Loads a backup copy of a user database, including its transaction log, that was created with dump database, as well as materialize archive databases that have been loaded with a database dump.

The target platform of a load database operation need not be the same platform as the source platform where the dump database operation occurred. dump database and load database are performed from either a big endian platform to a little endian platform, or from a little endian platform to a big endian platform.

Syntax

**Makes a routine database load:**

```sql
load database database_name
  from [compression=]stripe_device
  [at backup_server_name]
  [density = density_value, 
    blocksize = number_bytes, 
    dumpvolume = volume_name, 
    file = file_name]
  with verify only [= header | full]
  [stripe on [compression=]stripe_device
    [at backup_server_name]
    [density = density_value, 
      blocksize = number_bytes, 
      dumpvolume = volume_name, 
      file = file_name]] ...
  [with {
    density = density_value, 
    blocksize = number_bytes, 
    compression, 
    dumpvolume = volume_name, 
    file = file_name, 
    [dismount | nodismount], 
    [nounload | unload], 
    passwd = password, 
    notify = {client | operator_console}, 
    [override[]]}
```

**Returns header or file information without loading the backup:**

```sql
load database database_name
  from [compress::]stripe_device
  [at backup_server_name]
  [density = density_value, 
    blocksize = number_bytes,
```
dumpvolume = volume_name,
file = file_name

[stripe on [compress:::]stripe_device
 [at backup_server_name]
 [density = density_value,
 blocksize = number_bytes,
 dumpvolume = volume_name,
 file = file_name]

[[stripe on [compress:::]stripe_device
 [at backup_server_name]
 [density = density_value,
 blocksize = number_bytes,
 dumpvolume = volume_name,
 file = file_name]]...]

[with {
  density = density_value,
  blocksize = number_bytes,
  compression,
  dumpvolume = volume_name,
  file = file_name,
  [dismount | nodismount],
  [nounload | unload],
  passwd = password,
  listonly [= full],
  headeronly,
  notify = {client | operator_console}
}]

Materializes an archive database:

load database database_name
  from dump_device
    [ [stripe on stripe_device] ... ]
  [with [norecovery,][passwd=password]

Parameters

database_name

is the name of the database to receive the backup copy. It can be either a
database created with the for load option, or an existing database. Loading
dumped data to an existing database overwrites all existing data. The
receiving database must be at least as large as the dumped database. The
database name can be specified as a literal, a local variable, or a stored
procedure parameter.

For archive databases, database_name is the name of the archive database
into which you want to load.
compress::
    invokes the decompression of the archived database. For more information
    about the compress option, see Chapter 27, “Backing Up and Restoring User
    Databases” in the System Administration Guide.

Note Sybase recommends the native "compression = compression_level" option as
preferred over the older "compress::compression_level" option. If you use the
native option for dump database, you do not need to use
"compress::compression_level" when loading your database.

from dump_device
    specifies the name of the disk database dump from which you want to load
    the dump.

from stripe_device
    is the device from which data is being loaded. See “Specifying dump
devices” on page 441 for information about what form to use when
specifying a dump device. For a list of supported dump devices, see the
Adaptive Server installation and configuration guides.

at backup_server_name
    is the name of a remote Backup Server running on the machine to which the
dump device is attached. For platforms that use interfaces files, the
backup_server_name must appear in the interfaces file.

density = density_value
    is ignored. For more information, see the dump database command.

blocksize = number_bytes
    overrides the default block size for a dump device. If you specify a block
    size on UNIX systems, it should be identical to that used to make the dump.
    For more information, see the dump database command.

dumpvolume = volume_name
    is the volume name field of the ANSI tape label. load database checks this
    label when the tape is opened and generates an error message if the wrong
    volume is loaded.

Note When using load database, the dumpvolume option does not provide an
error message if an incorrect file name is given for the file=filename option. The
backup server searches the entire tape looking for that file, regardless of an
incorrect tape mounted.
file = file_name

is the name of a particular database dump on the tape volume. If you did not record the dump file names when you made the dump, use listonly to display information about all dump files.

stripe on stripe_device

is an additional dump device. You can use up to 32 devices, including the device named in the to stripe_device clause. The Backup Server loads data from all devices concurrently, reducing the time and the number of volume changes required. See “Specifying dump devices” on page 441 for more information.

compression

indicates that the database you are loading was compressed to a file on a remote server. You do not need to specify the compression level for load database.

Note  Sybase recommends the native "compression = compress_level" option as preferred over the older "compress::compression_level" option. If you use the native option for dump database, you do not need to use "compress::compression_level" when loading your database.

dismount | nodismount

on platforms that support logical dismount – determines whether tapes remain mounted. By default, all tapes used for a load are dismounted when the load completes. Use nodismount to keep tapes available for additional loads or dumps.

unload | nounload

determines whether tapes rewind after the load completes. By default, tapes do not rewind, allowing you to make additional loads from the same tape volume. Specify unload for the last dump file to be loaded from a multidump volume. This rewinds and unloads the tape when the load completes.

with [norecovery,]

indicates when materializing an archive database that the load database command will not run recovery, and that the database is brought online automatically after the load database command has completed.
load database

passwd = password

is the password you provided to protect the dump file from unauthorized users. The password must be between 6 and 30 characters long. You cannot use variables for passwords. For rules on passwords, see Chapter 14, “Managing Adaptive Server Logins, Database Users, and Client Connections,” in the System Administration Guide, Volume 1.

listonly [= full]
displays information about all dump files on a tape volume, but does not load the database. listonly identifies the database and device, the date and time the dump was made, and the date and time it can be overwritten. listonly = full provides additional details about the dump. Both reports are sorted by ANSI tape label.

After listing the files on a volume, the Backup Server sends a volume change request. The operator can either mount another tape volume or terminate the list operation for all dump devices.

Due to current implementation, the listonly option overrides the headeronly option.

Warning! Do not use load database with listonly on 1/4-inch cartridge tape.

with verify[only][=header | full]
performs a minimal header or structural row check on the data pages as they are being copied to the archives, but does not load the database. There are no structural checks done at this time to gam, oam, allocation pages, indexes, text, or log pages. The only other check is done on pages where the page number matches to the page header.
headeronly displays header information for a single dump file, but does not load the database. headeronly displays information about the first file on the tape unless you use the file = file\_name option to specify another file name. The dump header indicates:

- Type of dump (database or transaction log)
- Database ID
- File name
- Date the dump was made
- Character set
- Sort order
- Page count
- Next object ID

notify = \{client | operator\_console\} overrides the default message destination.

- On operating systems that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use client to route other Backup Server messages to the terminal session that initiated the dump database.

- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that initiated the dump database. Use operator\_console to route messages to the terminal on which the Backup Server is running.

override you must use with override to successfully load the database containing encryption keys that encrypt columns in other databases.

**Examples**

**Example 1** Reloads the database pubs2 from a tape device:

```
load database pubs2
from "/dev/nrmt0"
```

**Example 2** Loads the pubs2 database, using the Backup Server REMOTE\_BKP\_SERVER. This command names three devices:

```
load database pubs2
from "/dev/nrmt4" at REMOTE\_BKP\_SERVER
stripe on "/dev/nrmt5" at REMOTE\_BKP\_SERVER
```
Example 3  Loads the pubs2 database from a compressed dump file called dmp090100.dmp located at /opt/bin/Sybase/dumps:

    load database pubs2 from
    "compress:/opt/bin/Sybase/dumps/dmp090100.dmp"

Example 4  Loads the key_db database, which contains encryption keys, you must use with override if the encryption keys in key_db were used to encrypt columns in other databases:

    load database key_db from "/tmp/key_db.dat" with override

Usage

- If you use sp_hidetext followed by a cross-platform dump and load, you must manually drop and re-create all hidden objects.
- The listonly and headeronly options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
- To make sure databases are synchronized correctly so that all proxy tables have the correct schema to the content of the primary database you just reloaded, you may need to run the alter database dbname for proxy_update command on the server hosting the proxy database.
- Table 1-29 describes the commands and system procedures used to restore databases from backups:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>create database for load</td>
<td>Create a database for the purpose of loading a dump.</td>
</tr>
<tr>
<td>load database</td>
<td>Restore a database from a dump.</td>
</tr>
<tr>
<td>load transaction</td>
<td>Apply recent transactions to a restored database.</td>
</tr>
<tr>
<td>online database</td>
<td>Make a database available for public use after a normal load sequence or</td>
</tr>
<tr>
<td></td>
<td>after upgrading the database to the current version of Adaptive Server.</td>
</tr>
<tr>
<td>load {database</td>
<td>transaction}</td>
</tr>
<tr>
<td>with {headeronly</td>
<td>listonly}</td>
</tr>
<tr>
<td>sp_volchanged</td>
<td>Respond to Backup Server volume change messages.</td>
</tr>
</tbody>
</table>

- See “Encrypted columns and dump database” on page 327 in the section for dump database for more information about loading databases with encrypted columns.
Restrictions

- Component Integration Services only – Any proxy tables in the database are part of the database save set. The content data of proxy tables is not included in the save; only the pointer is saved and restored.

- You cannot load a dump that was made on a different platform.

- You cannot load a dump that was generated on a server earlier than version 11.9.

- If a database has cross-database referential integrity constraints, the sysreferences system table stores the name—not the ID number—of the external database. Adaptive Server cannot guarantee referential integrity if you use load database to change the database name or to load it onto a different server.

- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases can cause database corruption. Before dumping a database to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

- `load database` clears the suspect page entries pertaining to the loaded database from `master..sysattributes`.

- `load database` overwrites any existing data in the database.

- After a database dump is loaded, two processes may require additional time before the database can be brought online:

  - Backup Server zeroes the non-allocated pages that are in the source database’s space map. This zeroing is embedded as part of the physical load, and happens during the load database.

    If the target database is larger than the source, then the space above the ceiling of the source database’s space map is zeroed by Adaptive Server after Backup Server has completed the load.
R

- Recovery ignores transactions that completed before the checkpoint that was written by `dump database` at the start of its operation. Completed transactions in the active portion of the transaction log are rolled forward by recovery. In a load sequence, rollback of incomplete transactions happens at the end of that sequence, under `online database`.

- The receiving database must be as large as or larger than the database to be loaded. If the receiving database is too small, Adaptive Server displays an error message that gives the required size.

- You cannot load from the null device (on UNIX, `/dev/null`).

- You cannot use `load database` in a user-defined transaction.

- Once you load a database, Adaptive Server automatically identifies the endian type on the dump file and performs all necessary conversions while the `load database` and `online database` commands execute.

After Adaptive Server converts the index rows, the order of index rows may be incorrect. Adaptive Server marks the following indexes on user tables as suspect indexes during execution of `online database`:

- Nonclustered index on APL (all pages locked) table
- Clustered index on DOL (data-only locked) table
- Nonclustered index on DOL table

During cross-platform dump and load operations, suspect partitions are handled as follows:

- During the first `online database` command, after you execute `load database` across two platforms with different endian types, the hash partition is marked suspect.

- Any global clustered index on a round-robin partition, which has an internally generated partition condition with a `unichar` or `univarchar` partition key, is marked suspect.

- After the database is online, use `sp_post_xpload` to fix the suspect partitions and indexes.

**Note** See Chapter 1, “System Procedures,” in *Reference Manual: Procedures* for information about checking and rebuilding indexes on user tables using the `sp_post_xpload` stored procedure.

- `dump transaction` and `load transaction` are not allowed across platforms.
• dump database and load database to or from a remote backup server are not supported across platforms.

• You cannot load a password-protected dump file across platforms.

• If you perform dump database and load database for a parsed XML object, you must parse the text again after the load database command has completed.

• You cannot perform dump database and load database across platforms on Adaptive Servers versions earlier than 11.9.

• Adaptive Server cannot translate embedded data structures stored as binary, varbinary, or image columns.

• load database is not allowed on the master database across platforms.

• Stored procedures and other compiled objects are recompiled from the SQL text in syscomments at the first execution after the load database.

If you do not have permission to recompile from text, then the person who does has to recompile from text using dbcc upgrade_object to upgrade objects.

Locking out users during loads
• A database cannot be in use while it is being loaded. load database sets the status of the database to “offline.” No one can use the database while its status is “offline.” The “offline” status prevents users from accessing and changing the database during a load sequence.

• A database loaded by load database remains inaccessible until online database is issued.

Upgrading database and transaction log dumps
• To restore and upgrade a user database dump from a version 11.9 or later server to the current version of Adaptive Server:
  a Load the most recent database dump.
  b Load, in order, all transaction log dumps made since the last database dump.

Adaptive Server checks the timestamp on each dump to make sure that it is being loaded to the correct database and in the correct sequence.

  c Issue online database to do the upgrade and make the database available for public use.
Dump the newly upgraded database immediately after upgrade, to create a dump consistent with the current version of Adaptive Server.

Specifying dump devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You can specify a local device as:
  - A logical device name from the `sysdevices` system table
  - An absolute path name
  - A relative path name

The Backup Server resolves relative path names using the current working directory in Adaptive Server.

- When loading across the network, specify the absolute path name of the dump device. The path name must be valid on the machine on which the Backup Server is running. If the name includes characters other than letters, numbers, or the underscore (_), enclose the entire name in quotes.
- Ownership and permissions problems on the dump device may interfere with use of `load` commands.
- You can run more than one load (or dump) at the same time, as long as each load uses a different physical device.

Backup Servers

- You must have a Backup Server running on the same machine as Adaptive Server. The Backup Server must be listed in the `master..sysservers` table. This entry is created during installation or upgrade; do not delete it.
- If your backup devices are located on another machine, so that you load across a network, you must also have a Backup Server installed on the remote machine.

Volume names

- Dump volumes are labeled according to the ANSI tape labeling standard. The label includes the logical volume number and the position of the device within the stripe set.
• During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

**Note** When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

**Changing dump volumes**

• If the Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies the Backup Server by executing `sp_volchanged` on any Adaptive Server that can communicate with the Backup Server.

**Restoring the system databases**

• See the *System Administration Guide* for step-by-step instructions for restoring the system databases from dumps.

**Disk mirroring**

• At the beginning of a load, Adaptive Server passes Backup Server the primary device name of each logical database and log device. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before Backup Server completes its data transfer, Adaptive Server aborts the load.

• If you attempt to unmirror any named device while a load database is in progress, Adaptive Server displays a message. The user executing `disk unmirror` can abort the load or defer the disk unmirror until after the load completes.

• Backup Server loads the data onto the primary device, then `load database` copies it to the secondary device. `load database` takes longer to complete if any database device is mirrored.

**Materializing an archive database**

An archive database is a placeholder that is useful only once it has been loaded with a database dump. The load process does not actually copy pages, however, it materializes the database using page mapping.

**Note** You do not need to have Backup Server running when loading a database dump into an archive database.
Using load database with norecovery  The with norecovery option of the load database command allows a database dump to be loaded into an archive database without recovering anything, reducing the time required to load. Many database pages can be modified or allocated during recovery, causing them to be stored in the modified pages section. Therefore, skipping recovery consumes minimum space in the modified pages section. The with norecovery option allows a quick view into an archive database.

If you use with norecovery, the database is brought online automatically.

However, using load database with norecovery for a database that requires recovery may leave it transactionally and physically inconsistent. Running dbcc checks on a physically inconsistent database may produce many errors.

Once you have loaded an archive database with norecovery, you must have sa_role or database owner privileges to use it.

Using logical devices with an archive database  You can use sp_addumpdevice to create a logical device from which an archive database can be loaded:

```
sp_addumpdevice 'archive database', 'logical_name', 'physical_name'
```

After you have executed this command, use the `logical_name` instead of the `physical_name` as the `dump_device` or `stripe_device` in a load database command.

**Note**  You cannot use an archive database logical device as a device specification for a load into a traditional database or when dumping a traditional database.

**load database limitations with an archive database**  load database has the following limitations when used with an archive database:

- The database dump for an archive database is required to be a disk dump on a file system mounted on the local machine. This can be local storage or NFS storage. `load database ... at <remote server>` syntax is not supported, nor are database dumps on tape.

- Cross-architecture loads are not supported. The database dump and the load database command must be performed on the same architecture with respect to byte ordering.

- The dumped database must have the same page size as that used by the server that is hosting the archive database.
• The major version of the server on which the dump was taken must be earlier than or equal to the major version of the server hosting the archive database.

• The character set and sort order on the server on which the database dump was taken must be the same as the character set and sort order of the server hosting the archive database.

**load database and encrypted columns**

If you store keys in a database that is separate from the columns encrypted by those keys, you must load both databases from dumps that were made simultaneously, avoiding a problem where the encrypted column’s key is missing after the load.

After loading the databases for keys and data, bring both databases on line simultaneously.

You should not load your key database into a database with a different name because metadata dependencies exist between encrypted columns and their keys. If you must change the name of the key database:

1. Before dumping the database containing the encrypted columns, use `alter table` to decrypt the data.

2. Dump the databases containing keys and encrypted columns.

3. After loading the databases, use `alter table` to re-encrypt the data with the keys in the newly-named database.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Only a System Administrator, Database Owner, or user with the Operator role can execute `load database`.

**Auditing**

Values in event and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 43    | load         | `load database`           | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if `set proxy` in effect |
See also


Commands  alter database, dbcc, dump database, dump transaction, load transaction, online database

System procedures  sp_helpdb, sp_helpdevice, sp_hidetext, sp_volchanged
load transaction

Description
Loads a backup copy of the transaction log that was created with dump transaction.

Syntax

**Makes a routine log load:**
```sql
load transaction database_name
   from [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]
   [stripe on [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]]
   [[stripe on [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]]...]
   [with {]
   [density = density_value,]
   [blocksize = number_bytes,]
   [compression,]
   [dumpvolume = volume_name,]
   [file = file_name,]
   [dismount | nodismount],
   [nounload | unload],
   [notify = {client | operator_console}]
   [}]]
```

**Returns header or file information without loading the backup log:**
```sql
load transaction database_name
   from [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]
   [stripe on [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]]
   [[stripe on [compress::]stripe_device
   [at backup_server_name]
   [density = density_value,]
   [blocksize = number_bytes,]
   [dumpvolume = volume_name,]
   [file = file_name]]...]
   [with {]
   [density = density_value,]
   [blocksize = number_bytes,]
   [compression,]
   [dumpvolume = volume_name,]
   [file = file_name,]
   [dismount | nodismount],
   [nounload | unload],
   [notify = {client | operator_console}]
   [}]}
```
Denotes the name of the database to receive data from a dumped backup copy of the transaction log. The log segment of the receiving database must be at least as large as the log segment of the dumped database. The database name can be specified as a literal, a local variable, or a parameter of a stored procedure. For archive databases, database_name is the archive database into which you are loading the transaction log.

*compress::* invokes the decompression of the archived transaction log. See Chapter 27, “Backing Up and Restoring User Databases” in the System Administration Guide for more information about the compress option.

**Note** Sybase recommends the native "compression = compress_level" option as preferred over the older "compress::compression_level" option. If you use the native option for dump database, you do not need to use "compress::compression_level" when loading your database.

*from stripe_device* is the name of the dump device from which you are loading the transaction log. For information about the form to use when specifying a dump device, see “Specifying dump devices” on page 441. For a list of supported dump devices, see the Adaptive Server installation and configuration guides.
at \texttt{backup\_server\_name}

is the name of a remote Backup Server running on the machine to which the
dump device is attached. For platforms that use interfaces files, the
\texttt{backup\_server\_name} must appear in the interfaces file.

from \texttt{dump\_device}

is the local disk transaction log dump.

density = \texttt{density\_value}

overrides the default density for a tape device. \textbf{This option is ignored.}

\texttt{blocksize = number\_bytes}

overrides the default block size for a dump device. If you specify a block
size on UNIX systems, it should be identical to that used to make the dump.

dumpvolume = \texttt{volume\_name}

is the volume name field of the ANSI tape label. load transaction checks this
label when the tape is opened and generates an error message if the wrong
volume is loaded.

\texttt{file = file\_name}

is the name of a particular database dump on the tape volume. If you did not
record the dump file names when you made the dump, use \texttt{listonly} to display
information about all the dump files.

\texttt{stripe on stripe\_device}

is an additional dump device. You can use up to 32 devices, including the
device named in the to \texttt{stripe\_device} clause. The Backup Server loads data
from all devices concurrently, reducing the time and the number of volume
changes required. See “Specifying dump devices” on page 441 for
information about how to specify a dump device.

\texttt{compression}

indicates that the log you are loading was compressed to a file on a remote
server. You do not need to specify the compression level for load transaction.

The \texttt{with compression} option differs from the \texttt{compress} option, which you
use to load a compressed log from a local file.

\begin{itemize}
\item \textbf{Note} Sybase recommends the native \texttt{"compression = compress\_level"} option as
preffered over the older \texttt{"compress::compression\_level"} option. If you use the
native option for dump database, you do not need to use
\texttt{"compress::compression\_level"} when loading your database.
\end{itemize}
load transaction

dismount | nodismount

*on platforms that support logical dismount* – determines whether tapes remain mounted. By default, all tapes used for a load are dismounted when the load completes. Use nodismount to keep tapes available for additional loads or dumps.

nounload | unload

determines whether tapes rewind after the load completes. By default, tapes do not rewind, allowing you to make additional loads from the same tape volume. Specify unload for the last dump file to be loaded from a multidump volume. This rewinds and unloads the tape when the load completes.

listonly [= full]

displays information about all the dump files on a tape volume, but does not load the transaction log. listonly identifies the database and device, the date and time the dump was made, and the date and time it can be overwritten. listonly = full provides additional details about the dump. Both reports are sorted by ANSI tape label.

After listing the files on a volume, the Backup Server sends a volume change request. The operator can either mount another tape volume or terminate the list operation for all dump devices.

In the current implementation, listonly overrides headeronly.

**Warning!** Do not use load transaction with listonly on 1/4-inch cartridge tape.
headeronly displays header information for a single dump file, but does not load the database. headeronly displays information about the first file on the tape unless you use the file = file_name option to specify another file name. The dump header indicates:

- Type of dump (database or transaction log)
- Database ID
- File name
- Date the dump was made
- Character set
- Sort order
- Page count
- Next object ID
- Checkpoint location in the log
- Location of the oldest begin transaction record
- Old and new sequence dates

notify = {client | operator_console} overrides the default message destination.

- On operating systems that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use client to route other Backup Server messages to the terminal session that initiated the dump database.

- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that initiated the dump database. Use operator_console to route messages to the terminal on which the Backup Server is running.

until_time loads the transaction log up to a specified time in the transaction log. Only transactions committed before the specified time are saved to the database.

Examples

Example 1 Loads the transaction log for the database pubs2 tape:

```sql
load transaction pubs2 from "/dev/nrmt0"
```
load transaction

Example 2 Loads the transaction log for the pubs2 database, using the Backup Server REMOTE_BKP_SERVER:

```sql
load transaction pubs2
  from "/dev/nrmt4" at REMOTE_BKP_SERVER
  stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
  stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
```

Example 3 Loads the transaction log for pubs2, up to March 20, 1997, at 10:51:43:866 a.m:

```sql
load transaction pubs2
  from "/dev/ntmt0"
  with until_time = "mar 20, 1997 10:51:43:866am"
```

Usage

- If you use sp_hidetext followed by a cross-platform dump and load, you must manually drop and re-create all hidden objects.
- The listonly and headeronly options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
- Table 1-30 describes the commands and system procedures used to restore databases from backups:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>create database for load</td>
<td>Create a database for the purpose of loading a dump.</td>
</tr>
<tr>
<td>load database</td>
<td>Restore a database from a dump.</td>
</tr>
<tr>
<td>load transaction</td>
<td>Apply recent transactions to a restored database.</td>
</tr>
<tr>
<td>online database</td>
<td>Make a database available for public use after a normal load sequence or after upgrading the database to the current version of Adaptive Server.</td>
</tr>
<tr>
<td>load (database</td>
<td>transaction) with (headeronly</td>
</tr>
</tbody>
</table>

| sp_volchanged         | Respond to the Backup Server volume change messages. |

Restrictions

- You cannot load a dump that was generated on a version earlier than 11.9 server.
- The database and transaction logs must be at the same release level.
- Load transaction logs in chronological order.
- You cannot load from the null device (on UNIX, /dev/null).
• You cannot use load transaction after an online database command that performs an upgrade. The correct sequence for upgrading a database is load database, load transaction, online database.

• Do not issue online database until all transaction logs are loaded. The command sequence is:
  a. Load database
  b. Load transaction (repeat as needed)
  c. Online database

However, to load additional transaction logs while retaining read-only access to the database (a typical “warm backup” situation), use the dump tran for standby_access option to generate the transaction dumps. You can then issue online database for standby_access for read-only access.

• You cannot use the load transaction command in a user-defined transaction.

Restoring a database
• To restore a database:
  • Load the most recent database dump
  • Load, in order, all transaction log dumps made since the last database dump
  • Issue online database to make the database available for public use

  Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump both of the affected databases.

**Warning!** Loading earlier dumps of these databases can cause database corruption.

For more information on backup and recovery of Adaptive Server databases, see the *System Administration Guide*.

Recovering a database to a specified time
• You can use the until_time option for most databases that can be loaded or dumped. until_time does not apply to databases such as master, in which the data and logs are on the same device. Also, you cannot use it on any database that has had a truncated log since the last dump database, such as tempdb.
The `until_time` option is useful for the following reasons:

- It enables you to have a database consistent to a particular time. For example, in an environment with a decision-support system (DSS) database and an online transaction processing (OLTP) database, the System Administrator can roll the DSS database to an earlier specified time to compare data between the earlier version and the current version.

- If a user inadvertently destroys data, such as dropping an important table, you can use the `until_time` option to back out the errant command by rolling forward the database to a point just before the data was destroyed.

- To effectively use the `until_time` option after data has been destroyed, you must know the exact time the error took place. You can find out by executing a `select getdate ()` command immediately after the error. For a more precise time using milliseconds, use the `convert` function, for example:

```
select convert (char (26), getdate (), 109)
--------------------------
Feb 26 1997 12:45:59:650PM
```

- After you load a transaction log using `until_time`, Adaptive Server restarts the database’s log sequence. This means that until you dump the database again, you cannot load subsequent transaction logs after the load transaction using `until_time`. Dump the database before you dump another transaction log.

- Only transactions that committed before the specified time are saved to the database. However, in some cases, transactions committed shortly after the `until_time` specification are applied to the database data. This may occur when several transactions are committing at the same time. The ordering of transactions may not be written to the transaction log in time-ordered sequence. In this case, the transactions that are out of time sequence are reflected in the data that has been recovered. The time should be less than a second.

- For more information on recovering a database to a specified time, see the System Administration Guide.
Locking users out during loads

- A database cannot be in use when you load it. You cannot load a database that is already in use. `load transaction`, unlike `load database`, does not change the offline/online status of the database. `load transaction` leaves the status of the database the way it found it. `load database` sets the status of the database to “offline.” No one can use the database while it is “offline.” The “offline” status prevents users from accessing and changing the database during a load sequence.

- A database loaded by `load database` remains inaccessible until `online database` is issued.

Upgrading database and transaction log dumps

- To restore and upgrade a user database dump from a version 11.9 or later server to the current version of Adaptive Server:
  
  a. Load the most recent database dump.
  
  b. Load, in order, all transaction logs generated after the last database dump.
  
  c. Use `online database` to do the upgrade.
  
  d. Dump the newly upgraded database immediately after the upgrade, to create a dump that is consistent with the current version of Adaptive Server.

Specifying dump devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.

- When loading from a local device, you can specify the dump device as:
  
  a. An absolute path name
  
  b. A relative path name
  
  c. A logical device name from the `sysdevices` system table

  Backup Server resolves relative path names, using the current working directory in the Adaptive Server.

- When loading across the network, specify the absolute path name of the dump device. (You cannot use a relative path name or a logical device name from the `sysdevices` system table.) The path name must be valid on the machine on which the Backup Server is running. If the name includes any characters other than letters, numbers or the underscore (_), you must enclose it in quotes.
Ownership and permissions problems on the dump device may interfere with use of load commands. `sp_addumpdevice` adds the device to the system tables, but does not guarantee that you can load from that device or create a file as a dump device.

You can run more than one load (or dump) at the same time, as long as each one uses a different physical device.

Backup Servers

- You must have a Backup Server running on the same machine as your Adaptive Server. The Backup Server must be listed in the `master..sys.servers` table. This entry is created during installation or upgrade and should not be deleted.

- If your backup devices are located on another machine so that you load across a network, you must also have a Backup Server installed on the remote machine.

Volume names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.

- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

  *Note* When dumping and loading across a network, you must specify the same number of stripe devices for each operation.

Changing dump volumes

- If Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies Backup Server by executing `sp_volchanged` on any Adaptive Server that can communicate with Backup Server.

Restoring the system databases

- For step-by-step instructions for restoring the system databases from dumps, see the *System Administration Guide*. 
CHAPTER 1  Commands

Disk mirroring

- At the beginning of a load, Adaptive Server passes the primary device name of each logical database device and each logical log device to the Backup Server. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before the Backup Server completes its data transfer, Adaptive Server aborts the load.

- If you attempt to unmirror any of the named devices while a load transaction is in progress, Adaptive Server displays a message. The user executing disk unmirror can abort the load, or defer disk unmirror until after the load completes.

- Backup Server loads the data onto the primary device, then load transaction copies it to the secondary device. load transaction takes longer to complete if any database device is mirrored.

Loading a transaction log into an archive database

When you load a transaction log into an archive database, load transaction runs the recovery redo pass. Modified and new database pages are written to the permanent changes segment. You must have enough space in the modified pages section to accommodate these changes. If necessary, increase space for the modified pages section by using alter database to increase the normal database storage allocated to the archive database.

Unlike a traditional database, an archive database can be brought online in the middle of a load sequence without breaking the load sequence. When a traditional database is loaded and then brought online without using the for standby_access clause, it is no longer possible to load the next transaction log in the load sequence. An archive database however, can be brought online without the for standby_access clause and later, loaded with the next transaction log in the load sequence. This allows read-only operations like running consistency checks, at any time during the load sequence. This is possible because when loading a transaction log into the archive database, Adaptive Server automatically removes the disposable changes segment from the modified pages section. This effectively reverts the archive database to its state after the previous load was done, thereby allowing the next transaction log in the sequence to be loaded.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
load transaction permission defaults to the Database Owner and operators. It is not transferable.

Auditing
Values in event and extrainfo columns of sysaudits are:
### load transaction

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 44    | load         | load transaction          | • Roles – Current active roles  
|       |              |                           | • Keywords or options – NULL  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – NULL  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect |

See also

**Documents**  Chapter 28, “Backing Up and Restoring User Databases” in the *System Administration Guide*.

**Commands**  disk unmirror, dump database, dump transaction, load database, online database

**System procedures**  sp_dboption, sp_helpdb, sp_helpdevice, sp_hidetext, sp_volchanged
**lock table**

Description
Explicitly locks a table within a transaction.

Syntax
`lock table table_name in {share | exclusive} mode [wait [numsecs] | nowait]`

Parameters
- `table_name`
  specifies the name of the table to be locked.
- `share | exclusive`
  specifies the type of lock, shared or exclusive, to be applied to the table.
- `wait numsecs`
  specifies the number of seconds to wait, if a lock cannot be acquired immediately. If `numsecs` is omitted, specifies that the `lock table` command should wait until lock is granted.
- `nowait`
  causes the command to fail if the lock cannot be acquired immediately.

Examples

**Example 1** Tries to acquire a shared table lock on the titles table. If a session-level wait has been set with `set lock wait`, the `lock table` command waits for that period of time; otherwise, the server-level wait period is used:

```sql
begin transaction
lock table titles in share mode
```

**Example 2** Tries to acquire an exclusive table lock on the authors table. If the lock cannot be acquired within 5 seconds, the command returns an informational message. Subsequent commands within the transaction continue as they would have without `lock table`:

```sql
begin transaction
lock table authors in exclusive mode wait 5
```

**Example 3** If a table lock is not acquired within 5 seconds, the procedure checks the user’s role. If the procedure is executed by a user with `sa_role`, the procedure prints an advisory message and proceeds without a table lock. If the user does not have `sa_role`, the transaction is rolled back:

```sql
create procedure bigbatch
as
begin transaction
lock table titles in share mode wait 5
if @@error = 12207
begin
/*
** Allow SA to run without the table lock
** Other users get an error message
```
lock table

/*
 * if (proc_role ("sa_role") = 0)
 * begin
 * print "You cannot run this procedure at
 * this time, please try again later"
 * rollback transaction
 * return 100
 * end
 * else
 * begin
 * print "Couldn't obtain table lock,
 * proceeding with default locking."
 * end
 * end
*/ more SQL here */
commit transaction

Usage

- You can use lock table with an archive database.
- If you use lock table as the first statement after the set chained on command, this creates a new transaction.
- You can use lock table only within a transaction. The table lock is held for the duration of the transaction.
- The behavior of lock table depends on the wait-time options that are specified in the command or that are active at the session level or server level.
- If the wait and nowait option are not specified, lock table uses either the session-level wait period or the server-level wait period. If a session-level wait has been set using set lock wait, it is used, otherwise, the server-level wait period is used.
- If the table lock cannot be obtained with the time limit (if any), the lock table command returns message 12207. The transaction is not rolled back. Subsequent commands in the transaction proceed as they would have without the lock table command.
- You cannot use lock table on system tables or temporary tables.
- You can issue multiple lock table commands in the same transaction.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

You must have select access permission on the table to use lock table in share mode. You must have delete, insert, or update access permission on the table to use lock table in exclusive mode.

See also

Commands set
mount

Description

Use the mount command to attach a database to a destination or secondary Adaptive Server. The mount command decodes the information in the **manifest file** and makes the set of databases available. The mount command differs from other copying procedures such as the bcp bulk copy utility in that all required supporting activities are executed, including adding database devices, if necessary, and activating them, creating the catalog entries for the new databases, and recovering them.

If you are using different device names at the destination Adaptive Server when mounting the databases, use `mount with listonly` and modify the device path names at the destination server. Then use `mount` to actually mount the databases.

**Note** For every login that is allowed access to a database on the original Adaptive Server, it is more convenient to have a corresponding login for the same suid at the destination Adaptive Server, as this avoids user ID reconciliation issues.

For permissions to remain unchanged, the login maps at the destination Adaptive Server must be identical to that on the source Adaptive Server. For more information on login maps, see Chapter 15, “Managing Remote Servers” in *System Administration Guide, Volume 1*.

Syntax

```plaintext
mount database all | database_mapping[, database_mapping, ...]
from "manifest_file"
[using device_mapping[, device_mapping,...]
[with listonly]

database_mapping:
origdbname as newdbname
| newdbname = origdbname
| origdbname
| newdbname

device_mapping:
logical_device_name as new_physical_name
| new_physical_name = logical_device_name
| original_physical_name
| new_physical_name
```
mount

Parameters

*manifest_file*

The manifest file is the binary file that describes the databases that are present on a set of database devices.

Operations that can perform character translations of the file contents (such as ftp) corrupt the manifest file unless performed in binary mode.

Examples

**Example 1** Finds the path names listed on the manifest file from the source Adaptive Server:

```sql
mount database all from "/data/sybase2/mfile1" with listonly
```

```sql
go
```

```sql
[database]
mydb
```

```sql
[device]
"/data/sybase1/d0.dbs" = "1dev1"
"/data/sybase2/d14.dbs" = "1dev13"
```

When you use the path names different from the source ones, verify or modify them to meet your criteria at the destination Adaptive Server.

**Example 2** After the database devices are copied to the secondary Adaptive Server, you then mount it:

```sql
mount database all from "/data/sybase2/mfile1" using
"/data/sybase2/d0.dbs" = "1dev1",
"/data/sybase2/d14.dbs" = "1dev13"
```

When the mount process has completed, the database or databases are still offline. Use the online database command to bring them online. You need not restart the server.

**Example 3** The destination server can be the same as the source server. In this case, the database names must be mapped to a different name, and the logical device names will be internally renamed.

1. Create an exact copy of database `mydb` in the same server:

   ```bash
   1> quiesce database mydb_tag hold mydb for external dump to 
   "data/mydb.manifest"
   2> go
   ```

2. Copy the OS file:

   ```bash
   $ cp /data/sybase2/mydb.dbs /data/sybase2/mydb_copy.dbs
   ```

3. You can now mount it as a copy:

   ```bash
   1> quiesce database mydb_tag release
   2> go
   ```
1> mount database mydb as mydb_copy
2> from "/data/mydb.manifest"
3> using mydb_dev as "/data/sybase2/mydb_copy.dbs"
3> go

The physical device //data/sybase2/mydb_copy.dbs/ is automatically assigned a machine-generated logical name with the format $\text{Cccc}$\$<\text{mydb_dev}>$ where:

- $C$ – is [A–Z]
- $c$ – is [A–Z, 0–9], and refers to the encoded logical device number
- $\text{mydb}_\text{dev}$ – contains up to 26 characters from the old logical device name.

Database IDs for the transported databases should not exist on the destination Adaptive Server. Because the database has been mounted on the same server, the database ID had to be changed. The allocation pages in the mounted device keep the original database ID, and that information is used by the disk refit command. Use the dbcc checkalloc command to reconcile the dbid after running mount database so that disk refit can work on the mounted devices. Run checkalloc if the database is not being mounted for temporary use.

**Usage**

- The using clause allows you to define a mapping via the “=” sign or the “as” clause.

- If there are more than one device, a mapping can be one using “=” and another using “as.”

- In both databases and devices, you can map devices by name, specifying both logical and physical, and by order. If a database is mapped by name, all databases must be mapped by name and vice versa. The same happens for devices.

**Destination changes**

Once databases are mounted on the destination Adaptive Server, certain settings are cleared on the mounted database:

- Replication is turned off.
- Audit settings are cleared and turned off.
- Component Integration Services options, default remote location, and type are cleared.
- Cache bindings are dropped for both the mounted databases and their objects.
- Recovery order is dropped for the mounted databases and becomes the default dbid order.
System considerations
- You cannot use the mount command in a transaction.
- You cannot mount a database on server configured for high availability.

Performance considerations
When you mount databases onto an Adaptive Server, if you change the dbid of
the database you are mounting, all procedures are marked for recompilation in
the database. This increases the time it takes to recover the database at the
destination, and delays the first execution of the procedure.

Renaming devices
The manifest file contains the device paths known to the source Adaptive
Server that created the manifest file. If the destination Adaptive Server
accesses the devices with a different path, you can specify the new path to the
mount command.

1 Use the mount command with listonly to display the old path:

```sql
mount database all from "/work2/Mpubs_file" with listonly
go
```

```plaintext
[database]
mydb
[device]
"/work2/Devices/pubsdat.dat" = "pubs2dat"
```

2 If the new path for the device pubs2dat is `/work2/Devices/pubsdevice.dat`
   (the devices path in Windows), specify the new device in the mount
   command:

```sql
mount database all from "/work2/Mpubs_file" using
"/work2/datadevices/pubsdevice.dat" = "pubs2dat"
```

If the logical device names exist in the destination server, they will be
renamed using an automatically generated unique name.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.
Permissions mount requires an SA role.
Auditing  Values in event and extrainfo columns of sysaudits are:
### Event Audit option Command or access audited Information in extrainfo

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 101   | mount        | mount database            | • *Roles* – Current active roles  
                              |                           | • *Keywords or options* – NULL  
                              |                           | • *Previous value* – NULL  
                              |                           | • *Current value* – NULL  
                              |                           | • *Other information* – NULL  
                              |                           | • *Proxy information* – Original login name, if a set proxy is in effect |

See also **Commands** unmount, quiesce database

**Documentation** Chapter 7, “Database Mount and Unmount,” in *System Administration Guide Volume 2*
online database

Description
Marks a database available for public use after a normal load sequence; if needed, upgrades a loaded database to the current version of Adaptive Server; brings a database online after loading a transaction log dumped with the for standby_access option. You can also use online database to bring an archive database online.

Syntax
online database database_name [for standby_access]

Parameters
database_name
specifies the name of the database to be brought online.

for standby_access
brings the database online on the assumption that the database contains no open transactions.

Examples
Example 1 Makes the pubs2 database available for public use after a load sequence completes:

    online database pubs2

Example 2 Brings the database inventory_db online. Used after loading inventory_db with a transaction-log dump obtained through dump tran...with standby_access:

    online database inventory_db for standby_access

Usage
• online database brings a database online for general use after a normal database or transaction log load sequence.

• When load database is issued, the database’s status is set to “offline.” The offline status is set in the sysdatabases system table and remains set until online database completes.

• Do not issue online database until all transaction logs are loaded. The command sequence is:
  • load database
  • load transaction (there may be more than one load transaction)
  • online database

• If you execute online database against a currently online database, no processing occurs and no error messages are generated.
• You can only use online database...for standby_access with a transaction log that was dumped using dump transaction...with standby_access. If you use online database...for standby_access after loading a transaction log that was dumped without using dump transaction...with standby access, online database generates an error message and fails.

• You can use sp_helpdb to find out whether a database is currently online, online for standby access, or offline.

Upgrading databases

• online database initiates, if needed, the upgrade of a loaded database and transaction log dumps to make the database compatible with the current version of Adaptive Server. After the upgrade completes, the database is made available for public use. If errors occur during processing, the database remains offline.

• online database is required only after a database or transaction log load sequence. It is not required for new installations or upgrades. When you upgrade Adaptive Server to a new version, all databases associated with that server are automatically upgraded.

• online database only upgrades version 11.9 or later user databases.

• After you upgrade a database with online database, dump the newly upgraded database to create a dump that is consistent with the current version of Adaptive Server. You must dump the upgraded database before you can issue a dump transaction command.

Archive databases

The online database database_name command performs undo recovery during which modified and allocated pages may be remapped to the modified pages section.

You do not need to bring a database online if it has been loaded with norecovery, since the load automatically brings the database online without running the recovery undo pass.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Only a System Administrator, Database Owner, or user with the Operator role can execute online database.

Auditing Values in event and extrainfo columns of sysaudits are:
See also

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 83    | security     | online database           | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

**Commands** dump database, dump transaction, load database, load transaction

**System procedures** sp_helpdb
open

Description
Opens a cursor for processing.

Syntax
open cursor_name

Parameters
cursor_name
is the name of the cursor to open.

Examples
Opens the cursor named authors_crsr:

```
open authors_crsr
```

Usage
- open opens a cursor. Cursors allow you to modify or delete rows on an individual basis. You must first open a cursor to use the fetch, update, and delete statements. For more information about cursors, see the Transact-SQL User's Guide.
- Adaptive Server returns an error message if the cursor is already open or if the cursor has not been created with the declare cursor statement.
- Opening the cursor causes Adaptive Server to evaluate the select statement that defines the cursor (specified in the declare cursor statement) and makes the cursor result set available for processing.
- When the cursor is first opened, it is positioned before the first row of the cursor result set.
- You can use open with an archive database.
- When you set the chained transaction mode, Adaptive Server implicitly begins a transaction with the open statement if no transaction is currently active.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
open permission defaults to all users.

See also
Commands close, declare cursor, fetch
order by clause

Description

Returns query results in the specified columns in sorted order.

Syntax

```
[Start of select statement]

order by
    {[table_name,| view_name,]
        column_name | select_list_number | expression}
    [asc | desc]
    ,{[table_name,| view_name,]
        column_name | select_list_number | expression}
    [asc | desc][...]

[End of select statement]
```

Parameters

- `order by` sorts the results by columns.
- `asc` sorts the results in ascending order. If you do not specify `asc` or `desc`, `asc` is assumed.
- `desc` sorts the results in descending order.

Examples

**Example 1** Selects the titles whose price is greater than $19.99 and lists them with the titles in alphabetical order:

```
select title, type, price
from titles
where price > $19.99
order by title
```

<table>
<thead>
<tr>
<th>title</th>
<th>type</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>But Is It User Friendly?</td>
<td>popular_comp</td>
<td>22.95</td>
</tr>
<tr>
<td>Computer Phobic and Non-Phobic Individuals: Behavior Variations</td>
<td>psychology</td>
<td>21.59</td>
</tr>
<tr>
<td>Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean</td>
<td>trad_cook</td>
<td>20.95</td>
</tr>
<tr>
<td>Secrets of Silicon Valley</td>
<td>popular_comp</td>
<td>20.00</td>
</tr>
</tbody>
</table>

**Example 2** Lists the books from the `titles` table, in descending alphabetical order of the type, and calculates the average price and advance for each type:

```
select type, price, advance
```
compute avg (price), avg (advance) by type

data

Example 3  Lists the title IDs from the titles table, with the advances divided by the total sales, ordered from the lowest calculated amount to the highest:

```
select title_id, advance/total_sales
from titles
order by advance/total_sales
```

```
title_id
-------- ------------------------
MC3026    NULL
PC9999    NULL
MC2222    0.00
TC4203    0.26
PS3333    0.49
BU2075    0.54
MC3021    0.67
PC1035    0.80
PS2091    1.11
PS7777    1.20
BU1032    1.22
BU7832    1.22
BU1111    1.29
PC8888    1.95
TC7777    1.95
PS1372    18.67
TC3218    18.67
PS2106    54.05
```
order by clause

- If you sort by select_list_number, the columns to which the order by clause refers must be included in the select list, and the select list cannot be * (asterisk).
- Use order by to display your query results in a meaningful order. Without an order by clause, you cannot control the order in which Adaptive Server returns results.

Restrictions
- The maximum number of columns allowed in an order by clause is 31.
- You cannot use order by on text, unitext, or image datatype columns.
- Subqueries and view definitions cannot include an order by clause (or a compute clause or the keyword into). Conversely, you cannot use a subquery in an order by list.
- You cannot update the result set of a server- or language- type cursor if it contains an order by clause in its select statement. For more information about the restrictions applied to updatable cursors, see the Transact-SQL User’s Guide.
- If you use compute by, you must also use an order by clause. The expressions listed after compute by must be identical to or a subset of those listed after order by, must be in the same left-to-right order, must start with the same expression, and must not skip any expressions. For example, if the order by clause is:

```
order by a, b, c
```

the compute by clause can be any (or all) of these:

```
compute by a, b, c
compute by a, b
compute by a
```

You can also use the keyword compute can be used without by to generate grand totals, grand counts, and so on. In this case, order by is optional.

Collating sequences
- With order by, null values precede all others.
- The sort order (collating sequence) on your Adaptive Server determines how your data is sorted. The sort order choices are binary, dictionary, case-insensitive, case-insensitive with preference, and case- and accent-insensitive. Sort orders that are specific to national languages may also be provided.
Table 1-31: Effect of sort order choices

<table>
<thead>
<tr>
<th>Adaptive Server sort order</th>
<th>Effects on order by results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary order</td>
<td>Sorts all data according to the numeric byte-value of each character in the character set. Binary order sorts all uppercase letters before lowercase letters. Binary sort order is the only option for multibyte character sets.</td>
</tr>
<tr>
<td>Dictionary order</td>
<td>Sorts uppercase letters before their lowercase counterparts (case-sensitive). Dictionary order recognizes the various accented forms of a letter and sorts them after the unaccented form.</td>
</tr>
<tr>
<td>Dictionary order, case-insensitive</td>
<td>Sorts data in dictionary order but does not recognize case differences. Uppercase letters are equivalent to their lowercase counterparts and are sorted as described in “Sort rules” next.</td>
</tr>
<tr>
<td>Dictionary order, case-insensitive with preference</td>
<td>Sorts an uppercase letter in the preferred position, before its lowercase version. It does not recognize case difference when performing comparisons (for example, in where clauses).</td>
</tr>
<tr>
<td>Dictionary order, case- and accent-insensitive</td>
<td>Sorts data in dictionary order, but does not recognize case differences; treats accented forms of a letter as equivalent to the associated unaccented letter. This sort order intermingles accented and unaccented letters in sorting results.</td>
</tr>
</tbody>
</table>

- `sp_helpsort` reports the sort order installed on Adaptive Server.

Sort rules

- When two rows have equivalent values in the Adaptive Server sort order, the following rules are used to order the rows:
  - The values in the columns named in the `order by` clause are compared.
  - If two rows have equivalent column values, the binary value of the entire rows is compared byte by byte. This comparison is performed on the row in the order in which the columns are stored internally, not the order of the columns as they are named in the query or in the original `create table` clause. In brief, data is stored with all the fixed-length columns, in order, followed by all the variable-length columns, in order.
  - If rows are equal, row IDs are compared.

Given this table:

```sql
create table sortdemo (lname varchar (20),
                      init char (1) not null)
```

and this data:

```
<table>
<thead>
<tr>
<th>lname</th>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>B</td>
</tr>
<tr>
<td>SMITH</td>
<td>C</td>
</tr>
</tbody>
</table>
```
you get these results when you order by `lname`:

<table>
<thead>
<tr>
<th>lname</th>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>A</td>
</tr>
<tr>
<td>Smith</td>
<td>B</td>
</tr>
<tr>
<td>SMITH</td>
<td>C</td>
</tr>
</tbody>
</table>

Since the fixed-length char data (the init column) is stored first internally, the `order by` sorts these rows based on the binary values “Asmith”, “BSmith,” and “CSMITH”.

However, if the init is of type varchar, the `lname` column is stored first, and then the init column. The comparison takes place on the binary values “SMITHC”, “SmithB”, and “smithA”, and the rows are returned in that order.

### Descending scans

- Use of the keyword `desc` in an `order by` clause allows the query optimizer to choose a strategy that eliminates the need for a worktable and a sort step to return results in descending order. This optimization scans the page chain of the index in reverse order, following the previous page pointers on each index page.

To use this optimization, the columns in the `order by` clause must match the index order. They can be a subset of the keys, but must be a prefix subset, that is, they must include the first keys. You cannot use the descending scan optimization if the columns named in the `order by` clause are a superset of the index keys.

If the query involves a join, all tables can be scanned in descending key order, as long as the requirements for a prefix subset of keys are met. You can also use descending scan optimization for one or more tables in a join, while other tables are scanned in ascending order.

- If other user processes are scanning forward to perform updates or deletes, performing descending scans can cause deadlocks. Deadlocks may also be encountered during page splits and shrinks. You can use `sp_sysmon` to track deadlocks on your server, or you can use the configuration parameter `print deadlock information` to send deadlock information to the error log.

- If your applications must return results in descending order, but the descending scans optimization creates deadlock problems, some possible workarounds are:
• Use set transaction isolation level 0 scans for descending scans. For more information on the effect of isolation level 0 reads, see the set command, and Chapter 4, “Using Locking Commands” in Performance and Tuning Guide: Locking.

• Disable descending scan optimization with the configuration parameter allow backward scans so that all queries that use desc scan the table in ascending order and sort the result set into descending order. For more information, see the System Administration Guide.

• Break problematic descending scans into two steps, selecting the required rows into a temporary table in ascending order in the first step, and selecting from the temporary table in descending order in the second step.

• If a backward scan uses a clustered index that contains overflow pages because duplicate key values are present, the result set returned by the descending scan may not be in exact reverse order of the result set that is returned with an ascending scan. The specified key values are returned in order, but the order of the rows for the identical keys on the overflow pages may be different. For an explanation of how overflow pages in clustered indexes are stored, see Chapter 12, “How Indexes Work” in Performance and Tuning Guide: Basics.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Specifying new column headings in the order by clause of a select statement when the union operator is used is a Transact-SQL extension.

See also Commands compute clause, declare, group by and having clauses, select, where clause

System procedures sp_configure, sp_helpsort, sp_lock, sp_sysmon
prepare transaction

Description  Used by DB-Library in a two-phase commit application to see if a server is prepared to commit a transaction.

Syntax  prepare transaction

Usage  • For more information, see the Open Client DB-Library Reference Manual.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.

See also  Commands: begin transaction, begin transaction, rollback, save transaction
print

Prints a user-defined message on the user’s screen.

Syntax

\[
\text{print} \quad \{ \text{format\_string} | \text{@local\_variable} | \text{@@global\_variable} \} \\
\text{[, arg\_list}]
\]

Parameters

format\_string

can be either a variable or a string of characters. The maximum length of format\_string is 1023 bytes.

Format strings can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow format\_string when the text of the message is sent to the client.

To allow reordering of the arguments when format strings are translated to a language with a different grammatical structure, placeholders are numbered. A placeholder for an argument appears in this format:

```
\text{“ %nn !”}—a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation point (!). The integer represents the argument number in the string in the argument list. “%1!” is the first argument in the original version, “%2!” is the second argument, and so on.
```

Indicating the position of the argument in this way makes it possible to translate correctly, even when the order in which the arguments appear in the target language is different.

For example, assume the following is an English message:

```
%1! is not allowed in %2!.
```

The German version of this message is:

```
%1! ist in %2! nicht zulässig.
```

The Japanese version of this message is:

```
%2! の中で %1! は許されません。
```

In this example, “%1!” represents the same argument in all three languages, as does “%2!”.

This example shows the reordering of the arguments that is sometimes necessary in the translated form.

@local\_variable

must be of type char, nchar, varchar, or nvarchar, and must be declared within the batch or procedure in which it is used.
@@global_variable
must be of type char or varchar, or be automatically convertible to these
types, such as @@version. Currently, @@version is the only character-type
global variable.

arg_list
may be a series of either variables or constants separated by commas. arg_list
is optional unless a format string containing placeholders of the form
"%nn!" is provided. In that case, the arg_list must have at least as many
arguments as the highest numbered placeholder. An argument can be any
datatype except text or image; it is converted to a character datatype before
being included in the final message.

Examples

Example 1 Prints “Berkeley author” if any authors in the authors table live in
the 94705 postal code:

    if exists (select postalcode from authors
        where postalcode = '94705')
    print "Berkeley author"

Example 2 Declares a variable, assigns a value to the variable, and prints the
value:

    declare @msg char (50)
    select @msg = "What's up, doc?"
    print @msg

    What's up, doc?

Example 3 Demonstrates the use of variables and placeholders in messages:

    declare @tabname varchar (30)
    select @tabname = "titles"

    declare @username varchar (30)
    select @username = "ezekiel"

    print "The table '%1!' is not owned by the user '%2!'.", @tabname, @username

    The table 'titles' is not owned
    by the user 'ezekiel.'

Usage

- The maximum output string length of format_string plus all arguments
  after substitution is 1023 bytes.
If you use placeholders in a format string, keep this in mind: for each placeholder \( n \) in the string, the placeholders 1 through \( n-1 \) must also exist in the same string, although they do not have to be in numerical order. For example, you cannot have placeholders 1 and 3 in a format string without having placeholder 2 in the same string. If you omit a number in a format string, an error message is generated when \texttt{print} is executed.

- The \texttt{arg_list} must include an argument for each placeholder in the \texttt{format_string}, or the transaction is aborted. You can use more arguments than placeholders.

- To include a literal percent sign as part of the error message, use two percent signs ("%%") in the \texttt{format_string}. If you include a single percent sign ("%") in the \texttt{format_string} that is not used as a placeholder, Adaptive Server returns an error message.

- If an argument evaluates to NULL, it is converted into a zero-length character string. If you do not want zero-length strings in the output, use the \texttt{isnull} function. For example, if @arg is null, the following statement prints "I think we have nothing here."

  ```sql
  declare @arg varchar (30)
  select @arg = isnull (col1, "nothing") from table_a where ...
  print "I think we have %1! here", @arg
  ```

- You can add user-defined messages to the system table \texttt{sysusermessages} for use by any application. Use \texttt{sp_addmessage} to add messages to \texttt{sysusermessages}; use \texttt{sp_getmessage} to retrieve messages for use by \texttt{print} and \texttt{raiserror}.

- Use \texttt{raiserror} instead of \texttt{print} to print a user-defined error message and have the error number stored in \texttt{@@error}.

---

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

\texttt{print} permission defaults to all users. No permission is required to use it.

**See also**

\textbf{Commands} \quad declare, raiserror

\textbf{System procedures} \quad \texttt{sp_addmessage}, \texttt{sp_getmessage}
**quiesce database**

**Description**
Suspends and resumes updates to a specified list of databases.

**Syntax**
```
quiesce database tag_name hold database_list [for external dump]
[to manifest_file [with override]]
```

or:
```
quiesce database tag_name release
```

**Parameters**
- **tag_name**
  is a user-defined name that designates the list of databases to hold or release.
  The *tag_name* must conform to the rules for identifiers.

- **hold**
  when used with to manifest_file clause, holds the database and creates a manifest file.

**Warning!** Since the manifest file is binary, operations that perform character translations of the file contents (such as ftp) corrupt the file unless performed in binary mode.

- **database_list**
  is the list of the databases included in the quiesce database hold command.

- **for external dump**
  specifies that while updates to the databases in the list are suspended, you will physically copy all affected database devices, using some facility external to Adaptive Server. The copy operation serves as a replacement for the combination of dump database and load database.

- **manifest_file**
  the binary file that describes the databases that are present on a set of database devices. It can be created only if the set of databases that occupy those devices are isolated, self-contained on those devices.

  Since the manifest file is a binary file, operations that can perform character translations of the file contents (such as ftp) will corrupt the file unless performed in binary mode.

- **with override**
  overrides any restrictions that prevent you from successfully executing quiesce database on a database.

**Examples**

**Example 1** Suspends update activity on salesdb and ordersdb:
```
quiesce database report_dbs hold salesdb, ordersdb
```
Example 2  Resumes update activity on the databases labeled report_dbs:

    quiesce database report_dbs release

Example 3  Suspends update activity to the pubs2 database and signifies your intent to make an external copy of this database:

    quiesce database pubs_tag hold pubs2 for external dump

Example 4  Places the database in a hold status and builds the manifest file for a database to be copied to another Adaptive Server:

    quiesce database pubs_tag hold pubs2 for external dump to 
    "/work2/sybase1/mpubs_file", with override

    Once the command completes, control returns to the user.

Example 5  Copies the database devices, using the mount database with listonly to list all of the devices to be copied to view:

1> mount database all from "/data/sybase2/mfile1" with listonly
2> go

"/data/sybase1/d0.dbs" = "1dev1"

    You cannot create a manifest file if the set of databases that are quiesced contain references to databases outside of the set. Use with override option to bypass this restriction:

    quiesce database pubs2_tag release for external dump to Mpubs_file

Usage

- quiesce database used with the hold keyword suspends all updates to the specified database. Transactions cannot update data in suspended databases, and background tasks such as the checkpoint process and housekeeper process skip all databases that are in the suspended state.

- quiesce database used with the release keyword allows updates to resume on databases that were previously suspended.

- quiesce database used with the for external dump clause signifies that you intend to make an external copy of the database.

- The quiesce database hold and release commands need not be executed from the same user session.

- If the databases specified in the quiesce database hold command contain distributed or multidatabase transactions that are in the prepared state, Adaptive Server waits during a five-second timeout period for those transactions to complete. If the transactions do not complete during the timeout period, quiesce database hold fails.
**quiesce database**

- If Adaptive Server is executing a dump database or dump transaction command on a database specified in *quiesce database hold*, the database is suspended only after the dump command completes.

- If you execute a dump database or dump transaction command on a database while updates to the database are suspended, Adaptive Server blocks those commands until the database is released with *quiesce database release*.

- If you attempt to run a query against a database that is quiesced, Adaptive Server issues error message 880:

  
  Your query is blocked because it tried to write and  
  database '%.*s' is in quiesce state. Your query will  
  proceed after the DBA performs QUIESCE DATABASE  
  RELEASE

  
  The query is run once the database is no longer in a quiescent state.

- You can specify a maximum of eight databases in a single *quiesce database hold* command. If you must suspend updates to additional databases, execute additional *quiesce database hold* commands.

- To duplicate or copy databases, use *quiesce database* with the extension for creating the manifest file. The *quiesce database* effects the quiesce hold by blocking writes in the database, and then creates the manifest file. The command then returns control of the database to the user. You can now use a utility to copy the database to another Adaptive Server. These rules for *quiesce database hold* must be followed for the copy operation:

  - The copy operation cannot begin until the *quiesce database hold* process has completed.
  - Every device for every database in the *quiesce database* command must be copied.
  - The copy process must complete before you invoke *quiesce database release*.

**Encrypted columns and quiesce database**

You can use *quiesce database* when the database contains the encryption key. You must use with *override* to quiesce a database whose columns are encrypted with keys used in other databases.

*quiesce database key_db, col_db* is allowed where *key_db* is the database with the encryption key and *col_db* is the database with a table that has a column encrypted with the key in *key_db*. 

For example, the following commands will succeed where `key_db` contains the encryption key used to encrypt columns in `col_db`:

```
quiesce database key_tag hold key_db for external dump to "/tmp/keydb.dat"
```

```
quiesce database encr_tag hold col_db for external dump to "/tmp/col.dat" with override
```

```
quiesce database col_tag hold key_db, col_db for external dump to "/tmp/col.dat"
```

Permissions

`quiesce database` permission defaults to System Administrators.

Auditing

Values in event and extrainfo columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 96    | quiesce      | quiesce database          | • Roles – Current active roles  
|       |              |                           | • Keywords or options – NULL  
|       |              |                           | • Previous value – NULL      
|       |              |                           | • Current value – NULL       
|       |              |                           | • Other information – NULL   
|       |              |                           | • Proxy information – Original login name, if a set proxy is in effect |

See also

**Commands** dump database, dump transaction, mount, unmount

**System procedures** sp_helpdb, sp_who
raiserror

Description
Prints a user-defined error message on the user’s screen and sets a system flag to record that an error condition has occurred.

Syntax

```sql
raiserror error_number
    [({format_string | @local_variable}) [, arg_list]]
    [with errordata restricted_select_list]
```

Parameters

- **error_number**
  is a local variable or an integer with a value greater than 17,000. If the `error_number` is between 17,000 and 19,999, and `format_string` is missing or empty (""), Adaptive Server retrieves error message text from the `sysmessages` table in the master database. These error messages are used chiefly by system procedures.

  If `error_number` is 20,000 or greater and `format_string` is missing or empty, `raiserror` retrieves the message text from the `sysusermessages` table in the database from which the query or stored procedure originates. Adaptive Server attempts to retrieve messages from either `sysmessages` or `sysusermessages` in the language defined by the current setting of `@@langid`.

- **format_string**
  is a string of characters with a maximum length of 1024 bytes. Optionally, you can declare `format_string` in a local variable and use that variable with `raiserror` (see `@local_variable`). `raiserror` recognizes placeholders in the character string that is to be printed out. Format strings can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow `format_string`, when the text of the message is sent to the client.

  To allow reordering of the arguments, when format strings are translated to a language with a different grammatical structure, the placeholders are numbered. A placeholder for an argument appears in this format: “%nn!”—a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation point (!). The integer represents the argument number in the string in the argument list. “%1!” is the first argument in the original version, “%2!” is the second argument, and so on.

  Indicating the position of the argument in this way makes it possible to translate correctly, even when the order in which the arguments appear in the target language is different from their order in the source language.

  For example, assume the following is an English message:
%1! is not allowed in %2!.
The German version of this message is:
%1! ist in %2! nicht zulassig.
The Japanese version of this message is:
%2！の中で％1！は許されません。

In this example, “%1!” represents the same argument in all three languages, as does “%2!”. This example shows the reordering of the arguments that is sometimes necessary in the translated form.

@local_variable
is a local variable containing the format_string value. It must be of type char or varchar and must be declared within the batch or procedure in which it is used.

arg_list
is a series of variables or constants separated by commas. arg_list is optional unless a format string containing placeholders of the form “%nn!” is provided. An argument can be any datatype except text or image; it is converted to the char datatype before being included in the final string.

If an argument evaluates to NULL, Adaptive Server converts it to a zero-length char string.

with errordata
supplies extended error data for Client-Library™ programs.

restricted_select_list
consists of one or more of the following items:

- “*”, representing all columns in create table order.
- A list of column names in the order you want to see them. When selecting an existing IDENTITY column, you can substitute the syb_identity keyword, qualified by the table name, where necessary, for the actual column name.
- A specification to add a new IDENTITY column to the result table:
  column_name = identity (precision)
- A replacement for the default column heading (the column name), in the following forms:
  column_heading = column_name
column_name column_heading
column_name as column_heading

The column heading may be enclosed in quotation marks for any of these forms. The heading must be enclosed in quotation marks if it is not a valid identifier (that is, if it is a reserved word, if it begins with a special character, or if it contains spaces or punctuation marks).

- An expression (a column name, constant, function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery).
- A built-in function or an aggregate.
- Any combination of the items listed above.

The restricted_select_list can also perform variable assignment, in the form:

```sql
@variable = expression
[, @variable = expression ...]
```

Restrictions to restricted_select_list are:

- You cannot combine variable assignment with any of the other restricted_select_list options.
- You cannot use from, where, or other select clauses in restricted_select_list.
- You cannot use "*" to represent all columns in restricted_select_list.

For more information, see the Transact-SQL User's Guide.

Examples

**Example 1** This stored procedure example returns an error if it does not find the table supplied with the @tabname parameter:

```sql
create procedure showtable_sp @tabname varchar (18)
as
if not exists (select name from sysobjects
   where name = @tabname)
   begin
      raiserror 99999 "Table %1! not found.",
      @tabname
   end
else
   begin
      select sysobjects.name, type, crdate, indid
      from sysindexes, sysobjects
      where sysobjects.name = @tabname
      and sysobjects.id = sysindexes.id
   end
```
Example 2  This example adds a message to sysusermessages, then tests the message with raiserror, providing the substitution arguments:

```sql
sp_addmessage 25001,
"There is already a remote user named '%1!'
for remote server '%2!'.'
```

```sql
raiserror 25001, jane, myserver
```

Example 3  This example uses the with errordata option to return the extended error data column and server to a client application, to indicate which column was involved and which server was used:

```sql
raiserror 20100 "Login must be at least 5
characters long" with errordata "column" = "login", "server" = @@servername
```

Usage

- User-defined messages can be generated ad hoc, as in Example 1 and Example 3, or they can be added to the system table sysusermessages for use by any application, as shown in Example 2. Use sp_addmessage to add messages to sysusermessages; use sp_getmessage to retrieve messages for use by print and raiserror.

- Error numbers for user-defined error messages must be greater than 20,000. The maximum value is 2,147,483,647 (2^{31} - 1).

- The severity level of all user-defined error messages is 16. This level indicates that the user has made a nonfatal error.

- The maximum output string length of format_string plus all arguments after substitution is 1024 bytes.

- If you use placeholders in a format string, keep this in mind: for each placeholder $n$ in the string, the placeholders 1 through $n-1$ must exist in the same string, although they do not have to be in numerical order. For example, you cannot have placeholders 1 and 3 in a format string without having placeholder 2 in the same string. If you omit a number in a format string, an error message is generated when raiserror is executed.

- If there are too few arguments relative to the number of placeholders in format_string, an error message displays and the transaction is aborted. You can have more arguments than placeholders in format_string.

- To include a literal percent sign as part of the error message, use two percent signs ("%%") in the format_string. If you include a single percent sign ("%") in the format_string that is not used as a placeholder, Adaptive Server returns an error message.
- If an argument evaluates to NULL, it is converted into a zero-length char string. If you do not want zero-length strings in the output, use the isnull function.

- When raiserror is executed, the error number is placed in the global variable @@error, which stores the error number that was most recently generated by the system.

- Use raiserror instead of print if you want an error number stored in @@error.

- To include an arg_list with raiserror, put a comma after error_number or format_string before the first argument. To include extended error data, separate the first extended_value from error_number, format_string, or arg_list using a space (not a comma).

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
raiserror permission defaults to all users. No permission is required to use it.

See also
Commands declare, print
System procedures sp_addmessage, sp_getmessage
readtext

Description
Reads text, unitext, and image values, starting from a specified offset and
reading a specified number of bytes or characters.

Syntax
readtext [database.]owner.]table_name.column_name
text_pointer offset size
  [holdlock | noholdlock] [readpast]
  [using {bytes | chars | characters}]
  [at isolation {
    [read uncommitted | 0] |
    [read committed | 1] |
    [repeatable read | 2]]
    [serializable | 3]]

Parameters

- **table_name.column_name**
  is the name of the text, unitext, or image column. You must include the table
  name. Specify the database name if the table is in another database, and
  specify the owner’s name if more than one table of that name exists in the
database. The default value for **owner** is the current user, and the default
value for **database** is the current database.

- **text_pointer**
  is a varbinary (16) value that stores the pointer to the text, unitext, or image
data. Use the textptr function to determine this value. text, unitext, and image
data is not stored in the same set of linked pages as other table columns. It
is stored in a separate set of linked pages. A pointer to the actual location is
stored with the data; textptr returns this pointer.

- **offset**
  specifies the number of bytes or characters to skip before starting to read
text, unitext, or image data.

- **size**
  specifies the number of bytes or characters of data to read.

- **holdlock**
  causes the text value to be locked for reads until the end of the transaction.
  Other users can read the value, but they cannot modify it.

- **noholdlock**
  prevents the server from holding any locks acquired during the execution of
  this statement, regardless of the transaction isolation level currently in
  effect. You cannot specify both a holdlock and a noholdlock option in a query.

- **readpast**
  specifies that readtext should silently skip rows with exclusive locks,
  without waiting and without generating a message.
using

specifies whether readtext interprets the offset and size parameters as a number of bytes (bytes) or as a number of textptr characters (chars or characters are synonymous). This option has no effect when used with a single-byte character set or with image values (readtext reads image values byte by byte). If the using option is not given, readtext interprets the size and offset arguments as bytes.

at isolation

specifies the isolation level (0, 1, or 3) of the query. If you omit this clause, the query uses the isolation level of the session in which it executes (isolation level 1 by default). If you specify holdlock in a query that also specifies at isolation read uncommitted, Adaptive Server issues a warning and ignores the at isolation clause. For the other isolation levels, holdlock takes precedence over the at isolation clause.

read uncommitted

specifies isolation level 0 for the query. You can specify 0 instead of read uncommitted with the at isolation clause.

read committed

specifies isolation level 1 for the query. You can specify 1 instead of read committed with the at isolation clause.

repeatable read

specifies isolation level 2 for the query. You can specify 2 instead of serializable with the at isolation clause.

serializable

specifies isolation level 3 for the query. You can specify 3 instead of serializable with the at isolation clause.

Examples

Example 1 Selects the second through the sixth character of the copy column:

```
declare @val varbinary (16)
select @val = textptr (copy) from blurbs
where au_id = "648-92-1872"
readtext blurbs.copy @val 1 5 using chars
```

Example 2

```
declare @val varbinary (16)
select @val = textptr (copy) from blurbs readpast
where au_id = "648-92-1872"
readtext blurbs.copy @val 1 5 readpast using chars
```
The `textptr` function returns a 16-byte binary string (text pointer) to the text, unitext, or image column in the specified row or to the text, unitext, or image column in the last row returned by the query, if more than one row is returned. Declare a local variable to hold the text pointer, then use the variable with `readtext`.

The value in the global variable `@@textsize`, which is the limit on the number of bytes of data to be returned, supersedes the size specified for `readtext` if it is less than that size. Use `set textsize` to change the value of `@@textsize`.

When using bytes as the offset and size, Adaptive Server may find partial characters at the beginning or end of the text data to be returned. If it does, and character set conversion is on, the server replaces each partial character with a question mark (?) before returning the text to the client.

Adaptive Server must determine the number of bytes to send to the client in response to a `readtext` command. When the `offset` and `size` are in bytes, determining the number of bytes in the returned text is simple. When the offset and size are in characters, the server must calculate the number of bytes being returned to the client. As a result, performance may be slower when using characters as the `offset` and `size`. The using characters option is useful only when Adaptive Server is using a multibyte character set: it ensures that `readtext` does not return partial characters.

You cannot use `readtext` on text, unitext, or image columns in views.

If you attempt to use `readtext` on text values after changing to a multibyte character set, and you have not run `dbcc fix_text`, the command fails, and an error message instructs you to run `dbcc fix_text` on the table.

### Using `readtext` with unitext columns

When you issue `readtext` on a column defined for the unitext datatype, the `readtext offset` parameter specifies the number of bytes, or Unicode values, to skip before starting to read the unitext data. The `readtext size` parameter specifies the number of bytes, or 16-bit Unicode values, to read. If you specify using bytes (the default), the `offset` and `size` values are adjusted to always start and end on the Unicode character boundaries, if necessary.

If `enable surrogate processing` is on, `readtext` truncates only on the surrogate boundary, and starting/ending positions are also adjusted accordingly and returns whole Unicode characters. For this reason, issuing `readtext` against a column defined for unitext may return fewer bytes than specified.

In the following example, the unitext column `ut` includes the string `U+0101U+0041U+0042U+0043`:
declare @val varbinary (16)
select @val = textptr (ut) from unitable
where i = 1
readtext foo.ut @val 1 5

This query returns the value U+0041U+0042.

The offset position is adjusted to 2 since readtext cannot start from the second byte of a Unicode character. Unicode characters are always composed of an even number of bytes. Starting at the second byte (or ending in an odd number of bytes) shifts the result by one byte, and renders the result set inaccurate.

In the example above, the size value is adjusted to 4 since readtext cannot read the partial byte of the fourth character, U+0043.

In the following query, enable surrogate processing is enabled, and the ut column contains the string U+d800dc00U+00c2U+dbffdeffU+d800dc00:

```
declare @val varbinary (16)
select @val = textptr (ut) from unitable
where i = 2
readtext foo.ut @val 1 8
```

This query returns the value U+00c2U+dbffdeff. The starting position is reset to 2, and the actual result size is 6 bytes rather than 8 since readtext does not break in the middle of a surrogate pair. Surrogate pairs (in this example, the first value in the range d800..dbff and the second in the range dc00..dfff) require 4-byte boundaries, and the rules of Unicode conformance for UTF-16 do not allow the division of these 4-byte characters.

Using the readpast option

- readpast applies only to data-only-locked tables, and is ignored if it is specified for an allpages-locked table.
- readpast is incompatible with the holdlock option. If both are specified in a command, an error is generated and the command terminates.
- If readtext specifies at isolation read uncommitted, readpast generates a warning, but does not terminate the command.
- If the statement isolation level is set to 3, readpast generates an error and terminates the command.
- If the session-wide isolation level is 3, readpast is silently ignored.
- If the session-wide isolation level is 0, readpast generates a warning, but does not terminate the command.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.
Permissions

readtext requires select permission on the table. readtext permission is transferred when select permission is transferred.

See also

Commands set, writetext

System procedures text, image, and unitext datatypes
reconfigure

**Description**
The `reconfigure` command currently has no effect; it is included to allow existing scripts to run without modification.

**Syntax**
reconfigure

**Usage**

**Note** If you have scripts that include `reconfigure`, change them at your earliest convenience. Although `reconfigure` is included in this version, it may not continue to be supported in subsequent versions.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
`reconfigure` permission defaults to System Administrators and is not transferable.

**See also**
- System procedures
  - `sp_configure`
remove java

Description
Removes one or more Java-SQL classes, packages, or JARs from a database.
Use when Java classes are installed in the database. See Java in Adaptive Server Enterprise for more information.

Syntax

```
remove java
   class class_name[, class_name]...
   | package package_name[, package_name]...
   | jar jar_name[, jar_name]...[retain classes]
```

Parameters
class class_name
the name of one or more Java classes to be removed from the database. The classes must be installed in the current database.

package package_name
the name of one or more Java packages to be removed. The packages must be stored in the current database.

jar jar_name
either a SQL identifier or character string value of up to 30 bytes that contains a valid SQL identifier.

Each jar_name must be equal to the name of a retained JAR in the current database.

retain classes
specifies that the named JARs are no longer retained in the database, and the retained classes have no associated JAR.

Usage

- If a remove java statement is contained in a stored procedure, the current database is the database that is current when the procedure is created, not the database that is current when the procedure is called.

- If a remove java statement is not contained in a stored procedure, the current database is the database that is current when the remove statement is executed.

- If class or package is specified and any removed class has an associated JAR, then an exception is raised.

- If any stored procedure, table, or view contains a reference to a removed class as the datatype of a column, variable, or parameter, then an exception is raised.

- All removed classes are:
  - Deleted from the current database.
remove java

- Unloaded from the Java Virtual Machine (Java VM) of the current connection. The removed classes are not unloaded from the Java VMs of other connections.

- If any exception is raised during the execution of remove java, then all actions of remove java are cancelled.

- You cannot remove a Java-SQL class if that class is directly referenced by a SQLJ stored procedure or function.

- To remove a Java-SQL class from the database, you must:
  a Delete all SQLJ stored procedures or functions that directly reference the class using drop procedure and drop function.
  b Delete the Java-SQL class from the database using remove java.

Locks
- When you use remove java, an exclusive table lock is placed on sysxtypes.
- If jar is specified, then an exclusive table lock is placed on sysjars.

Permissions
You must be a System Administrator or Database Owner to use remove java.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>remove java</td>
<td>remove java</td>
<td>Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keywords or options – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proxy information – Original login name, if a set proxy is in effect</td>
</tr>
</tbody>
</table>

See also

System procedures: sp_helpjava

System tables: sysjars, sysxtypes

Utilities: extractjava, installjava
reorg

Description
Reclaims unused space on pages, removes row forwarding, or rewrites all rows in the table to new pages, depending on the option used.

Syntax
reorg compact table_name [partition partition_name]
   [with {resume, time = no_of_minutes}]
reorg forwarded_rows table_name [partition partition_name]
   [with {resume, time = no_of_minutes}]
reorg rebuild table_name [index_name [partition index_partition_name]]
   [with {resume, time = no_of_minutes}]
reorg reclaim_space table_name [index_name] [partition partition_name]
   [with {resume, time = no_of_minutes}]

Parameters
compact
   combines the functions of reorg reclaim_space and reorg forwarded_rows to both reclaim space and undo row forwarding in the same pass.
forwarded_rows
   removes row forwarding.

index_partition_name
   is the name of the index partition on which you are running reorg. update statistics performs a check to validate that index_partition_name is an index partition.

indexname
   specifies the name of the index to be reorganized.

partition_name
   is the name of the partition on which you are running reorg.

tablename
   specifies the name of the table to be reorganized. If indexname is specified, only the index is reorganized.

rebuild
   if a table name is specified, rewrites all rows in a table to new pages, so that the table is arranged according to its clustered index (if one exists), with all pages conforming to current space management settings and with no forwarded rows and no gaps between rows on a page. If the table has an index, all indexes are dropped and re-created. If an index name is specified, reorg rebuilds that index while leaving the table accessible for read and update activities.

Note reorg rebuild is not supported for system catalogs.
reclaim_space

reclaims unused space left by deletes and updates. For each data page in a
table, if there is unused space resulting from committed deletes or
row-shortening updates, reorg reclaim_space rewrites the current rows
contiguously, leaving all unused space at the end of the page. If there are no
rows on the page, the page is deallocated.

Note reorg reclaim_space only affects tables with variable-length rows, and
only frees up space within a page. To reduce the number of pages used, use the
reorg rebuild command.

with resume

initiates reorganization from the point at which a previous reorg command
terminated. Used when the previous reorg command specified a time limit
(with time = no_of_minutes).

with time = no_of_minutes

specifies the number of minutes that the reorg command is to run.

Example 1 Reclaims unused page space in the titles table:

    reorg reclaim_space titles

Example 2 Reclaims unused page space in the index titleind:

    reorg reclaim_space titles titleind

Example 3 Initiates reorg compact on the titles table. reorg starts at the
beginning of the table and continues for 120 minutes. If the reorg completes
within the time limit, it returns to the beginning of the table and continues until
the full time period has elapsed:

    reorg compact titles with time = 120

Example 4 Initiates reorg compact at the point where the previous reorg
compact stopped and continues for 30 minutes:

    reorg compact titles with resume, time = 30

Example 5 Runs reorg forwarded_rows on the smallsales partition of the titles
table:

    reorg forwarded_rows titles partition smallsales

Example 6 Runs reorg forwarded_rows on the authors table:

    reorg forwarded_rows authors

Example 7 Runs reorg reclaim_space on the bigsales partition of titles:

    reorg reclaim_space titles partition bigsales
reorg reclaim_space titles partition bigsales

**Example 8** Runs `reorg compact` on the bigsales partition of titles:

```
reorg compact titles partition bigsales
```

**Example 9** Runs `reorg rebuild` on the index partition `idx_p2` of index `local_idx` on table `sales`:

```
reorg rebuild sales local_idx partition idx_p2
```

**Usage**

- The table specified in `reorg`—excluding `reorg rebuild`—must have a datarows- or datapages-locking scheme.

- You cannot issue `reorg` within a transaction.

- `reorg rebuild` requires that you set the database option `select into/bulkcopy/pllsort` to `true` and run checkpoint in the database.

- `reorg rebuild` requires additional disk space equal to the size of the table and its indexes. You can find out how much space a table currently occupies by using `sp_spaceused`. You can use `sp_helpsegment` to check the amount of space available.

- After running `reorg rebuild`, you must dump the database before you can dump the transaction log.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

You must be a System Administrator or the object owner to issue the `reorg` command.

**See also**

Documents  For more information, see the *System Administration Guide*.

System procedures  `sp_chgattribute`
## return

### Description
Exits from a batch or procedure unconditionally and provides an optional return status. Statements following `return` are not executed.

### Syntax
```
return [integer_expression] [plan "abstract_plan"]
```

### Parameters
- **integer_expression**
  - is the integer value returned by the procedure. Stored procedures can return an integer value to a calling procedure or an application program.

- **plan "abstract_plan"**
  - specifies the abstract plan to use to optimize the query. The abstract plan can be a full or partial plan specified in the abstract plan language. Plans can be specified only for optimizable SQL statements, that is, queries that access tables. See Chapter 16, “Creating and Using Abstract Plans,” in the *Performance and Tuning Guide: Optimizer and Abstract Plans* for more information.

### Examples
**Example 1** If no user name is given as a parameter, the `return` command causes the procedure to exit after a message has been sent to the user’s screen. If a user name is given, the names of the rules created by that user in the current database are retrieved from the appropriate system tables:

```sql
create procedure findrules @nm varchar (30) = null as
if @nm is null
  begin
    print "You must give a user name"
    return
  end
else
  begin
    select sysobjects.name, sysobjects.id, sysobjects.uid
    from sysobjects, master..syslogins
    where master..syslogins.name = @nm
    and sysobjects.uid = master..syslogins.suid
    and sysobjects.type = "R"
  end
```

**Example 2** If the updates cause the average price of business titles to exceed $15, the `return` command terminates the batch before any more updates are performed on titles:

```sql
print "Begin update batch"
update titles
  set price = price + $3
  where title_id = 'BU2075'
```
update titles
  set price = price + $3
  where title_id = 'BU1111'
if (select avg (price) from titles
    where title_id like 'BU\%' ) > $15
begin
  print "Batch stopped; average price over $15"
  return
end
update titles
  set price = price + $2
  where title_id = 'BU1032'

Example 3 This procedure creates two user-defined status codes: a value of 1
is returned if the contract column contains a 1; a value of 2 is returned for any
other condition (for example, a value of 0 on contract or a title_id that did not
match a row):

create proc checkcontract @param varchar (11)
as
declare @status int
if (select contract from titles where title_id = @param) = 1
  return 1
else
  return 2

Usage
• The return status value can be used in subsequent statements in the batch
  or procedure that executed the current procedure, but must be given in the
  form:
    execute @retval = procedure_name
See execute for more information.
• Adaptive Server reserves 0 to indicate a successful return, and negative
  values in the range -1 to -99 to indicate different reasons for failure. If no
  user-defined return value is provided, the Adaptive Server value is used.
  User-defined return status values cannot conflict with those reserved by
  Adaptive Server. Numbers 0 and -1 through -14 are currently in use:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Procedure executed without error</td>
</tr>
<tr>
<td>-1</td>
<td>Missing object</td>
</tr>
<tr>
<td>-2</td>
<td>Datatype error</td>
</tr>
<tr>
<td>-3</td>
<td>Process was chosen as deadlock victim</td>
</tr>
</tbody>
</table>
### return

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>Permission error</td>
</tr>
<tr>
<td>-5</td>
<td>Syntax error</td>
</tr>
<tr>
<td>-6</td>
<td>Miscellaneous user error</td>
</tr>
<tr>
<td>-7</td>
<td>Resource error, such as out of space</td>
</tr>
<tr>
<td>-8</td>
<td>Nonfatal internal problem</td>
</tr>
<tr>
<td>-9</td>
<td>System limit was reached</td>
</tr>
<tr>
<td>-10</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-11</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-12</td>
<td>Table or index is corrupt</td>
</tr>
<tr>
<td>-13</td>
<td>Database is corrupt</td>
</tr>
<tr>
<td>-14</td>
<td>Hardware error</td>
</tr>
</tbody>
</table>

Values -15 to -99 are reserved for future Adaptive Server use.

- If more than one error occurs during execution, the status with the highest absolute value is returned. User-defined return values always take precedence over Adaptive Server-supplied return values.

- The `return` command can be used at any point where you want to exit from a batch or procedure. Return is immediate and complete: statements after `return` are not executed.

- A stored procedure cannot return a NULL return status. If a procedure attempts to return a null value, for example, using `return @status` where `@status` is NULL, a warning message is generated, and a value in the range of 0 to -14 is returned.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
`return` permission defaults to all users. No permission is required to use it.

**See also**  
Commands `begin...end`, `execute`, `if...else`, `while`
revoke

Description
Revokes permissions or roles from users, groups, or roles.

Syntax
To revoke permission to access database objects:

revoke [grant option for]
   {all [privileges] | permission_list}
   on {table_name [(column_list)]
       | view_name [(column_list)]
       | stored_procedure_name}
   from {public | name_list | role_list}
   [cascade]

To revoke permission to select built-in functions:

revoke select
   on [builtin] builtin
   to {name_list | role_list}

To revoke permission to create database objects, execute set proxy, or execute set session authorization:

revoke {all [privileges] | command_list}
   from {public | name_list | role_list}

To revoke a role from a user or another role:

revoke role {role_name [, role_list ...]}
   from {grantee [, grantee ...]}

To revoke access on some dbcc commands:

revoke dbcc {dbcc_command [on {all | database}]
   [, dbcc_command [on {all | database}], ...]}
   from {user_list | role_list}

To revoke permission from other users, groups, and roles to create encryption keys.

revoke create encryption key from user | role | group

To revoke decrypt permission on a table or a list of columns in a table:

revoke decrypt on [owner.] tablename[(columnname [{, columnname}])]
   from user | group | role

To revoke the default permissions from public:

revoke default permissions on system tables
revoke

Parameters

**all**
when used to revoke permission to access database objects (the first syntax format), all revokes all permissions applicable to the specified object. All object owners can use revoke all with an object name to revoke permissions on their own objects.

Only the System Administrator or the Database Owner can revoke permission to revoke create command permissions (the third syntax format). When used by the System Administrator, **revoke all** revokes all create permissions (create database, create default, create procedure, create rule, create table, and create view). When the Database Owner uses **revoke all**, Adaptive Server revokes all create permissions except create database, and prints an informational message.

**all** does not apply to set proxy, set session authorization, create encryption key, and create trigger.

**permission_list**
is a list of permissions to revoke. If more than one permission is listed, separate them with commas. The following table illustrates the access permissions that can be granted and revoked on each type of object:

<table>
<thead>
<tr>
<th>Object</th>
<th>permission_list can include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>select, insert, delete, update references, update statistics, delete statistics, and truncate table, decrypt</td>
</tr>
<tr>
<td>View</td>
<td>select, insert, delete, update, decrypt</td>
</tr>
<tr>
<td>Column</td>
<td>select, update, references, decrypt</td>
</tr>
<tr>
<td></td>
<td>Column names can be specified in either <strong>permission_list</strong> or <strong>column_list</strong>.</td>
</tr>
<tr>
<td>Stored procedure</td>
<td>execute</td>
</tr>
<tr>
<td>Encryption key</td>
<td>select</td>
</tr>
</tbody>
</table>

Permissions can be revoked only by the user who granted them.

**builtin**
is a built-in function. Specifying built-in functions allows you to differentiate between a table and a revocable built-in function with the same name. The functions are set_appcontext, get_appcontext, list_appcontext, and rm_appcontext.
**command_list**

is a list of commands. If more than one command is listed, separate them with commas. The command list can include create database, create default, create procedure, create rule, create table, create view, create encryption key, set proxy, or set session authorization. create database permission can be revoked only by a System Administrator and only from within the master database.

set proxy and set session authorization are identical; the only difference is that set session authorization follows the SQL standard, and set proxy is a Transact-SQL extension. Revoking permission to execute set proxy or set session authorization revokes permission to become another user in the server. Permissions for set proxy or set session authorization can be revoked only by a System Security Officer, and only from within the master database.

**table_name**

is the name of the table on which you are revoking permissions. The table must be in your current database. Only one object can be listed for each revoke statement.

**column_list**

is a list of columns, separated by commas, to which the privileges apply. If columns are specified, only select and update permissions can be revoked. References permissions also can be revoked on columns.

**view_name**

is the name of the view on which you are revoking permissions. The view must be in your current database. Only one object can be listed for each revoke statement.

**stored_procedure_name**

is the name of the stored procedure on which you are revoking permissions. The stored procedure must be in your current database. Only one object can be listed for each revoke statement.

**keyname**

is the name of the key from which you are revoking permission. The encryption key must be in your current database. Only one object can be listed for each revoke statement. You can revoke only select permission from a key.
public

is all users. For object access permissions, public excludes the object owner. For object creation permissions or set proxy authorizations, public excludes the Database Owner. You cannot grant permissions with grant option to “public” or to other groups or roles.

name_list

is a list of user and group names, separated by commas.

role

is the name of a system or user-defined role. Use revoke role to revoke revoked roles from roles or users.

role_name

is the name of a system or user-defined role. This allows you to revoke permissions from all users who have been revoked a specific role. The role name can be either a system role or a user-defined role created by a System Security Officer with create role. Either type of role can be revoked to a user with the revoke role command. In addition, you can use sp_role to revoke system roles.

grantee

is the name of a system role, user-defined role, or a user, from whom you are revoking a role.

grant option for

revokes with grant option permissions, so that the users specified in name_list can no longer grant the specified permissions to other users. If those users have granted permissions to other users, you must use the cascade option to revoke permissions from those users. The user specified in name_list retains permission to access the object, but can no longer grant access to other users.

grant option for applies only to object access permissions, not to object creation permissions.

cascade

revokes the specified object access permissions from all users to whom the revokee granted permissions. Applies only to object access permissions, not to object creation permissions. When you use revoke without grant option for, permissions granted to other users by the revokee are also revoked: the cascade occurs automatically.

dbcc_command

is the name of the dbcc command you are revoking. It cannot be a variable. Table 1-33 on page 499 lists the valid revoke dbcc commands.
database

is the name of the database on which you are revoking permissions. It is used with database-specific dbcc commands to revoke permission only on the target database. The revokee must be a valid user in the target database. 
database conforms to the rules for identifiers and cannot be a variable.

If there are multiple revoked actions in the same command, database must be unique.

See “on all | database parameter and server-level commands” on page 500 for more information.

user_list

is a list of users from whom you are revoking the permission, and cannot be a variable.

role_list

is a list of the name of system or user-defined roles from whom you are revoking the permission, and cannot be a variable.

Note You cannot grant or revoke dbcc commands to public or groups.

default permissions on system tables

specifies that you revoke the default permissions for the system tables listed in “revoking default permissions on system tables” on page 500.

Examples

Example 1 Revokes insert and delete permissions on the titles table from Mary and the “sales” group:

   revoke insert, delete
   on titles
   from mary, sales

Example 2 Revokes select permission on the get_appcontext function from “public” (which includes all users):

   revoke select on builtin get_appcontext from public

Compare this to the following, which revokes select permission on a table called get_appcontext, if a table with that name exists:

   revoke select on get_appcontext from public

Example 3 Two ways to revoke update permission on the price and advance columns of the titles table from “public:”

   revoke update
   on titles (price, advance)
from public

or:

revoke update (price, advance)
on titles
from public

Example 4 Revokes permission from Mary and John to use the create
database and create table commands. Because create database permission is
being revoked, this command must be executed by a System Administrator
from within the master database. Mary’s and John’s create table permission is
revoked only within the master database:

revoke create database, create table from mary, john

Example 5 Revokes permission from Harry and Billy to execute either set
proxy or set session authorization to impersonate another user in the server:

revoke set proxy from harry, billy

Example 6 Revokes permission from users with sso_role to execute either set
proxy or set session authorization:

revoke set session authorization from sso_role

Example 7 Revokes permission from users with vip_role to impersonate
another user in the server. vip_role must be a role defined by a System Security
Officer with the create role command:

revoke set proxy from vip_role

Example 8 Revokes all object creation permissions from Mary in the current
database (except create encryption key):

revoke all from mary

Example 9 Revokes all object access permissions on the titles table from Mary
(except decrypt permission):

revoke all on titles from mary

Example 10 Two ways to revoke Tom’s permission to create a referential
integrity constraint on another table that refers to the price and advance
columns in the titles table:

revoke references
on titles (price, advance)
from tom

or:

revoke references (price, advance)
on titles
from tom

Example 11 Revokes permission to execute new_sproc from all users who have been granted the “operator” role:

revoke execute on new_sproc from oper_role

Example 12 Revokes John’s permission to grant insert, update, and delete permissions on the authors table to other users. Also revokes from other users any such permissions that John has granted:

revoke grant option for insert, update, delete on authors from john cascade

Example 13 Revokes “doctor_role” from “specialist_role”:

revoke role doctor_role from specialist_role

Example 14 Revokes “doctor_role” and “surgeon_role” from “specialist_role” and “intern_role”, and from users Mary and Tom:

revoke role doctor_role, surgeon_role from specialist_role, intern_role, mary, tom

Example 15 Revokes dbcc privileges from Frank:

1> use pubs2
2> go
1> revoke dbcc checkdb on pubs2 from checkdb_role
2> go
1> use master
2> go
1> revoke dbcc checkdb on all from frank
2> go
...

Example 16 Revokes truncate table and update statistics privileges from Harry on the authors table:

revoke truncate table on authors from harry
revoke update statistics on authors from harry

Example 17 Revokes the delete statistics privileges from user Billy on the authors table:

revoke delete statistics on authors from billy
Example 18 Revokes truncate table and update and delete statistics privileges from all users with the oper_role:

- revoke truncate table on authors from oper_role
- revoke update statistics on authors from oper_role
- revoke delete statistics on authors from oper_role

Users Billy and Harry can no longer run these commands on authors.

Example 19 Revokes decrypt permissions from public:

- revoke decrypt on customer from public

Example 20 Revokes create encryption key permissions from user joe:

- revoke create encryption key from joe

Example 21 Revokes select on permission for the ssn_key from the database owner.

- grant select on ssn_key to dbo

Usage

- See the grant command for more information about permissions.
- You can revoke permissions only on objects in your current database.
- You can revoke only permissions that were granted by you.
- You cannot revoke a role from a user while the user is logged in.
- grant and revoke commands are order-sensitive. When there is a conflict, the command issued most recently takes effect.
- You can substitute the word to for the word from in the revoke syntax.
- If you do not specify grant option for in a revoke statement, with grant option permissions are revoked from the user along with the specified object access permissions. In addition, if the user has granted the specified permissions to any other users, all of those permissions are revoked. In other words, the revoke cascades.
- A grant statement adds one row to the sysprotects system table for each user, group, or role that receives the permission. If you subsequently revoke the permission from the user or group, Adaptive Server removes the row from sysprotects. If you revoke the permission from only selected group members, but not from the entire group to which it was granted, Adaptive Server retains the original row and adds a new row for the revoke.
• Permission to issue create trigger is granted to users by default. When you
revoke permission for a user to create triggers, a revoke row is added in
the sysprotects table for that user. To grant permission to issue create
trigger, you must issue two grant commands. The first command removes
the revoke row from sysprotects; the second inserts a grant row. If you
revoke permission to create triggers, the user cannot create triggers even
on tables that the user owns. Revoking permission to create triggers from
a user affects only the database where the revoke command was issued.

Using the cascade option
• revoke grant option for revokes the user’s ability to grant the specified
permission to other users, but does not revoke the permission itself from
that user. If the user has granted that permission to others, you must use the
cascade option; otherwise, you receive an error message and the revoke
fails.

For example, say you revoke the with grant option permissions from the
user Bob on titles, with this statement:

    revoke grant option for select
    on titles
    from bob
    cascade

• If Bob has not granted this permission to other users, this command
revokes his ability to do so, but he retains select permission on the
titles table.

• If Bob has granted this permission to other users, you must use the
cascade option. If you do not, you receive an error message and the
revoke fails. cascade revokes this select permission from all users to
whom Bob has granted it, as well as their ability to grant it to others.

• You cannot use revoke with the cascade option to revoke privileges
granted by the table owner. For example, the owner of a table (UserA) can
grant privileges to another user (UserB) as in this scenario:

    create table T1 (...)
    grant select on T1 to UserB

    revoke select on T1 from UserA cascade

However, the System Administrator cannot revoke UserB’s privileges
using the revoke privileges command with the cascade option as in this
statement:

    revoke select on T1 from UserA cascade

This statement revokes the select privileges of the table owner, but does
not revoke those privileges from UserB.
By default, all data manipulation language (DML) operations are revoked implicitly for users other than the table owner. Because the sysprotects table contains no records indicating that the table owner has granted and then revoked privileges, the cascade option is not invoked.

You must revoke explicitly the select privilege from UserB.

Revoking set proxy and set session authorization

- To revoke set proxy or set session authorization permission, or to revoke roles, you must be a System Security Officer, and you must be in the master database.

- set proxy and set session authorization are identical, with one exception: set session authorization follows the SQL standard. If you are concerned about using only SQL standard commands and syntax, use set session authorization.

- revoke all does not include set proxy or set session authorization permissions.

Revoking from roles, users and groups

- Permissions granted to roles override permissions granted to individual users or groups. Therefore, if you revoke a permission from a user who has been granted a role, and the role has that same permission, the user retains it. For example, say John has been granted the System Security Officer role, and sso_role has been granted permission on the sales table. If John’s individual permission on sales is revoked, he can still access sales because his role permissions override his individual permissions.

- Revoking a specific permission from “public” or from a group also revokes it from users who were individually granted the permission.

- Database user groups allow you to grant or revoke permissions to more than one user at a time. A user is always a member of the default group, “public” and can be a member of only one other group. The Adaptive Server installation script assigns a set of permissions to “public.” Create groups with sp_addgroup and remove groups with sp_dropgroup. Add new users to a group with sp_adduser. Change a user’s group membership with sp_changegroup. To display the members of a group, use sp_helpgroup.

revoke dbcc command options

Table 1-33 lists the valid revoke dbcc commands.
Table 1-33: *dbcc command options*

<table>
<thead>
<tr>
<th>Command name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkalloc</td>
<td>Checks the specified database to make sure all of its pages are correctly allocated, and that there are no unused allocated pages.</td>
</tr>
<tr>
<td>checkcatalog</td>
<td>Checks for consistency in and between system tables.</td>
</tr>
<tr>
<td>checkdb</td>
<td>Runs the same checks as checktable, but on each table in the specified database, including syslogs.</td>
</tr>
<tr>
<td>checkindex</td>
<td>Checks the specified index to make sure that:</td>
</tr>
<tr>
<td></td>
<td>- Index and data pages are correctly linked.</td>
</tr>
<tr>
<td></td>
<td>- Indexes are correctly sorted.</td>
</tr>
<tr>
<td></td>
<td>- All pointers are consistent.</td>
</tr>
<tr>
<td></td>
<td>- Data information on each page is reasonable.</td>
</tr>
<tr>
<td></td>
<td>- Page offsets are reasonable.</td>
</tr>
<tr>
<td>checkstorage</td>
<td>Checks the specified database for:</td>
</tr>
<tr>
<td></td>
<td>- Allocation</td>
</tr>
<tr>
<td></td>
<td>- OAM page entries</td>
</tr>
<tr>
<td></td>
<td>- Page consistency</td>
</tr>
<tr>
<td></td>
<td>- Text-valued columns</td>
</tr>
<tr>
<td></td>
<td>- Allocation of text-valued columns</td>
</tr>
<tr>
<td></td>
<td>- Text-column chains</td>
</tr>
<tr>
<td>checktable</td>
<td>Checks the specified table to make sure that:</td>
</tr>
<tr>
<td></td>
<td>- Index and data pages are correctly linked.</td>
</tr>
<tr>
<td></td>
<td>- Indexes are correctly sorted.</td>
</tr>
<tr>
<td></td>
<td>- All pointers are consistent.</td>
</tr>
<tr>
<td></td>
<td>- Data information on each page is reasonable.</td>
</tr>
<tr>
<td></td>
<td>- Page offsets are reasonable.</td>
</tr>
<tr>
<td>checkverify</td>
<td>Verifies the results of the most recent run of dbcc checkstorage for the specified database.</td>
</tr>
<tr>
<td>fix_text</td>
<td>Upgrades text values after any Adaptive Server character set is converted to a new multibyte character set.</td>
</tr>
<tr>
<td>indexalloc</td>
<td>Checks the specified index to make sure all pages are correctly allocated, and that there are no unused allocated pages.</td>
</tr>
<tr>
<td>reindex</td>
<td>Checks the integrity of indexes on user tables by running a fast version of dbcc checktable.</td>
</tr>
<tr>
<td>tablealloc</td>
<td>Checks the specified table to make sure that all pages are correctly allocated, and that there are no unused allocated pages.</td>
</tr>
<tr>
<td>textalloc</td>
<td>Checks for a violation of the format of the root page of a text or image index.</td>
</tr>
<tr>
<td>tune</td>
<td>Enables or disables tuning flags for special performance situations.</td>
</tr>
</tbody>
</table>

All of the options in Table 1-33 on page 499 are database-level commands except for *tune*, which is a server-level command.
See Chapter 25, “Checking Database Consistency” in the *System Administration Guide* for more information on these *dbcc* commands.

**on all | database parameter and server-level commands**

The *on database* parameter specifies the database on which to invoke the database-level *revoke dbcc* command. Because *on master* revokes the ability to use *dbcc* commands on all databases, *on master* is the same as *on all*. You must be in the *master* database to use either the *on all* and *on master* parameters.

Neither the *on database* nor on all parameters work when invoking a server-level *revoke dbcc* command such as *dbcc tune*, because by doing so, you are forcing a server-level command to restrict itself to individual databases. For this reason, using the server-level *revoke dbcc tune on master* command raises an error.

**revoking default permissions on system tables**

default permissions on system tables revokes *sysobjects* (*audflags*) permissions from “public.”

Table 1-34 lists the system tables you can revoke the default permissions for when you issue the command from any database.

### Table 1-34: System tables with revocable permissions issued from any database

<table>
<thead>
<tr>
<th>sysalternates</th>
<th>sysjars</th>
<th>sysprotects</th>
<th>systabstats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysattributes</td>
<td>syskeys</td>
<td>sysqueryplans</td>
<td>systhresholds</td>
</tr>
<tr>
<td>syscolumns</td>
<td>syslogs</td>
<td>sysreferences</td>
<td>systypes</td>
</tr>
<tr>
<td>syscomments</td>
<td>sysobjects</td>
<td>sysroles</td>
<td>sysusermessages</td>
</tr>
<tr>
<td>sysconstraints</td>
<td>syspartitions</td>
<td>syssegments</td>
<td>sysusers</td>
</tr>
<tr>
<td>sysdepends</td>
<td>sysprocedures</td>
<td>sysstatistics</td>
<td>sysxtypes</td>
</tr>
<tr>
<td>sysindexes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1-35 lists the system tables you revoke the default permissions for when you issue this command from the *master* database.

sysalternates, sysjars, sysobjects (audflags), sysroles, sysstatistics, sysusermessages
Table 1-35: System tables with revocable permissions issued from master

<table>
<thead>
<tr>
<th>sysdatabases</th>
<th>sysprocesses</th>
<th>syscurconfigs</th>
<th>systimeranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>sydevices</td>
<td>sytransactions</td>
<td>syslanguages</td>
<td>sysresourcelimits</td>
</tr>
<tr>
<td>syslocks</td>
<td>sysusages</td>
<td>syscharsets</td>
<td>syslogins</td>
</tr>
<tr>
<td>sysmessages</td>
<td>sysconfigures</td>
<td>sysservers</td>
<td>sysremotelogins</td>
</tr>
</tbody>
</table>

Revoking permissions for update statistics, delete statistics, and truncate table

Adaptive Server allows you to revoke permissions for users, roles, and groups for the update statistics, delete statistics, and truncate table commands. Table owners can also provide permissions through an implicit grant by adding update statistics, delete statistics, and truncate table to a stored procedure and then granting execute permissions on that procedure to a user or role.

You cannot revoke permissions for update statistics at the column level. You must have the sso_role to run update statistics or delete statistics on sysroles, syssrvroles, and sysloginroles security tables.

By default, users with the sa_role have permission to run update statistics and delete statistics on system tables other than sysroles, syssrvroles, and sysloginroles, and can transfer this privilege to other users.

You can also issue grant all to grant permissions on update statistics, delete statistics, and truncate table.

Note Once you revoke permission to execute update statistics from a user, they also lose permission to execute variations of this command, such as update all statistics, update partition statistics, update index statistics, update statistics table, and so on. For example, the following revokes Billy permission from running all variations of update statistics on the authors table:

revoke update statistics on authors to billy

If you revoke a user’s permission to execute update statistics, you also revoke their ability to execute the variations of this command.

You cannot revoke variants of update statistics (for example, update index statistics) separately. That is, you cannot issue:

revoke update all statistics from harry

You cannot grant and revoke delete statistics permissions at the column level.

For more information, see the “Usage” section of grant on page 360.
The command fails and generates an error if a user issues update statistics, delete statistics, or truncate table and they:

- Do not own the table.
- Do not have the sa_role.
- Are not a database owner who has successfully used setuser to become the user who is the owner of the table.
- Have not been granted update statistics, delete statistics, or truncate table privileges.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

- **Command execution**  Only a System Administrator can revoke create database permission, and only from the master database. Only a System Security Officer can revoke create trigger and create encryption key permission.

- **Database consistency checking**  Only System Administrators can run revoke dbcc commands. Database Owners cannot run revoke dbcc.

- **Database object access**  revoke permission for database objects defaults to object owners. An object owner can revoke permission from other users on his or her own database objects.

- **Functions**  Only System Administrators can revoke permissions on built-in functions.

- **Proxy and session authorization**  Only a System Security Officer can revoke set proxy or set session authorization, and only from the master database.

- **Roles**  You can revoke roles only from the master database. Only a System Security Officer can revoke sso_role, oper_role, or a user-defined role from a user or a role. Only System Administrators can revoke sa_role from a user or a role. Only a user who has both sa_role and sso_role can revoke a role that includes sa_role.

- **Tables**  Database Owners can revoke default permissions on system tables.

  Table owners and the System Security Officer can revoke decrypt permission on a table or a list of columns in a table.

Auditing

Values in event and extrainfo columns of sysaudits are:
### Commands

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 47    | revoke       | revoke                    | - Roles – Current active roles  
- Keywords or options – NULL  
- Previous value – NULL  
- Current value – NULL  
- Other information – NULL  
- Proxy information – Original login name, if set proxy in effect |
| 85    | roles        | create role, drop role, alter role, grant role, or revoke role | - Roles – Current active roles  
- Keywords or options – NULL  
- Previous value – NULL  
- Current value – NULL  
- Other information – NULL  
- Proxy information – Original login name, if set proxy in effect |

See also
- **Commands** grant, setuser, set
- **Functions** proc_role

**System procedures** sp_activeroles, sp_adduser, sp_changedbowner, sp_changegroup, sp_displaylogin, sp_displayroles, sp_dropgroup, sp_dropuser, sp_helpgroup, sp_helpprotect, sp_helpuser, sp_modifylogin, sp_role
rollback

Description
Rolls back a user-defined transaction to the named savepoint in the transaction or to the beginning of the transaction.

Syntax
rollback [tran | transaction | work]
  [transaction_name | savepoint_name]

Parameters
tran | transaction | work
specifies that you want to roll back the transaction or the work. If you specify tran, transaction, or work, you can also specify the transaction_name or the savepoint_name.

transaction_name
is the name assigned to the outermost transaction. It must conform to the rules for identifiers.

savepoint_name
is the name assigned to the savepoint in the save transaction statement. The name must conform to the rules for identifiers.

Examples
Rolls back the transaction:

begin transaction
delete from publishers where pub_id = "9906"
rollback transaction

• rollback transaction without a transaction_name or savepoint_name rolls back a user-defined transaction to the beginning of the outermost transaction.

• rollback transaction transaction_name rolls back a user-defined transaction to the beginning of the named transaction. Though you can nest transactions, you can roll back only the outermost transaction.

• rollback transaction savepoint_name rolls a user-defined transaction back to the matching save transaction savepoint_name.

Usage

• If no transaction is currently active, the commit or rollback statement has no effect.

• The rollback command must appear within a transaction. You cannot roll back a transaction after commit has been entered.

Rolling back an entire transaction

• rollback without a savepoint name cancels an entire transaction. All the transaction’s statements or procedures are undone.
• If no savepoint_name or transaction_name is given with the rollback command, the transaction is rolled back to the first begin transaction in the batch. This also includes transactions that were started with an implicit begin transaction using the chained transaction mode.

Rolling back to a savepoint
• To cancel part of a transaction, use rollback with a savepoint_name. A savepoint is a marker set within a transaction by the user with the command save transaction. All statements or procedures between the savepoint and the rollback are undone.

After a transaction is rolled back to a savepoint, it can proceed to completion (executing any SQL statements after that rollback) using commit, or it can be canceled altogether using rollback without a savepoint. There is no limit on the number of savepoints within a transaction.

Rollbacks within triggers and stored procedures
• In triggers or stored procedures, rollback statements without transaction or savepoint names roll back all statements to the first explicit or implicit begin transaction in the batch that called the procedure or fired the trigger.

• When a trigger contains a rollback command without a savepoint name, the rollback aborts the entire batch. Any statements in the batch following the rollback are not executed.

• A remote procedure call (RPC) is executed independently from any transaction in which it is included. In a standard transaction (that is, not using Open Client™ DB-Library two-phase commit), commands executed via an RPC by a remote server are not rolled back with rollback and do not depend on commit to be executed.

• For complete information on using transaction management statements and on the effects of rollback on stored procedures, triggers, and batches, see the Transact-SQL User’s Guide.

Standards
ANSI SQL – Compliance level: Entry-level compliant.

Transact-SQL extensions
The rollback transaction and rollback tran forms of the statement and the use of a transaction name.

Permissions
rollback permission defaults to “public.” No permission is required to use it.

See also
Commands begin transaction, commit, create trigger, save transaction
rollback trigger

Description
Rolls back the work done in a trigger, including the data modification that caused the trigger to fire, and issues an optional raiserror statement.

Syntax
rollback trigger
[with raiserror_statement]

Parameters
with raiserror_statement
specifies a raiserror statement, which prints a user-defined error message and sets a system flag to record that an error condition has occurred. This provides the ability to raise an error to the client when rollback trigger is executed so that the transaction state in the error reflects the rollback. For information about the syntax and rules defining raiserror_statement, see the raiserror command.

Examples
Rolls back a trigger and issues the user-defined error message 25002:

        rollback trigger with raiserror 25002
        "title_id does not exist in titles table."

Usage
• When rollback trigger is executed, Adaptive Server aborts the currently executing command and halts execution of the rest of the trigger.

• If the trigger that issues rollback trigger is nested within other triggers, Adaptive Server rolls back all work done in these triggers up to and including the update that caused the first trigger to fire.

• Adaptive Server ignores a rollback trigger statement that is executed outside a trigger and does not issue a raiserror associated with the statement. However, a rollback trigger statement executed outside a trigger but inside a transaction generates an error that causes Adaptive Server to roll back the transaction and abort the current statement batch.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
rollback trigger permission defaults to “public.” No permission is required to use it.

See also
Commands create trigger, raiserror, rollback
save transaction

Description
Sets a savepoint within a transaction.

Syntax
save transaction savepoint_name

Parameters
savepoint_name
is the name assigned to the savepoint. It must conform to the rules for identifiers.

Examples
After updating the royaltyper entries for the two authors, insert the savepoint percentchanged, then determine how a 10 percent increase in the book’s price affects the authors’ royalty earnings. The transaction is rolled back to the savepoint with rollback transaction:

```sql
begin transaction royalty_change

update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

save transaction percentchanged

update titles
set price = price * 1.1
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

rollback transaction percentchanged

commit transaction
```

Usage
- For complete information on using transaction statements, see the Transact-SQL User’s Guide.
save transaction

- A savepoint is a user-defined marker within a transaction that allows portions of a transaction to be rolled back. `rollback savepoint_name` rolls back to the indicated savepoint; all statements or procedures between the savepoint and the rollback are undone.

  Statements preceding the savepoint are not undone—but neither are they committed. After rolling back to the savepoint, the transaction continues to execute statements. A rollback without a savepoint cancels the entire transaction. A commit allows it to proceed to completion.

- If you nest transactions, `save transaction` creates a savepoint only in the outermost transaction.

- There is no limit on the number of savepoints within a transaction.

- If no `savepoint_name` or `transaction_name` is given with the `rollback` command, all statements back to the first `begin transaction` in a batch are rolled back, and the entire transaction is canceled.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

`save transaction` permission defaults to “public.” No permission is required to use it.

See also

`Commands`  `begin transaction`, `commit`, `rollback`
select

Description
Retrieves rows from database objects.

Syntax
select ::=  
  select [all | distinct]  
  [top unsigned_integer]  
  select_list  
  into_clause  
  from_clause  
  where_clause  
  group_by_clause  
  having_clause  
  order_by_clause  
  compute_clause  
  read_only_clause  
  isolation_clause  
  browse_clause  
  plan_clause

select_list ::=  

Note For details on select_list, see the “Parameters” section.

into_clause ::=  
  into [([database.] owner.) table_name  
    [([colname encrypt [with [database.[owner].]keyname] [,  
    colname encrypt_clause])  
    [[external table at]  
      'server_name.[database].[owner].object_name'  
      | external directory at 'pathname'  
      | external file at 'pathname' [column delimiter 'string']]  
    [on segment_name]  
    [partition_clause]  
    [lock {datarows | datapages | allpages}]  
    [with [, into_option, into_option] ...]]]

| into existing table table_name

partition_clause ::=  
  partition by range (column_name[, column_name]...)  
  ([partition_name] values <= ([constant | MAX]  
    [, (constant | MAX)] ...) [on segment_name]  
  [, [partition_name] values <= ([constant | MAX]  
    [, (constant | MAX)] ...) [on segment_name]...  
  | partition by hash (column_name[, column_name]...)  
  ( [partition_name [on segment_name]  
    [, partition_name [on segment_name]]...)  
  | number_of_partitions
select

| partition by list (column_name)
| (partition_name) values (constant[, constant] ...)
| [on segment_name]
| , [partition_name] values (constant[, constant] ...)
| [on segment_name][...]

| partition by roundrobin
| (partition_name [on segment_name]
| , partition_name [on segment_name]...)
| number_of_partitions
| [on (segment_name [, segment_name][...])]

into_option ::= |
| max_rows_per_page = num_rows
| exp_row_size = num_bytes
| reservepagegap = num_pages
| identity_gap = gap

from_clause ::= from table_reference [, table_reference]...

table_reference ::= table_view_name | ANSI_join

table_view_name ::= [database]. [owner]. {{table_name | view_name}
| as [correlation_name]
| (index {index_name | table_name})
| parallel [degree_of_parallelism]
| [prefetch size][lru | mru]}
| [holdlock | noholdlock]
| [readpast]
| [shared]

ANSI_join ::= table_reference join_type join table_reference
| join_conditions
| join_type ::= inner | left [outer] | right [outer]
| join_conditions ::= on search_conditions

where_clause ::= where search_conditions

group_by_clause ::= group by [all] aggregate_free_expression
| [ , aggregate_free_expression]...

having_clause ::= having search_conditions

order_by_clause ::= order by sort_clause [, sort_clause]...
sort_clause ::= 
  { [[ database ] [ owner ] ] [ table_name | view_name ] } column_name 
  | select_list_number 
  | expression 
  [ asc | desc ]

compute_clause ::= 
  compute row_aggregate ( column_name ) 
  [ , row_aggregate ( column_name ) ] ... 
  [ by column_name [ , column_name ] ... ]

read_only_clause ::= 
  for { read only | update [ of column_name_list ] }

isolation_clause ::= 
  at isolation 
  { read uncommitted | 0 } 
  | { read committed | 1 } 
  | { repeatable read | 2 } 
  | { serializable | 3 } 

browse_clause ::= 
  for browse

plan_clause ::= 
  plan "abstract plan"

Parameters

all
includes all rows in the results. all is the default.
distinct
includes only unique rows in the results. distinct must be the first word in the select list. distinct is ignored in browse mode.

Null values are considered equal for the purposes of the keyword distinct: only one NULL is selected, no matter how many are encountered.
top unsigned_integer
is used with select...into statements to limit the number of rows inserted in
the target table. This is different from set rowcount, which is ignored during
a select...into.

- When used with delete, update, or in a view, you cannot specify
ordering. If there is an implied order on the table from a clustered index,
that order applies; otherwise, the results are unpredictable, as they can
be in any order.

- $n$ is an unsigned 32-bit value between 0 through $2^{32}-1$ (4GB-1 or

- When used with cursors, top $n$ limits the overall size of the result set.
Specifying set cursor rowcount limits the results of a single fetch.

- When a view definition contains select top $n$ and a query with a where
clause uses it, the results may be inconsistent.

**select_list**
consists of one or more of the following items:

- “*”, representing all columns in create table order.

- A list of column names in the order in which you want to see them.
When selecting an existing IDENTITY column, you can substitute the
syb_identity keyword, qualified by the table name, where necessary, for
the actual column name.

- A specification to add a new IDENTITY column to the result table:

  $column_name = identity (int | smallint | tinyint | precision)$

If you specify int, smallint, or tinyint, the resulting column is an integer.
If you specify precision, the result is numeric datatype.

- A replacement for the default column heading (the column name), in
one of these forms:

  $column_heading = column_name$

  $column_name column_heading$
column_name as column_heading

The column heading can be enclosed in quotation marks for any of these forms. The heading must be enclosed in quotation marks if it is not a valid identifier (that is, if it is a reserved word, if it begins with a special character, or if it contains spaces or punctuation marks).

- An expression (a column name, constant, function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery).
- A built-in function or an aggregate.
- Any combination of the items listed above.

The select_list can also assign values to variables, in the form:

@variable = expression 
[, @variable = expression ... ]

You cannot combine variable assignment with any other select_list option.

into

except when used with existing table, creates a new table based on the columns specified in the select list and the rows chosen in the where clause. See “Using select into” on page 530.

colname encrypt

Specifies encryption on colname in the target table. By default, Adaptive Server decrypts data selected from the source table. You must use the encrypt keyword to preserve the data encryption or to encrypt a column in the target database that was not encrypted in the source database.

with database...key

Specifies the key used on the source data, or a different key.

external [[table] | directory | file]

indicates that the type of the external object is a table, directory, or file. If you do not indicate a file, directory, or table, select into assumes that you are using a table.

Note You cannot specify an external location when using any part of the partition_clause. Partitions can be created only on tables on the current server and database.
select

'server_name.[database].[owner].object_name'
indicates that you are selecting into a table or view found on the remote server_name.

at 'path_name'
indicates the full, operating system-specific path name of the external file or directory you are selecting into. All directories in path_name must be accessible to Adaptive Server.

column delimeter 'string'
indicates the delimeter that you are using to separate columns after converting the column’s data to string format. string can have as many as 16 characters. If you do not specify a delimiter, select into uses the tab character.

existing table table_name
indicates that you are selecting data into a proxy table. You cannot use this select into with any other table type except proxy. The column list in the select list must match the type, length, and number in the proxy table.

on segment_name
specifies that the table is to be created on the named segment. Before the on segment_name option can be used, the device must be initialized with disk init, and the segment must be added to the database with sp_addsegment. See your System Administrator or use sp_helpsegment for a list of the segment names available in your database.

partition by range
specifies records are to be partitioned according values in the partitioning column or columns. Each partitioning column value is compared with sets of user-supplied upper and lower bounds to determine partition assignment.

column_name
when used in the partition_clauses, specifies a partition key column.

partition_name
specifies the name of a new partition on which table records are to stored. Partition names must be unique within the set of partitions on a table or index. Partition names can be delimited identifiers if set quoted_identifier is on. Otherwise, they must be valid identifiers.

If partition_name is omitted, Adaptive Server creates a name in the form table_name_partition_id. Adaptive Server truncates partition names that exceed the allowed maximum length.
values <= constant | MAX
specifies the inclusive upper bound of values for a named partition.
Specifying a constant value for the highest partition bound imposes an
implicit integrity constraint on the table. The keyword MAX specifies the
maximum value in a given datatype.

on segment_name
when used in the partition_clause, specifies the name of the segment on
which to place the partition. When using on segment_name, the logical
device must already have been assigned to the database with create database
or alter database, and the segment must have been created in the database
with sp_addsegment. See your System Administrator or use sp_helpsegment
for a list of the segment names available in your database.

partition by hash
specifies records are to be partitioned by a system-supplied hash function.
The function computes the hash value of the partition keys that specify the
partition to which records are assigned.

partition by list
specifies records are to be partitioned according to literal values specified in
the named column. The partition key contains only one column. You can list
up to 250 constants as the partition values for each list partition.

partition by roundrobin
specifies records are to be partitioned in a sequential manner. A round-robin
partitioned table has no partitioning key. Neither the user nor the optimizer
knows in which partition a particular record resides.

lock datarows | datapages | allpages
specifies the locking scheme to be used for a table created with a select into
command. The default is the server-wide setting for the configuration
parameter lock scheme.

max_rows_per_page
limits the number of rows on data pages for a table created with select into.
Unlike fillfactor, the max_rows_per_page value is maintained when data is
inserted or deleted. max_rows_per_page is not supported on
data-only-locked tables.
exp_row_size = num_bytes

specifies the expected row size for a table created with the `select into` command. Valid only for datarows and datapages locking schemes and only for tables that have variable-length rows. Valid values are 0, 1, and any value greater than the minimum row length and less than the maximum row length for the table. The default value is 0, which means that a server-wide default is used.

reservepagegap = num_pages

specifies a ratio of filled pages to empty pages that is to be left as `select into` allocates extents to store data. This option is valid only for the `select into` command. For each specified `num_pages`, one empty page is left for future expansion of the table. Valid values are 0 – 255. The default value is 0.

readpast

specifies that the query should silently skip rows with exclusive locks, without waiting and without generating a message.

with identity_gap

specifies the identity gap for the table. This value overrides the system identity gap setting for this table only.

value

is the identity gap amount.

If you are creating a table in a `select into` statement from a table that has a specific identity gap setting, the new table does not inherit the identity gap setting from the parent table. Instead, the new table uses the identity burning set factor setting. To give the new table a specific `identity_gap` setting, specify the identity gap in the `select into` statement. You can give the new table an identity gap that is the same as or different from the parent table.

from

indicates which tables and views to use in the `select` statement, from required except when the `select` list contains no column names (that is, it contains constants and arithmetic expressions only):

```
select 5 x, 2 y, "the product is", 5*2 Result

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>the product is 10</td>
</tr>
</tbody>
</table>
```
At most, a query can reference 50 tables and 14 worktables (such as those created by aggregate functions). The 50-table limit includes:

- Tables (or views on tables) listed in the `from` clause
- Each instance of multiple references to the same table (self-joins)
- Tables referenced in subqueries
- Tables being created with `into`
- Base tables referenced by the views listed in the `from` clause

`view_name`, `table_name` lists tables and views used in the `select` statement. Specify the database name if the table or view is in another database, and specify the owner’s name if more than one table or view of that name exists in the database. The default value for `owner` is the current user, and the default value for `database` is the current database.

If there is more than one table or view in the list, separate their names by commas. The order of the tables and views following the keyword `from` does not affect the results.

You can query tables in different databases in the same statement.

Table names and view names can be given correlation names (aliases), either for clarity or to distinguish the different roles that tables or views play in self-joins or subqueries. To assign a correlation name, give the table or view name, then a space, then the correlation name, like this:

```sql
select pub_name, title_id
from publishers pu, titles t
where t.pub_id = pu.pub_id
```

All other references to that table or view (for example, in a `where` clause) must use the correlation name. Correlation names cannot begin with a numeral.

`index index_name` specifies the index to use to access `table_name`. You cannot use this option when you select from a view, but you can use it as part of a `select` clause in a `create view` statement.

`parallel` specifies a parallel partition or index scan, if Adaptive Server is configured to allow parallel processing.
degree_of_parallelism

specifies the number of worker processes that will scan the table or index in parallel. If set to 1, the query executes serially.

prefetch size

specifies the I/O size, in kilobytes, for tables bound to caches with large I/Os configured. You cannot use this option when you select from a view, but you can use it as part of a select clause in a create view statement. sp_helpcache shows the valid sizes for the cache an object is bound to or for the default cache. To configure the data cache size, use sp_cacheconfigure.

When using prefetch and designating the prefetch size (size), the minimum is 2K and any power of two on the logical page size up to 16K. prefetch size options in kilobytes are:

<table>
<thead>
<tr>
<th>Logical page size</th>
<th>Prefetch size options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2, 4, 8, 16</td>
</tr>
<tr>
<td>4</td>
<td>4, 8, 16, 32</td>
</tr>
<tr>
<td>8</td>
<td>8, 16, 32, 64</td>
</tr>
<tr>
<td>16</td>
<td>16, 32, 64, 128</td>
</tr>
</tbody>
</table>

The prefetch size specified in the query is only a suggestion. To allow the size specification, configure the data cache at that size. If you do not configure the data cache to a specific size, the default prefetch size is used.

If Component Integration Services is enabled, you cannot use prefetch for remote servers.

lru | mru

specifies the buffer replacement strategy to use for the table. Use lru to force the optimizer to read the table into the cache on the MRU/LRU (most recently used/least recently used) chain. Use mru to discard the buffer from cache and replace it with the next buffer for the table. You cannot use this option when you select from a view, but you can use it as part of a select clause in a create view statement.
holdlock

makes a shared lock on a specified table or view more restrictive by holding it until the transaction completes (instead of releasing the shared lock as soon as the required data page is no longer needed, whether or not the transaction has completed).

The holdlock option applies only to the table or view for which it is specified, and only for the duration of the transaction defined by the statement in which it is used. Setting the transaction isolation level 3 option of the set command implicitly applies a holdlock for each select statement within a transaction. The keyword holdlock is not permitted in a select statement that includes the for browse option. You cannot specify both a holdlock and a noholdlock option in a query.

If Component Integration Services is enabled, you cannot use holdlock for remote servers.

noholdlock

prevents the server from holding any locks acquired during the execution of this select statement, regardless of the transaction isolation level currently in effect. You cannot specify both a holdlock and a noholdlock option in a query.

shared

instructs Adaptive Server to use a shared lock (instead of an update lock) on a specified table or view. This allows other clients to obtain an update lock on that table or view. You can use the shared keyword only with a select clause included as part of a declare cursor statement. For example:

```sql
declare shared_crsr cursor
for select title, title_id
from titles shared
where title_id like "BU%"
```

You can use the holdlock keyword in conjunction with shared after each table or view name, but holdlock must precede shared.

ANSL join

an inner or outer join that uses the ANSI syntax. The from clause specifies the tables to be joined.

inner

includes only the rows of the inner and outer tables that meet the conditions of the on clause. The result set of a query that includes an inner join does not include any null-supplied rows for the rows of the outer table that do not meet the conditions of the on clause.
select

outer
includes all the rows from the outer table whether or not they meet the
conditions of the on clause. If a row does not meet the conditions of the on
clause, values from the inner table are stored in the joined table as null
values. The where clause of an ANSI outer join restricts the rows that are
included in the query result.

left
left joins retain all the rows of the table reference listed on the left of the join
clause. The left table reference is referred to as the outer table or
row-preserving table.

In the queries below, T1 is the outer table and T2 is the inner table:

```
T1 left join T2
T2 right join T1
```

right
right joins retain all the rows of the table reference on the right of the join
clause (see example above).

search_conditions
used to set the conditions for the rows that are retrieved. A search condition
can include column names, expressions, arithmetic operators, comparison
operators, the keywords not, like, is null, and, or, between, in, exists, any, and
all, subqueries, case expressions, or any combination of these items. See
where clause on page 630 for more information.

group by
finds a value for each group. These values appear as new columns in the
results, rather than as new rows.

When group by is used with standard SQL, each item in the select list must
either have a fixed value in every row in the group or be used with aggregate
functions, which produce a single value for each group. Transact-SQL has
no such restrictions on the items in the select list. Also, Transact-SQL allows
you to group by any expression (except by a column alias); with standard
SQL, you can group by a column only.

You can use the aggregates listed in Table 1-36 with group by (expression is
almost always a column name):

<table>
<thead>
<tr>
<th>Aggregate function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum ([all</td>
<td>distinct] expression)</td>
</tr>
<tr>
<td>avg ([all</td>
<td>distinct] expression)</td>
</tr>
<tr>
<td>count ([all</td>
<td>distinct] expression)</td>
</tr>
</tbody>
</table>
See group by and having clauses on page 390 for more information.

A table can be grouped by any combination of columns—that is, groups can be nested within each other. You cannot group by a column heading; you must use a column name, an expression, or a number representing the position of the item in the select list.

**group by all** includes all groups in the results, even those that do not have any rows that meet the search conditions. See group by and having clauses on page 390 for an example.

**aggregate_free_expression**

is an expression that includes no aggregates.

**having**

sets conditions for the group by clause, similar to the way that where sets conditions for the select clause. There is no limit on the number of conditions that can be included.

You can use a having clause without a group by clause.

If any columns in the select list do not have aggregate functions applied to them and are not included in the query’s group by clause (illegal in standard SQL), the meanings of having and where are somewhat different.

In this situation, a where clause restricts the rows that are included in the calculation of the aggregate, but does not restrict the rows returned by the query. Conversely, a having clause restricts the rows returned by the query, but does not affect the calculation of the aggregate. See group by and having clauses on page 390 for examples.

**order by**

sorts the results by columns. In Transact-SQL, you can use order by for items that do not appear in the select list. You can sort by a column name, a column heading (or alias), an expression, or a number representing the position of the item in the select list (the *select_list_number*). If you sort by select list number, the columns to which the order by clause refers must be included in the select list, and the select list cannot be * (asterisk).

<table>
<thead>
<tr>
<th>Aggregate function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>`count_big ([all</td>
<td>distinct] expression)`</td>
</tr>
<tr>
<td><code>count (*)</code></td>
<td>Number of selected rows returned as an integer.</td>
</tr>
<tr>
<td><code>count_big (*)</code></td>
<td>Number of selected rows returned as a bigint.</td>
</tr>
<tr>
<td><code>max (expression)</code></td>
<td>Highest value in the column.</td>
</tr>
<tr>
<td><code>min (expression)</code></td>
<td>Lowest value in the column.</td>
</tr>
</tbody>
</table>
asc
sorts results in ascending order (the default).

desc
sorts results in descending order.

compute
used with row aggregates (sum, avg, min, max, count, and count_big) to
generate control break summary values. The summary values appear as
additional rows in the query results, allowing you to see detail and summary
rows with one statement.

You cannot use a select into clause with compute.

If you use compute by, you must also use an order by clause. The columns
listed after compute by must be identical to or a subset of those listed after
order by, and must be in the same left-to-right order, start with the same
expression, and not skip any expressions.

For example, if the order by clause is order by a, b, c, the compute by
clause can be any (or all) of these:

  compute by a, b, c
  compute by a, b
  compute by a

The keyword compute can be used without by to generate grand totals, grand
counts, and so on. order by is optional if you use compute without by. See
compute clause on page 66 for details and examples.

If Component Integration Services is enabled, compute is not forwarded to
remote servers.

for {read only | update}
specifies that a cursor result set is read-only or updatable. You can use this
option only within a stored procedure and only when the procedure defines
a query for a cursor. In this case, the select is the only statement allowed in
the procedure. It defines the for read only or for update option (instead of the
declare cursor statement). This method of declaring cursors provides the
advantage of page-level locking while fetching rows.

If the select statement in the stored procedure is not used to define a cursor,
Adaptive Server ignores the for read only | update option. See the Embedded
SQL™ documentation for more information about using stored procedures
to declare cursors. For information about read-only or updatable cursors, see
the Transact-SQL User’s Guide.
of column_name_list

is the list of columns from a cursor result set defined as updatable with the
for update option.

at isolation

specifies the isolation level (0, 1, 2 or 3) of the query. If you omit this clause,
the query uses the isolation level of the session in which it executes
(isolation level 1 by default). The at isolation clause is valid only for single
queries or within the declare cursor statement. Adaptive Server returns a
syntax error if you use at isolation:

- With a query using the into clause
- Within a subquery
- With a query in the create view statement
- With a query in the insert statement
- With a query using the for browse clause

If there is a union operator in the query, you must specify the at isolation
clause after the last select. If you specify holdlock, noholdlock, or shared in a
query that also specifies at isolation read uncommitted, Adaptive Server
issues a warning and ignores the at isolation clause. For the other isolation
levels, holdlock takes precedence over the at isolation clause. For more
information about isolation levels, see the Transact-SQL User’s Guide.

If Component Integration Services is enabled, you cannot use at isolation for
remote servers.

read uncommitted | 0

specifies isolation level 0 for the query.

read committed | 1

specifies isolation level 1 for the query.

repeatable read | 2

specifies transaction isolation level 2 for the query.

serializable | 3

specifies isolation level 3 for the query.

for browse

must be attached to the end of a SQL statement sent to Adaptive Server in a
DB-Library browse application. See the Open Client DB-Library Reference
Manual for details.
plan "abstract plan" specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. See Chapter 30, “Creating and Using Abstract Plans,” in the Performance and Tuning Guide for more information.

Examples

**Example 1** Selects all rows and columns from the publishers table:

```
select * from publishers
```

<table>
<thead>
<tr>
<th>pub_id</th>
<th>pub_name</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>New Age Books</td>
<td>Boston</td>
<td>MA</td>
</tr>
<tr>
<td>0877</td>
<td>Binnet &amp; Hardley</td>
<td>Washington</td>
<td>DC</td>
</tr>
<tr>
<td>1389</td>
<td>Algodata Infosystems</td>
<td>Berkeley</td>
<td>CA</td>
</tr>
</tbody>
</table>

**Example 2** Selects all rows from specific columns of the publishers table:

```
select pub_id, pub_name, city, state from publishers
```

**Example 3** Selects all rows from specific columns of the publishers table, substituting one column name and adding a string to the output:

```
select "The publisher's name is", Publisher = pub_name, pub_id
from publishers
```

<table>
<thead>
<tr>
<th>Publisher</th>
<th>pub_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>The publisher’s name is New Age Books</td>
<td>0736</td>
</tr>
<tr>
<td>The publisher’s name is Binnet &amp; Hardley</td>
<td>0877</td>
</tr>
<tr>
<td>The publisher’s name is Algodata Infosystems</td>
<td>1389</td>
</tr>
</tbody>
</table>

**Example 4** Selects all rows from specific columns of the titles table, substituting column names:

```
select type as Type, price as Price
from titles
```

**Example 5** Specifies the locking scheme and the reserve page gap for select into:

```
select title_id, title, price
into bus_titles
lock datarows with reservepagegap = 10
from titles
where type = "business"
```
Example 6 Encrypts the creditcard column when selecting into the bigspenders table:

```sql
select creditcard, custid, sum(amount) into #bigspenders (creditcard encrypt with cust.database.new_cc_key) from daily_xacts
group by creditcard having sum(amount) > $5000
```

Example 7 Selects only the rows that are not exclusively locked. If any other user has an exclusive lock on a qualifying row, that row is not returned:

```sql
select title, price
from titles readpast
    where type = "news"
    and price between $20 and $30
```

Example 8 Selects specific columns and rows, placing the results into the temporary table #advance_rpt:

```sql
select pub_id, total = sum (total_sales)
    into #advance_rpt
from titles
where advance < $10000
    and total_sales is not null
group by pub_id
having count (*) > 1
```

Example 9 Selects the top 3 rows from au_lname from the authors table:

```sql
select top 3 au_lname from authors
```

Example 10 Concatenates two columns and places the results into the temporary table #tempnames:

```sql
select "Author_name" = au_fname + " " + au_lname
    into #tempnames
from authors
```

Example 11 Selects specific columns and rows, returns the results ordered by type from highest to lowest, and calculates summary information:

```sql
select type, price, advance from titles
order by type desc
compute avg (price), sum (advance) by type
compute sum (price), sum (advance)
```

Example 12 Selects specific columns and rows, and calculates totals for the price and advance columns:

```sql
select type, price, advance from titles compute sum (price), sum (advance)
```
Example 13  Creates the coffeetabletitles table, a copy of the titles table which includes only books priced over $20:

    select * into coffeetabletitles from titles
    where price > $20

Example 14  Creates the newtitles table, an empty copy of the titles table:

    select * into newtitles from titles
    where 1 = 0

Example 15  Gives an optimizer hint:

    select title_id, title
        from titles (index title_id_ind prefetch 16)
        where title_id like "BU%"

Example 16  Selects the IDENTITY column from the sales_east and sales_west tables by using the syb_identity keyword:

    select sales_east.syb_identity,
        sales_west.syb_identity
    from sales_east, sales_west

Example 17  Creates the newtitles table, a copy of the titles table with an IDENTITY column:

    select *, row_id = identity (10)
    into newtitles from titles

Example 18  Specifies a transaction isolation level for the query.

    select pub_id, pub_name
    from publishers
    at isolation read uncommitted

Example 19  Selects from titles using the repeatable read isolation level. No other user can change values in or delete the affected rows until the transaction completes:

    begin tran
    select type, avg (price)
        from titles
        group by type
    at isolation repeatable read

Example 20  Gives an optimizer hint for the parallel degree for the query:

    select ord_num from salesdetail
    (index salesdetail parallel 3)
Example 21  Joins the titleauthor and the titles tables on their title_id columns. The result set only includes those rows that contain a price greater than 15:

```sql
select au_id, titles.title_id, title, price
from titleauthor inner join titles
on titleauthor.title_id = titles.title_id
and price > 15
```

Example 22  The result set contains all the authors from the authors table. The authors who do not live in the same city as their publishers produce null values in the pub_name column. Only the authors who live in the same city as their publishers, Cheryl Carson and Abraham Bennet, produce a non-null value in the pub_name column:

```sql
select au_fname, au_lname, pub_name
from authors left join publishers
on authors.city = publishers.city
```

Example 23  Create a new table (newtable) from the existing table (oldtable) with an identity gap, you specify it in the select into statement:

```sql
select identity into newtable
with identity_gap = 20
from oldtable
```


Example 24  Creates a new table sales_report from an existing table sales_detail. The new table is partitioned by range on the qty column.

```sql
select * into sales_report partition by range (qty)
(smallorder values <= (500) on seg1,
bigorder values <= (5000) on seg2)
from sales_detail
```

Example 25  Use this query to find the statements that incur too many IOs as the candidates for tuning.

```sql
select lio_avg, qtext from sysquerymetrics order by lio_avg
```

**Usage**

- The keywords in the select statement, as in all other statements, must be used in the order shown in the syntax statement.
- The maximum number of expressions in a select statement is 4096.
The keyword all can be used after select for compatibility with other implementations of SQL. all is the default. Used in this context, all is the opposite of distinct. All retrieved rows are included in the results, whether or not some are duplicates.

Except in create table, create view, and select into statements, column headings may include any characters, including blanks and Adaptive Server keywords, if the column heading is enclosed in quotes. If the heading is not enclosed in quotes, it must conform to the rules for identifiers.

The character string indicated by like cannot be longer than 255 bytes.

You cannot use the select...for browse option on tables containing more than 255 columns.

Column headings in create table, create view, and select into statements, as well as table aliases, must conform to the rules for identifiers.

To insert data with select from a table that has null values in some fields into a table that does not allow null values, you must provide a substitute value for any NULL entries in the original table. For example, to insert data into an advances table that does not allow null values, this example substitutes “0” for the NULL fields:

```
insert advances
    select pub_id, isnull (advance, 0) from titles
```

Without the isnull function, this command would insert all the rows with non-null values into the advances table, and produce error messages for all rows where the advance column in the titles table contained NULL.

If you cannot make this kind of substitution for your data, you cannot insert data containing null values into the columns with the NOT NULL specification.

Two tables can be identically structured, and yet be different as to whether null values are permitted in some fields. Use sp_help to see the null types of the columns in your table.

The default length of the text, unitext, or image data returned with a select statement is 32K. Use set textsize to change the value. The size for the current session is stored in the global variable @@textsize. Certain client software may issue a set textsise command on logging in to Adaptive Server.
• Data from remote Adaptive Servers can be retrieved through the use of remote procedure calls. See \texttt{create procedure} and \texttt{execute} for more information.

• A \texttt{select} statement used in a cursor definition (through \texttt{declare cursor}) must contain a \texttt{from} clause, but it cannot contain a \texttt{compute}, \texttt{for browse}, or \texttt{into} clause. If the \texttt{select} statement contains any of the following constructs, the cursor is considered read-only and not updatable:
  
  • distinct option
  
  • group by clause
  
  • Aggregate functions
  
  • union operator

If you declare a cursor inside a stored procedure with a \texttt{select} statement that contains an \texttt{order by} clause, that cursor is also considered read-only. Even if it is considered updatable, you cannot delete a row using a cursor that is defined by a \texttt{select} statement containing a join of two or more tables. See \texttt{declare cursor} for more information.

• If a \texttt{select} statement that assigns a value to a variable returns more than one row, the last returned value is assigned to the variable. For example:

\begin{verbatim}
declare @x varchar (40)
select @x = pub_name from publishers
print @x
\end{verbatim}

(3 rows affected)

Algodata Infosystems

Using ANSI join syntax

• Before you write queries using the ANSI inner and outer join syntax, read “Outer Joins” in Chapter 4, “Joins: Retrieving Data From Several Tables,” in the \textit{Transact-SQL User’s Guide}. 
Using \texttt{select into}

- \texttt{select into} is a two-step operation. The first step creates the new table, and the second step inserts the specified rows into the new table.

\textbf{Note} You can \texttt{select into} a Component Integration Services existing table.

Because the rows inserted by \texttt{select into} operations are not logged, \texttt{select into} commands cannot be issued within user-defined transactions, even if the \texttt{ddl in tran} database option is set to \texttt{true}. Page allocations during \texttt{select into} operations are logged, so large \texttt{select into} operations may fill the transaction log.

If a \texttt{select into} statement fails after creating a new table, Adaptive Server does not automatically drop the table or deallocate its first data page. This means that any rows inserted on the first page before the error occurred remain on the page. Check the value of the \texttt{@@error} global variable after a \texttt{select into} statement to be sure that no error occurred. Use the \texttt{drop table} statement to remove the new table, then reissue the \texttt{select into} statement.

- The name of the new table must be unique in the database and must conform to the rules for identifiers. You can also \texttt{select into} temporary tables (see Examples 7, 8, and 11).

- Any rules, constraints, or defaults associated with the base table are not carried over to the new table. Bind rules or defaults to the new table using \texttt{sp_bindrule} and \texttt{sp_bindefault}.

- \texttt{select into} does not carry over the base table’s \texttt{max_rows_per_page} value, and it creates the new table with a \texttt{max_rows_per_page} value of 0. Use \texttt{sp_chgattribute} to set the \texttt{max_rows_per_page} value.

- The \texttt{select into/bulkcopy/pllsort} option must be set to \texttt{true} (by executing \texttt{sp_dboption}) in order to \texttt{select into} a permanent table. You do not have to set the \texttt{select into/bulkcopy/pllsort} option to \texttt{true} in order to \texttt{select into} a temporary table, since the temporary database is never recovered.

After you have used \texttt{select into} in a database, you must perform a full database dump before you can use the \texttt{dump transaction} command. \texttt{select into} operations log only page allocations and not changes to data rows. Therefore, changes are not recoverable from transaction logs. In this situation, issuing the \texttt{dump transaction} statement produces an error message instructing you to use \texttt{dump database} instead.
By default, the select into/bulkcopy/pllsort option is set to false in newly created databases. To change the default situation, set this option to true in the model database.

- `select into` can be used with an archive database.
- `select into` runs more slowly while a dump database is taking place.
- You can use `select into` to create a duplicate table with no data by having a false condition in the `where` clause (see Example 12).
- You must provide a column heading for any column in the select list that contains an aggregate function or any expression. The use of any constant, arithmetic or character expression, built-in functions, or concatenation in the select list requires a column heading for the affected item. The column heading must be a valid identifier or must be enclosed in quotation marks (see Examples 7 and 8).
- Datatypes and nullability are implicitly assigned to literal values when `select into` is used, such as:

  ```sql
  select x = getdate () into mytable
  ```

  This results in a non-nullable column, regardless of whether allow nulls by default is on or not. It depends upon how the select commands are used and with what other commands within the syntax.

  The `convert` syntax allows you to explicitly specify the datatype and nullability of the resulting column, not the default.

  Wrap `getdate` with a function that does result in a null, such as:

  ```sql
  select x = nullif (getdate (), "1/1/1900") into mytable
  ```

  Or, use the `convert` syntax:

  ```sql
  select x = convert (datetime null, getdate ()) into mytable
  ```

- You cannot use `select into` inside a user-defined transaction or in the same statement as a `compute` clause.

- To select an IDENTITY column into a result table, include the column name (or the `syb_identity` keyword) in the select statement’s `column_list`. The new column observes the following rules:

  - If an IDENTITY column is selected more than once, it is defined as NOT NULL in the new table. It does not inherit the IDENTITY property.
• If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as NULL if any column in the expression allows nulls; otherwise, it is created as NOT NULL.

• If the select statement contains a group by clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created NULL; others are NOT NULL.

• An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a NULL column, the new column is defined as NULL. Otherwise, it is defined as NOT NULL.

• You cannot use select into to create a new table with multiple IDENTITY columns. If the select statement includes both an existing IDENTITY column and a new IDENTITY specification of the form column_name = identity (precision), the statement fails.

• If Component Integration Services is enabled, and if the into table resides on Adaptive Server, Adaptive Server uses bulk copy routines to copy the data into the new table. Before doing a select into with remote tables, set the select into/bulkcopy database option to true.

• For information about the Embedded SQL command select into host_var_list, see the Open Client Embedded SQL Reference Manual.

Converting the NULL properties of a target column with select...into

• Use the convert command to change the nullability of a target column into which you are selecting data. For example, the following selects data from the titles table into a target table named temp_titles, but converts the total_sales column from null to not null:

```
select title, convert (char (100) not null, total_sales)
  total_sales
into #tempsales
from titles
```

Specifying a lock scheme using select...into

• The lock option, used with select...into, allows you to specify the locking scheme for the table created by the command. If you do not specify a locking scheme, the default locking scheme, as set by the configuration parameter lock scheme, is applied.
When you use the lock option, you can also specify the space management properties max_rows_per_page, exp_row_size, and reservepagegap.

You can change the space management properties for a table created with select into, using sp_chgattribute.

Specifying a partition strategy using select...into

- The partitions_clause, when used with select...into, allows you to specify the partition properties of the table created by the command. (See create table for more information.) If you do not specify a partition type, Adaptive Server creates an unpartitioned table. If any row to be inserted does not satisfy the criteria for any partition in the target table, select...into fails.

Using index, prefetch, and lru | mru

- The index, prefetch and lru | mru options specify the index, cache and I/O strategies for query execution. These options override the choices made by the Adaptive Server optimizer. Use them with caution, and always check the performance impact with set statistics io on. For more information about using these options, see the Performance and Tuning Guide.

Using parallel

- The parallel option reduces the number of worker threads that the Adaptive Server optimizer can use for parallel processing. The degree_of_parallelism cannot be greater than the configured max parallel degree. If you specify a value that is greater than the configured max parallel degree, the optimizer ignores the parallel option.

- When multiple worker processes merge their results, the order of rows that Adaptive Server returns may vary from one execution to the next. To get rows from partitioned tables in a consistent order, use an order by clause, or override parallel query execution by using parallel 1 in the from clause of the query.

- A from clause specifying parallel is ignored if any of the following conditions is true:
  - The select statement is used for an update or insert.
  - The from clause is used in the definition of a cursor.
  - parallel is used in the from clause within any inner query blocks of a subquery.
  - The select statement creates a view.
  - The table is the inner table of an outer join.
The query specifies \text{min} or \text{max} on the table and specifies an index.

An unpartitioned clustered index is specified or is the only parallel option.

The query specifies exists on the table.

The value for the configuration parameter \text{max scan parallel degree} is 1 and the query specifies an index.

A nonclustered index is covered. For information on index covering, see Chapter 9, “How Indexes Work” in the \textit{Performance and Tuning Guide}.

The table is a system table or a virtual table.

The query is processed using the OR strategy. For an explanation of the OR strategy, see the \textit{Performance and Tuning Guide}.

The query returns a large number of rows to the user.

Using \textit{readpast}

- The \textit{readpast} option allows a \texttt{select} command to access the specified table without being blocked by incompatible locks held by other tasks. \textit{readpast} queries can only be performed on data-only-locked tables.

- If the \textit{readpast} option is specified for an allpages-locked table, the \textit{readpast} option is ignored. The command operates at the isolation level specified for the command or session. If the isolation level is 0, dirty reads are performed, and the command returns values from locked rows and does not block. If the isolation level is 1 or 3, the command blocks when pages with incompatible locks must be read.

- The interactions of session-level isolation levels and \textit{readpast} on a table in a \texttt{select} command are shown in Table 1-37.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Session isolation level & Effects \tabularnewline\hline
0, read uncommitted & \textit{readpast} is ignored, and rows containing uncommitted transactions are returned to the user. A warning message is printed. \tabularnewline (dirty reads) & \textit{readpast} is ignored, and rows containing uncommitted transactions are returned to the user. A warning message is printed. \tabularnewline
1, read committed & Rows or pages with incompatible locks are skipped; no locks are held on the rows or pages \textit{readpast} may produce duplicates and adding the \texttt{distinct} clause does not clear this problem. To resolve this, when using \textit{readpast}, use a \texttt{group by} clause \textit{in addition to} a \texttt{distinct} clause to avoid duplicates. \tabularnewline
\hline
\end{tabular}
\caption{Effects of session-level isolation levels and \textit{readpast}}
\end{table}
CHAPTER 1   Commands

<table>
<thead>
<tr>
<th>Session isolation level</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, repeatable read</td>
<td>Rows or pages with incompatible locks are skipped; shared locks are held on all rows or pages that are read until the end of the statement or transaction; holds locks on all pages read by the statement until the transaction completes.</td>
</tr>
<tr>
<td>3, serializable</td>
<td>readpast is ignored, and the command executes at level 3. The command blocks on any rows or pages with incompatible locks.</td>
</tr>
</tbody>
</table>

- select commands that specify readpast fail with an error message if they also include any of the following:
  - An at isolation clause, specifying 0 or read uncommitted
  - An at isolation clause, specifying 3 or serializable
  - The holdlock keyword on the same table
- If at isolation 2 or at isolation repeatable read is specified in a select query that specifies readpast, shared locks are held on the readpast tables until the statement or transaction completes.
- If a select command with the readpast option encounters a text column that has an incompatible lock on it, readpast locking retrieves the row, but returns the text column with a value of null. No distinction is made, in this case, between a text column containing a null value and a null value returned because the column is locked.

Expanded select * syntax

When the source text of a stored procedure or trigger is stored in the system table syscomments, a query using select * is stored in syscomments expanding the column list referenced in the select *.

For example, a select * from a table containing the columns col1 and col2 is stored as:

```
select <table>.col1, <table>.col2 from <table>
```

In Adaptive Server version 12.5.4, the expanding of the column-list has been enhanced to check whether identifiers (table-names, column-names and so on) comply with the rules for identifiers.

For example, if a table includes the columns col1 and 2col, the second column-name starts with a number, which can only be included by using brackets in the create table statement.

When performing a select * in a stored procedure or trigger from this table, the text in syscomments looks similar to:
select <table>.col1, <table>[2col] from <table>

For all identifiers used in the text that expands a select *, brackets are added when the identifier does not comply with the rules for identifiers.

You must add brackets around identifiers to make sure Adaptive Server can use the SQL-text while performing an upgrade to a more recent release.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

The following are Transact-SQL extensions:

- select into to create a new table
- lock clauses
- compute clauses
- Global and local variables
- index clause, prefetch, parallel and lru | mru
- holdlock, noholdlock, and shared keywords
- “column_heading = column_name”
- Qualified table and column names
- select in a for browse clause
- The use, within the select list, of columns that are not in the group by list and have no aggregate functions
- at isolation repeatable read | 2 option

Permissions

select permission defaults to the owner of the table or view, who can transfer it to other users.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>select</td>
<td>select from a table</td>
<td>• Roles – Current active roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Keywords or options – select, select into, or readtext</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Previous value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Current value – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other information – NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Proxy information – Original login name, if set proxy in effect</td>
</tr>
<tr>
<td>Event</td>
<td>Audit option</td>
<td>Command or access audited</td>
<td>Information in extrainfo</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
</tbody>
</table>
| 63    | select       | select from a view        | • Roles – Current active roles  
          |               |                           | • Keywords or options – select, select into, or readtext  
          |               |                           | • Previous value – NULL  
          |               |                           | • Current value – NULL  
          |               |                           | • Other information – NULL  
          |               |                           | • Proxy information – Original login name, if set proxy in effect |

See also

**Commands**  compute clause, create index, create trigger, delete, group by and having clauses, insert, order by clause, set, union operator, update, where clause

**Functions**  avg, count, isnull, max, min, sum

**System procedures**  sp_cachestrategy, sp_chgattribute, sp_dboption
set

Description
Sets Adaptive Server query-processing options for the duration of the user’s work session; sets some options inside a trigger or stored procedure.

Syntax
set advanced_aggregation on/off
set @variable = expression [, @variable = expression...]
set ansi_nulls {on | off}
set ansi_permissions {on | off}
set arithabortion [arith_overflow | numeric_truncation] {on | off}
set arithignore [arith_overflow] {on | off}
set bulk_array_size number
set bulk_batch_size number
set (chained, close on endtran, nocommit, noexec, parseonly, self_recursion, showplan, sort_resources) {on | off}
set char_convert {off | on [with {error | no_error}] | charset [with {error | no_error}]}
set cis_rpc_handling {on | off}
set [clientname client_name | clienthostname host_name | clientapplname application_name]
set cursor_rows number for cursor_name
set (datefirst number, dateformat format, language language)
set delayed_commit {on | off | default}
set export_options {on | off}
set fipsflagger {on | off}
set flushmessage {on | off}
set fmtonly {on | off}
set forceplan {on | off}
set identity_insert [database.[owner.][table_name] on | off]
set identity_update table_name {on | off}
set index_union on | off
set literal_autoparam on | off
set lock {wait [numsecs] | nowait}
set metrics_capture on | off
set offsets {select, from, order, compute, table, procedure, statement, param, execute} {on | off}
set option show
set opttimeoutlimit
set parallel_degree number
set plan {dump | load} [group_name] {on | off}
set plan exists check {on | off}
set plan for show
set plan optgoal {allrows_mix | allrows_dss}
set plan opttimeoutlimit number
set plan replace {on | off}
set prefetch [on|off]
set proc_output_params {on | off}
set proc_return_status {on | off}
set process_limit_action {abort | quiet | warning}
set proxy login_name
set quoted_identifier {on | off}
set repartition_degree number
set resource_granularity number
set role {"sa_role" | "sso_role" | "oper_role" | role_name [with passwd "password"]} {on | off}
set {rowcount number, textsize number}
set scan_parallel_degree number
set session authorization login_name
set switch [serverwide] {on | off} trace_flag [, trace_flag,] [with option [, option]
set show_sqltext {on | off}
set statistics {io, subquerycache, time, plancost} {on | off}
set statistics simulate {on | off}
set strict_dtm_enforcement {on | off}
set string_rtruncation {on | off}
set textsize {number}
set tracefile [filename] [off] [for spid]
set transaction isolation level { [read uncommitted | 0] |
[read committed | 1] |
[repeatable read | 2] |
[serializable | 3]}
set transactional_rpc {on | off}
Parameters

set advanced_aggregation
enables and disables advanced aggregation at the session level.

set @variable = expression
allows multiple variable assignments in one statement. The
set @variable = expression command is an identical—and an alternative—command to select @variable = expression in Transact-SQL.

• expression includes constant, function, any combination of constants, and functions connected by arithmetic or bitwise operators, or a subquery.

set ansi_null {on | off}
impacts on both aggregate and comparison behaviors. See “Aggregate behavior” on page 575 for more information on aggregate and comparison behaviors.

set ansi_permissions {on | off}
determines whether ANSI SQL permission requirements for delete and update statements are checked. The default is off. Table 1-38 summarizes permission requirements:

<table>
<thead>
<tr>
<th>Command</th>
<th>Off</th>
<th>On</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>• update permission on columns where values are being set</td>
<td>• update permission on columns where values are being set&lt;br&gt;• select permission on all columns appearing in where clause&lt;br&gt;• select permission on all columns on right side of set clause</td>
</tr>
<tr>
<td>delete</td>
<td>• delete permission on table</td>
<td>• delete permission on table&lt;br&gt;• select permission on all columns appearing in where clause</td>
</tr>
</tbody>
</table>
set arithabort [arith_overflow | numeric_truncation] {on | off}
determines how Adaptive Server behaves when an arithmetic error occurs. The two arithabort options, arithabort arith_overflow and arithabort numeric_truncation, handle different types of arithmetic errors. You can set each option independently or set both options with a single set arithabort on or set arithabort off statement.

- **arithabort arith_overflow** – specifies the Adaptive Server behavior following a divide-by-zero error, range overflow during an explicit or implicit datatype conversion, or a domain error. This type of error is serious. The default setting, arithabort arith_overflow on, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, arithabort arith_overflow on does not roll back earlier commands in the batch; however, Adaptive Server does not execute any statements in the batch that follow the error-generating statement.

  Setting arith_overflow to on refers to the execution time, not to the level of normalization to which Adaptive Server is set.

  If you set arithabort arith_overflow off, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.

- **arithabort numeric_truncation** – specifies the Adaptive Server behavior following a loss of scale by an exact numeric type during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, arithabort numeric_truncation on, aborts the statement that causes the error, but Adaptive Server continues to process other statements in the transaction or batch. If you set arithabort numeric_truncation off, Adaptive Server truncates the query results and continues processing.

set arithignore [arith_overflow] {on | off}
determines whether Adaptive Server displays a message after a divide-by-zero error or a loss of precision. By default, the arithignore option is set to off. This causes Adaptive Server to display a warning message after any query that results in numeric overflow. To have Adaptive Server ignore overflow errors, use set arithignore on. You can omit the optional arith_overflow keyword without any effect.
set bulk array size number
establishes the number of rows that are buffered in local server memory before being transferred using the bulk copy interface.

Use this option only with Component Integration Services for transferring rows to a remote server using select into.

View your current setting using the @@bulkarraysize global variable.

number – indicates the number of rows to buffer. If the rows being transferred contain text, unitext, image or java ADTs, then the bulk copy interface ignores the current setting for array size and uses a value of 1. Also, the array size actually used will never exceed the value of @@bulkbatchsize. If @@bulkbatchsize is smaller than array size, then the smaller value is used.

The initial value of the array size is inherited by new connections from the current setting of the configuration property cis bulk insert array size, which defaults to 50. Setting this value to 0 will reset the value to the default.

set bulk batch size number
establishes the number of rows transferred to a remote server via select into proxy_table when the bulk interface is used. The bulk interface is available to all Adaptive Servers, as well as DirectConnect for Oracle version 12.5.1.

Use this option only with Component Integration Services for transferring rows to a remote server using select into.

View your current setting using the @@bulkbatchsize global variable.

The bulk interface allows a commit after a specified number of rows. This allows the remote server to free any log space being consumed by the bulk transfer operation, and enables the transfer of large data sets from one server to another without filling the transaction log.

The initial value of the batch size is inherited by new connections from the current setting of the configuration property cis bulk insert batch size, which by default is 0. A value of 0 indicates that no rows should be committed until after the last row is transferred.
set {chained, close on endtran, nocount, noexec, parseonly, self_recursion, showplan, sort_resources} {on | off}

- chained – begins a transaction just before the first data retrieval or data modification statement at the beginning of a session and after a transaction ends. In chained mode, Adaptive Server implicitly executes a begin transaction command before the following statements: delete, fetch, insert, lock table, open, select, and update. You cannot execute set chained within a transaction.

- close on endtran – causes Adaptive Server to close all cursors opened within a transaction at the end of that transaction. A transaction ends by the use of either the commit or rollback statement. However, only cursors declared within the scope that sets this option (stored procedure, trigger, and so on) are affected. For more information about cursor scopes, see the Transact-SQL User’s Guide.

For more information about the evaluated configuration, see the System Administration Guide.

- nocount – controls the display of rows affected by a statement. set nocount on disables the display of rows; set nocount off reenables the count of rows.

- noexec – compiles each query but does not execute it. noexec is often used with showplan. After you set noexec on, no subsequent commands are executed (including other set commands) until you set noexec off.

- parseonly – checks the syntax of each query and returns any error messages without compiling or executing the query. Do not use parseonly inside a stored procedure or trigger.

- self_recursion – determines whether Adaptive Server allows triggers to cause themselves to fire again (this is called self recursion). By default, Adaptive Server does not allow self recursion in triggers. You can turn this option on only for the duration of a current client session; its effect is limited by the scope of the trigger that sets it. For example, if the trigger that sets self_recursion on returns or causes another trigger to fire, this option reverts to off. This option works only within a trigger and has no effect on user sessions.

- showplan – generates a description of the processing plan for the query. The results of showplan are of use in performance diagnostics. showplan does not print results when it is used inside a stored procedure or trigger. For parallel queries, showplan output also includes the adjusted query plan at runtime, if applicable. For more information, see the Performance and Tuning Guide.
• `sort_resources` – generates a description of the sorting plan for a `create index` statement. The results of `sort_resources` are of use in determining whether a sort operation is done serially or in parallel. When `sort_resources` is on, Adaptive Server prints the sorting plan but does not execute the `create index` statement. For more information, see Chapter 24, “Parallel Sorting,” in the *Performance and Tuning Guide*.

The `set char_convert {off | on [with {error | no_error}] | charset [with {error | no_error}]]` command enables or disables character set conversion between Adaptive Server and a client. If the client is using Open Client DB-Library release 4.6 or later, and the client and server use different character sets, conversion is turned on during the login process and is set to a default based on the character set the client is using. You can also use `set char_convert charset` to start conversion between the server character set and a different client character set.

The `charset` can be either the character set’s ID or a name from `syscharsets` with a type value of less than 2000.

- `set char_convert off` turns conversion off so that characters are sent and received unchanged. `set char_convert on` turns conversion on if it is turned off. If character set conversion was not turned on during the login process or by the `set char_convert` command, `set char_convert on` generates an error message.

If you request character set conversion with `set char_convert charset`, and Adaptive Server cannot perform the requested conversion, the conversion state remains the same as it was before the request. For example, if conversion is set to `off` prior to the `set char_convert charset` command, conversion remains turned off if the request fails.

When the `with no_error` option is included, Adaptive Server does not notify an application when characters from Adaptive Server cannot be converted to the client’s character set. Error reporting is initially turned on when a client connects with Adaptive Server: if you do not want error reporting, you must turn it off for each session with `set char_convert {on | charset} with no_error`. To turn error reporting back on within a session, use `set char_convert {on | charset} with error`.

Whether or not error reporting is turned on, the bytes that cannot be converted are replaced with ASCII question marks (?).

See the *System Administration Guide* for a more complete discussion of error handling in character set conversion.
set cis_rpc_handling {on | off}

determines whether Component Integration Services handles outbound remote procedure call (RPC) requests by default.

set [clientname client_name | clienthostname host_name | clientapplname application_name]
assigns names to the client.

- clientname client_name – assigns a client an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same client name. After you assign a new name to a user, they appear in the sysprocesses table under the new name.

  client_name is the new name you assign to the user.

- clienthostname host_name – assigns a host an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same host name. After you assign a new name to a host, it appears in the sysprocesses table under the new name.

  host_name is the new name you assign to the host.

- clientapplname application_name – assigns an application an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same application name. After you assign a new name to an application, it appears in the sysprocesses table under the new name.

  application_name is the new name you assign to the application.

set cursor rows number for cursor_name

causes Adaptive Server to return the number of rows for each cursor fetch request from a client application. The number can be a numeric literal with no decimal point or a local variable of type integer. If the number is less than or equal to zero, the value is set to 1. You can set the cursor rows option for a cursor, whether it is open or closed. However, this option does not affect a fetch request containing an into clause. cursor_name specifies the cursor for which to set the number of rows returned.
set {datefirst number, dateformat format, language language}

specifies the following settings:

- datefirst number – uses numeric settings to specify the first day of the week. The us_english language default is Sunday. To set the first day of the week, use the following:

<table>
<thead>
<tr>
<th>To set the first day of the week as</th>
<th>Use this setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>1</td>
</tr>
<tr>
<td>Tuesday</td>
<td>2</td>
</tr>
<tr>
<td>Wednesday</td>
<td>3</td>
</tr>
<tr>
<td>Thursday</td>
<td>4</td>
</tr>
<tr>
<td>Friday</td>
<td>5</td>
</tr>
<tr>
<td>Saturday</td>
<td>6</td>
</tr>
<tr>
<td>Sunday (us_english language default)</td>
<td>7</td>
</tr>
</tbody>
</table>

  **Note** Regardless of which day you set as the first day of the week, the value of that first day becomes 1. This value is not the same as the numeric setting you use in set datefirst n. For example, if you set Sunday as your first day of the week, its value is 1. If you set Monday as your first day of the week, Monday’s value becomes 1. If you set Wednesday as your first day of the week, Wednesday’s value becomes 1, and so on.

- dateformat format – sets the order of the date parts month/day/year for entering datetime, smalldatetime, date or time data. Valid arguments are mdy, dmy, ymd, ydm, myd, and dym. The us_english language default is mdy.

- language language – is the official name of the language that displays system messages. The language must be installed on Adaptive Server. The default is us_english.

set delayed_commit {on | off | default}

determines when log records are written to disk. With the delayed_commit parameter set to true, the log records are asynchronously written to the disk and control is returned to the client without waiting for the IO to complete.

The session-level setting overrides any existing the database-level setting. Change delayed_commit to its default to revert back to the database-level setting.

**Note** Use delayed_commit only after careful consideration of your application.
set export_options [on | off]
Adaptive Server’s default behavior is to reset any set parameter changes that are set by a trigger or system procedure after they finish running. Enabling set export_options allows you to retain the session settings that are set by a system procedure or trigger for the duration of the session.

For example, this enables set export_options:

```plaintext
set export_options on
```

This disables set export_options and returns Adaptive Server to the default behavior:

```plaintext
set export_options off
```

set fipsflagger {on | off}
determines whether Adaptive Server displays a warning message when Transact-SQL extensions to entry-level ANSI SQL are used. By default, Adaptive Server does not tell you when you use nonstandard SQL. This option does not disable SQL extensions. Processing completes when you issue the non-ANSI SQL command.

set flushmessage {on | off}
determines when Adaptive Server returns messages to the user. By default, messages are stored in a buffer until the query that generated them is completed or the buffer is filled to capacity. Use set flushmessage on to return messages to the user immediately, as they are generated.

set fmtonly {on | off}
captures plans in stored procedures without actually executing them.

set forceplan {on | off}
causes the query optimizer to use the order of the tables in the from clause of a query as the join order for the query plan. forceplan is generally used when the optimizer fails to choose a good plan. Forcing an incorrect plan can have severely bad effects on I/O and performance. For more information, see the Performance and Tuning Guide.

**Note** The query optimizer ignores attempts to force illegal join orders with outer joins, such as in the following:

```plaintext
1> set forceplan on
2> select * from table1, table2
   where table2.id *= table1.id
```
set identity_insert [database.[owner.]]table_name {on | off}

determines whether explicit inserts into a table’s IDENTITY column are
allowed. (Updates to an IDENTITY column are never allowed.) This option
can be used only with base tables. It cannot be used with views or set within
a trigger.

After setting identity_insert on for the table, the table owner, Database
Owner, or System Administrator can manually insert any legal value greater
than 5. For example, inserting a value of 55 would create a large gap in
IDENTITY column values:

```
insert stores_cal
    (syb_identity, stor_id, stor_name)
values (55, "5025", "Good Reads")
```

```
select syb_identity from stores_cal

  ------
  1
  5
  55
```
If `identity_insert` is then set to `off`, Adaptive Server assigns an IDENTITY column value of 55 + 1, or 56, for the next insertion. If the transaction that contains the `insert` statement is rolled back, Adaptive Server discards the value 56 and uses a value of 57 for the next insertion.

Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the inserted value; you can insert any positive integer.

The table owner, Database Owner, or System Administrator can use the `set identity_insert table_name on` command on a table with an IDENTITY column to enable the manual insertion of a value into an IDENTITY column. However, only the following users can actually insert a value into an IDENTITY column, when `identity_insert` is on:

- Table owner
- Database Owner:
  - If granted explicit insert permission on the column by the table owner
  - Impersonating the table owner by using the `setuser` command

Setting `identity_insert table_name off` restores the default behavior by prohibiting explicit inserts to IDENTITY columns. At any time, you can use `set identity_insert table_name on` for a single database table within a session.

```
set identity_update table_name {on | off}
```

With `set identity_update on`, you can explicitly update the value of the IDENTITY column on a table. `identity_update` changes the identity column value for the qualified rows. When `identity_update` is enabled, you can update the identity value to any value greater than 0. However, if the input value is greater than the `identity burn max` value, a new set of ID values is allocated, and the `identity burn max` value on the OAM page is updated accordingly. If `update` is included in a transaction, the new `identity burn max` value cannot be rolled back. You can use `syb_identity` to point to the identity column for update. For example:

```
update table_name set syb_identity = value where clause
```

Adaptive Server does not check for duplicates entries or verify that entries are unique. You can update an existing value to any positive integer within the range allowed by the column's declared precision. You can check for duplicate entries by creating a unique index on the identity column.
set index_union on | off
when enabled, sets limits the scan of a table with an or clause.

Index unions (also known as an or strategy) are used for queries that contain or clauses. For example:

select * from titleauthor where au_id = "409-56-7008" or title_id = "PC8888"

If index_union is:

- Enabled – this example uses an index on au_id to find the row IDs (RIDs) of all titleauthor tuples with au_id = "409-56-7008", and uses an index on title_id to find the RIDs of all titleauthor tuples with title_id = "PC8888". Adaptive Server then performs a union on all RIDs to eliminate duplicates. The resulting RIDs are joined with a RidJoin to access the data tuples.

- Disabled – Adaptive Server does not use an index union strategy in a query to limit the table scan. Instead, it uses other access paths on the table (in the example above, it would use a table scan for table titleauthor), and applies the or clause as a filter in the scan operator.

set literal_autoparam on | off
enables and disables literal parameterization at the session level.

set lock {wait [numsecs] | nowait}
specifies the settings for a lock.

- wait – specifies the length of time that a command waits to acquire locks before aborting and returning an error.

- numsecs – specifies the number of seconds a command is to wait to acquire a lock. Valid values are from 0 to 2147483647, the maximum value for an integer.

- lock nowait – specifies that if a command cannot acquire a lock immediately, it returns an error and fails. set lock nowait is equivalent to set lock wait 0.

set metrics_capture {on | off}
enables the capture of QP metrics at the session level, set the capture to “on”. Query processing (QP) metrics identify and compare empirical metric values in query execution. When a query is executed, it is associated with a set of defined metrics that are the basis for comparison in QP metrics.
set offsets \{select, from, order, compute, table, procedure, statement, param, execute\} \{on \mid off\}

returns the position of specified keywords (with relation to the beginning of the query) in Transact-SQL statements. The keyword list is a comma-separated list that can include any of the following Transact-SQL constructs: select, from, order, compute, table, procedure, statement, param, and execute. Adaptive Server returns offsets if there are no errors.

This option is used in Open Client DB-Library only.
set option show_option {normal | brief | long | on | off} generates diagnostics output in text format.

The valid values for show_option are:

- show – shows the basic syntax common to all modules
- show_lop – shows the logical operators (scans, joins, etc.) used
- show_managers – shows data structure managers used during optimization.
- show_log_props – shows the logical properties (row count, selectivity, etc.) evaluated.
- show_parallel – shows details of parallel query optimization
- show_histograms – shows the processing of histograms associated with SARG/Join columns
- show_abstract_plan – shows the details of an abstract plan
- show_search_engine – shows the details of the join ordering algorithm
- show_counters – shows the optimization counters
- show_best_plan – shows the details of the best query plan selected by the optimizer
- show_pio_costing – shows estimates of physical input/output (reads/writes from/to the disk)
- show_lio_costing – shows estimates of logical input/output (reads/writes from/to memory)
- show_elimination – shows partition elimination
- show_missing_stats – shows details of useful statistics missing from SARG/Join columns

See Chapter 4, “Displaying Query Optimization Strategies And Estimates,” in Query Optimizer for more information.

set opttimeoutlimit
sets the timeout limit for the optimizer. The valid range of values for opttimeoutlimit 0 to 4000 ms, with 0 indicating no optimization limit.
set parallel_degree number

specifies an upper limit for the number of worker processes used in the parallel execution of a query. This number must be less than or equal to the number of worker processes per query, as set by the max parallel degree configuration parameter. The @@parallel_degree global variable stores the current setting.

set plan {dump | load} [group_name] {on | off}

introduces an abstract plan command.

- dump – enables or disables capturing abstract plans for the current connection. If a group_name is not specified, the plans are stored in the default group, ap_stdout.

- load – enables or disables loading abstract plans for the current connection. If a group_name is not specified, the plans are loaded from the default group, ap_stdin.

- group_name – is the name of the abstract plan group to use for loading or storing plans.

For more information, see Chapter 30, “Creating and Using Abstract Plans,” in the Performance and Tuning Guide.

set plan exists check {on | off}

when used with set plan load, stores hash keys for up to 20 queries from an abstract plan group in a per-user cache.
Adaptive Server Enterprise

set plan for *show*
generates an XML document for the diagnostic output. The valid values for *show* are:

- *show_exec_xml* – gets the compiled plan output in XML, showing each of the query plan operators.
- *show_execio_xml* – gets the plan output along with estimated and actual IOs. This also includes the query text.
- *show_opt_xml* – gets optimizer diagnostic output, which shows all of the different components like logical operators, output from the managers, some of the search engine diagnostics, and the best query plan.
- *show_lop_xml* – gets the output logical operator tree in XML.
- *show_managers_xml* – shows the output of the different component managers during the preparation phase of the query optimizer.
- *show_log_props_xml* – shows the logical properties for a given equivalence class (one or more group of relations in the query).
- *show_parallel_xml* – shows the diagnostics related to the optimizer while generating parallel query plans.
- *show_histograms_xml* – shows diagnostics related to histograms and the merging of histograms.
- *show_abstract_plan_xml* – shows the AP generation/application.
- *show_search_engine_xml* – shows the search engine related diagnostics.
- *show_counters_xml* – shows plan object construction/destruction counters.
- *show_best_plan_xml* – shows the best plan in XML.
- *show_pio_costing_xml* – shows actual PIO costing in XML.
- *show_lio_costing_xml* – shows actual LIO costing in XML.
- *show_elimination_xml* – shows partition elimination in XML.
- *client* – when specified, output goes to the client.
- *message* – when specified, output goes to an internal message buffer.

See Chapter 4, “Displaying Query Optimization Strategies And Estimates,” in *Query Processor* for more information.
set plan optgoal {allrows_mix | allrows_dss}
sets the optimization goal.

- allrows_mix – is the default optimization goal, and the most useful goal in a mixed-query environment. It balances the needs of OLTP and DSS query environments.
- allrows_dss – is the most useful goal for operational DSS queries of medium to high complexity. Currently, this goal is provided on an experimental basis.

See Chapter 1, “Understanding Query Processing in Adaptive Server” in *Query Processor* for more information about optimization plans.

set plan opttimeoutlimit number
sets the timeout at the session level, where $n$ is any integer between 0 and 1000. See Chapter 1, “Understanding Query Processing in Adaptive Server” in *Query Processor* for more information about optimization plans.

set plan replace {on | off}
enables or disables replacing existing abstract plans during plan capture mode. By default, plan replacement is off.

set prefetch {on | off}
enables or disables large I/Os to the data cache.

set proc_output_params {on | off}
controls sending of output parameters that a stored procedure generates back to the client. set proc_output_params off suppresses sending the output parameters back to the client. The default for this parameter is on.

set proc_return_status {on | off}
controls sending of a return status TDS token back to the client. set proc_return_status off suppresses sending the return status token to the client, and *isql* client does not display the (return status = 0) message. The default for this parameter is on.

**Warning!** If the client application that executes a procedure relies on the success or failure of the procedure based on the return status, then do not use the set proc_return_status off option.
set process_limit_action {abort | quiet | warning}
specifies whether Adaptive Server executes parallel queries when an insufficient number of worker processes is available. Under these circumstances, if:

- process_limit_action is set to quiet, Adaptive Server silently adjusts the plan to use a degree of parallelism that does not exceed the number of available processes.

- process_limit_action is set to warning when an insufficient number of worker processes are available, Adaptive Server issues a warning message when adjusting the plan

- process_limit_action is set to abort, Adaptive Server aborts the query and issues an explanatory message an insufficient number of worker processes are available.

set proxy login_name
allows you to assume the permissions, login name, and suid (server user ID) of login_name. For login_name, specify a valid login from master..syslogins, enclosed in quotation marks. To revert to your original login name and suid, use set proxy with your original login_name.

Note Without explicit permission, neither the “sa_role” nor the “sso_role” can issue the set proxy login_name command. To use set proxy login_name, any user, including the System Security Officer, must have permission explicitly granted by the System Security Officer.

See “Using proxies” on page 578 for more information.
set quoted_identifier {on | off}
determines whether Adaptive Server recognizes delimited identifiers within
double quotation marks. By default, quoted_identifier is off and all identifiers
must either:

- Conform to the rules for valid identifiers.
- Be enclosed in brackets.

If you use set quoted_identifier on, double quotation marks behave the way
brackets do, and you can use table, view, and column names that begin with
a nonalphabetic character, including characters that would not otherwise be
allowed, or are reserved words, by enclosing the identifiers within double
quotation marks. Delimited identifiers cannot exceed 28 bytes, may not be
recognized by all front-end products, and may produce unexpected results
when used as parameters to system procedures.

When quoted_identifier is on, all character strings enclosed within double
quotes are treated as identifiers. Use single quotes around character or
binary strings.

set repartition_degree number
is the maximum degree to which any intermediate data stream is
re-partitioned for semantic purposes. See Chapter 2, “Parallel Query
Processing” in Query Processor for more information about setting the
value of max repartition degree for a session.

set resource_granularity number
overrides the global value max resource granularity and sets it to a session
specific value, which influences whether Adaptive Server uses
memory-intensive operation or not. See Chapter 2, “Parallel Query
Processing” in Query Processor for more information.
set role \{"sa_role" | "sso_role" | "oper_role" | role_name \[with passwd "password"]\} 
\{on | off\}

turns the specified role on or off during the current session. When you log in, all system roles that have been granted to you are turned on.

Use `set role role_name off` to turn a role off, and `set role role_name on` to turn it back on again, as needed.

- "sa_role" | "sso_role" | "oper_role" are system roles. If you are not a user in the current database, and if there is no “guest” user, you cannot set `sa_role` off, because there is no server user ID for you to assume.

- `role_name` – is the name of any user-defined role created by the System Security Officer. User-defined roles are not turned on by default. To set user-defined roles to activate at login, the user or the System Security Officer must use `set role on`.

- `with passwd` – specifies the password to activate the role. If a user-defined role has an attached password, you must specify the password to activate the role.

`set {rowcount number, textsize number}` causes Adaptive Server to stop processing the query (select, insert, update, or delete) after the specified number of rows are affected. The `number` can be a numeric literal with no decimal point or a local variable of type integer. To turn this option off, use:

```sql
set rowcount 0
```

You can determine the current value for `set rowcount` with the \(@@setrowcount\) global variable. For example:

```sql
select @@setrowcount
```

37

`set scan_parallel_degree number` specifies the maximum session-specific degree of parallelism for hash-based scans (parallel index scans and parallel table scans on nonpartitioned tables). This number must be less than or equal to the current value of the `max scan parallel degree` configuration parameter. The \(@@scan_parallel_degree\) global variable stores the current setting.

`set session authorization login_name` is identical to `set proxy`, with this exception: `set session authorization` follows the SQL standard, while `set proxy` is a Transact-SQL extension.
set show_sqltxt {on | off}
allows you to print the SQL text for ad-hoc queries, stored procedures, cursors, and dynamic prepared statements.

You do not need to enable the `set show_sqltxt` before you execute the query (as you do with commands like `set showplan on`) to collect diagnostic information for a SQL session. Instead you can enable it while the commands are running to help determine which query is performing poorly and diagnose their problems.

Before you enable `show_sqltext`, you must first enable `dbcc traceon` to display the output to standard out:

```
dbcc traceon(3604)
```

The syntax for `set show_sqltext` is:

```
set show_sqltext {on | off}
```

For example, this enables `show_sqltext`:

```
set show_sqltext on
```

Once `set show_sqltext` is enabled, Adaptive Server prints all SQL text to standard out for each command or system procedure you enter. Depending on the command or system procedure you run, this output can be extensive.

To disable `show_sqltext`, enter:

```
set show_sqltext off
```
set statistics {io, subquerycache, time, plancost, simulate} {on | off}

- **io** – displays statistics for each table referenced in the statement:
  - The number of times the table is accessed (scan count)
  - The number of logical reads (pages accessed in memory)
  - The number of physical reads (database device accesses)

For each command, statistics **io** displays the number of buffers written. If Adaptive Server has been configured to enforce resource limits, statistics **io** also displays the total I/O cost.

- **subquerycache** – displays the number of cache hits, misses, and the number of rows in the subquery cache for each subquery.

- **time** – displays the amount of time Adaptive Server used to parse and compile for each command. For each step of the command, statistics **time** displays the amount of time Adaptive Server used to execute the command. Times are given in milliseconds and timeticks, the exact value of which is machine-dependent.

- **plancost** – displays the query statistics in a tree format.

---

**Note** When you enable `set statistics plancost`, Adaptive Server abbreviates the names for lio, pio, and row to l, p, and r, respectively.

- **simulate** – specifies that the optimizer should use simulated statistics to optimize the query.

For more information, see Chapter 34, “Using the set statistics Commands” in the Performance and Tuning Guide.

set strict_dtm_enforcement {on | off}

determines whether the server propagates transactions to servers that do not support Adaptive Server transaction coordination services. The default value is inherited from the value of the `strict dtm enforcement` configuration parameter.
set string_rtruncation {on | off}

determines whether Adaptive Server raises a SQLSTATE exception when an insert or update command truncates a char, unichar, varchar or univarchar string. If the truncated characters consist only of spaces, no exception is raised. The default setting, off, does not raise the SQLSTATE exception, and the character string is silently truncated.

set switch [serverwide] {on | off} trace_flag,[trace_flag] [,.with option [, option]> allows you to set trace flags and switch names locally and server-wide.

- serverwide – optional and will set a switch serverwide ON or OFF. The default is session-specific.
- on – trace flags are switched on.
- off – trace flags are switched off.
- trace_flag – a sequence of numbers (the old traceflag numbers) and/or switch names.
- option – an optional sequence of switch options. Valid values are:
  - override – this option is necessary to enable a non-documented switch names or trace flags
  - no_info – this option is used to surpress any informational warnings

set textsize [number]
specifies the maximum size in bytes of text, unitext, or image type data that is returned with a select statement. The @@textsize global variable stores the current setting.

The default setting is 32K in isql. Some client software sets other default values. To reset textsize to the default size (32K), use:

set textsize 0

set tracefile [filename] [off] [for spid]

once enabled, saves all SQL text for the current session to the specified file, each SQL text batch appending to the previous batch.

The syntax to enable tracing is:

set tracefile file_name [off] [for spid]

The syntax to disable tracing is:
set tracefile off [for spid]

Where:

- **file_name** – is the full path to the file in which you are saving the SQL text. If you do not specify a directory path, Adaptive Server creates the file in $SYBASE.

**Note** If **file_name** contains special characters (";", "/", and so on) other than numbers and letters, you must include **file_name** in quotes. For example, this **file_name** must be in quotes because of the "/" for the directory structure:

```sql
set tracefile '/tmp/mytracefile.txt' for 25
```

If **file_name** does not contain special characters and you want to save it to $SYBASE, it does not require quotes. For example, this **file_name** does not need to be in quotes:

```sql
set tracefile mytracefile.txt
```

- **off** – disables the tracing for this session or spid.

- **spid** – server process ID whose SQL text you want saved to a trace file. Only the users with the SA or SSO role can enable tracing for other spids. You cannot save the SQL text for system tasks (such as the housekeeper or the port manager).

**Note** After you use `set tracefile` for a particular session, the diagnostic output of all successive set commands or DBCC traces are then redirected to a tracefile.

Make sure to switch off all the diagnostic commands you turned on before issuing `set tracefile off`, or else the output that should go to the tracefile will instead go to the client.
set transaction isolation level {
  read uncommitted | 0
  | read committed | 1
  | repeatable read | 2
  | serializable | 3
}
sets the transaction isolation level for your session. After you set this option, any current or future transactions operate at that isolation level.

- **read uncommitted | 0** – scans at isolation level 0 do not acquire any locks. Therefore, the result set of a level 0 scan may change while the scan is in progress. If the scan position is lost due to changes in the underlying table, a unique index is required to restart the scan. In the absence of a unique index, the scan may be aborted.

By default, a unique index is required for a level 0 scan on a table that does not reside in a read-only database. You can override this requirement by forcing the Adaptive Server to choose a nonunique index or a table scan, as follows:

```sql
select * from table_name (index table_name)
```
Activity on the underlying table may cause the scan to be aborted before completion.

- **read committed | 1** – by default, the Adaptive Server transaction isolation level is read committed or 1, which allows shared read locks on data.

- **repeatable read | 2** – prevents nonrepeatable reads.

- **serializable | 3** – specifies isolation level 3, Adaptive Server applies a holdlock to all select and readtext operations in a transaction, which holds the queries’ read locks until the end of that transaction. If you also set chained mode, that isolation level remains in effect for any data retrieval or modification statement that implicitly begins a transaction.

set transactional_rpc {on | off}
controls the handling of remote procedure calls. If this option is set to on, when a transaction is pending, the RPC is coordinated by Adaptive Server. If this option is set to off, the remote procedure call is handled by the Adaptive Server site handler. The default value is inherited from the value of the enable xact coordination configuration parameter.

**Examples**

**Example 1** Tells Adaptive Server to evaluate NULL-valued operands of equality (=) and inequality (!=) comparisons and aggregate functions in compliance with the entry level ANSI SQL standard:

```sql
set ansinull on
```
When you use `set ansinull on`, aggregate functions and row aggregates raise the following SQLSTATE warning when Adaptive Server finds null values in one or more columns or rows:

| Warning - null value eliminated in set function |

If the value of either the equality or the inequality operands is NULL, the comparison’s result is UNKNOWN. For example, the following query returns no rows in `ansinull` mode:

```sql
select * from titles where price = null
```

If you use `set ansinull off`, the same query returns rows in which `price` is NULL.

**Example 2** Activates character set conversion, setting it to a default based on the character set the client is using. Adaptive Server also notifies the client or application when characters cannot be converted to the client’s character set:

```sql
set char_convert on with error
```

**Example 3** Specifies that Component Integration Services handles outbound RPC requests by default:

```sql
set cis_rpc_handling on
```

**Example 4** Assigns this user:

- The client name alison
- The host name money1
- The application name webserver2

```sql
set clientname 'alison'
set clienthostname 'money1'
set clientapplname 'webserver2'
```

**Example 5** Returns five rows for each succeeding `fetch` statement requested by a client using `test_cursor`:

```sql
set cursor rows 5 for test_cursor
```

**Example 6** Tells Adaptive Server to retain the session settings that are set by a system procedure or trigger for the duration of the session:

```sql
set export_options on
```

To disable `set export_options` and return Adaptive Server to the default behavior, use:

```sql
set export_options off
```
You can export these optimization settings using `set export_options on`.

---

**Note** By default, `set export_options` are enabled for login triggers.

---

**Example 7** Tells Adaptive Server to display a warning message if you use a Transact-SQL extension:

```
set fipsflagger on
```

Then, if you use nonstandard SQL, like this:

```
use pubs2
go
```

Adaptive Server displays:

```
SQL statement on line number 1 contains Non-ANSI text.  
The error is caused due to the use of use database.
```

**Example 8** Inserts a value of 100 into the IDENTITY column of the `stores_south` table, then prohibits further explicit inserts into this column. Note the use of the `syb_identity` keyword; Adaptive Server replaces the keyword with the name of the IDENTITY column:

```
set identity_insert stores_south on
insert stores_south (syb_identity) 
values (100)
go
set identity_insert stores_south off
```

**Example 9** Enables `identity_update` and updates tables with values 1 and 10, respectively, then disables `identity_update`:

```
set identity_update t1 on
update t1 set c2 = 10 where c1 =1
select * from t1 
c1    c2
  1    10

set identity_update t1 off
```

**Example 10** Subsequent commands in the session or stored procedure return an error and fail if they cannot get requested locks immediately:

```
set lock nowait
```
**Example 11** Subsequent commands in the current session or stored procedure wait indefinitely long to acquire locks:

```
set lock wait
```

**Example 12** Subsequent commands in the session or stored procedure wait 5 seconds to acquire locks before generating an error message and failing:

```
set lock wait 5
```

**Example 13** Enables capturing abstract plans to the `dev_plans` group:

```
set plan dump dev_plans on
```

**Example 14** Enables loading of abstract plans from the `dev_plans` group for queries in the current session:

```
set plan load dev_plans on
```

**Example 15** Suppresses the output of parameter information:

```
1> create procedure sp_pout (@x int output) as select @x = @x + 1
2> go
1> set proc_output_params off
2> go
1> declare @x int
2> select @x = 1
3> exec sp_pout @x output
4> print "Value of @x returned from sproc is: %1!", @x
5> go
```

(1 row affected)
(return status = 0)

Value of @x returned from sproc is: 1

If you do not perform `set proc_output_params off`, the output after `(return status = 0)` includes the following:

```
Return parameters:

-----------
2
```

**Example 16** Suppresses the output of both parameters and the return status TDS token:

```
set proc_output_params OFF
```
set proc_return_status OFF

declare @x int
select @x = 2
exec sp_pout @x output
print "Value of @x returned from sproc is: %1!", @x

go

(1 row affected)
Value of @x returned from sproc is: 2
(1 row affected)

In addition, you can also suppress the lines reporting the number of rows affected to generate output with no extra messages using the set nocount on option before running this batch.

**Example 17** The user executing this command now operates within the server as the login “mary” and Mary’s server user ID:

```
set proxy "mary"
```

**Example 18** For each insert, update, delete, and select statement, Adaptive Server stops processing the query after it affects the first four rows. For example:

```
select title_id, price from titles

<table>
<thead>
<tr>
<th>title_id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>BU1032</td>
<td>19.99</td>
</tr>
<tr>
<td>BU1111</td>
<td>11.95</td>
</tr>
<tr>
<td>BU2075</td>
<td>2.99</td>
</tr>
<tr>
<td>BU7832</td>
<td>19.99</td>
</tr>
</tbody>
</table>
```

(4 rows affected)

```
set rowcount 4
```

**Example 19** Tells Adaptive Server to treat any character string enclosed in double quotes as an identifier. The table name "!*&strange_table" and the column name "emp's_name" are legal identifier names while quoted_identifier is on:

```
set quoted_identifier on

go
create table "!*&strange_table"
  ("emp's_name" char (10)),
```
Example 20  Treats a character string enclosed in brackets as an identifier. The table name ![strange_table] and the column name [emp’s_name] are legal identifier names because they are enclosed in brackets, even though quoted_identifier is off:

```sql
set quoted_identifier off
go
```

```sql
create table ![strange_table]
    ([emp’s_name] char (10),
     age int)
go
```

See “Delimited identifiers” on page 576 for usage information about bracket identifiers.

Example 21  Activates the “doctor” role. This command is used by users to specify the roles they want activated:

```sql
set role doctor_role on
```

Example 22  Deactivates the user’s System Administrator role for the current session:

```sql
set role "sa_role" off
```

Example 23  Activates the “doctor” role when the user enters the password:

```sql
set role doctor_role with passwd "physician" on
```

Example 24  Deactivates the “doctor” role:

```sql
set role doctor_role off
```

Example 25  Specifies a maximum degree of parallelism of 4 for parallel index scans and parallel table scans on nonpartitioned tables:

```sql
set scan_parallel_degree 4
```

Example 26  An alternative way of stating example 5:

```sql
set session authorization "mary"
```

Example 27  For each query, returns a description of the processing plan, but does not execute it:

```sql
set showplan, noexec on
go
select * from publishers
```
Example 28 Displays the statistics for the query in a tree format:

```sql
set statistics plancost on
select * from authors
```

<table>
<thead>
<tr>
<th>au_id</th>
<th>au_lname</th>
<th>au_fname</th>
<th>phone</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>country</th>
<th>postalcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>172-32-1176</td>
<td>White</td>
<td>Johnson</td>
<td>408</td>
<td>496-7223 10932 Bigge Rd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>213-46-8915</td>
<td>Green</td>
<td>Marjorie</td>
<td>415</td>
<td>986-7020 309 63rd St. #411</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>998-72-3567</td>
<td>Ringer</td>
<td>Albert</td>
<td>801</td>
<td>826-0752 67 Seventh Av.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Lava Operator Tree*

```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emit</td>
</tr>
<tr>
<td>(VA = 1)</td>
</tr>
<tr>
<td>23 rows est: 23</td>
</tr>
<tr>
<td>cpu: 0</td>
</tr>
<tr>
<td>TableScan</td>
</tr>
<tr>
<td>authors</td>
</tr>
<tr>
<td>(VA = 0)</td>
</tr>
<tr>
<td>23 rows est: 23</td>
</tr>
<tr>
<td>lio: 1 est: 2</td>
</tr>
<tr>
<td>pio: 0 est: 2</td>
</tr>
</tbody>
</table>

(23 rows affected)

Example 29 Causes Adaptive Server to generate an exception when truncating a char, unichar, or nchar string:

```sql
set string_rtruncation on
```

If an insert or update statement would truncate a string, Adaptive Server displays:

string data, right truncation
Example 30  Sets the limit on text, unitext, or image data returned with a select statement to 100 bytes:

```
set textsize 100
```

Example 31  Sets the serverwide switch on to set traceflags for 110, an undocumented traceflag, with no additional informational warnings:

```
set switch serverwide on 110 with override, no_info
```

Example 32  Opens a trace file named \texttt{sql\_text\_file} for the the current session:

```
set tracefile '/var/sybase/REL1502/text\_dir/sql\_text\_file'
```

Subsequent outputs from \texttt{set showplan}, \texttt{set statistics io}, and \texttt{dbcc traceon(100)} are saved in \texttt{sql\_text\_file}.

Example 33  Does not specify a directory path, so the trace file is saved in \texttt{$SYBASE/sql\_text\_file}:

```
set tracefile 'sql\_text\_file' for 11
```

Any SQL run on spid 11 is saved to this tracefile.

Example 34  Saves the SQL text for spid 86:

```
set tracefile '/var/sybase/REL1502/text\_dir/sql\_text\_file' for 86
```

Example 35  Specifies that when a transaction is pending, the RPC is handled by the Component Integration Services access methods rather than by the Adaptive Server site handler:

```
set transactional_rpc on
```

Example 36  All subsequent queries in the session run at the repeatable reads transaction isolation level:

```
set transaction isolation level 2
```

Example 37  Implements read-locks with each \texttt{select} statement in a transaction for the duration of that transaction:

```
set transaction isolation level 3
```

Usage  \texttt{fipsflagger}, \texttt{string\_rtruncation}, \texttt{ansinull}, \texttt{ansi\_permissions}, \texttt{arithabort}, and \texttt{arithignore} affect aspects of Adaptive Server error handling and compliance to SQL standards.

- You can use the \texttt{cis\_rpc\_handling} and \texttt{transactional\_rpc} options only when Component Integration Services is enabled.
- The async log service option and \texttt{delayed\_commit} are mutually exclusive. \texttt{delayed\_commit} will not work if async log service is set to “true.”
parallel_degree and scan_parallel_degree limit the degree of parallelism for queries, if Adaptive Server is configured for parallelism. When you use these options, you give the optimizer a hint to limit parallel queries to use fewer worker processes than allowed by the configuration parameters. Setting these parameters to 0 restores the server-wide configuration values.

If you specify a number that is greater than the numbers allowed by the configuration parameters, Adaptive Server issues a warning message and uses the value set by the configuration parameter.

If you use the set command inside a trigger or stored procedure, most set options revert to their former settings after the trigger or procedure executes.

The following options do not revert to their former settings after the procedure or trigger executes, but remain for the entire Adaptive Server session or until you explicitly reset them:

- datefirst
- dateformat
- identity_insert
- language
- quoted_identifier

If you specify more than one set option, the first syntax error causes all following options to be ignored. However, the options specified before the error are executed, and the new option values are set.

If you assign a user a client name, host name, or application name, these assignments are only active for the current session. You must reassign these the next time the user logs in. Although the new names appear in sysprocesses, they are not used for permission checks, and sp_who still shows the client connection as belonging to the original login. For more information about setting user processes, see the System Administration Guide.

All set options except showplan and char_convert take effect immediately. showplan takes effect in the following batch. Here are two examples that use set showplan on:

```sql
set showplan on
select * from publishers
go
```
Adaptive Server Enterprise

<table>
<thead>
<tr>
<th>pub_id</th>
<th>pub_name</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>New Age Books</td>
<td>Boston</td>
<td>MA</td>
</tr>
<tr>
<td>0877</td>
<td>Binnet &amp; Hardley</td>
<td>Washington</td>
<td>DC</td>
</tr>
<tr>
<td>1389</td>
<td>Algodata Infosystems</td>
<td>Berkeley</td>
<td>CA</td>
</tr>
</tbody>
</table>

(3 rows affected)

But:

set showplan on

```
SELECT * FROM publishers
```

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT

FROM TABLE
publishers
Nested iteration
Table Scan
Ascending Scan.
Positioning at start of table.

(3 rows affected)

- Adaptive Server automatically stores one or more spaces in clientname, clienthostname, and clientapplname columns. For this reason, a query using any of these three columns that includes “is null” does not return an expected result set.

- set proxy issue the following warning when they are issued while set fipsflagger option is enabled:

```
SQL statement on line number 1 contains Non-ANSI text. The error is caused due to the use of DBCC.
```

- If you use a login trigger to set current execution properties, any exportable set option that you enable or disable inside a login trigger takes affect in the current process.

- Some set options can be grouped together:
parseonly, noexec, prefetch, showplan, rowcount, and nocount control the way a query is executed. It does not make sense to set both parseonly and noexec on. The default setting for rowcount is 0 (return all rows); the default for the others is off.

The statistics options display performance statistics after each query. The default setting for the statistics options is off. For more information about noexec, prefetch, showplan and statistics, see the *Performance and Tuning Guide*.

You can update up to 1024 columns in the set clause using literals, variables, or expressions returned from a subquery.

offsets is used in DB-Library to interpret results from Adaptive Server. The default setting for this option is on.

datefirst, dateformat, and language affect date functions, date order, and message display. If used within a trigger or stored procedure, these options do not revert to their previous settings.

In the default language, us_english, datefirst is 1 (Sunday), dateformat is mdy, and messages are displayed in us_english. Some language defaults (including us_english) produce Sunday=1, Monday=2, and so on; others produce Monday=1, Tuesday=2, and so on.

set language implies that Adaptive Server should use the first weekday and date format of the language it specifies, but does not override an explicit set datefirst or set dateformat command issued earlier in the current session.

cursor rows and close on endtran affect the way Adaptive Server handles cursors. The default setting for cursor rows with all cursors is 1. The default setting for close on endtran is off.

chained and transaction isolation level allow Adaptive Server to handle transactions in a way that is compliant with the SQL standards.

**Compile-time changes for some set parameters**

In version 15.0.2 and later, Adaptive Server changes the compile-time behavior for some abstract plan set parameters when you use them to create stored procedures or run them in Transact-SQL batches.
In earlier versions of Adaptive Server, the set parameters took effect after the stored procedure was executed or recompiled. Adaptive Server 15.0.2 allows you to use optimizer set parameters at compile time to affect the optimizer in stored procedures or batches.

**Note** This changed behavior may effect the composition of the result set. Sybase recommends that you review the result set created by the 15.0.2 versions of the set parameters before using them in your production systems.

You must reset the set parameter before returning from the stored procedure or the execution of subsequent stored procedures may be affected. If you intend to propagate this change to subsequent stored procedures, use export_options parameter.

Table 1-39 shows the optimizer options that you can export when you use set export_options on.

<table>
<thead>
<tr>
<th>Option</th>
<th>Export Option</th>
<th>Export Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>optgoal</td>
<td>store_index</td>
<td>showmanagers</td>
</tr>
<tr>
<td>opttimeout</td>
<td>bushy_space_search</td>
<td>showlogprops</td>
</tr>
<tr>
<td>merge_join</td>
<td>parallel_query</td>
<td>showparallel</td>
</tr>
<tr>
<td>hash_join</td>
<td>replicated_partitioning</td>
<td>showhistograms</td>
</tr>
<tr>
<td>nl_join</td>
<td>basic_optimization</td>
<td>showabstractplan</td>
</tr>
<tr>
<td>distinct_sorted</td>
<td>index_intersection</td>
<td>showsearchengine</td>
</tr>
<tr>
<td>distinct_sorting</td>
<td>index_union</td>
<td>showcounters</td>
</tr>
<tr>
<td>distinct_hashing</td>
<td>multi_tt_store_index</td>
<td>showbestplan</td>
</tr>
<tr>
<td>group_sorted</td>
<td>opportunistic_grouping</td>
<td>showfinalplan</td>
</tr>
<tr>
<td>group_hashing</td>
<td>opportunistic_distinct</td>
<td>showcodegen</td>
</tr>
<tr>
<td>group_inserting</td>
<td>auto_query_tuning</td>
<td>showpiocosting</td>
</tr>
<tr>
<td>order_sorting</td>
<td>streaming_sort</td>
<td>showliocosting</td>
</tr>
<tr>
<td>addend_union_all</td>
<td>nary_nl_join</td>
<td>showelimination</td>
</tr>
<tr>
<td>merge_union_all</td>
<td>query_tuning_mem_limit</td>
<td>showpilcosting</td>
</tr>
<tr>
<td>merge_union_distinct</td>
<td>query_tuning_time_limit</td>
<td>shownostats</td>
</tr>
<tr>
<td>hash_union_distinct</td>
<td>showlop</td>
<td>showexecicio</td>
</tr>
</tbody>
</table>
Aggregate behavior

`ansinull` determines whether evaluation of NULL-valued operands in aggregate functions is compliant with the ANSI SQL standard. If you use `set ansi_null on`, Adaptive Server generates a warning when an aggregate function eliminates a null-valued operand from the calculation.

For example, if you perform the following query on the `titles` table with `set ansi_null off` (the default value):

```sql
select avg (total_sales) from titles
```

Adaptive Server returns:

```
-----------
6090
```

However, if you perform the same query with `set ansi_null on`, Adaptive Server returns the following:

```sql
1> use pubs2
2> go
1> select avg (total_sales) from titles
2> go

----------
6090
(1 row affected)
```

This message indicates that some entries in `total_sales` contain NULL instead of a real amount, so you do not have complete data on total sales for all books in this table. However, of the available data, the value returned is the highest.

Comparison behavior

The SQL standard requires that if either one of the two operands of an equality comparison is NULL, the result is UNKNOWN. Transact-SQL treats NULL values differently. If one of the operands is a column, parameter, or variable, and the other operand is the NULL constant or a parameter or variable whose value is NULL, the result is either TRUE or FALSE:
Sybase NULL mode – “val = NULL” is true when “val” is NULL

ANSI NULL mode – “val = NULL” is unknown when “val” is NULL

The ANSI rule for the where and on clauses return rows that are true, and rejects rows that are both false and unknown.

The ANSI rule for a check constraint rejects values that are false. For this reason, unknown or true results are not rejected.

If you:

• Enable ansinull mode – do not use the Sybase NULL comparisons
  \( \text{val} = \text{NULL} \text{ or } \text{val} \neq \text{NULL} \).

• Expect to use ANSI-null mode during insert and update – do not use the
  Sybase NULL comparisons in check constraints.

Instead, use the ANSI IS NULL or IS NOT NULL syntax to prevent from having unexpected results.

**Delimited identifiers**

When the quoted_identifier option is set to on, you do not need to use double quotes around an identifier if the syntax of the statement requires that a quoted string contain an identifier. For example:

```
set quoted_identifier on
create table "lone" (c1 int)
```

However, object_id requires a string, so you must include the table name in quotes to select the information:

```
select object_id ('lone')
```

```
-----------------------
896003192
```

You can include an embedded double quote in a quoted identifier by doubling the quote:

```
create table "embedded""quote" (c1 int)
```

However, there is no need to double the quote when the statement syntax requires the object name to be expressed as a string:

```
select object_id ('embedded"quote')
```

**Bracketed identifiers**  Adaptive Server supports an alternative to quoted identifiers that uses brackets to surround an identifier. The behavior of bracketed identifiers is identical to that of quoted identifiers, with the exception that you do not have to use set quoted_identifier on to use them.

```
set quoted_identifier on
create table "lone" (c1 int)
```

```
set quoted_identifier on
create table "embedded""quote" (c1 int)
```
When you use bracketed identifiers instead of quoted identifiers to create objects, your objectname should have at least one valid character, such as:

create table [table name]
create database [database name]

All trailing spaces are removed from the objectname, so the following are all treated identically:

[tabl<space><space>]
[tabl<space><space>]
[tabl]
[tabl<space><space><space>]
tabl

tabl

This applies to all objects that can be created using bracketed identifiers.

The following are restrictions when using delimited identifiers in Adaptive Server:

- A dot (.) cannot appear in an identifier name, however delimited
- Object names as stored procedure parameters – Adaptive Server stored procedure object names can be treated as strings, and do not need delimiters. For example, the following gives correct results if a table named table actually exists:

  exec sp_help 'dbo.table'

  However, the brackets are not stripped from the object name in the following:

  exec sp_help 'dbo.[table]'

Roles and set options
- When you log in to Adaptive Server, all system-defined roles granted to you are automatically activated. User-defined roles granted to you are not automatically activated. To automatically activate user-defined roles granted to you, use sp_modifylogin. See sp_modifylogin in Reference Manual: Procedures. Use set role role_name on or set role role_name off to turn roles on and off.

  For example, if you have been granted the System Administrator role, you assume the identity (and user ID) of Database Owner in the current database. To assume your real user ID, execute this command:

  set role "sa_role" off

  If you are not a user in the current database, and if there is no “guest” user, you cannot set sa_role off.
If the user-defined role you intend to activate has an attached password, you must specify the password to turn the role on. Thus, you would enter:

```
set role "role_name" with passwd "password" on
```

Distributed transactions, CIS, and set options

- The behavior of the cis rpc handling configuration property and the set transactional_rpc commands changed with the introduction of ASTC. In versions earlier than 12.0, enabling cis rpc handling caused all RPCs to be routed through CIS's Client-Library connection. As a result, whenever cis rpc handling was enabled, transactional_rpc behavior occurred whether or not it had been specifically set. As of Adaptive Server 12.0, this behavior has changed. If cis rpc handling is enabled and transactional_rpc is off, RPCs within a transaction are routed through the site handler. RPCs executed outside a transaction are sent via CIS's Client-Library connection.

- When Adaptive Server distributed transaction management services are enabled, you can place RPCs within transactions. These RPCs are called **transactional RPCs**. A transactional RPC is an RPC whose work can be included in the context of a current transaction. This remote unit of work can be committed or rolled back along with the work performed by the local transaction.

To use transactional RPCs, enable CIS and distributed transaction management with `sp_configure`, then issue the `set transactional_rpc` command. When `set transactional_rpc` is on and a transaction is pending, the Adaptive Server (as opposed to the Adaptive Server site handler) coordinates the RPC.

The `set transactional_rpc` command default is off. The `set cis_rpc_handling` command overrides the `set transactional_rpc` command. If you set `cis_rpc_handling` on, all outbound RPCs are handled by Component Integration Services.

- See the *Component Integration Services User's Guide* for a discussion of using `set transactional_rpc`, `set cis_rpc_handling`, and `sp_configure`.

Using proxies

**Note** Without explicit permission, neither the “sa_role” nor the “sso_role” can issue the `set proxy login_name` command. To use `set proxy login_name`, any user, including the System Security Officer, must have permission explicitly granted by the System Security Officer.
Before you can use the set proxy or set session authorization command, a System Security Officer must grant permission to execute set proxy or set session authorization from the master database.

You can switch your server user identity to any other server login and limit its use based on the target login roles by using:

```
grant set proxy to user_or_role_list
[restrict role role_list | all | system]
```

See grant on page 360 for more information.

Executing set proxy or set session authorization with the original login_name reestablishes your previous identity.

You cannot execute set proxy or set session authorization from within a transaction.

Adaptive Server permits only one level of login identity change. Therefore, after you use set proxy or set session authorization to change identity, you must return to your original identity before changing it again. For example, assume that your login name is “ralph”. To create a table as “mary”, create a view as “joe”, then return to your own login identity. Use the following statements:

```
set proxy "mary"
create table mary_sales
    (stor_id char (4),
     ord_num varchar (20),
     date datetime)
grant select on mary_sales to public
set proxy "ralph"
set proxy "joe"
create view joes_view (publisher, city,
    state)
    as select stor_id, ord_num, date
    from mary_sales
set proxy "ralph"
```
If a user issues `set proxy` to assume the permissions, login name, and `suid` of another user, Adaptive Server checks the proxy user’s access to database objects, rather than the original user’s access. Adaptive Server uses the name and password information of the user who logged in to check for automatic access to encryption keys using login credentials. Adaptive Server does not have access to the proxy user’s password. Access to keys through the login password is on behalf of the user who logs in, not on behalf of the user assumed through an alias, `set proxy`, or `setuser`. Access to copies of encryption keys that were set up for login association, but which are still encrypted by the system encryption password, is treated similarly.

Using `lock wait`  
- By default, an Adaptive Server task that cannot immediately acquire a lock waits until incompatible locks are released, then continues processing. This is equivalent to `set lock wait` with no value specified in the `numsecs` parameter.
- You can set a server-wide lock wait period by using `sp_configure` with the `lock wait period` option.
- `lock wait period`, with the session-level setting `set lock wait nnn`, is only applicable for user-defined tables. These settings have no influence on system tables.
- A lock wait period defined at the session level or in a stored procedure with the `set lock` command overrides a server-level lock-wait period.
- If `set lock wait` is used by itself, with no value for `numsecs`, all subsequent commands in the current session wait indefinitely to acquire requested locks.
- `sp_sysmon` reports the number of times that tasks waiting for a lock could not acquire the lock within the waiting period.

Repeatable-reads transaction isolation level  
- The repeatable-reads isolation level, also known as transaction isolation level 2, holds locks on all pages read by the statement until the transaction completes.
- A nonrepeatable read occurs when one transaction reads rows from a table and a second transaction can modify the same rows and commit the changes before the first transaction completes. If the first transaction rereads the rows, they now have different values, so the initial read is not repeatable. Repeatable reads hold shared locks for the duration of a transaction, blocking transactions that update the locked rows or rows on the locked pages.

Using simulated statistics
- You can load simulated statistics into a database using the simulate mode of the optdiag utility program. If set statistics simulate on has been issued in a session, queries are optimized using simulated statistics, rather than the actual statistics for a table.

Global variables affected by `set` options
- Table 1-40 lists the global variables that contain information about the session options controlled by the set command.

### Table 1-40: Global variables containing session options

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@char_convert</code></td>
<td>Contains 0 if character set conversion not in effect. Contains 1 if character set conversion is in effect.</td>
</tr>
<tr>
<td><code>@@isolation</code></td>
<td>Contains the current isolation level of the Transact-SQL program. <code>@@isolation</code> takes the value of the active level (0, 1, or 3).</td>
</tr>
<tr>
<td><code>@@options</code></td>
<td>Contains a hexadecimal representation of the session’s <code>set</code> options.</td>
</tr>
<tr>
<td><code>@@parallel_degree</code></td>
<td>Contains the current maximum parallel degree setting.</td>
</tr>
<tr>
<td><code>@@rowcount</code></td>
<td>Contains the number of rows affected by the last query. <code>@@rowcount</code> is set to 0 by any command that does not return rows, such as an <code>if</code>, <code>update</code>, or <code>delete</code> statement. With cursors, <code>@@rowcount</code> represents the cumulative number of rows returned from the cursor result set to the client, up to the last <code>fetch</code> request. <code>@@rowcount</code> is updated even when <code>nocount</code> is on.</td>
</tr>
<tr>
<td><code>@@scan_parallel_degree</code></td>
<td>Contains the current maximum parallel degree setting for nonclustered index scans.</td>
</tr>
<tr>
<td><code>@@textsize</code></td>
<td>Contains the limit on the number of bytes of text, unitext or image data a <code>select</code> returns. Default limit is 32K bytes for <code>isql</code>; the default depends on the client software. Can be changed for a session with <code>set textsize</code>. If you use <code>enable surrogate processing</code>, Unicode surrogates (two 16-bit values) are returned as single characters, even though the actual return size may be less than the <code>@@textsize</code> value.</td>
</tr>
<tr>
<td><code>@@tranchained</code></td>
<td>Contains the current transaction mode of the Transact-SQL program. <code>@@tranchained</code> returns 0 for unchained or 1 for chained.</td>
</tr>
</tbody>
</table>
Using `fipsflagger` with Java in the database

- When `fipsflagger` is on, Adaptive Server displays a warning message when these extensions are used:
  - The `installjava` utility
  - The `remove java` command
  - Column and variable declarations that reference Java classes as datatypes
  - Statements that use Java-SQL expressions for member references

- The status of `fipsflagger` does not affect arithmetic expressions performed by Java methods.

- For more information about Java in the database, see *Java in Adaptive Server Enterprise*.

Restrictions using `set tracefile`

- You cannot save the SQL text for system tasks (such as the housekeeper or the port manager).
- You must have the sa or sso roles, or be granted `set tracing` permission, to run enable or disable tracing.
- `set tracefile` is not allowed to open an existing file as a tracefile.
- During an SA or SSO session, if you enable `set tracfile` for a specific spid, all subsequent tracing commands executed take effect on that spid, not the SA or SSO spid.
- If Adaptive Server runs out of file space while writing the tracefile, it closes the file and disables the tracing.
- If an `isql` session starts tracing for a spid, but the `isql` session quits without disabling the tracing, another `isql` session can begin tracing this spid.
- Tracing occurs for the session for which it is enabled only, not for the session that enabled it.
- You cannot trace more than one session at a time from a single sa or sso session. If you attempt to open a tracefile for a session for which there is already a trace file open, Adaptive Server issues this error message: `tracefile is already open for this session`.
- You cannot trace the same session from multiple sa or sso sessions.
- The file storing the trace output is closed when the session being traced quits or when you disable tracing.
Before you allocate resources for tracing, keep in mind that each tracing requires one file descriptor per engine.

Set options that save diagnostic information to a trace file

You can use `set tracefile` in combination with other `set` commands and options that provide diagnostic information for a better understanding of slow-running queries. These are the `set` commands and options that save diagnostic information to a file:

- `set show_sqltext [on | off]`
- `set showplan [on | off]`
- `set statistics io [on | off]`
- `set statistics time [on | off]`
- `set statistics plancost [on | off]`

These are the `set` options:

- `set option show [normal | brief | long | on | off]`
- `set option show_lop [normal | brief | long | on | off]`
- `set option show_parallel [normal | brief | long | on | off]`
- `set option show_search_engine [normal | brief | long | on | off]`
- `set option show_statistics [normal | brief | long | on | off]`
- `set option show_managers [normal | brief | long | on | off]`
- `set option show_histograms [normal | brief | long | on | off]`
- `set option show_abstract_plan [normal | brief | long | on | off]`
- `set option show_best_plan [normal | brief | long | on | off]`
- `set option show_code_gen [normal | brief | long | on | off]`
- `set option show_pio_costing [normal | brief | long | on | off]`
- `set option show_lio_costing [normal | brief | long | on | off]`
- `set option show_log_props [normal | brief | long | on | off]`
- `set option show_elimination [normal | brief | long | on | off]`

Restrictions for `show_sqltext`

- You must have the `sa` or `sso` roles to run `show_sqltext`.
- You cannot use `show_sqltext` to print the SQL text for triggers.
You cannot use `show_sqltext` to show a binding variable or a view name.

Exporting `set` options from a login trigger

Adaptive Server enables `set` options inside login triggers to remain valid for the entire user session. The following `set` options are automatically exported:

- `altnames`
- `ansi_permissions`
- `ansinull`
- `arithabort [overflow | numeric_truncation]`
- `arithignore [overflow]`
- `cis_rpc_handling`
- `close on endtran`
- `colnames`
- `command_status_reporting`
- `dup_in_subquery`
- `explicit_transaction_required`
- `fipsflagger`
- `flushmessage`
- `fmtonly`
- `forceplan`
- `format`
- `noverview`
- `or_strategy`
- `prefetch`
- `proc_output_params`
- `proc_return_status`
- `procid`
- `quoted_identifier`
- `raw_object_serialization`
- `remote_indexes`
- `replication`
- `rowcount`
- `self_recursion`
- `showplan`
- `sort_merge`
- `sort_resources`
- `statistics io`
- `statement_cache`
- `strict_dtm_enforcement`
- `string_rtruncation`
- `textptr_parameters`
- `transactional_rpc`
- `triggers`

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

The ANSI SQL standard specifies behavior that differs from Transact-SQL behavior in earlier Adaptive Server versions. Compliant behavior is enabled by default for all Embedded-SQL precompiler applications. Other applications needing to match this standard of behavior can use the `set` options listed in Table 1-41.

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ansi_permissions</code></td>
<td>on</td>
</tr>
<tr>
<td><code>ansinull</code></td>
<td>on</td>
</tr>
<tr>
<td><code>arithabort</code></td>
<td>off</td>
</tr>
<tr>
<td><code>arithabort numeric_truncation</code></td>
<td>on</td>
</tr>
<tr>
<td><code>arithignore</code></td>
<td>off</td>
</tr>
<tr>
<td><code>chained</code></td>
<td>on</td>
</tr>
<tr>
<td><code>close on endtran</code></td>
<td>on</td>
</tr>
<tr>
<td><code>fipsflagger</code></td>
<td>on</td>
</tr>
<tr>
<td><code>quoted_identifier</code></td>
<td>on</td>
</tr>
<tr>
<td><code>string_rtruncation</code></td>
<td>on</td>
</tr>
<tr>
<td><code>transaction isolation level</code></td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1-41: Options to set for entry level ANSI SQL compliance

Adaptive Server Enterprise
Permissions

In general, set permission defaults to all users and no special permissions are required to use it. Exceptions include set role, set proxy, and set session authorization.

To use set role, a System Administrator or System Security Officer must have granted you the role. If you gain entry to a database only because you have a certain role, you cannot turn that role off while you are using the database. For example, if you are not normally authorized to use a database info_plan, but you use it as a System Administrator, Adaptive Server returns an error message if you try to set sa_role off while you are still in info_plan.

To use set proxy or set session authorization, you must have been granted permission by a System Security Officer.

Auditing

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 88    | security     | set proxy or set session authorization | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – Previous suid  
• Current value – New suid  
• Other information – NULL  
• Proxy information – Original login name, if set proxy or set session authorization had no parameters; otherwise, NULL. |

See also

**Commands** create trigger, fetch, grant, insert, lock table, revoke, set

**Functions** convert

**Utilities** isql, optdiag
set encryption passwd

Description
Sets the encryption password.

Syntax
set encryption passwd 'password_phrase'
    for {key | column} {keyname | column_name}

Parameters

password_phrase
  is the explicit password specified with the create encryption key or alter
  encryption key command to protect the key

key
  indicates that Adaptive Server uses this password to decrypt the key when
  accessing any column encrypted by the named key

keyname
  may be supplied as a fully qualified name. For example:
    [[database.][owner].]keyname

column
  specifies that Adaptive Server uses this password only in the context of
  encrypting or decrypting the named column. End users do not necessarily
  know the name of the key that encrypts a given column

column_name
  name of the column on which you are setting an encryption password. 
  Supply column_name as
    [[database.][owner]. ]table_name.column_name

Examples

Example 1 The key custodian shares the password required to access employee.ssn with “susan”. He does not need to disclose the name of the key to do this:

Example 2 If “susan” has select and decrypt permission on employee, she can select employee data using the password given to her for employee.ssn:

    set encryption passwd "Ynot387" for column employee.ssn
    select ename from employee where ssn = '111-22-3456'
    ename
    --------------------
    Priscilla Kramnik

Example 3 If “susan” attempts to select data from payroll without specifying the password for payroll.ssn, the following select fails (even if “susan” has select and decrypt permission on payroll):

    select base_salary from payroll where ssn = '111-22-3456'

You cannot execute 'SELECT' command because the user encryption password
has not been set.

To avoid this error, “susan” must first enter:

```
set encryption passwd "Ynot387" for column payroll.ssn
```

**Usage**

- If a stored procedure or a trigger references a column encrypted by a user specified password, you must set the encryption password before executing the procedure or the statement that fires the trigger.

- Sybase does not recommend that you place the `set encryption passwd` statement inside the trigger or procedure because this could lead to unintentional exposure of the password through `sp_helptext`. Additionally, hard-coded passwords require you to change the procedure or trigger when a password is changed.
setuser

Description
Allows a Database Owner to impersonate another user.

Syntax
setuser ["user_name"]

Examples
The Database Owner temporarily adopts Mary’s identity in the database in order to grant Joe permissions on authors, a table owned by Mary:

```
setuser "mary"
go
grant select on authors to joe
setuser
go
```

Usage
- The Database Owner uses `setuser` to adopt the identity of another user in order to use another user’s database object, to grant permissions, to create an object, or for some other reason.

- When the Database Owner uses the `setuser` command, Adaptive Server checks the permissions of the user being impersonated instead of the permissions of the Database Owner. The user being impersonated must be listed in the `sysusers` table of the database.

- `setuser` affects permissions only in the local database. It does not affect remote procedure calls or accessing objects in other databases.

- `setuser` remains in effect until another `setuser` command is given or until the current database is changed with the `use` command.

- `setuser` has no effect when creating a database.

- Executing `setuser` with no user name reestablishes the Database Owner’s original identity.

- System Administrators can use `setuser` to create objects that are owned by another user. However, since a System Administrator operates outside the permissions system, she or he cannot use `setuser` to acquire another user’s permissions.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
`setuser` permission defaults to the Database Owner and is not transferable.

Auditing
Values in `event` and `extrainfo` columns of `sysaudits` are:
<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>setuser</td>
<td>setuser</td>
<td></td>
</tr>
</tbody>
</table>

- **Roles** – Current active roles
- **Keywords or options** – NULL
- **Previous value** – NULL
- **Current value** – NULL
- **Other information** – Name of the user being set
- **Proxy information** – Original login name, if a set proxy is in effect

See also

**Commands** grant, revoke, use
**shutdown**

**Description**
Shuts down the Adaptive Server from which the command is issued, its local Backup Server, or a remote Backup Server.

**Syntax**
```
shutdown [srvname] [with {wait [="hh:mm:ss"] | nowait}]
```

**Parameters**
- **srvname**
  is the logical name by which the Backup Server is known in the Adaptive Server sysservers system table. This parameter is not required when shutting down the local Adaptive Server.

- **with wait**
  is the default. This shuts down the Adaptive Server or Backup Server gracefully.

- **hh:mm:ss**
  is an optional setting that specifies the maximum time the server waits for all running or sleeping processes to finish their job.

- **with nowait**
  shuts down the Adaptive Server or Backup Server immediately, without waiting for currently executing statements to finish.

**Note** Use of shutdown with nowait can lead to gaps in IDENTITY column values.

**Examples**

**Example 1** Shuts down the Adaptive Server from which the shutdown command is issued:
```
shutdown
```

**Example 2** Shuts down the Adaptive Server immediately:
```
shutdown with nowait
```

**Example 3** Shuts down the local Backup Server:
```
shutdown SYB_BACKUP
```

**Example 4** Shuts down the remote Backup Server REM_BACKUP:
```
shutdown REM_BACKUP
```

**Usage**
- Unless you use the nowait option, shutdown attempts to bring Adaptive Server down gracefully by:
  - Disabling logins (except for the System Administrator)
  - Performing a checkpoint in every database
• Waiting for currently executing SQL statements or stored procedures to finish

Shutting down the server without the nowait option minimizes the amount of work that must be done by the automatic recovery process.

• Unless you use the nowait option, shutdown backup_server waits for active dumps and loads to complete. Once you issue a shutdown command to a Backup Server, no new dumps or loads that use this Backup Server can start.

• Use shutdown with nowait only in extreme circumstances. In Adaptive Server, issue a checkpoint command before executing a shutdown with nowait.

• You can halt only the local Adaptive Server with shutdown; you cannot halt a remote Adaptive Server.

• You can halt a Backup Server only if:
  • It is listed in your sysservers table. Use sp_addserver to add entries to sysservers.
  • It is listed in the interfaces file for the Adaptive Server where you execute the command.

• Use sp_helpserver to determine the name by which a Backup Server is known to the Adaptive Server. Specify the Backup Server’s name—not its network_name—as the srvname parameter. For example:

<table>
<thead>
<tr>
<th>name</th>
<th>network_name</th>
<th>status</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>REM_BACKUP</td>
<td>WHALE_BACKUP</td>
<td>timeouts, no net password encryption</td>
<td>3</td>
</tr>
<tr>
<td>SYB_BACKUP</td>
<td>SLUG_BACKUP</td>
<td>timeouts, net password encryption</td>
<td>1</td>
</tr>
<tr>
<td>eel</td>
<td>eel</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>whale</td>
<td>whale</td>
<td>timeouts, no net password encryption</td>
<td>2</td>
</tr>
</tbody>
</table>

To shut down the remote Backup Server named WHALE_BACKUP, use:

`shutdown REM_BACKUP`
3 Waits for all running or sleeping processes to finish their job
4 Performs another checkpoint on the databases, this time with a flag that informs you that you need to flush:
   • All the dynamic thresholds in mixed log-data databases
   • All the object statistics
   • The values of the identity fields to avoid holes after recovery

When you use `with wait` with the `hh:mm:ss` option, the time you specify is not the maximum total time Adaptive Server takes to shut itself down. Instead, Adaptive Server takes into account the time it takes to perform the first checkpoint, and automatically subtracts this from the time you specified.

For example, if you specify a maximum wait time of 20 minutes and the first checkpoint takes 3 minutes, Adaptive Server allows up to 17 minutes for the processes to finish. If for some reason the second checkpoint takes longer, however, this is not calculated into the `with wait hh:mm:ss` parameter you specify.

Adaptive Server also accommodates a checkpoint that takes longer than the time you specify in `with wait hh:mm:ss`. For example, if you specify a wait time of 10 minutes but the first checkpoint takes 20 minutes to complete, Adaptive Server does not interrupt checkpoint midway, but instead waits for checkpoint to complete. When this occurs, Adaptive Server immediately begins to shut down after checkpoint is complete, since the time you specified has passed, and runs the last checkpoint with the flag informing you of the flushes you must perform.

### Standards
ANSI SQL – Compliance level: Transact-SQL extension.

### Permissions
`shutdown` permission defaults to System Administrators and is not transferable.

### Auditing
Values in `event` and `extrainfo` columns of `sysaudits` are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 51    | security     | Server shutdown           | • Roles – Current active roles  
|       |              |                           | • Keywords or options – shutdown  
|       |              |                           | • Previous value – NULL  
|       |              |                           | • Current value – NULL  
|       |              |                           | • Other information – NULL  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect  

592 Adaptive Server Enterprise
See also

<table>
<thead>
<tr>
<th>Commands</th>
<th>alter database</th>
</tr>
</thead>
<tbody>
<tr>
<td>System procedures</td>
<td>sp_addserver, sp_helpserver</td>
</tr>
</tbody>
</table>
**truncate table**

Description

Removes all rows from a table or partition.

Syntax

```
truncate table [[database.]owner.]table_name
    [partition partition_name]
```

Parameters

- `table_name` is the name of the table to truncate. Specify the database name if the table is in another database, and specify the owner’s name if more than one table of that name exists in the database. The default value for `owner` is the current user, and the default value for `database` is the current database.

- `partition_name` specifies the name of the partition to truncate.

Examples

**Example 1** Removes all data from the `authors` table:

```
truncate table authors
```

**Example 2** Removes all data from the `smallsales` partition of the `titles` table:

```
truncate table titles partition smallsales
```

Usage

- `truncate table` deletes all rows from a table. The table structure and all the indexes continue to exist until you issue a `drop table` command. The rules, defaults, and constraints that are bound to the columns remain bound, and triggers remain in effect.

- Adaptive Server no longer uses distribution pages; statistical information is now stored in the tables `sysstatistics` and `systabstats`. During `truncate table`, statistical information is no longer deleted (deallocated), so you need not run `update statistics` after adding data. `truncate table` does not delete statistical information for the table.

- `truncate table` is equivalent to—but faster than—a `delete` command without a `where` clause. `delete` removes rows one at a time and logs each deleted row as a transaction; `truncate table` deallocates whole data pages and makes fewer log entries. Both `delete` and `truncate table` reclaim the space occupied by the data and its associated indexes.

- Truncating a partition does not affect the data in other partitions.

- You can truncate only one partition at a time.

- Truncating a table locks the entire table until the truncation process is complete.
• Because the deleted rows are not logged individually, truncate table cannot fire a trigger.

• You cannot use truncate table if another table has rows that reference it. Delete the rows from the foreign table, or truncate the foreign table, then truncate the primary table.

• You can grant and revoke permissions to users and roles to use truncate table on tables with the grant and revoke commands.

Standards
ANSI SQL – Compliance level: Entry-level compliant.

Permissions
truncate table permission defaults to the table owner. Table owners can grant permissions for truncate table to other users. To truncate a system audit table (sysaudits_01, sysaudits_02, sysaudits_03, and so on, through sysaudits_08), you must have the a System Security Officer (SSO) role.

Auditing
Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 64    | truncate     | truncate table            | • Roles – Current active roles  
|       |              |                           | • Keywords or options – NULL  
|       |              |                           | • Previous value – NULL     
|       |              |                           | • Current value – NULL      
|       |              |                           | • Other information – NULL  
|       |              |                           | • Proxy information – Original login name, if set proxy in effect |

See also Commands alter table, create table, create trigger, delete, drop table, grant, revoke
union operator

Returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the all keyword is specified.

Syntax

```sql
select [top unsigned_integer] select_list
   [into clause] [from clause] [where clause]
   [group by clause] [having clause]
   [union [all]]
select [top unsigned_integer] select_list
   [from clause] [where clause]
   [group by clause] [having clause]][...
   [order by clause]
[compute clause]
```

Parameters

top unsigned_integer

The top limit applies to the individual selects that form a union, not to the union as a whole.

into

creates a new table based on the columns specified in the select list and the rows chosen in the where clause. The first query in the union operation is the only one that can contain an into clause.

union

creates the union of data specified by two select statements.

all

includes all rows in the results; duplicates are not removed.

Examples

**Example 1** The result set includes the contents of the stor_id and stor_name columns of both the sales and sales_east tables:

```sql
select stor_id, stor_name from sales
union
select stor_id, stor_name from sales_east
```

**Example 2** The into clause in the first query specifies that the results table holds the final result set of the union of the specified columns of the publishers, stores, and stores_east tables:

```sql
select pub_id, pub_name, city into results
from publishers
union
select stor_id, stor_name, city from stores
union
select stor_id, stor_name, city from stores_east
```
Example 3 First, the union of the specified columns in the sales and sales_east tables is generated. Then, the union of that result with publishers is generated. Finally, the union of the second result and authors is generated:

```sql
select au_lname, city, state from authors
union
((select stor_name, city, state from sales
union
select stor_name, city, state from sales_east)
union
select pub_name, city, state from publishers)
```

Example 4 Returns six rows. The top limit applies to the individual selects that form a union, not to the union as a whole:

```sql
select top 3 au_lname from authors
union all
select top 3 title from titles
```

Usage

- The maximum number of subqueries within a single side of a union is 50.
- The total number of tables that can appear on all sides of a union query is 256.
- You can use union in select statements, for example:

  ```sql
  create view
  select * from Jan1998Sales
  union all
  select * from Feb1998Sales
  union all
  ```

- The order by and compute clauses are allowed only at the end of the union statement to define the order of the final results or to compute summary values.
- The group by and having clauses can be used only within individual queries and cannot be used to affect the final result set.
- The default evaluation order of a SQL statement containing union operators is left-to-right.
- Since union is a binary operation, parentheses must be added to an expression involving more than two queries to specify evaluation order.
- The first query in a union statement may contain an into clause that creates a table to hold the final result set. The into statement must be in the first query, or you receive an error message (see Example 2).
The union operator can appear within an insert...select statement. For example:

```
insert into sales.overall
select * from sales
union
select * from sales_east
```

All select lists in a SQL statement must have the same number of expressions (column names, arithmetic expressions, aggregate functions, and so on). For example, the following statement is invalid because the first select list contains more expressions than the second:

```
/* Example of invalid command--shows imbalance */
in select list items */
select au_id, title_id, au_ord from titleauthor
union
select stor_id, date from sales
```

Corresponding columns in the select lists of union statements must occur in the same order, because union compares the columns one-to-one in the order given in the individual queries.

The column names in the table resulting from a union are taken from the first individual query in the union statement. To define a new column heading for the result set, do it in the first query. Also, to refer to a column in the result set by a new name (for example, in an order by statement), refer to it by that name in the first select statement. For example, the following query is correct:

```
select Cities = city from stores
union
select city from stores_east
order by Cities
```

The descriptions of the columns that are part of a union operation do not have to be identical. Table 1-42 lists the rules for the datatypes and the corresponding column in the result table.

<table>
<thead>
<tr>
<th>Datatype of columns in union operation</th>
<th>Datatype of corresponding column in result table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not datatype-compatible (data conversion is not handled implicitly by Adaptive Server)</td>
<td>Error returned by Adaptive Server.</td>
</tr>
<tr>
<td>Both are fixed-length character with lengths L1 and L2</td>
<td>Fixed-length character with length equal to the greater of L1 and L2.</td>
</tr>
<tr>
<td>Both are fixed-length binary with lengths L1 and L2</td>
<td>Fixed-length binary with length equal to the greater of L1 and L2.</td>
</tr>
</tbody>
</table>
Restrictions

- You cannot use the union operator in a subquery.
- You cannot use the union operator with the for browse clause.

Standards

ANSI SQL – Compliance level: Entry-level compliant

The following are Transact-SQL extensions:

- The use of union in the select clause of an insert statement
- Specifying new column headings in the order by clause of a select statement when the union operator is present in the select statement

See also

Commands  compute clause, declare, group by and having clauses, order by clause, select, where clause

Functions  convert
unmount

Description

The `unmount` command shuts down the database and drops it from the Adaptive Server. The devices are also deactivated and dropped. The database and its pages are not altered when they are unmounted. The database pages remain on the OS devices. Once the unmount command completes, you can disconnect and move the devices at the source Adaptive Server if necessary. Use the `manifest_file` extension to create the manifest file for use at the secondary Adaptive Server.

The `unmount` command limits the number of databases to eight in a single command.

**Warning!** The `unmount` command removes a database and all its information from the Adaptive Server. Use the `unmount` command only when you want to remove the database from one Adaptive Server to another Adaptive Server.

Syntax

```
unmount database dbname_list to manifest_file
```

Parameters

- `dbname_list`
  - the database being unmounted. You can unmount more than one database.

- `manifest_file`
  - the binary file that describes the databases that are present on a set of database devices. It can be created only if the set of databases that occupy those devices are isolated and self-contained on those devices.

Since the manifest file is a binary file, operations that perform character translations of the file contents (such as `ftp`) will corrupt the file unless done in binary mode.

Examples

Unmount databases from an Adaptive Server and create the manifest file for the database:

```
unmount database pubs2 to "/work2/Devices/Mpubs2_file"
```

Usage

You cannot:

- Unmount system databases. However, you can unmount `sys$systemprocs`.
- Unmount proxy databases or user created temporary databases.
- Use the `unmount` command in a transaction.
- Unmount a database on an HA-configured server.
Encrypted columns and _unmount database_

When columns are encrypted by keys from other databases, unmount all related databases as a set. The interdependency of the databases containing the encrypted columns and the databases containing the keys is similar to the interdependency of databases that use referential integrity.

You use the _override_ option to unmount a database containing columns encrypted by a key in another database.

With the following commands the encryption key created in _key_db_ has been used to encrypt columns in _col_db_. These commands successfully unmount the named databases:

```
unmount database key_db, col_db
unmount database key_db with override
unmount database col_db with override
```

These commands will fail with an error message without the _override_:

```
unmount database key_db
unmount database col_db
```

Standards

ANSI SQL – Compliance level: Entry-level compliant.

Auditing

Values in event and extrainfo columns of _sysaudits_ are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 102   | unmount      | unmount database          | • Roles – Current active roles  
• Keywords or options – NULL  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if a set proxy is in effect |

See also _Commands_ mount, quiesce database
update

Description
Changes data in existing rows, either by adding data or by modifying existing data.

Syntax
update [top unsigned_integer]
   | [database.]owner.][table_name | view_name]
   | set [([database.]owner.][table_name | view_name.])
   |  column_name1 =
   |  {expression1 | NULL | (select_statement)} |
   |  variable_name1 =
   |  {expression1 | NULL | (select_statement)} |
   |  [, column_name2 =
   |  {expression2 | NULL | (select_statement)}... |
   |  [, variable_name2 =
   |  {expression2 | NULL | (select_statement)}... |
   |  [from [([database.]owner.][view_name [readpast]]
   |   table_name
   |   [(index {index_name | table_name}
   |        [prefetch size][lru|mru]])]
   |   [readpast]
   |   [([database.]owner.][view_name [readpast]]
   |   table_name
   |   [(index {index_name | table_name}
   |        [prefetch size][lru|mru]])]
   |   [readpast] ...]
   | [where search_conditions]
   | [plan "abstract plan"]

Parameters

- **table_name | view_name**
is the name of the table or view to update. Specify the database name if the table or view is in another database, and specify the owner’s name if more than one table or view of that name exists in the database. The default value for **owner** is the current user, and the default value for **database** is the current database.

---

Adaptive Server Enterprise
top unsigned_integer
inserts the top n clause immediately after the keyword, and limits the number of rows updated.

set
specifies the column name or variable name and assigns the new value. The value can be an expression or a NULL. When more than one column name or variable name and value are listed, they must be separated by commas.

from
uses data from other tables or views to modify rows in the table or view you are updating.

readpast
causes the update command to modify unlocked rows only on datarows-locked tables, or rows on unlocked pages, for datapages-locked tables. update...readpast silently skips locked rows or pages rather than waiting for the locks to be released.

where
is a standard where clause (see where clause).

index (index_name | table_name)
index_name specifies the index to be used to access table_name. You cannot use this option when you update a view.

prefetch size
specifies the I/O size, in kilobytes, for tables bound to caches with large I/Os configured. You cannot use this option when you update a view.
sp_helpcache shows the valid sizes for the cache to which an object is bound or for the default cache. To configure the data cache size, use sp_cacheconfigure.

When using prefetch and designating the prefetch size (size), the minimum is 2K and any power of two on the logical page size up to 16K. prefetch size options in kilobytes are:

<table>
<thead>
<tr>
<th>Logical page size</th>
<th>Prefetch size options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2, 4, 8, 16</td>
</tr>
<tr>
<td>4</td>
<td>4, 8, 16, 32</td>
</tr>
<tr>
<td>8</td>
<td>8, 16, 32, 64</td>
</tr>
<tr>
<td>16</td>
<td>16, 32, 64, 128</td>
</tr>
</tbody>
</table>
The prefetch size specified in the query is only a suggestion. To allow the size specification configure the data cache at that size. If you do not configure the data cache to a specific size, the default prefetch size is used.

If Component Integration Services is enabled, you cannot use prefetch for remote servers.

\texttt{lru | mru}

specifies the buffer replacement strategy to use for the table. Use lru to force the optimizer to read the table into the cache on the MRU/LRU (most recently used/least recently used) chain. Use mru to discard the buffer from cache and replace it with the next buffer for the table. You cannot use this option when you update a view.

\texttt{where current of}

causes Adaptive Server to update the row of the table or view indicated by the current cursor position for \texttt{cursor_name}.

\texttt{index name}

is the name of the index to be updated. If an index name is not specified, the distribution statistics for all the indexes in the specified table are updated.

\texttt{plan "abstract plan"}

specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. See Chapter 30, “Creating and Using Abstract Plans,” in the \textit{Performance and Tuning Guide} for more information.

**Examples**

**Example 1** All the McBaddens in the authors table are now MacBaddens:

```
update authors
set au_lname = "MacBadden"
where au_lname = "McBadden"
```

**Example 2** Modifies the total sales column to reflect the most recent sales recorded in the sales and salesdetail tables. This assumes that only one set of sales is recorded for a given title on a given date, and that updates are current:

```
update titles
set total_sales = total_sales + qty
from titles, salesdetail, sales
where titles.title_id = salesdetail.title_id
    and salesdetail.stor_id = sales.stor_id
    and salesdetail.ord_num = sales.ord_num
    and sales.date in
        (select max (sales.date) from sales)
```
Example 3 Changes the price of the book in the titles table that is currently pointed to by title_crsr to $24.95:

```
update titles
set price = 24.95
where current of title_crsr
```

Example 4 Finds the row for which the IDENTITY column equals 4 and changes the price of the book to $18.95. Adaptive Server replaces the syb_identity keyword with the name of the IDENTITY column:

```
update titles
set price = 18.95
where syb_identity = 4
```

Example 5 Updates the titles table using a declared variable:

```
declare @x money
select @x = 0
update titles
    set total_sales = total_sales + 1,
    @x = price
where title_id = "BU1032"
```

Example 6 Updates rows on which another task does not hold a lock:

```
update salesdetail set discount = 40
    from salesdetail readpast
    where title_id like "BU1032"
    and qty > 100
```

Usage

- Use update to change values in rows that have already been inserted. Use insert to add new rows.
- You can refer to as many as 15 tables in an update statement.
- update interacts with the ignore_dup_key, ignore_dup_row, and allow_dup_row options set with the create index command. See create index for more information.
- You can define a trigger that takes a specified action when an update command is issued on a specified table or on a specified column in a table.
- In pre-12.5.2 versions of Adaptive Server, queries that used update and delete on views with a union all clause were sometimes resolved without using worktables, which occasionally lead to incorrect results. In Adaptive Server 12.5.2, queries that use update and delete on views with a union all clause are always resolved using worktables in tempdb.
Using variables in update statements

- You can assign variables in the set clause of an update statement, similarly to setting them in a select statement.

- Before you use a variable in an update statement, you must declare the variable using declare, and initialize it with select, as shown in Example 5.

- Variable assignment occurs for every qualified row in the update.

- When a variable is referenced on the right side of an assignment in an update statement, the current value of the variable changes as each row is updated. The current value is the value of the variable just before the update of the current row. The following example shows how the current value changes as each row is updated.

Suppose you have the following statement:

```
declare @x int
select @x=0
update table1
    set C1=C1+@x, @x=@x+1
where column2=xyz
```

The value of C1 before the update begins is 1. The following table shows how the current value of the @x variable changes after each update:

<table>
<thead>
<tr>
<th>Row</th>
<th>Initial C1 value</th>
<th>Initial @x value</th>
<th>Calculations: C1+@x = updated C1</th>
<th>Updated C1 value</th>
<th>Calculations: @x+1 = updated @x</th>
<th>Updates value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1+0</td>
<td>1</td>
<td>0+1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1+1</td>
<td>2</td>
<td>1+1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
<td>2+2</td>
<td>4</td>
<td>2+1</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3</td>
<td>4+3</td>
<td>7</td>
<td>3+1</td>
<td>4</td>
</tr>
</tbody>
</table>

- When multiple variable assignments are given in the same update statement, the values assigned to the variables can depend on their order in the assignment list, but they might not always do so. For best results, do not rely on placement to determine the assigned values.

- If multiple rows are returned and a nonaggregating assignment of a column to a variable occurs, then the final value of the variable is the last row processed; therefore, it might not be useful.

- An update statement that assigns values to variables need not set the value of any qualified row.

- If no rows qualify for the update, the variable is not assigned.
• A variable that is assigned a value in the update statement cannot be referenced in subquery in that same update statement, regardless of where the subquery appears in that update statement.

• A variable that is assigned a value in the update statement cannot be referenced in a where or having clause in that same update statement.

• In an update driven by a join, a variable that is assigned a value in the right hand side of the update statement uses columns from the table that is not being updated. The result value depends on the join order chosen for the update and the number of rows that qualify from the joined table.

• Updating a variable is not affected by a rollback of the update statement because the value of the updated variable is not stored on disk.

Using `update` with transactions

• When you set chained transaction mode on, and no transaction is currently active, Adaptive Server implicitly begins a transaction with the update statement. To complete the update, you must either commit the transaction or rollback the changes. For example:

```sql
update stores set city = 'Concord'
   where stor_id = '7066'
   if exists (select t1.city, t2.city
               from stores t1, stores t2
               where t1.city = t2.city
               and t1.state = t2.state
               and t1.stor_id < t2.stor_id)
       rollback transaction
   else
       commit transaction
```

This batch begins a transaction (using chained transaction mode) and updates a row in the stores table. If it updates a row containing the same city and state information as another store in the table, it rolls back the changes to the stores table and ends the transaction. Otherwise, it commits the updates and ends the transaction.

• Adaptive Server does not prevent you from issuing an update statement that updates a single row more than once in a given transaction. For example, both of these updates affect the price of the book with title_id MC2022, since its type id "mod_cook":

```sql
begin transaction
update titles
set price = price + $10
where title_id = "MC2222"
```
Using joins in updates

- Performing joins in the FROM clause of an update is an Transact-SQL extension to the ANSI standard SQL syntax for updates. Because of the way an update statement is processed, updates from a single statement do not accumulate. That is, if an update statement contains a join, and the other table in the join has more than one matching value in the join column, the second update is not based on the new values from the first update but on the original values. The results are unpredictable, since they depend on the order of processing. Consider this join:

```sql
update titles set total_sales = total_sales + qty
    from titles t, salesdetail sd
    where t.title_id = sd.title_id
```

The `total_sales` value is updated only once for each `title_id` in `titles`, for one of the matching rows in `salesdetail`. Depending on the join order for the query, on table partitioning, or on the indexes available, the results can vary each time. But each time, only a single value from `salesdetail` is added to the `total_sales` value.

If the intention is to return the sum of the values that match the join column, the following query, using a subquery, returns the correct result:

```sql
update titles set total_sales = total_sales +
    (select isnull (sum (qty), 0)
     from salesdetail sd
     where t.title_id = sd.title_id)
from titles t
```

Using update with character data

- Updating variable-length character data, or text or unitext columns with the empty string (""") inserts a single space. Fixed-length character columns are padded to the defined length.

- All trailing spaces are removed from variable-length column data, except when a string contains only spaces. Strings that contain only spaces are truncated to a single space. Strings longer than the specified length of a char, nchar, unichar, varchar, univarchar, or nvarchar column are silently truncated unless you set string_rtruncation on.

- An update to a text or unitext column initializes the text or unitext column, assigns it a valid text pointer, and allocates at least one text page.
Using `update` with cursors

- You cannot update a scrollable cursor.
- To update a row using a cursor, define the cursor with `declare cursor`, then open it. The cursor name cannot be a Transact-SQL parameter or a local variable. The cursor must be updatable, or Adaptive Server returns an error. Any update to the cursor result set also affects the base table row from which the cursor row is derived.

- The `table_name` or `view_name` specified with an `update...where current of` must be the table or view specified in the first `from` clause of the `select` statement that defines the cursor. If that `from` clause references more than one table or view (using a join), you can specify only the table or view being updated.

After the update, the cursor position remains unchanged. You can continue to update the row at that cursor position, provided another SQL statement does not move the position of that cursor.

- Adaptive Server allows you to update columns that are not specified in the list of columns of the cursor’s `select_statement`, but that are part of the tables specified in the `select_statement`. However, when you specify a `column_name_list` with `for update`, and you are declaring the cursor, you can update only those specific columns.

Updating IDENTITY columns

- You cannot update a column with the IDENTITY property, either through its base table or through a view. To determine whether a column was defined with the IDENTITY property, use `sp_help` on the column’s base table.

- An IDENTITY column selected into a result table observes the following rules with regard to inheritance of the IDENTITY property:
  - If an IDENTITY column is selected more than once, it is defined as NOT NULL in the new table. It does not inherit the IDENTITY property.
  - If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as NULL if any column in the expression allows nulls; otherwise, it is NOT NULL.
• If the select statement contains a group by clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created NULL; others are created NOT NULL.

• An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a NULL column, the new column is defined as NULL. Otherwise, it is defined as NOT NULL.

Updating data through views

• You cannot update views defined with the distinct clause.

• If a view is created with check option, each row that is updated through the view must remain visible through the view. For example, the stores_cal view includes all rows of the stores table where state has a value of “CA”.

   The with check option clause checks each update statement against the view’s selection criteria:

   ```sql
   create view stores_cal
   as select * from stores
   where state = "CA"
   with check option
   ```

An update statement such as this one fails if it changes state to a value other than “CA”:

```sql
update stores_cal
set state = "WA"
where store_id = "7066"
```

• If a view is created with check option, all views derived from the base view must satisfy the view’s selection criteria. Each row updated through a derived view must remain visible through the base view.

Consider the view stores_cal30, which is derived from stores_cal. The new view includes information about stores in California with payment terms of “Net 30”:

```sql
create view stores_cal30
as select * from stores_cal
where payterms = "Net 30"
```

Because stores_cal was created with check option, all rows updated through stores_cal30 must remain visible through stores_cal. Any row that changes state to a value other than “CA” is rejected.
Notice that stores_cal30 does not have a with check option clause of its own. Therefore, you can update a row with a payterms value other than “Net 30” through stores_cal30. For example, the following update statement would be successful, even though the row would no longer be visible through stores_cal30:

```sql
update stores_cal30
    set payterms = "Net 60"
    where stor_id = "7067"
```

- You cannot update a row through a view that joins columns from two or more tables, unless both of the following conditions are true:
  - The view has no with check option clause, and
  - All columns being updated belong to the same base table.

- update statements are allowed on join views that contain a with check option clause. The update fails if any of the affected columns appear in the where clause in an expression that includes columns from more than one table.

- If you update a row through a join view, all affected columns must belong to the same base table.

Using `index`, `prefetch`, or `lru | mru`

- `index`, `prefetch`, and `lru | mru` override the choices made by the Adaptive Server optimizer. Use them with caution, and always check the performance impact with `set statistics io on`. For more information about using these options, see the Performance and Tuning Guide.

Using `readpast`

- The `readpast` option applies only to data-only-locked tables. `readpast` is ignored if it is specified for an allpages-locked table.

- The `readpast` option is incompatible with the holdlock option. If both are specified in the same select command, an error is generated and the command terminates.

- If the session-wide isolation level is 3, the `readpast` option is ignored.

- If the transaction isolation level for a session is 0, update commands using `readpast` do not issue warning messages. For datapages-locked tables, these commands modify all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, they affect all rows that are not locked with incompatible locks.
If an update command with the readpast option applies to two or more text columns, and the first text column checked has an incompatible lock on it, readpast locking skips the row. If the column does not have an incompatible lock, the command acquires a lock and modifies the column. Then, if any subsequent text column in the row has an incompatible lock on it, the command blocks until it can obtain a lock and modify the column.

For more information on readpast locking, see the *Performance and Tuning Guide*.

**Standards**

ANSI SQL – Compliance level: Entry-level compliant.

The following are Transact-SQL extensions:

- The use of a from clause or a qualified table or column name are Transact-SQL extensions detected by the FIPS flagger. Updates through a join view or a view of which the target list contains an expression are Transact-SQL extensions that cannot be detected until run time and are not flagged by the FIPS flagger.

- The use of variables.

- readpast

**Permissions**

update permission defaults to the table or view owner, who can transfer it to other users.

If set ansi_permissions is on, you need update permission on the table being updated and, in addition, you must have select permission on all columns appearing in the where clause and on all columns following the set clause. By default, ansi_permissions is off.

**Auditing**

Values in event and extrainfo columns of sysaudits are:

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 70    | update       | update to a table         | • Roles – Current active roles  
• Keywords or options – update or writetext  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

Adaptive Server Enterprise
<table>
<thead>
<tr>
<th>Event</th>
<th>Audit option</th>
<th>Command or access audited</th>
<th>Information in extrainfo</th>
</tr>
</thead>
</table>
| 71    | update       | update to a view         | • Roles – Current active roles  
• Keywords or options – update or writetext  
• Previous value – NULL  
• Current value – NULL  
• Other information – NULL  
• Proxy information – Original login name, if set proxy in effect |

See also

**Commands** alter table, create default, create index, create rule, create trigger, insert, where clause

**System procedures** sp_bindefault, sp_bindrule, sp_help, sp_helppartition, sp_helpindex, sp_unbindefault, sp_unbindrule
update all statistics

Description
Updates all statistics information for a given table. You can run update all statistics on a single data partition.

Syntax
update all statistics table_name [partition data_partition_name]

Parameters

- **table_name**
  is the name of the table for which statistics are being updated.

- **data_partition_name**
  is the name of the partition to be updated. Statistics for each local index partition on the data partition is updated. Does not update statistics for global indexes.

Examples

**Example 1** Updates all statistics for the salesdetail table:

```
update all statistics salesdetail
```

**Example 2** Updates all statistics for the smallsales partition on salesdetail table:

```
update all statistics salesdetail partition smallsales
```

Usage

- **update all statistics** updates all the statistics information for a given table. Adaptive Server keeps statistics about the distribution of pages within a table, and uses these statistics when considering whether or not to use a parallel scan in query processing on partitioned tables, and which index (es) to use in query processing. The optimization of your queries depends on the accuracy of the stored statistics.

- Histogram statistics are created on each column, either through an index scan of a leading column or a projection of the column into a work table, followed by a sort.

- Density statistics are created for all the prefix subsets of the columns of index (es) whose statistics is being updated. For example, if an index is on columns c1, c2 and c3, then the prefix subsets are (c1,c2) and (c1, c2, c3).

- When you run update all statistics on a single data partition, histograms are generated for each leading column of the local indexes using an index scan. For all other columns, including leading columns of a global index, update all statistics performs a data scan followed by a sort.

- **update statistics** commands create partition-specific statistics. Global statistics are implicitly created during partition statistics creation. The partition statistics serve as input to global statistics creation and enable per-partition DDL operations. Global statistics are used by the optimizer.
• **update all statistics** regenerates and update the table statistics stored in systabstats for each data and index partition of the table. If the `update all statistics` command is run for a specific data partition, the table statistics are generated and updated only for that data partition and any local index partitions. Global indexes are skipped.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`update all statistics` permission defaults to the table owner and is not transferrable.

**See also**

**Commands** update statistics, update index statistics, update statistics, update table statistics
**update index statistics**

**Description**
Updates the statistics for all columns in an index.

**Syntax**
```
update index statistics
    table_name [[partition data_partition_name] |
    [index_name [partition index_partition_name]]]
    [using step values]
    [with consumers = consumers] [, sampling=N percent]
```

**Parameters**
- **table_name**
  when used with update statistics, `table_name` is the name of the table with which the index is associated. `table_name` is required, since Transact-SQL does not require index names to be unique in a database.

- **data_partition_name**
  is the name of the partition to be updated. Statistics for each local index partition on the data partition is updated. Does not update statistics for global indexes.

- **index_name**
  is the name of the index to be updated. If an index name is not specified, the distribution statistics for all the indexes in the specified table are updated.

- **index_partition_name**
  is the name of the index partition to be updated.

- **using step values**
  specifies the number of histogram steps. The default value is 20, for columns where no statistics exist. If you need to change the default for this, use `sp_configure` to modify the `number of histogram steps` parameter. If statistics for a column already exist in `sysstatistics`, the default value is the current number of steps.

  The steps are applied to each partition of a partitioned table—for example, `update index statistics` uses the default value of 20 steps for each data and index partition involved in the scan for updating statistics. If global statistics are generated through an index scan of a global index, then 20 steps are applied by default. If partition statistics are generated, either through a data scan or local index scan, then 20 steps are applied by default for each partition.

  If the histogram steps specified through `using step values` is M, and the `histogram tuning factor` parameter is N, then `update index statistics` uses between M and M*N steps, depending on the number of frequency cells that `update index statistics` isolates.
with consumers = consumers
specifies the number of consumer processes to be used for a sort when
column_list is provided and parallel query processing is enabled. The
consumers option specifies the degree of parallelism applied to a sort
performed for statistics update on a single data partition. For example, if
update statistics with a column list is applied to a table with three data
partitions, data from each of the three partitions is sorted separately and the
consumers option is applied during each of the sort. The three sorts
themselves are not performed in parallel.

with sampling = N percent
specifies the percentage of the column to be randomly sampled in order to
gather statistics. The value for N is any number between 1 and 100.

Examples

Example 1 Generates statistics for all columns in all indexes of the authors
table:

update index statistics authors

Example 2 Generates statistics for all columns in the au_names_ix index of the
authors table:

update index statistics authors au_names_ix

Example 3 Generates statistics on all inner columns of the au_names_ix index
using a sampling rate of 20 percent.

update index statistics authors au_names_ix
with sampling = 20 percent

The statistics for the leading column of au_names_ix is gathered using a full
scan of the index pages; sampling is not applied on this column.

Example 4 Generates statistics for all the columns of an index partition:

update index statistics publishers publish1_idx
partition p1

Usage

• update index statistics, when used with a table name and an index name,
  updates statistics for all columns in the specified index. If update index
  statistics is used with just a table name, it updates statistics for all columns
in all indexes of the table.

• If you run update index statistics against large tables, the command fails
  with error number 1105 if tempdb is not large enough to process the
  command.

• Specifying the name of an unindexed column or the nonleading column of
an index generates statistics for that column without creating an index.
**update index statistics**

- Histogram statistics are created for each column of index (es) whose statistics is being updated.
- Density statistics are created for all the prefix subsets of the columns of index (es) whose statistics are being updated.
- If you use `update index statistics` on a specific partition, you update global statistics implicitly as well.
- The partition statistics serve as input to global statistics creation and enable per-partition DDL operations. Global statistics are used by the optimizer.
- `update index statistics` also regenerates and updates the table statistics stored in `systabstats` for each data and index partition of the table the command updates. If you run the `update index statistics` command for a specific data partition, the table statistics are generated and updated only for that data partition and for any local index partitions. Global indexes are skipped. If you run the `update index statistics` for a specific index partition, only the table statistics for that index partition are updated.
- The `with consumers` clause is designed for use on partitioned tables on RAID devices, which appear to Adaptive Server as a single I/O device, but which are capable of producing the high throughput required for parallel sorting. For more information, see Chapter 24, “Parallel Sorting,” in the *Performance and Tuning Guide*.
- The `update index statistics` command generates a series of update statistics operations that use the same locking, scanning, and sorting as the equivalent index-level and column-level command. For example, if the salesdetail table has a nonclustered index named `sales_det_ix` on `salesdetail (stor_id, ord_num, title_id)`, the `update index statistics salesdetail` command performs these update statistics operations:
  ```sql
  update statistics salesdetail sales_det_ix
  update statistics salesdetail (ord_num)
  update statistics salesdetail (title_id)
  ```

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`update index statistics` permission defaults to the table owner and is not transferable. The command can also be executed by the Database Owner, who can impersonate the table owner by running the `setuser` command.

**See also**

Commands: `delete statistics`, `update all statistics`, `update statistics`, `update table statistics`

**Documentation**

*Performance and Tuning Guide*
update statistics

Description
Updates information about the distribution of key values in specified indexes, for all columns in an index, table, or partition.

Syntax
update statistics table_name
   [([partition data_partition_name] (column_list)) | index_name [partition index_partition_name]]
   [using step values]
   [with consumers = consumers][, sampling=N percent]

Parameters

- **table_name**
  when used with update statistics, table_name is the name of the table with which the index is associated. table_name is required, since Transact-SQL does not require index names to be unique in a database.

- **index_name**
  is the name of the index to be updated. If an index name is not specified, the distribution statistics for all the indexes in the specified table are updated.

- **data_partition_name**
  is the name of the partition to be updated. Statistics for each local index partition on the data partition is updated. Does not update statistics for global indexes.

- **index_partition_name**
  is the name of the index partition to be updated.

- **column_list**
  is a comma-separated list of columns.
using *step values*

specifies the number of histogram steps. The default value is 20, for columns where no statistics exist. If you need to change the default for this, use *sp_configure* to modify the *number of histogram steps* parameter. If statistics for a column already exist in *sysstatistics*, the default value is the current number of steps.

The steps are applied to each partition of a partitioned table—for example, *update statistics* uses the default value of 20 steps for each data and index partition involved in the scan for updating statistics. If global statistics are generated through an index scan of a global index, then 20 steps are applied by default. If partition statistics are generated, either through a data scan or local index scan, then 20 steps are applied by default for each partition.

If the histogram steps specified through *using step values* is *M*, and the *histogram tuning factor* parameter is *N*, then *update statistics* uses between *M* and *M*\(^*N*) steps, depending on the number of frequency cells that *update statistics* isolates.

with *consumers = consumers*

specifies the number of consumer processes to be used for a sort when *column list* is provided and parallel query processing is enabled. The *consumers* option specifies the degree of parallelism applied to a sort performed for statistics update on a single data partition. For example, if *update statistics* with a column list is applied to a table with three data partitions, data from each of the three partitions is sorted separately and the *consumers* option is applied during each of the sort. The three sorts themselves are not performed in parallel.

with *sampling = N percent*

specifies the percentage of the column to be randomly sampled in order to gather statistics. The value for *N* is any number between 1 and 100. Sampling applies to all *update statistics* types:

- update statistics *table_name*
- update statistics *table_name* (*col_name*)
- update index statistics
- update all statistics

*index*

specifies that statistics for all columns in an index are to be updated.

**Examples**

**Example 1** Generates statistics for the *price* column of the *titles* table:

```
update statistics titles (price) using 40 values
```
Example 2 Updates statistics on the data partition smallsales. Adaptive Server creates histograms for each leading column and densities for the composite columns of each local index of the data partition. Statistics are not updated for global indexes:

```
update statistics titles partition smallsales
```

Example 3 Updates statistics on the data partition smallsales. Adaptive Server creates histograms on column col1 and creates densities for the composite columns col1 and col2:

```
update statistics titles partition smallsales (col1, col2)
```

Usage

- Adaptive Server keeps statistics about the distribution of the key values in each index, and uses these statistics in its decisions about which index(es) to use in query processing.
- When you create a nonclustered index on a table that contains data, `update statistics` is automatically run for the new index. When you create a clustered index on a table that contains data, `update statistics` is automatically run for all indexes.
- The optimization of your queries depends on the accuracy of the statistics. If there is significant change in the key values in your index, you should rerun `update statistics` on that index or column. Use the `update statistics` command if a great deal of data in an indexed column has been added, changed, or removed (that is, if you suspect that the distribution of key values has changed).
- You should also run `update statistics` on system tables with a large number of rows. If you have permission to run the command on a user table, it is no different with respect to system table. Without statistics, there is always a chance for system stored procedures to perform poorly.
- `update statistics` skips global indexes when you run the command on a data partition.
- `update statistics`, when used with a table name and an index name, updates statistics for the leading column of an index. If `update statistics` is used with just a table name, it updates statistics for the leading columns of all indexes on the table.
- Specifying the name of an unindexed column or the nonleading column of an index generates statistics for that column without creating an index.
update statistics

- Specifying more than one column in a column list generates or updates a histogram for the first column, and density statistics for all prefix subsets of the list of columns.

- If you use update statistics to generate statistics for a column or list of columns, update statistics must scan the table and perform a sort.

- If you use update statistics on a specific partition, you update global statistics implicitly as well.

- update statistics regenerates and updates the table statistics stored in systabstats for each data and index partition of the table the command updates. If you run the update statistics command for a specific data partition, the table statistics are generated and updated only for that data partition and for any local index partitions. Global indexes are skipped. If you run the update statistics for a specific index partition, only the table statistics for that index partition are updated.

- The with consumers clause is designed for use on partitioned tables on RAID devices, which appear to Adaptive Server as a single I/O device, but which are capable of producing the high throughput required for parallel sorting. For more information, see Chapter 24, “Parallel Sorting,” in the Performance and Tuning Guide.

- Table 1-43 shows the types of scans performed during update statistics, the types of locks acquired, and when sorts are needed.

<table>
<thead>
<tr>
<th>Table name specifying</th>
<th>Scans and sorts performed</th>
<th>Locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allpages-locked table</td>
<td>Table scan, plus a leaf-level scan of each nonclustered index</td>
<td>Level 1; shared intent table lock, shared lock on current page</td>
</tr>
<tr>
<td>Data-only-locked table</td>
<td>Table scan, plus a leaf-level scan of each nonclustered index and the clustered index, if one exists</td>
<td>Level 0; dirty reads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table name and clustered index name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Allpages-locked table</td>
<td>Table scan</td>
<td>Level 1; shared intent table lock, shared lock on current page</td>
</tr>
<tr>
<td>Data-only-locked table</td>
<td>Leaf level index scan</td>
<td>Level 0; dirty reads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table name and nonclustered index name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Allpages-locked table</td>
<td>Leaf level index scan</td>
<td>Level 1; shared intent table lock, shared lock on current page</td>
</tr>
<tr>
<td>Data-only-locked table</td>
<td>Leaf level index scan</td>
<td>Level 0; dirty reads</td>
</tr>
</tbody>
</table>
The update index statistics command generates a series of update statistics operations that use the same locking, scanning, and sorting as the equivalent index-level and column-level command. For example, if the salesdetail table has a nonclustered index named sales_det_ix on salesdetail (stor_id, ord_num, title_id), the update index statistics salesdetail command performs these update statistics operations:

```
update statistics salesdetail sales_det_ix
update statistics salesdetail (ord_num)
update statistics salesdetail (title_id)
```

- update statistics is not run on system tables in the master database during upgrade from earlier versions. Indexes exist on columns queried by most system procedures, and running update statistics on these tables is not required for normal usage. However, running update statistics is allowed on all system tables in all databases, except those that are not normal tables. These tables, which are built from internal structures when queried, include syscurconfigs, sysengines, sysgams, syslisteners, syslocks, syslogs, syslogshold, syslogshold, sysmonitors, sysprocesses, syssecmechs, systestlog and systransactions.

You do not need to run update statistics on Replication Server RSSD tables. Running updates statistics on these tables can result in Replication Server errors if you run it while Replication Server attempts to access the RSSD tables. RSSD tables and their format are specific to Replication Server processing.

**update statistics and sampling**

Sampling is not performed for leading columns of indexes. If you specify an index in update statistics, such as in the following:

```
update statistics table_name [index_name] with sampling = N percent
```

This command creates and updates statistics on the leading column of all indexes on the specified table, or the leading column of a specified index.

When you use the `sampling = N percent` option with the `using steps value`, you must specify the `sampling = N percent` option last:

```
update statistics titles (type)
```
If you do not, you get an error message:

```sql
update statistics titles (type)
with sampling = 10 percent
using 40 value
```

Msg 156, Level 15, State 2:
Line 1:
Incorrect syntax near the keyword 'using'.

create index and stored procedures
Adaptive Server automatically recompiles stored procedures after executing `update statistics` statements. Although ad hoc queries that you start before executing `update statistics` still continue to work, they do not take advantage of the new statistics.

In Adaptive Server versions 12.5 and earlier, `update statistics` was ignored by cached stored procedures.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
`update statistics` permission defaults to the table owner and is not transferable. The command can also be executed by the Database Owner, who can impersonate the table owner by running the `setuser` command.

See also
- `Commands` delete statistics, update all statistics, update index statistics, update table statistics

Documentation  Performance and Tuning Guide
update table statistics

**Description**
update table statistics updates statistics that are stored in systabstats table, such as rowcount, cluster ratios, and so on. update table statistics does not affect column statistics stored in sysstatistics.

**Syntax**
```
update table statistics table_name
    [partition data_partition_name]
    [index_name [partition index_partition_name]]
```

**Parameters**
- `table_name` is the name of the table you are updating the statistics for.
- `data_partition_name` is the name of the data partition for which you are updating the statistics for. If you do not include this, table statistics for all the data partitions are updated.
- `index_name` is the name of index associated with the partition.
- `index_partition_name` is the name of the index partition.

**Examples**
- **Example 1** Performs a table statistics update on the smallsales partition:
  
  `update table statistics titles partition smallsales`

- **Example 2** Performs a table statistics update on all of the partitions in the titles table:
  
  `update table statistics titles`

**Usage**
- update table statistics does not update statistics for index partitions. To generate table-level statistics for index partitions, use update statistics.
- Because running update table statistics incurs the I/O cost of running update statistics, use update statistics to generate both column and table statistics.

You can create, and then drop, a global index to generate global statistics.

When you run update statistics on a single partition, you create global statistics by merging partition statistics. However, these merged global statistics are less accurate than the global statistics created as a side-effect of global index creation. Avoid generating column statistics that overwrite more accurate, earlier versions of column statistics.

When you specify:
update table statistics

- **index_name** – *update table statistics* updates statistics for all the index partitions of the index.
- **index_partition** – *update table statistics* updates statistics for the specific index partition.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

*update table statistics* permission defaults to the table owner and is not transferable. The command can also be executed by the Database Owner, who can impersonate the table owner by running the `setuser` command.

**See also**

Commands: update all statistics, update index statistics, update statistics

Documentation: *Performance and Tuning Guide*
**use**

**Description**
Specifies the database with which you want to work.

**Syntax**
use *database_name*

**Parameters**
- *database_name*
  is the name of the database to open.

**Examples**
```
use pubs2
go
```

The current database is now pubs2.

**Usage**
- Allowed with an archive database.
- The `use` command must be executed before you can reference objects in a database.
- `use` cannot be included in a stored procedure or a trigger.
- `sp_addalias` adds an alias, which permits a user to use a database under another name to gain access to that database.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
If the database has a “guest” account, all users can use the database. If the database does not have a “guest” account, you must be a valid user in the database, have an alias in the database, or be a System Administrator or System Security Officer.

**See also**
- **Commands** create database, drop database
- **System procedures** sp_addalias, sp_adduser, sp_modifylogin
waitfor

Description
Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.

Syntax
waitfor {delay time | time time | errorexit | processexit | mirrorexit}

Parameters

\textit{delay}

- instructs Adaptive Server to wait until the specified amount of time has passed, up to a maximum of 24 hours.

\textit{time}

- a time in one of the acceptable formats for date/time data, or a variable of character type. You cannot specify dates—the date portion of the date/time value is not allowed. You can use the datatype \textit{time} for this information.

\textit{errorexit}

- instructs Adaptive Server to wait until a kernel or user process terminates abnormally.

\textit{processexit}

- instructs Adaptive Server to wait until a kernel or user process terminates for any reason.

\textit{mirrorexit}

- instructs Adaptive Server to wait for a mirror failure.

Examples

**Example 1** At 2:20 p.m., the chess table is updated with my next move, and a procedure called sendmail inserts a row in a table owned by Judy, notifying her that a new move now exists in the chess table:

\begin{verbatim}
begin
  waitfor time "14:20"
  insert chess (next_move)
  values ('Q-KR5')
  execute sendmail 'judy'
end
\end{verbatim}

**Example 2** After 10 seconds, Adaptive Server prints the message specified:

\begin{verbatim}
declare @var char (8)
select @var = "00:00:10"
begin
  waitfor delay @var
  print "Ten seconds have passed. Your time is up."
\end{verbatim}
Example 3 After any process exits abnormally, Adaptive Server prints the message specified:

```sql
begin
    waitfor errorexit
    print "Process exited abnormally!"
end
```

Usage

- After issuing the `waitfor` command, you cannot use your connection to Adaptive Server until the time or event that you specified occurs.
- You can use `waitfor errorexit` with a procedure that kills the abnormally terminated process, to free system resources that would otherwise be taken up by an infected process.
- To find out which process terminated, check the `sysprocesses` table with `sp_who`.
- The time you specify with `waitfor time` or `waitfor delay` can include hours, minutes, and seconds. Use the format “hh:mm:ss”, as described in “Date and time datatypes” on page 20 in Chapter 1, “System and User-Defined Datatypes” of Reference Manual: Building Blocks.

The following example instructs Adaptive Server to wait until 4:23 p.m:

```sql
waitfor time "16:23"
```

This statement instructs Adaptive Server to wait for 1 hour and 30 minutes:

```sql
waitfor delay "01:30"
```

- Changes in system time (such as setting the clock back for Daylight Savings Time) can delay the `waitfor` command.
- You can use `waitfor errorexit` within a DB-Library program to notify users when there is a mirror failure.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

`waitfor` permission defaults to all users. No permission is required to use it.

See also

- **Commands** `begin...end`
- **Datatypes** Date and time datatypes
- **System procedures** `sp_who`
where clause

Description
Sets the search conditions in a select, insert, update, or delete statement.

Syntax
Search conditions immediately follow the keyword where in a select, insert, update, or delete statement. If you use more than one search condition in a single statement, connect the conditions with and or or.

where [not] expression comparison_operator expression
where [(not] expression comparison_operator expression) | {...}
where [not] expression [not] like "match_string"
   [escape "escape_character"]
where [not] expression is [not] null
where [not] expression [not] between expression and expression
where [not] expression [not] in ((value_list | subquery))
where [not] exists (subquery)
where [not] expression comparison_operator (any | all) (subquery)
where [not] column_name join_operator column_name
where [not] logical_expression
where [not] expression (and | or) [not] expression

Parameters

expression
is a column name, a constant, a function, a subquery, or any combination of
column names, constants, and functions connected by arithmetic or bitwise
operators. For more information about expressions, see “Expressions” on
page 335 in Chapter 4, “Expressions, Identifiers, and Wildcard Characters”
comparison_operator

is one of the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Not greater than</td>
</tr>
<tr>
<td>!&lt;</td>
<td>Not less than</td>
</tr>
</tbody>
</table>

In comparing char, nchar, unichar, varchar, univarchar, and nvarchar data, < means closer to the beginning of the alphabet and > means closer to the end of the alphabet.

Case and special character evaluations depend on the collating sequence of the operating system on the machine on which Adaptive Server is located. For example, lowercase letters may be greater than uppercase letters, and uppercase letters may be greater than numbers.

Trailing blanks are ignored for the purposes of comparison. For example, “Dirk” is the same as “Dirk ”.

In comparing dates, < means earlier and > means later. Put quotes around all character and date data used with a comparison operator. For example:

```
  = "Bennet"
  > "94609"
```

like

is a keyword indicating that the following character string (enclosed by single or double quotes) is a matching pattern. like is available for char, varchar, unichar, univarchar, nchar, nvarchar, datetime, date and time, text, and untext columns, but not to search for seconds or milliseconds.

You can use the keyword like and wildcard characters with datetime and date data as well as with char and varchar. When you use like with datetime or date and time values, Adaptive Server converts the dates to standard datetime format, then to varchar. Since the standard storage format does not include seconds or milliseconds, you cannot search for seconds or milliseconds with like and a pattern.

It is a good idea to use like when you search for date/time values, since date/time entries may contain a variety of date parts. For example, if you insert the value “9:20” into a column named arrival_time, the following clause would not find it because Adaptive Server converts the entry into “Jan 1, 1900 9:20AM.”:

```sql
where arrival_time = '9:20'
```

However, the following clause would find it:

```sql
where arrival_time like '%9:20%'
```

match_string

is a string of characters and wildcard characters enclosed in quotes. Table 1-44 lists the wildcard characters.

**Table 1-44: Wildcard characters**

<table>
<thead>
<tr>
<th>Wildcard character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Any string of 0 or more characters</td>
</tr>
<tr>
<td>_</td>
<td>Any single character</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character within the specified range ([a-f]) or set ([abcdef])</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character that is not within the specified range ([^a-f]) or set ([^abcdef])</td>
</tr>
</tbody>
</table>

escape

specifies an escape character with which you can search for literal occurrences of wildcard characters.

**escape_character**

is any single character. For more information, see “Using the escape clause” on page 358 in Chapter 4, “Expressions, Identifiers, and Wildcard Characters” of Reference Manual: Building Blocks.

is null

searches for null values.
between
is the range-start keyword. Use and for the range-end value. The following range is inclusive:

\[ \text{where} \ \text{@val between} \ x \ \text{and} \ y \]

The following range is not:

\[ x \ \text{and} \ \text{@val} < y \]

Queries using between return no rows if the first value specified is greater than the second value.

and
joins two conditions and returns results when both of the conditions are true.

When more than one logical operator is used in a statement, and operators are usually evaluated first. However, you can change the order of execution with parentheses.

in
allows you to select values that match any one of a list of values. The comparator can be a constant or a column name, and the list can be a set of constants or, more commonly, a subquery. For information on using in with a subquery, see the *Transact-SQL User’s Guide*. Enclose the list of values in parentheses.

value_list
is a list of values. Put single or double quotes around character values, and separate each value from the following one with a comma (see example 7). The list can be a list of variables, for example:

in \((\text{@a, @b, @c})\)

However, you cannot use a variable containing a list, such as the following, for a values list:

\[ \text{@a} = "'1', '2', '3'" \]

exists
is used with a subquery to test for the existence of some result from the subquery. For more information, see the *Transact-SQL User’s Guide*.

subquery
is a restricted select statement (order by and compute clauses and the keyword into are not allowed) inside the where or having clause of a select, insert, delete, or update statement, or a subquery. For more information, see the *Transact-SQL User’s Guide*. 
any
is used with >, <, or = and a subquery. It returns results when any value
retrieved in the subquery matches the value in the where or having clause of
the outer statement. For more information, see the Transact-SQL User's
Guide.

all
is used with > or < and a subquery. It returns results when all values retrieved
in the subquery match the value in the where or having clause of the outer
statement. For more information, see the Transact-SQL User's Guide.

column_name
is the name of the column used in the comparison. Qualify the column name
with its table or view name if there is any ambiguity. For columns with the
IDENTITY property, you can specify the syb_identity keyword, qualified by
a table name where necessary, rather than the actual column name.

join_operator
is a comparison operator or one of the join operators =* or *=. For more
information, see the Transact-SQL User's Guide.

logical_expression
is an expression that returns TRUE or FALSE.

or
joins two conditions and returns results when either of the conditions is true.
When more than one logical operator is used in a statement, or operators are
normally evaluated after and operators. However, you can change the order
of execution with parentheses.

Examples

Example 1

where advance * $2 > total_sales * price

Example 2 Finds all the rows in which the phone number does not begin with
415:

where phone not like '415%'

Example 3 Finds the rows for authors named Carson, Carsen, Karsen, and
Karson:

where au_lname like "%[CK]ars[eo]n"

Example 4 Finds the row of the sales_east table in which the IDENTITY
column has a value of 4:

where sales_east.syb_identity = 4
CHAPTER 1 Commands

Example 5

where advance < $5000 or advance is null

Example 6

where (type = "business" or type = "psychology") and advance > $5500

Example 7

where total_sales between 4095 and 12000

Example 8 Finds the rows in which the state is one of the three in the list:

where state in ('CA', 'IN', 'MD')

Usage

• where and having search conditions are identical, except that aggregate functions are not permitted in where clauses. For example, this clause is legal:

  having avg (price) > 20

This clause is not legal:

  where avg (price) > 20

For examples, see Chapter 2, “Transact-SQL Functions” in Reference Manual: Building Blocks for information on the use of aggregate functions, and group by and having clauses on page 390.

• Joins and subqueries are specified in the search conditions: see the Transact-SQL User's Guide for full details.

• You can use the keyword like to search a unitext column for a specific pattern. However, the like clause is not optimized when it is used with a unitext column. like pattern matching for unitext depends on the default Unicode sort order, which is also used for like pattern matching for unichar and univarchar datatypes.

• The number of and and or conditions in a where clause is limited only by the amount of memory available to run the query.

• The pattern string included in the like predicate is limited only by the size of string that can be placed in a varchar.

• There are two ways to specify literal quotes within a char or varchar entry. The first method is to use two quotes. For example, if you began a character entry with a single quote, and you want to include a single quote as part of the entry, use two single quotes:

  'I don't understand.'

Or use double quotes:
"He said, "It's not really confusing.""

The second method is to enclose a quote in the opposite kind of quotation mark. In other words, surround an entry containing double quotes with single quotes (or vice versa). Here are some examples:

'I George said, "There must be a better way."'
"Isn't there a better way?"
'George asked, "Isn't there a better way?"

- To enter a character string that is longer than the width of your screen, enter a backslash (\) before going to the next line.

- If a column is compared to a constant or variable in a where clause, Adaptive Server converts the constant or variable into the datatype of the column so that the optimizer can use the index for data retrieval. For example, float expressions are converted to int when compared to an int column. For example:

  where int_column = 2

  selects rows where int_column = 2.

- When Adaptive Server optimizes queries, it evaluates the search conditions in where and having clauses, and determines which conditions are search arguments (SARGs) that can be used to choose the best indexes and query plan. All of the search conditions are used to qualify the rows. For more information on search arguments, see the Performance and Tuning Guide.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

See also

Commands delete, execute, group by and having clauses, insert, select, update

Datatypes Date and time datatypes

System procedures sp_helpjoins
while

Description
Sets a condition for the repeated execution of a statement or statement block. The statements are executed repeatedly, as long as the specified condition is true.

Syntax
while logical_expression [plan "abstract plan"] statement

Parameters
logical_expression
is any expression that returns TRUE, FALSE, or NULL.

plan "abstract plan"
specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for optimizable SQL statements, that is, queries that access tables. See Chapter 30, “Creating and Using Abstract Plans,” in the Performance and Tuning Guide for more information.

statement
can be a single SQL statement, but is usually a block of SQL statements delimited by begin and end.

Examples
If the average price is less than $30, double the prices of all books in the titles table. As long as it is still less than $30, the while loop keeps doubling the prices. In addition to determining the titles whose price exceeds $20, the select inside the while loop indicates how many loops were completed (each average result returned by Adaptive Server indicates one loop):

```sql
while (select avg (price) from titles) < $30
begin
    select title_id, price
    from titles
    where price > $20
    update titles
    set price = price * 2
end
```

Usage
- The execution of statements in the while loop can be controlled from inside the loop with the break and continue commands.
- The continue command causes the while loop to restart, skipping any statements after the continue. The break command causes an exit from the while loop. Any statements that appear after the keyword end, which marks the end of the loop, are executed. The break and continue commands are often activated by if tests.

For example:

```sql
while (select avg (price) from titles) < $30
```
begin
    update titles
    set price = price * 2
    if (select max (price) from titles) > $50
        break
    else
        if (select avg (price) from titles) > $30
            continue
        print "Average price still under $30"
end

select title_id, price from titles
where price > $30

This batch continues to double the prices of all books in the titles table as long as the average book price is less than $30. However, if any book price exceeds $50, the break command stops the while loop. The continue command prevents the print statement from executing if the average exceeds $30. Regardless of how the while loop terminates (either normally or because of the break command), the last query indicates which books are priced over $30.

- If two or more while loops are nested, the break command exits to the next outermost loop. All the statements after the end of the inner loop run, then the next outermost loop restarts.

Warning! If a create table or create view command occurs within a while loop, Adaptive Server creates the schema for the table or view before determining whether the condition is true. This may lead to errors if the table or view already exists.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
while permission defaults to all users. No permission is required to use it.

See also
Commandsthbegin...end, break, continue, goto label
writetext

Description
Permits minimally logged, interactive updating of an existing text, unitext or image column.

Syntax
writetext [database.]owner.table_name.column_name
text_pointer [readpast] [with log] data

Parameters
table_name.column_name
is the name of the table and text, unitext or image column to update. Specify the database name if the table is in another database, and specify the owner’s name if more than one table of that name exists in the database. The default value for owner is the current user, and the default value for database is the current database.

text_pointer
a varbinary (16) value that stores the pointer to the text, unitext or image data. Use the textptr function to determine this value. text, unitext or image data is not stored in the same set of linked pages as other table columns. It is stored in a separate set of linked pages. A pointer to the actual location is stored with the data; textptr returns this pointer.

readpast
specifies that the command should modify only unlocked rows. If the writetext command finds locked rows, it skips them, rather than waiting for the locks to be released.

with log
logs the inserted text, unitext or image data. The use of this option aids media recovery, but logging large blocks of data quickly increases the size of the transaction log, so make sure that the transaction log resides on a separate database device. See create database, sp_logdevice, and the System Administration Guide for details.

data
is the data to write into the text, unitext or image column. text and unitext data must be enclosed in quotes. image data must be preceded by “0x”. Check the information about the client software you are using to determine the maximum length of text, unitext or image data that can be accommodated by the client.

Examples

Example 1 Puts the text pointer into the local variable @val. Then, writetext places the text string “hello world” into the text field pointed to by @val:

declare @val varbinary (16)
select @val = textptr (copy) from blurbs
where au_id = "409-56-7008"
writetext blurbs.copy @val with log "hello world"

Example 2

```
declare @val varbinary (16)
select @val = textptr (copy)
from blurbs readpast
    where au_id = "409-56-7008"
writetext blurbs.copy @val readpast with log "hello world"
```

Example 3 writetext includes information about unitext datatypes, and places the string “Hello world” into the unitext field that @val points to:

```
declare @val varbinary (16)
select @val = textptr (ut) from unitable
    where i = 100
writetext unitable.ut @val with log "Hello world"
```

The varchar constant is implicitly converted to unitext before the column is updated.

Usage

- The maximum length of text that can be inserted interactively with writetext is approximately 120K bytes for text, unitext or image data.
- By default, writetext is a minimally logged operation; only page allocations and deallocations are logged, but the text, unitext or image data is not logged when it is written into the database. To use writetext in its default, minimally logged state, a System Administrator must use sp_dboption to set select into/bulkcopy/pllsort to true.
- writetext updates text data in an existing row. The update completely replaces all of the existing text.
- writetext operations are not caught by an insert or update trigger.
- writetext requires a valid text pointer to the text, unitext or image column. For a valid text pointer to exist, a text, or unitext column must contain either actual data or a null value that has been explicitly entered with update.

Given the table textnull with columns textid and x, where x is a text column that permits nulls, this update sets all the text values to NULL and assigns a valid text pointer in the text column:

```
update textnull
set x = null
```

No text pointer results from an insert of an explicit null:

```
insert textnull values (2,null)
```
And, no text pointer results from an insert of an implicit null:

\[
\text{insert textnull (textid)} \\
\text{values (2)}
\]

- `insert` and `update` on `text` columns are logged operations.
- You cannot use `writetext` on `text` and `image` columns in views.
- If you attempt to use `writetext` on `text` values after changing to a multibyte character set, and you have not run `dbcc fix_text`, the command fails, and an error message is generated, instructing you to run `dbcc fix_text` on the table.
- `writetext` in its default, non-logged mode runs more slowly while a `dump database` is taking place.
- The Client-Library functions `dbwritetext` and `dbmoretext` are faster and use less dynamic memory than `writetext`. These functions can insert up to 2GB of text data.

Using the `readpast` option

- The `readpast` option applies only to data-only-locked tables. `readpast` is ignored if it is specified for an allpages-locked table.
- If the session-wide isolation level is 3, the `readpast` option is silently ignored.
- If the transaction isolation level for a session is 0, `writetext` commands using `readpast` do not issue warning messages. These commands at session isolation level 0 modify the specified text column if the text column is not locked with incompatible locks.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

`writetext` permission defaults to the table owner, who can transfer it to other users.

**See also**

- [Commands](#)
- [readtext](#)
- [Datatypes](#)
- Converting text and image datatypes
CHAPTER 2

Interactive SQL Commands

This chapter describes Interactive SQL commands. These commands are entered in top pane of the Interactive SQL display. These commands are intended only for Interactive SQL and are not sent to Adaptive Server for execution. For information about Interactive SQL, see Chapter 9, “Using Interactive SQL” in the Utility Guide, and the Adaptive Server Plug-in online help.

Table 2-1: DBISQL commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear on page 644</td>
<td>Clears the Interactive SQL panes.</td>
</tr>
<tr>
<td>configure on page 645</td>
<td>Opens the Interactive SQL Options dialog.</td>
</tr>
<tr>
<td>connect on page 646</td>
<td>Establishes a connection to a database.</td>
</tr>
<tr>
<td>disconnect on page 649</td>
<td>Drops the current connection to a database.</td>
</tr>
<tr>
<td>exit on page 650</td>
<td>Leaves Interactive SQL.</td>
</tr>
<tr>
<td>input on page 651</td>
<td>Imports data into a database table from an external file or from the keyboard.</td>
</tr>
<tr>
<td>output on page 656</td>
<td>Imports data into a database table from an external file or from the keyboard.</td>
</tr>
<tr>
<td>parameters on page 661</td>
<td>Specifies parameters to an Interactive SQL command file.</td>
</tr>
<tr>
<td>read on page 662</td>
<td>Reads Interactive SQL statements from a file.</td>
</tr>
<tr>
<td>set connection on page 664</td>
<td>Changes the current database connection to another server.</td>
</tr>
<tr>
<td>set option on page 665</td>
<td>Use this statement to change the values of Interactive SQL options.</td>
</tr>
<tr>
<td>start logging on page 666</td>
<td>Use this statement to start logging executed SQL statements to a log file.</td>
</tr>
<tr>
<td>stop logging on page 667</td>
<td>Use this statement to stop logging of SQL statements in the current session.</td>
</tr>
<tr>
<td>system on page 668</td>
<td>Use this statement to launch an executable file from within Interactive SQL.</td>
</tr>
</tbody>
</table>
clear

Description
Clears the Interactive SQL panes.

Syntax
clear

Usage
- Use the clear statement to clear the SQL Statements and Messages panes and the Results, Messages, Plan, and Plan tabs in the Results pane.
- clear closes the cursor associated with the data being cleared.

Permissions
Any user can execute this command.
configure

Description
Opens the Interactive SQL Options dialog.

Syntax
configure

Usage
- The `configure` statement opens the Interactive SQL Options dialog and displays the current settings of all Interactive SQL options. It does not display or allow you to modify database options.

- You can configure Interactive SQL settings in this dialog. If you select Make Permanent, the options are saved for use in subsequent Interactive SQL sessions. If you do not choose Make Permanent, and instead click OK, the options are set temporarily and remain in effect for the current database connection only.

Permissions
Any user can run `configure`.

See also
set
connect

Description
Establishes a connection to a database.

Syntax
```sql
connect
    [to engine_name]
    [database database_name]
    [as connection_name]
    [user] user_id identified by password
    engine_name, database_name, connection_name, user_id,
         password : {identifier | string | hostvar}

connect using connect_string : {identifier | string | hostvar}
```

Parameters
- `engine_name` is the name of the engine to which you are connecting.
- `database_name` is the name of the database to which you are connecting. It must conform to the rules for identifiers and cannot be a variable.
- `as` allows multiple connections to the same database, or multiple connections to the same or different database servers, all simultaneously. Each connection has its own associated transaction. You may get locking conflicts between your transactions if, for example, you modify the same record in the same database from two different connections.
- `connection_name` is the login name you are using to make the connection.
- `user` indicates that you are connecting to Adaptive Server as a user.
- `user_id` is the ID of the user who is connecting.
- `identified by password` indicates that the user will need to include a password when they connect.
- `password` is the password of the user connecting to Adaptive Server.
- `identifier` is the identifier you are using for the connection information.
- `string` is the string you are using for the connection information.
hostvar

is the variable information for the host name and port.

connect_string

is a list of parameter settings of the form keyword = value, separated by
semicolons, and must be enclosed in single quotes.

Examples

Example 1 Connects to a database from Interactive SQL. Interactive SQL
prompts for a user ID and a password:

connect

Example 2 Connects to the default database as DBA from Interactive SQL.
Interactive SQL prompts for a password:

connect user "DBA"

Example 3 As user dba, with password sql1, connects to the pubs2 database
of an Adaptive Server running on host “tribble” at port number 5000:

connect to "tribble:5000"
database pubs2
user dba
identified by sql

Example 4 As user dba, with password sql1, connects to an Adaptive Server
named “tribble” (defined in interfaces file):

connect to tribble
user dba
identified by sql

Usage

• connect establishes a connection to the database identified by
database_name running on the server identified by engine_name.

• No statements are allowed until a successful connect statement has been
executed.

• Interactive SQL behavior – if you do not specify a database or server in the
connect statement, Interactive SQL remains connected to the current
database, rather than to the default server and database. If you do specify
a database name without a server name, Interactive SQL attempts to
connect to the specified database on the current server. If you specify a
server name without a database name, Interactive SQL connects to the
default database on the specified server.

• In the user interface, if the password or the user ID and password are not
specified, the user is prompted to type the missing information.
When Interactive SQL is running in command-prompt mode (-noGui is specified when you start Interactive SQL from a command prompt) or batch mode, or if you execute connect without an as clause, an unnamed connection is opened. If there is another unnamed connection already opened, the old one is automatically closed. Otherwise, existing connections are not closed when you run connect.

Multiple connections are managed through the concept of a current connection. After a successful connect statement, the new connection becomes the current one. To switch to a different connection, use the set connection statement. Use the disconnect statement to drop connections.

In Interactive SQL, the connection information (including the database name, your user ID, and the database server) appears in the title bar above the SQL Statements pane. If you are not connected to a database, Not Connected appears in the title bar.

Permissions
Any user can execute this command.

See also
disconnect, set connection
**disconnect**

**Description**
Drops the current connection to a database.

**Syntax**
```
disconnect [{identifier | string | hostvar} | current | all]
```

**Parameters**
- `{identifier | string | hostvar}` is the login name you are using to make the connection.
- `identifier` – is the identifier you are using for the connection information.
- `string` – is the string you are using for the connection information.
- `hostvar` – is the variable information for the host name and port.
- `current` indicates that you are disconnecting the current connection.
- `all` indicates that you are disconnecting all connections.

**Examples**
Disconnects all connections:
```
disconnect all
```

**Usage**
- `disconnect` drops a connection to the database server and releases all resources used by it. If the connection to be dropped was named on the `connect` statement, the name can be specified. Specifying `all` drops all of the application’s connections to all database environments. `current` is the default, and drops the current connection.
- An implicit rollback is executed on connections that are dropped.

**Permissions**
Any user can execute this command.

**See also**
`connect`, `set connection`
**exit**

**Description**  
Leaves Interactive SQL.

**Syntax**  
```
{exit | quit | bye} [{number | connection_variable}]
```

**Parameters**  
```
exit | quit | bye
```

closes your connection with the database, then closes the Interactive SQL environment.

{(number | connection_variable)}

can be used in batch files to indicate success or failure of the commands in an Interactive SQL command file. The default return code is 0.

- `number` – is the number of the return code.
- `connection_variable` – is a variable indicating a specific connection.

**Usage**  
Before closing the database connection, Interactive SQL automatically executes a commit statement if the `commit_on_exit` option is set to on. If this option is set to off, Interactive SQL performs an implicit rollback. By default, the `commit_on_exit` option is set to on.

**Permissions**  
Any user can execute this command.
**input**

**Description**
Imports data into a database table from an external file or from the keyboard.

**Syntax**

```
input into [ owner.]table_name
    [ from filename | prompt]
    [ format { ascii | dbase | dbasell | dbaselll | excel | fixed | foxpro | lotus }]
    [ escape character character ]
    [ escapes { on | off }]
    [ by order | by name ]
    [ delimited by string ]
    [ column widths (integer , . . . ) ]
    [ nostrip ]
    [ ( column_name, . . . ) ]
    [ encoding { identifier | string }]
```

**Parameters from clause**

- is the file name that is passed to the server as a quoted string. The string is therefore subject to the same formatting requirements as other SQL strings. In particular:
  - To indicate directory paths, you must represent the backslash character (\) by two backslashes. To load data from the file `c: emp\input.dat` into the employee table:
    ```sql
    input into employee
        from 'c: nn temp nn input.dat'
    ```
  - The path name is relative to the machine on which Interactive SQL is running.

- `prompt` allows the user to enter values for each column in a row. When running in windowed mode, a dialog appears where the user can enter the values for the new row. If the user is running Interactive SQL on the command line, Interactive SQL prompts the user to type the value for each column on the command line.
format

each set of values must be in the format specified by the format clause, or the
set option input_format statement if the format clause is not specified. When
input is entered by the user, a dialog is provided for the user to enter one row
per line in the input format.

Certain file formats contain information about column names and types.

Using this information, the input statement creates the database table if it
does not already exist. This is a very easy way to load data into the database.
The formats that have enough information to create the table are: dbasell,
dbaselll, foxpro, and lotus.

Input from a command file is terminated by a line containing end. Input from
a file is terminated at the end of the file.

Allowable input formats are:

• ascii – input lines are assumed to be ASCII characters, one row per line,
  with values separated by commas. Alphabetic strings may be enclosed
  in apostrophes (single quotes) or quotation marks (double quotes).
  Strings containing commas must be enclosed in either single or double
  quotes. If the string itself contains single or double quotes, double the
  quote character to use it within the string. Optionally, you can use the
delimited by clause to specify a delimiter string other than the default,
  which is a comma.

Three other special sequences are also recognized. The two characters
represent a new line character, “\”, represents a single (\), and the
sequence \xDD represents the character with hexadecimal code DD.

• dbase – the file is in DBASEII or DBASEIII format. Interactive SQL
  will attempt to determine which format, based on information in the
  file. If the table does not exist, it is created.

• dbasell – the file is in DBASEII format. If the table does not exist, it is
  created.

• dbaselll – the file is in DBASEIII format. If the table does not exist, it is
  created.

• excel – input file is in the format of Microsoft Excel 2.1. If the table does
  not exist, it is created.

• fixed – input lines are in fixed format. Use the column widths clause to
  specify column widths. If they are not specified, column widths in the
  file must be the same as the maximum number of characters required by
  any value of the corresponding database column’s type.
You cannot use the fixed format with binary columns that contain embedded new line and End of File character sequences.

- **foxpro** – the file is in FoxPro format. If the table does not exist, it is created.
- **lotus** – the file is a Lotus WKS format worksheet. input assumes that the first row in the Lotus WKS format worksheet is column names. If the table does not exist, it is created. In this case, the types and sizes of the columns created may not be correct because the information in the file pertains to a cell, not to a column.

**escape character**

is the default escape character for hexadecimal codes and symbols is a backslash (\), so \x0A is the linefeed character, for example.

You can change the escape character using the escape character clause. For example, to use the exclamation mark as the escape character, enter:

```sql
... escape character '!'
```

Only one single-byte character can be used as an escape character.

**escapes**

with escapes enabled (the default), characters following the backslash character are recognized and interpreted as special characters by the database server. New line characters can be included as the combination \n, other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\) is interpreted as a single backslash. A backslash followed by any character other than n, x, X or ‘\’ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q.

**by**

allows the user to specify whether the columns from the input file should be matched up with the table columns based on their ordinal position in the lists (order, the default) or by their names (name). Not all input formats have column name information in the file. name is allowed only for those formats that do. They are the same formats that allow automatic table creation: dbasell, dbaselll, foxpro, and lotus.

**delimited**

allows you to specify a string to be used as the delimiter in ASCII input format.
column widths
can be specified for fixed format only; it specifies the widths of the columns
in the input file. If column widths is not specified, the widths are determined
by the database column types. Do not use this clause if you are inserting long
varchar or binary data in fixed format.

nostrip
normally, for ASCII input format, trailing blanks are stripped from unquoted
strings before the value is inserted. nostrip can be used to suppress trailing
blank stripping. Trailing blanks are not stripped from quoted strings,
regardless of whether the option is used. Leading blanks are stripped from
unquoted strings, regardless of the nostrip option setting.

If the ASCII file has entries such that a column appears to be null, it is
treated as NULL. If the column in that position cannot be NULL, a zero is
inserted in numeric columns, and an empty string in character columns.

encoding
allows you to specify the encoding that is used to read the file. encoding can
be used only with the ASCII format.

If encoding is not specified, Interactive SQL determines the code page that
is used to read the file as follows, where code page values occurring earlier
in the list take precedence over those occurring later in the list:

- The code page specified with the default_isql_encoding option (if this
  option is set)
- The code page specified with the -codepage option when Interactive
  SQL was started
- The default code page for the computer Interactive SQL is running on

Examples
Is an input statement from an ASCII text file:

    input into employee
    from new_emp.inp
    format ASCII

Usage
- The input statement allows efficient mass insertion into a named database
table. Lines of input are read either from the user via an input window (if
prompt is specified) or from a file (if you specify from file_name). If neither
is specified, the input is read from the command file that contains the input
statement. In Interactive SQL, this can even be directly from the SQL
Statements pane. In this case, input is ended with a line containing only the
string end.
• If a column list is specified for any input format, the data is inserted into
the specified columns of the named table. By default, the input statement
assumes that column values in the input file appear in the same order in
which they appear in the database table definition. If the input file’s
column order is different, you must list the input file’s actual column order
at the end of the input statement.

In this example, you create a table called inventory. To import ASCII data
from the input file that contains the name value before the quantity value,
you must list the input file’s actual column order at the end of the input
statement for the data to be inserted correctly:

create table inventory (  
  quantity int,
  item varchar(60)
)

The ASCII data from the input file stock.txt that contains the name value
before the quantity value:

'Shirts', 100  
'Shorts', 60

The input file’s actual column order at the end of the input statement for
the data to be inserted correctly:

input into inventory  
from stock.txt  
FORMAT ASCII  
(item, quantity)

• By default, input stops when it attempts to insert a row that causes an error.
Errors can be treated in different ways by setting the on_error and
conversion_error options (see “set option” on page 665). Interactive SQL
prints a warning in the Messages pane if any string values are truncated on
input. Missing values for NOT NULL columns are set to zero for numeric
types and to the empty string for non-numeric types. If input attempts to
insert a NULL row, the input file contains an empty row.

Permissions  You must have insert permission on the table or view.
output

Description
Imports data into a database table from an external file or from the keyboard.

Syntax
output to filename
  [ append ]
  [ verbose ]
  [ format { ascii | dbase | dbasell | dbaselll | excel | fixed | foxpro | lotus | sql | xml }]
  [ escape character character ]
  [ escapes { on | off }]
  [ delimited by string ]
  [ quote string [ all ]]
  [ column widths (integer . . . ) ]
  [ hexidecimal { on | off | asis }]
  [ encoding { string | identifier }]

Parameters
append
appends the results of the query to the end of an existing output file without overwriting the previous contents of the file. If the append clause is not used, the output statement overwrites the contents of the output file by default. The append keyword is valid if the output format is ASCII, fixed, or SQL.
verbose

Writes error messages about the query, the SQL statement used to select the data, and the data itself to the output file. Lines that do not contain data are prefixed by two hyphens. If you omit verbose (the default) only the data is written to the file. verbose is valid if the output format is ASCII, fixed, or SQL. Allowable output formats are:

- **ascii** – the output is an ASCII format file with one row per line in the file. All values are separated by commas, and strings are enclosed in apostrophes (single quotes). You can change the delimiter and quote strings using the delimited by and quote clauses. If all is specified in the quote clause, all values (not just strings) are quoted.

Three other special sequences are also used. The two characters represent a new line character, "\", represents a single \, and the sequence \xDD represents the character with hexadecimal code DD. This is the default output format.

- **dbaseII** – the output is in DBASEII which includes column definitions. A maximum of 32 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.

- **dbaseIII** – the output is a dBASE III format file which includes column definitions. A maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.

- **excel** – the output is an Excel 2.1 worksheet. The first row of the worksheet contains column labels (or names if there are no labels defined). Subsequent worksheet rows contain the actual table data.

- **fixed** – the output is fixed format with each column having a fixed width. You can specify the width for each column with column widths. No column headings are output in this format.

  If the column widths clause is omitted, the width for each column is computed from the datatype for the column, and is large enough to hold any value of that datatype. The exception is that long varchar and long binary data default to 32K.

- **foxpro** – the output is a FoxPro format file which includes column definitions. A maximum of 128 columns can be output. Column names are truncated to 11 characters. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
• html – the output is in the HyperText Markup Language format.

• lotus – the output is a Lotus WKS format worksheet. Column names are placed as the first row in the worksheet. There are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file Interactive SQL can produce.

• SQL – the output is an Interactive SQL input statement required to recreate the information in the table.

• XML – the output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings. The input statement does not accept XML as a file format.

escape character
is the default escape character for characters\ stored as hexadecimal codes and symbols is a backslash (\), so, for example, \x0A is the linefeed character.

You can change the default escape character using escape character. For example, to use the exclamation mark as the escape character, enter:

... escape character '!' 

escapes
if enabled (the default), characters following the backslash character are recognized and interpreted as special characters by the database server. New line characters can be included as the combination \n, and other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\) is interpreted as a single backslash. A backslash followed by any character other than n, x, X or \ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q.

delimited by
for the ASCII output format only. The delimiter string is placed between columns (default comma).

quote
for the ASCII output format only. The quote string is placed around string values. The default is a single quote character. If all is specified in the quote clause, the quote string is placed around all values, not just around strings.

column width
specifies the column widths for the fixed format output
hexidecimal

specifies how binary data is to be unloaded for the ASCII format only. When
set to on, binary data is unloaded in the format 0xabcd. When set to off,
binary data is escaped when unloaded (\xab\xcd). When set to asis, values
are written as is, that is, without any escaping—even if the value contains
control characters. asis is useful for text that contains formatting characters
such as tabs or carriage returns.

encoding

allows you to specify the encoding that is used to write the file. encoding can
be used only with the ASCII format.

If encoding is not specified, Interactive SQL determines the code page that
is used to write the file as follows, where code page values occurring earlier
in the list take precedence over those occurring later in the list:

• The code page specified with default_isql_encoding (if this option is set)
• The code page specified with the -codepage option when Interactive
  SQL was started
• The default code page for the computer Interactive SQL is running

Examples

Example 1 Places the contents of the employee table in a file in ASCII format:

```sql
select *
from employee
go
output to employee.txt
format ASCII
```

Example 2 Place the contents of the employee table at the end of an existing
file, and includes any messages about the query in this file as well:

```sql
select *
from employee
go
output to employee.txt append verbose
```

Example 3 In this example, you need to export a value that contains an
embedded line feed character. A line feed character has the numeric value 10,
which you can represent as the string ‘\x0a’ in a SQL statement. If you execute
the following statement, with hexidecimal set to on:

```sql
select 'line1 n x0aline2'
go
output to file.txt hexidecimal on
```

You see a file with one line in it containing the following text:
However, if you execute the same statement with `hexidecimal` set to `off`, you see the following:

```
line1
line2
```

Finally, if you set `hexidecimal` to `asis`, you see a file with two lines:

```
line1
line2
```

You get two lines when you use `asis` because the embedded line feed character has been exported without being converted to a two-digit hexadecimal representation, and without being prefixed by anything.

### Usage

- The output statement copies the information retrieved by the current query to a file.
- You can specify the output format with the optional `format` clause. If you do not specify the `format` clause, the Interactive SQL `output_format` option setting is used.
- The current query is the `select` or `input` statement that generated the information appearing on the Results tab in the Results pane. The output statement reports an error if there is no current query.
- In Interactive SQL, the Results tab displays only the results of the current query. All previous query results are replaced with the current query results.

### Permissions

Any user can execute this command.
parameters

Description
Specifies parameters to an Interactive SQL command file.

Syntax
parameters parameter1, parameter2, ...

Examples
This Interactive SQL command file takes two parameters:

```
parameters department_id, file;
select emp_lname
from employee
where dept_id = {department_id}
>#{file}.dat
```

If you save this script in a file named `test.sql`, you can run it from Interactive SQL using the following command:

```
read test.SQL [100] [data]
```

Usage

- The parameters statement names the parameters for a command file, so that they can be referenced later in the command file.
- Parameters are referenced by putting `{parameter1}` into the file where you want the named parameter to be substituted. There cannot be any spaces between the braces and the parameter name.
- If a command file is invoked with less than the required number of parameters, Interactive SQL prompts for values of the missing parameters.

Permissions
Any user can execute this command.

See also
read
### read

**Description**

Reads Interactive SQL statements from a file.

**Syntax**

```
read [ encoding {identifier | string}] file_name [ parameters ]
```

**Parameters**

- `encoding {identifier | string}` allows you to specify the encoding that is used to write the file. `encoding` can be used only with the ASCII format.
  - `identifier` – is the identifier you are using to indicate the file you are reading.
  - `string` – is the string you are using to indicate the file you are reading.

- `file_name` is the name of the file you are reading.

- `parameters` correspond to the parameters listed in the statement file.

**Examples**

These are examples of the `read` statement:

```
READ status.rpt '160'
```

```
READ birthday.SQL [>= '1988-1-1'] [<= '1988-1-30']
```

**Usage**

- The `read` statement reads a sequence of Interactive SQL statements from the named file. This file can contain any valid Interactive SQL statement, including other `read` statements. `read` statements can be nested to any depth. If the file name does not contain an absolute path, Interactive SQL searches for the file. Interactive SQL first searches the current directory, and then the directories specified in the environment variable `SQLPATH`, and then the directories specified in the environment variable `PATH`. If the named file has no file extension, Interactive SQL searches each directory for the same file name with the extension `.SQL`.

- The encoding argument allows you to specify the encoding that is used to read the file. The `read` statement does not process escape characters when it reads a file. It assumes that the entire file is in the specified encoding. If encoding is not specified, Interactive SQL determines the code page that is used to read the file as follows, where code page values occurring earlier in the list take precedence over those occurring later in the list:
  - The code page specified with the `default_isql_encoding` option (if this option is set)
  - The code page specified with the `-codepage` option when Interactive SQL was started
• The default code page for the computer Interactive SQL is running on

• Parameters can be listed after the name of the command file. These parameters correspond to the parameters named on the parameters statement at the beginning of the statement file. Interactive SQL substitutes the corresponding parameter wherever the source file contains \{parameter_name\}, where parameter_name is the name of the appropriate parameter.

• The parameters passed to a command file can be identifiers, numbers, quoted identifiers, or strings. When quotes are used around a parameter, the quotes are placed into the text during the substitution. You must enclose in square brackets ([ ]) parameters that are not identifiers, numbers, or strings (contain spaces or tabs). This allows for arbitrary textual substitution in the command file.

• If not enough parameters are passed to the command file, Interactive SQL prompts for values for the missing parameters.

Permissions

Any user can execute this command.
## set connection

**Description**
Changes the current database connection to another server.

**Syntax**
```
set connection {identifier | string | hostvar}
```

**Parameters**
- `identifier` is the login name identifier you are using for the connection information.
- `string` is the string you are using for the connection information.
- `hostvar` is the variable information for the host name and port.

**Usage**
The `set connection` statement changes the active database connection to another server. The current connection state is saved, and resumes again when it again becomes the active connection. If you omit `connection_name` and there is a connection that was not named, that connection becomes the active connection.

**Permissions**
Any user can execute this command.

**See also**
`connect`, `disconnect`
**set option**

**Description**
Changes the values of Interactive SQL options.

**Syntax**
```
set [ temporary] option
  [{identifier | string | hostvar}. | public.]
  {identifier | string | hostvar} = [option_value]
```

```
sset permanent

set
```

**Parameters**
- **temporary**
  indicates that the options you set are only for this session.
- **option**
  indicates that you are setting the subsequent option.
- **public**
  indicates that the option is being set for all users.
- **option_value**
  is the value of the option you are setting.
- **identifier**
  is the identifier you are using for the connection information.
- **string**
  is the string you are using for the connection information.
- **hostvar**
  is the variable information for the host name and port

**Usage**
- **set permanent** in Syntax 2 saves all current Interactive SQL options. These settings are automatically established every time Interactive SQL is started for the current user ID.
- Syntax 3 displays all of the current option settings. If there are temporary options set for Interactive SQL or the database server, these are displayed; otherwise, the permanent option settings are displayed.
**start logging**

**Description**
Starts logging executed SQL statements to a log file.

**Syntax**
```
start logging file_name
```

**Parameters**
- `file_name` is the file to which you are logging the session.

**Examples**
Starts logging to a file called `filename.sql`, located in the c: directory:
```
start logging 'c: \n filename.sql'
```

**Usage**
The `start logging` statement starts copying all subsequent executed SQL statements to the log file that you specify. If the file does not exist, Interactive SQL creates it. Logging continues until you explicitly stop the logging process with the `stop logging` statement, or until you end the current Interactive SQL session. You can also start and stop logging by selecting SQL | Start Logging and SQL | Stop Logging.

**Permissions**
Any user can execute this command.

**See also**
- `stop logging`
stop logging

Description
Stops logging of executed SQL statements in the current session.

Syntax
stop logging

Examples
Stops the current logging session:

stop logging

Usage
The stop logging statement stops Interactive SQL from writing each SQL statement you execute to a log file. You can start logging with the start logging statement. You can also start and stop logging by selecting SQL | Start Logging and SQL | Stop Logging.

Permissions
Any user can execute this command.

See also
start logging
**system**

**Description**
Launches an executable file from within Interactive SQL.

**Syntax**

```
system [path] file_name
```

**Parameters**

- `path`
  is the path to the Notepad program

- `file_name`
  is the file name of the program you are launching.

**Examples**
Launches the Notepad program, assuming that the Notepad executable is in your path.

```
system 'notepad.exe'
```

**Usage**
Launches the specified executable file.

- The `system` statement must be entirely contained on one line.
- Comments are not allowed at the end of a `system` statement.
- Enclose the path and file name in single quotation marks.

**Permissions**
Any user can execute this command.

**See also**
connect
Symbols

* (asterisk) select and 225
@ (at sign) local variable name 251–252, procedure parameters and 347, rule arguments and 152
\ (backslash) character string continuation with 636
:= (BNF notation) in SQL statements xiv, (comma) in SQL statements xiv
{} (curly braces) in SQL statements xiv = (equals sign) for assigning variables 513, for renaming column headings 512
! (exclamation point) error message placeholder 463
() (parentheses) in SQL statements xiv
% (percent sign) error message literal 465, error message placeholder 463
%nn! (placeholder format) 463
# (pound sign), temporary table identifier prefix 165
?? (question marks) for partial characters 477
“ ” (quotation marks) literal specification of 635
<$ix_command and <$ix_command 110

Numerics

0 return status in stored procedures 137
“0x” in defaults 89, in rules 152
writetext command and image data 639
2 isolation level (repeatable reads) 523

A

abbreviations
chars for characters, readtext 476
exec for execute 346
out for output 131, 347
tran for transaction, rollback command 504
abstract plans, creating with create plan 127
accent sensitivity
compute and 73
dictionary sort order and 459
group by and 402
access, ANSI restrictions on tapes 344
access, object. See permissions; users
activation keyword, alter role 19
add keyword
alter role 19
alter table 26, 33
adding
columns to a table 24
constraints for tables 24
messages to sysusermessages 465
mirror device 276–279
mutually exclusive user-defined roles 19
objects to tempdb 188
passwords to roles 19
roles 150
rows to a table or view 406–415
space to a database 7–12
table constraints 24
user-defined roles 150
aggregate functions
group by clause and 390, 393
having clause and 391, 393
scalar aggregates 393
vector aggregates, group by and 393
aggregate-free expression, grouping by 391
aliases for table correlation names 517
aliases, column
compute clauses allowing 71
Index

prohibited after group by 391, 392

all keyword 361, 379
group by 390
negated by having clause 391
revoke 490
select 511, 528
union 596, 600
where 634

allocation map. See Object Allocation Map (OAM).
allow nested triggers configuration parameter 215
allow_dup_row option, create index 112
alter database command 7–12
default keyword 7
dumping databases and 10
for load keyword 8
for proxy update keyword 8
log on keyword 8
offline databases and 10
on keyword 7
with override keyword 8
alter encryption key command 13–18
alter role command 19–23
activation keyword 19
add keyword 19
drop keyword 19
exclusive keyword 19
membership keyword 19
passwd keyword 19
alter table command 24–57
add keyword 26, 33
asc option 29
check option 33
clustered constraint 29
constraint keyword 28
default keyword 27
desc option 29
drop keyword 33
encrypted columns and 42
exp_row_size option 34
fillfactor option 30
foreign key constraint 32
identity keyword 28
lock allpages option 34
lock datapages option 34
lock datarows option 34
locking scheme 24
max_rows_per_page option 31
nonclustered constraint 29
on keyword 32, 174
partition clause 34
primary key constraint 29
references constraint 32
replace keyword 34
reservepagegap option 31
sp_dboption and changing lock scheme 55
unique constraint 29
unpartition clause 34
user keyword 27
when is data copy required 51
and keyword
range-end 633
in search conditions 633
ANSI tape label
dumpvolume option to dump database 313
dumpvolume option to dump transaction 334
listonly option to load database 422
listonly option to load transaction 436
ansinull option, set 540
any keyword in where clause 634
archive database access
compatibility 327
compressed dumps 326
create archive database command, using 79
logical devices 430
materializing an archive database 429
with no recovery 430
arguments
See also logical expressions
numbered placeholders for, in print command 463, 464
in user-defined error messages 471
where clause, number allowed 636
arthabort option, set
arith_overflow and 541
arithmetic option, set
arith_overflow and 541
as keyword for renaming column headings 512
asc index option
alter table command 29, 44
create index command 109
create table command 169
Index

Reference Manual: Commands

ascending index order, specifying 24
ascending indexes 29
ascending order, asc keyword 456, 522
asterisk (*)
  select and 225
at option
  create existing table 96
  create proxy_table 145
  create table 175
  dump database 312
  dump transaction 332
  load database 420
  load transaction 435
at sign (@)
  local variable name 251–252
  procedure parameters and 347
  rule arguments and 152
@@char_convert global variable 581
@@error global variable
  select into and 530
  stored procedures and 134
  user-defined error messages and 465, 474
@@identity global variable 412
@@isolation global variable 581
@@langid global variable 470
@@nestlevel global variable 351
  nested procedures and 137
  nested triggers and 216
@@options global variable 581
@@parallel_degree global variable 581
  set parallel_degree and 553
@@rowcount global variable 581
  set nocount and 581
  triggers and 214
@@scan_parallel_degree global variable 581
  set scan_parallel_degree and 558
@@textsize global variable 581
  readtext and 477
  set textsise and 561
@@tranchained global variable 581
@@version global variable 464
attributes
  remote tables 98
  authority. See permissions.
  automatic operations
  checkpoints 61
datatype conversion 187
  triggers 206

B
backslash (\) for character string continuation 636
backups
  See also dump, database; dump, transaction log;
  load, database; load, transaction log
disk mirroring and 277, 290
disk remirroring and 285
  incremental. See dump, transaction log
master database 10
Backus Naur Form (BNF) notation xiii, xiv
base tables. See tables.
batch processing
  create default and 90
  execute 346, 350
  return status 486–488
  set options for 571
bcp (bulk copy utility)
  changing locking scheme during 56
begin transaction command 59
  commit and 65
  rollback to 505
begin...end command 58
if...else and 403
  triggers and 207
between keyword
  check constraint using 196
where 633
binary datatypes
  "0x" prefix 89, 152
binary operation, union 597
binary sort order of character sets
  order by and 459
binding
  defaults 89
  rules 154
  unbinding and 294
blanks, character datatypes and 409, 608
blocking process 417
blocksize option
  dump database 312
  dump transaction 333
Index

load database 420
load transaction 435
BNF notation in SQL statements xiii, xiv
boolean (logical) expressions, select statements in 404
brackets. See square brackets [ ]
branching 359
break command 60, 637–638
browse mode in select 523
B-trees, index and fillfactor 110
bulk array size option, set
bulk array size and 542
bulk batch size option, set
bulk batch size and 542
by row aggregate subgroup 66
bytes
See also size
per row 40
bytes option, readtext 476

C
canceling
See also rollback command
command at rowcount 558
duplicate updates or inserts 112
queries with adjusted plans 556
transactions with arithmetic errors 541
triggers 506
capacity option
dump database 313
dump transaction 333
cascade option, revoke 492, 497
cascading changes (triggers) 211
case sensitivity
compute and 72
group by and 401
in SQL xv
sort order and 459
chained option, set 543
chained transaction mode
commit and 65
delete and 264
fetch and 354
insert and 410
open and 455
update and 607
chains of pages
partitions 34, 48
unpartitioning 34
changes, canceling. See rollback command.
changing
See also updating
constraints for tables 24
database size 7–12
locking scheme 24, 34
passwords for user-defined roles 22
table constraints 24
tables 24–57
user-defined roles 19
view definitions 225
changing database size 287–288
char datatype
row sort order and 460
@@char_convert global variable 581
char_convert option, set 544
character sets
conversion between client and server 544
fix_text upgrade after change in 235
multibyte, changing to 235
set char_convert 544
character strings
empty 409
truncation 409, 561
characters
“0x” 152
not converted with char_convert 544
chars or characters option, readtext 476
check constraints
column definition conflict with 196
insert and 409
check option
alter table 33
create table 173
checkalloc option, dbcc 231
checkcatalog option, dbcc 232
checkdb option, dbcc 232
checker, consistency. See dbcc command.
checkpoint command 61–62
checkpoint process 61–62
See also recovery; savepoints
checkstorage option, dbcc 232
checktable option, dbcc 233–248
checkverify option, dbcc 233
CIS. See Component Integration Services.
cis_rpc_handling option, set command 545
clear Interactive SQL command 644
client, character set conversion 544
clientappname option, set command 545
clienthostname option, set command 545
clientname option, set command 545
close command 63
close on endtran option, set 543
closing cursors 63
clustered constraint
alter table 29
create table 169
clustered indexes
See also indexes
<Six_command and 110
creating 108
migration of tables to 118, 188
segments and 113, 118
cntrltype option
disk init 271
disk reinit 282
collating sequence. See sort order
collision of database creation requests 85
column name
aliasing 471, 512
grouping by 391, 392
union result set 598
views and 221
columns
adding data with insert 408
adding to table 24
check constraints conflict with definitions of 196
creating indexes on 107–126
defaults for 89–91, 409
gaps in IDENTITY values 549
group by and 391
list and insert 406
maximum number per table 40
null values and check constraints 196
null values and default 90, 154
order by 521
per table 40
permissions on 361
permissions revoked 490
rules 409
rules conflict with definitions of 154
union of 598
variable-length, and sort order 459
views and 221
columns per table 40
command execution delay. See waitfor command
command permissions 375–377
See also object permissions; permissions
grant all 379
grant assignment of 360–389
levels 375
revoking 491
commands
alter database 7–12
alter encryption key 13–18
alter role 19–23
begin transaction 59
begin...end 58
break 60
checkpoint 61–62
close 63
commit 64–65
compute 66–74
connect to 75–77
continue 78
create archive database 79–80
create encryption key 92–94
create existing table 95–100
create function 101–103
create function (SQLJ) 104–106
create index 107–126
create plan 127–128
create procedure 129–140
create procedure (SQLJ) 141–144
create proxy_table 145–148
create role 149–151
create rule 152–155
create scheme 156–157
create service 158–162
create table 163–205
create trigger 206–220
dbcc 230–249
deallocate cursor 250
declare 251–252
Index

declare cursor 253–259
declare cursor, cannot use with scrollable cursor 255
delete 260–267
delete statistics 268–269
disconnect 75–77
disk init 270–275
disk mirror 276–279
disk reinit 281–284
disk remirror 285–286
disk resize 287–288
disk unmirror 289–291
drop database 292–293
fetch, multiple rows per 356
order-sensitive 377, 496
rowcount range for 558
statistics io for 560
statistics time information on 560
Transact-SQL, summary table 1–6
update, cannot use with scrollable cursor 609
commit command 64–65
begin transaction and 59, 65
rollback and 65, 505
commit work command. See commit command.
commiting a transaction in prepare state 246
common keys 169
See also foreign keys; joins; primary keys
compact option, reorg command 483
comparing values
datatype conversion for 636
for sort order 459–460
in where clause 636
comparison operators
where clause 631
compatibility for an archive database 327
compatibility, data
create default and 90
of rule to column datatype 153
compiling
exec with recompile and 348
time (statistics time) 560
without execution (noexec) 543
create procedure, dbcc 234
Component Integration Services
constraints for remote servers and 28, 33
Component Integration Services commands
connect to 75
create existing table 95
create proxy_table 145
composite indexes 108, 124
compressed backups
making 312, 332
unloading 420, 434
compressed dumps
with archived databases 326
compute clause 66–74
order by and 458, 522
select 522
without by 70
conceptual (logical) tables 211, 212
configuration parameters 480
configure Interactive SQL command 645
conflicting roles 21
connect Interactive SQL command 646
connect to command 75–77
consistency check. See dbcc command.
constants, return parameters in place of 350
constraint keyword
alter table 28
create table 168
cross-database 195, 307
dropping table 24
error messages 191
indexes created by and max_rows_per_page 31
referential integrity 193
unique 192
consumer process 111
consumers option, update statistics command 617, 620
continuation lines, character string 636
continue command 78
while loop 637
control-of-flow language
begin...end and 58
create procedure and 131
conventions
See also syntax
Transact-SQL syntax xiii
used in the Reference Manual xiii
conversion columns 187
dates used with like keyword 632
null values and automatic 187
where clause and datatype 636
copying databases with create database 86–87
the model database 85
rows with insert...select 407
tables with select into 530
correlation names and table names 517
corrupt indexes. See reindex option, dbcc.
counters, while loop. See while loop.
create archive database command 79–80
create database command 81–88
default option 81
disk init and 274
for load keyword 82
for proxy_update keyword 82
log on keyword 81
on keyword 81
permission 379
with dbid keyword 82
with default_location keyword 82
with override keyword 82
create default command 89–91
batches and 90
create encryption key command 92–94
create existing table command 95–100
datatype conversions and 98
defining remote procedures 99
mapping to remote tables 95
server class changes 99
create function (SQLJ) command 104–106
create function command 101–103
create index command 107–126
index options and locking modes 123
insert and 408
space management properties 122
create plan command 127–128
create procedure (SQLJ) command 141–144
create procedure command 129–140
See also stored procedures; extended stored procedures (ESPs)
order of parameters in 347, 350
return status and 137–138
select * in 135
create proxy_table command 145–148
mapping proxy tables to remote tables 145
create role command 149–151
grant all and 151
create rule command 152–155
create scheme command 156–157
create service command 158–162
examples 159
parameters 158
syntax 158
create table command 163–205
column order and 459
locking scheme specification 198
mapping proxy tables to remote tables 200
null values and 27, 168
space management properties 198
create trigger command 206–220, 378, 497
create view command 221–229
SQL derived tables and 223
creating archive databases 79–80
databases 81–88
defaults 89–91
encryption keys 92–94
extended stored procedures 129–140
indexes 107–126
rules 152–155
schemas 156–157
services 158–162
SQLJ stored procedures 141–144
tables 163–205, 513
tables, with identity column 198
trigger 206–220
triggers 378, 497
user-defined roles 149
views 221–229
views from SQL derived tables 227
cross-platform dump and load, handling suspect partitions 426
curly braces ({} in SQL statements xiv
current database changing 627
current locks, sp_lock system procedure 417
current processes. See processes (server tasks)
cursor result set 257
datatypes and 354
returning rows 353
cursor rows option, set 545
cursors
  closing 63
  compute clause and 70
datatype compatibility 354
deallocating 250
declaring 253–259
deleting rows 265
fetching 353–358
grant and 377
group by and 393
Halloween problem 258
opening 455
order by and 458
read-only 257
scans 257
scope 255
select and 529
union prohibited in updatable 597
updatable 257
updating rows 609

D
damaged database, removing and repairing 234
data dictionary. See system tables
data integrity 409
  See also referential integrity constraints
data modification
text and image with writetext 639
update 602
database consistency checker. See dbcc command.
database devices
  alter database and 7
  new database 81
  transaction logs on separate 278, 286
database dump. See dump, database; dump devices
database object owners
  See also database owners; ownership
database objects
  adding to tempdb 187
  permissions when creating procedures 140
  permissions when creating triggers 219
  permissions when creating views 228
  permissions when executing procedures 140
  permissions when executing triggers 219
  permissions when invoking views 228
  referencing, create procedure and 135
  select_list 512–513
database owners
  See also database object owners; permissions
  permissions granted by 361
  use of setuser 375
databases
  backing up 311–329
  checkalloc option (dbcc) 231
  checkdb option (dbcc) 232
  checkstorage option (dbcc) 232, 233
  creating 81
  creating with separate log segment 339
  creation permission 88
  default size 84
  dropping 292
  dumping 311–329
  increasing size of 7
  loading 418–432
  number of server 85
  offline, altering 10
  recovering 418–432
  removing and repairing damaged 234
  selecting 627
  suspending 466
  upgrading database dumps 427, 441
  use command 627
data-only locked tables
  restrictions for adding, dropping, or modifying columns 52
dataserver utility 278
  See also Utility Guide manual
disk mirror and 278
disk remirror and 286
datatype conversions
  column definitions and 187
datatypes
  comparison in union operations 598
  compatibility of column and default 90
  cursor result set and 354
invalid in \textbf{group by} and \textbf{having} clauses 392
local variables and 251
date parts
order of 546
datefirst option, \textbf{set} 546
dateformat option, \textbf{set} 546
dates
display formats 546
display formats, \textbf{waitfor} command 629
datetime datatype
\textit{See also} \textbf{set} command
dbcc
command options 381
\textbf{on all} and guest 383
permissions 388
server-level commands and \textbf{on all | database} 383
dbcc command
\textit{See also individual dbcc options}
dbcc (database consistency checker)
\textbf{readtext} and 477
dbcc command 230–249
dbcc complete_xact 1pc command 246
dbcc pravailabletempdbs and tempdbs 236
dbcc traceon 239
dbcc tune 239
DB-Library programs
browse mode 523
db writext and dbmoretext, writext compared to 641
prepare transaction 462
\textbf{set} options for 551, 573
\textbf{waitfor} mirrorexit and 629
dbrepair option, \textbf{dbcc} 234
deactivation of disk mirroring 289–291
deadlocks
descending scans and 460
deallocate cursor command 250
deallocating cursors 250
debugging aids
\textbf{set showplan on} 543
\textbf{set sort_resources on} 544
\textbf{set statistics io on} 560
triggers and 216
declare command 251–252
declare cursor command 253–259
scrollable cursors and 255
declaring
local variables 251
parameters 130
default database size configuration parameter in \textit{sysconfigures} 84
default keyword
\textbf{alter database} 7
\textbf{alter table} 27
\textbf{create table} 166
default option
\textbf{create database} command 81
default segment
extending 11
default settings
parameters for stored procedures 130
\textbf{set} command options 572
weekday order 573
default values
datatypes when no length specified 130
defaults 409
column 27
creating 89–91
definitions and \textbf{create default} 89–91
dropping 294
IDENTITY columns and 50
rules and 90, 154
defining local variables 251–252
degree of parallelism
\textbf{select} and \textbf{parallel} 518
delayed execution (\textbf{waitfor}) 628
delete command 260–267
\textbf{readpast} option 260
triggers and 212
\textbf{truncate table} compared to 594
delete statistics command 268–269
\textbf{deleted} table
triggers and 211, 212
deleting
\textit{See also} dropping, shared statistics 268
unlocked rows 260
density option
dump database 312
dump transaction 333
load database 420
load transaction 435
Index

dependencies, database object
  sp_depends system procedure 187

desc index option
  alter table command 44
  create index command 109
  create table command 169

desc option
  alter table 29
  descending index order, specifying 24
  descending indexes 29
  descending order (desc keyword) 456, 522
  descending scans 460
  deadlocks and 460
  overflow pages and 461

descriptions
  grant dbcc 360
  revoke dbcc 489

device failure
  dumping transaction log after 335, 338

device fragments number of 85

device initialization. See initializing devices

disk mirroring to 276–279
  master 10
  numbering 270, 281
  secondary 277

dictionary sort order 459

dirty pages updating 61–62

disabling mirroring. See disk mirroring

disconnect command 75–77

disconnect Interactive SQL command 649

disk controllers 271, 282

disk devices
  adding 270–275
  mirroring 276–279
  unmirroring 289–291

disk init command 270–275
  master database backup after 274

disk mirror command 276–279

disk mirroring 276–279
  database dump and 325
  database load and 429
  restarting 285–286
  transaction log dump and 345

disconnect Interactive SQL command 649

distinct keyword
  create view 221
  select 511, 528

distributed transaction processing (DTP) 234

dividing tables into groups. See group by clause.
domain rules 409

doubled quotes in character strings 635

drop database command 292–293
  damaged databases and 234
  drop default command 294
  drop encryption key command 295
  drop function (SQLJ) command 297
  drop function command 296
  drop index command 298–299
  drop keyword
  alter role 19
  alter table 33
  drop procedure command 300–301
  grouped procedures and 300, 347
  drop role command 302
  drop rule command 304

deadlocks and 460
overflow pages and 461

transaction log load and 443
unmirroring and 289–291
waitfor mirrorexit 628

disk refit command 280
create database and 86
disk reinit command 281–284
See also disk init command
disk remirror command 285–286
See also disk mirroring
disk resize command 287–288
disk unmirror command 289–291
See also disk mirroring
dismount option
  dump database 314
  dump transaction 334
  load database 421
  load transaction 436

display
  create procedure statement text 139
  procedures for information 132
  setting for command-affected rows 543

“don’t recover” status of databases created for load 87

double quotes in character strings 635

drop database command 292–293
  damaged databases and 234

drop default command 294

drop encryption key command 295

drop function (SQLJ) command 297

drop function command 296

drop index command 298–299

drop keyword
  alter role 19
  alter table 33

drop procedure command 300–301
  grouped procedures and 300, 347

drop role command 302

drop rule command 304
drop service command 305
drop table command 306–308
drop trigger command 309
drop view command 310
dropdb option, dbcc dbrepair 234
dropping
  constraints for tables 24
  corrupt indexes 237
  damaged database 234
  databases 292–293
dbcc dbrepair database 234
defaults 90, 294
  grouped procedures 129
  indexes 298–299
  passwords from roles 19
  procedures 300–301
  roles in a mutually exclusive relationship 19
  rows from a table 260–267, 306
  rows from a table using truncate table 594
rules 304
services 305
table constraints 24
tables 306–308
tables with triggers 213
triggers 213, 309
user-defined roles 302
views 310
dump 327
dump database
  across platforms 318
  compress option 312
dump database command 311–329
  See also dump, database
  after using create database 86
  after using disk init 274
  after using dump transaction with no_log 332
dump transaction and 318
master database and 319
select into and 531
dump devices
  See also database devices; log device
dump, database and 312
dump, transaction log and 332
  naming 312, 332, 341
  number required 428
dump striping
database dumps and 314
transaction dumps and 334
dump transaction command 330–345
  See also dump, transaction log
  after using disk init 274
  compress option 332
  permissions for execution 345
  select into/bulkcopy/plsort and 337
  standby_access option 336
  trunc log on chkpt and 337
  with no_log option 340
  with no_truncate option 335, 338
  with truncate_only option 339
dump, database
  across networks 319
  appending to volume 324–325
  Backup Server and 321
  Backup Server, remote 312
  block size 312
  commands used for 337
  dismounting tapes 314
  dump devices 312, 320
  dump striping 314
  dynamic 319
  expiration date 315
  file name 315, 321
  initializing/appending 315
  loading 87, 418–432
  master database 320
  message destination 315
  new databases and 319
  overwriting 315, 324–325
  remote 321
  rewinding tapes after 314
  scheduling 319–320
  successive 324, 343
  system databases 320
  tape capacity 313
  tape density 312
  thresholds and 320
  volume changes 324
  volume name 313, 324
  dump, transaction log
  across networks 340
  appending dumps 335
  appending to volume 344
Index

Backup Server, remote 342
command used for 337
dismounting tapes 334
dump striping 334
expiration date 334
file name 335, 342–343
initializing tape 335
initializing volume 344
insufficient log space option 340
loading 433–444
message destination 335
permissions problems 337
remote 342, 343
rewinding tapes after 334
scheduling 340
tape capacity 333
thresholds and 341
volume name 334, 343
dumpvolume option
dump database 313
dump transaction 334
load database 420
load transaction 435
duplicate rows
indexes and 108, 112
removing with union 596
duplication
of space for a new database 87
of a table with no data 531
dynamic dumps 319, 340
dynamic execution of Transact-SQL commands 346

e
else keyword. See if...else conditions
empty string ("" ) or (’ ’)
as a single space 409
updating an 607
enable xact coordination configuration parameter 563
alter table command and 42
encryption
quiesce database 468
unmount 601
encryption keys
creating 92–94
end keyword 58
engine option, dbcc 235
@@error global variable
select into and 530
stored procedures and 134
user-defined error messages and 465, 474
error handling
in character set conversion 544
dbcc and 245
triggers and 216
error messages
12207 445, 446
character conversion 544
printing user-defined 465
user-defined 470–474
errorexit keyword, waitfor 628
errors
See also error messages; SQLSTATE codes
allocation 231, 236, 239
datatype conversion 166
numbers for user-defined 470
return status values 487
escape keyword
where 632
evaluation order 597
tables 597
examples
grant dbcc 366
revoke dbcc 493
exception report, dbcc tablealloc 236, 238
exclamation point (!)
error message placeholder 463
exclusive keyword
alter role 19
exclusive option, lock table 445
execute command 346–352
create procedure and 134
executing
extended stored procedures 346
procedures 346
Transact-SQL commands 346
user-defined procedures 346
execution delay. See waitfor command.
exeuction, specifying times for 628
exists keyword
where 633
680
Adaptive Server Enterprise
exit
  unconditional, and return command 486–488
  waitfor command 628
exit Interactive SQL command 650
exp_row_size option
  create table 173, 199
  select into 516
  setting before alter table...lock 47
  specifying with create table 173
  specifying with select into 516
explicit values for IDENTITY columns 411, 548
exporting set options 584
expressions
  evaluation order in 597
  grouping by 392
  insert and 407
  summary values for 70
extended columns, Transact-SQL 395, 397
extended stored procedures
  C runtime signals not allowed 135
  creating 129–140
  dropping 300
  executing 346
extending
  database storage 7
  extensions, Transact-SQL 395
extents 118
  create table and 183
  dbcc index alloc report on index 236
  dbcc report on table 238
external option
  create existing table 95
  create proxy_table 145
  create table 174

F
failures, media
  See also recovery automatic failover and 289
disk remirror and 285
fast option
  dbcc index alloc 235
dbcc table alloc 236, 238
fetch command 353–358
multiple rows per 356
fetching cursors 353–358
file names
database dumps 321
DLL 131
listing database dump with listonly 422
listing transaction log with listonly 436
transaction log dumps 335, 435
file option
dump database 315
dump transaction 335
load database 421
load transaction 435
files
  See also tables; transaction log
  mirror device 276
fillfactor option
  alter table 30
  create index 110, 122
  create table 170, 198
fillfactor values
  alter table...lock 45
FIPS flagger
  insert extension not detected by 414
  set option for 547
  update extensions not detected by 612
fipsflagger option, set 547
first column parameter. See keys
fix option
  dbcc 231, 236, 239
dbcc index alloc 235
dbcc table alloc 231
fix_text option, dbcc 235, 245
fixed-length columns
  stored order of 459
flushmessage option, set 547
fmtonly option, set 547
for browse option, select 523
  union prohibited in 599
for load keyword
  alter database 8
  create database command 82
for load option
  create database 87
for proxy_update keyword
  alter database 8
create database command 82
for read only option, declare cursor 253
for update option, declare cursor 253
forceplan option, set 547
forcing offline pages online 318
foreign key constraint
alter table 32
create table 172
foreign keys 192
forget_xact option, dbcc 234
forwarded_rows option, reorg command 483
format strings
print 463
raiserror 470
in user-defined error messages 470
from keyword
delete 260
grant 374
load database 420
load transaction 434
select 516
update 603
full option
dbcc indexalloc 235
dbcc tablealloc 236, 238

G
German language print message example 463
goto keyword 359
grammatical structure, numbered placeholders and 463
grand totals
compute 70
order by 458
grant command 77, 360–389
all keyword 361
drop role permission not included in 302
public group and 362
roles and 380
grant dbcc
described 360
examples 366
parameters 363
syntax 360

H
Halloween problem 258
having clause 390–402
aggregate functions and 390, 393
having clause and 390–402
having clause and, in standard SQL 394
having clause and, in Transact-SQL 395
having clause and, sort orders 401
select 520–521
views and 226
without having clause 401
grants
multiple trigger actions 207
procedures of the same name 129, 300, 347
table rows 394
groups
See also “public” group
grant and 381
revoke and 498
table rows 390
guest users
permissions 381
select  519

I/O
  devices, disk mirroring to  276
  displaying total actual cost (statistics io)  560
  prefetch and delete  261
  prefetch and select  518
  prefetch and update  603

identifiers
  select  528

identities
  sa_role and Database Owner  577
  set proxy and  579
  set session authorization and  579
  setuser command  588

identity burning set factor configuration parameter  411

IDENTITY columns
  adding, dropping, or modifying with alter table  53
  creating tables with  197
  defaults and  50
  gaps in values  549
  inserting values into  406
  inserts into tables with  411
  maximum value of  411
  null values and  412
  selecting  412, 531–532
  updates not allowed  609
  views and  226

identity gap
  setting  198
  @@identity global variable  412

identity keyword
  alter table  28
  create table  167

identity of user. See aliases; logins; users

identity_insert option, set  548

if update clause, create trigger  206, 207, 214
if...else conditions  403–405
  continue and  78
  local variables and  252

ignore_dup_key option, create index  112

ignore_dup_row option, create index  112

image datatype
  length of data returned  528, 561

order by
  not allowed  458
  pointer values in readtext  475
  storage on separate device  475
  triggers and  212
  writetext to  639

immediate shutdown  590

impersonating a user. See setuser command.
in keyword
  alter table and  33
  check constraint using  196
  where  633

inactive transaction log space  332
  included groups. group by query  395
  incremental backups. See dump, transaction log

index keys
  asc option for ordering  119
  desc option for ordering  119
  ordering  119

index pages
  fillfactor effect on  30, 110, 170
  leaf level  30, 108, 110, 170

indexalloc option, dbcc  235

indexes
  ascending  29
  composite  124
  creating  107–126
  dbcc indexalloc and  235
  descending  29
  dropping  298–299
  integrity checks (dbcc)  237
  joins and  117
  key values  617, 621
  listing  298
  max_rows_per_page and  31, 171
  naming  108
  nonclustered  108
  object allocation maps of  236
  page allocation check  235
  specifying order of  24
  specifying sort order with alter table  44
  specifying sort order with create index  119
  specifying sort order with create table  189
  truncate table and  594
types of 108
update index statistics on 616–618
update statistics on 117
infected processes
waitfor errorexit and 629
information (server)
  display procedures 132
  space usage 124
  text 139
information messages (server). See error messages; severity
  levels, error.
init option
dump database 315
dump transaction 335
initializing
disk reinit and 274, 281–284
disk space 270–275
in-memory map 10
input Interactive SQL command 651
insert command 406–415
create default and 89
IDENTITY columns and 411
null/not null columns and 225
triggers and 212, 214
update and 408
views and 226, 412–413
inserted table
  triggers and 211, 212
integrity of data
  constraints 189
  methods 190
integrity. See dbcc command.
Interactive SQL commands 643–668
clear 644
configure 645
connect 646
disconnect 649
exit 650
input 651
output 656
parameters 661
read 662
set connection 664
set option 665
start logging 666
stop logging 667
system 668
internal datatypes of null columns 187
interval, automatic checkpoint 61
into keyword
fetch 353
insert 406
select 513, 530
union 596, 600
is null keyword
where 632
isnull system function
  insert and 410
  print and 465
  select and 528
@@isolation global variable 581
isolation levels
  readpast option and 534
  repeatable reads 523
J
Japanese character sets
  print message example 463, 471
Java columns, adding 52
Java items
  remove java command 481
  joins
    indexes and 117
    table groups and 397
K
key columns
  dropping with alter table 52
  key values 617, 621
  keys, table 192
    See also common keys; indexes
kill command 416–417
L
labels
  dump volumes 324, 428, 442
Index

goto label 359
@@langid global variable 470
language option, set 546
languages, alternate
structure and translation 463
system messages and 546
weekday order and 573
leaf levels of indexes
clustered index 30, 108, 110, 170
leaving a procedure. See return command.
levels
nested procedures and 137, 351
nesting triggers 216
@@nestlevel 137
permission assignment 375
like keyword
alter table and 33
check constraint using 196
where 632
listing
existing defaults 294
user group members 381
listonly option
load database 422
load transaction 436
lists
commands 1–6
error return values 487
reserved return status value 487
sort order choices and effects 459
load 327
load database
across platforms 426
compress option 420
load database command 418–432
load database with no recovery 430
load transaction
compress option 434
load transaction command 433–444
load, database 418–432
across networks 428
Backup Server and 428
block size 420
cross-platform not supported 425
dismounting tapes after 421
file name, listing 422
header, listing 423
load striping 421
message destination 423, 442
new database 87
remote 428
restricting use 427, 441
rewinding tapes after 421
size required 426
updates prohibited during 426
volume name 420
load, transaction log 433–444
disk mirroring and 443
dismounting tape after 436
dump devices 434
file name, listing 436
header, listing 437
load striping 434
message destination 437
point-in-time recovery 437
rewinding tape after 436
until_time 437
volume name 435
loading databases 429, 430
local variables
declare (name and datatype) 251
raiserror and 471
in screen messages 463
in user-defined error messages 471
location of new database 81
lock allpages option
alter table 34
create table command 173
select into command 515
lock datapages option
alter table 34
create table command 173
select into command 515
lock datarows option
alter table 34
alter table command 55
create table command 173
select into command 515
lock nowait option, set lock command 550
lock table command 445
lock wait option, set command 550
locking
Index

tables with lock table command 445
text for reads 475
locking scheme
  changing 24, 34
  changing with alter table 24
create table and 198
modifying 34
specifying with select into 515
locks
deletes skipping locked rows 260
selects skipping locked rows 532, 533
updates skipping locked rows 602
log device
  See also transaction logs
purging a 320
space allocation 86, 245
log on keyword
alter database 8
create database 81
log segment
  dbcc checktable report on 233
  not on its own device 248
logging
  select into 530
  text or image data 639
  triggers and unlogged operations 213
  writetext command 639
logical (conceptual) tables 211, 212
logical consistency. See dbcc command.
logical device name
disk mirroring 276
disk remirroring 285
disk unmirroring 289
new database 81
logical devices and archive database access 430
logical expressions
  if...else 403
  syntax 60
login triggers
  and set options 584
login triggers
  See also remote logins; users
  char_convert setting for 544
  disabling 590
loans. See segments; transaction logs.
loops
  break and 60
  continue and 78
  goto label 359
  trigger chain infinite 216
  while 60, 637
lowercase letters, sort order and 459

M
making compressed backups 312, 332
mapping
  system and default segments 11
  markers, user-defined. See placeholders; savepoints
master 384
master database
  See also recovery of master database; databases
alter database and 10
backing up 339
create database and 86
disk init and 274
disk mirror and 277
disk refit and 280
disk reinit and 281
disk remirror and 285
disk unmirror and 290
dropping databases and 292
transaction log purging 320, 339
master device 10
materializing an archive database 429
max_rows_per_page option
  alter table 31, 45
  create index 111, 122
  create table 171, 198
  select into 515
maximum number of columns 40
maximum row size 40
members keyword
  alter role 19
memory
  See also space
  releasing with deallocate cursor 250
messages
  language setting for 546
  printing user-defined 463–465
Index

revoke 497
screen 463–465
trigger 212, 309
migration
of system log to another device 274
of tables to clustered indexes 118, 188
mirror keyword, disk mirror 276
mirrorexit keyword in waitfor command 628
mistakes, user. See errors
mode option, disk unmirr 289
model database, copying the 85
modifying
databases 7
locking scheme 34
roles 19
tables 24
mount command 447–451
See also quiesce database
See also unmount
multibyte character sets
changing to 235
fix_text upgrade for 235, 245
readtext and 477
readtext using characters for 477
writetext and 641
multicolumn index. See composite indexes
multiple rows per fetch 356
multiple trigger actions 207
multitable views 611
See also views
delete and 225, 263
mutually exclusive roles 19

N
name of device
disk mirroring and 276
disk remirroring and 285
disk unmirroring and 289
dump device 312, 332
physical, disk reinit and 281
remote dump device 428
name option
disk init 270
disk reinit 281

names
alias for table 517
column, in views 221
parameter, in create procedure 130
segment 32, 35, 113, 171, 174, 514, 515
setuser 588
sorting groups of view 310

naming
columns in views 221
cursors 255
database device 270
file 270
indexes 108
stored procedures 135
tables 165
temporary tables 188
triggers 206
views 221
nested select statements. See select command; subqueries.
nesting
begin...end blocks 58
if...else conditions 405
levels 137
levels of triggers 216
stored procedures 135, 351
triggers 216
while loops 638
while loops, break and 60
@@nestlevel global variable 351
nested procedures and 137
nested triggers and 216
%!nn! (placeholder format) 463

no_log option, dump transaction 332
no_truncate option, dump transaction 335
nocount option, set 543
nodismount option
dump database 314
dump transaction 334
load database 421
load transaction 436
noexec option, set 543
nofix option, dbcc
checkalloc and 231
indexalloc and 236
Index

tablealloc and 239
noholdlock keyword, select 475, 519
noinit option
dump database 315
dump transaction 335
nonclustered constraint
alter table 29
create table 169
nonclustered indexes 108
noserial option, disk mirror 276
not keyword
where 630
not null keyword
create table 27, 168
not null values
dropping defaults for 294
insert and 410
select statements and 528
views and 225
notify option
dump database 315
dump transaction 335
load database 423
load transaction 437
nounload option
dump database 314
dump transaction 334
load database 421
load transaction 436
nowait option
lock table command 445
set lock command 550
nowait option, shutdown 590
null keyword
create table 27, 166, 168
null values
check constraints and 196
column defaults and 90, 154
defining 90, 187
dropping defaults for 294
group by and 392
inserting substitute values for 410
new column 90
new rules and column definition 154
null defaults and 90, 154
select statements and 528
sort order of 458
stored procedures cannot return 488
text and image columns 409
triggers and 214
number (quantity of)
active dumps or loads 321, 341, 428, 442
arguments and placeholders 464
arguments, in a where clause 636
bytes in returned text 477
bytes per row 40
clustered indexes 108
databases server can manage 85
device fragments 85
different triggers 212
having clause search arguments 391
logical reads (statistics io) 560
named segments 85
nesting levels 137
nesting levels, for triggers 216
nonclustered indexes 108
parameters in a procedure 252
physical reads (statistics io) 560
placeholders in a format string 465
scans (statistics io) 560
steps for distribution histogram 113
stored procedure parameters 134
tables allowed in a query 517
tables per database 183
updates 216
user-defined roles 150
number of columns
in an order by clause 458
per table 40, 183
in a view 224
number of pages
in an extent 118, 183
statistics io and 560
written (statistics io) 560
numbers
error return values (server) 487
placeholder (%nn!) 463
same name group procedure 129, 300, 347
select list 521
statistics io 560
virtual device 270, 281
weekday names and 546
Index

O

object allocation map (OAM) pages 236

**dbcc indexalloc** and 236

**dbcc** report on table 238

object names, database as parameters 130
in stored procedures 137, 138

object owners. *See* database object owners

object permissions

*See also* command permissions; permissions

**grant** 360–389

**grant all** 379

of option, **declare cursor** 253

offline databases and **alter database** command 10

offset position, **readtext** command 475

**offsets** option, **set** 551

on keyword

**alter database** 7

**alter table** 32, 174

**create database** command 81

**create index** 113, 118

**create table** 35, 171, 514, 515

one-phase commit transactions with **dbcc**

**complete_xact** 246

**online database** command 427, 452–453

bringing databases online 427

**dump transaction** and 439

**load transaction** and 438

upgrades and 441

Open Client applications

keywords 551

**set** options for 551, 573

**open** command 455

opening cursors 455

**optdiag** utility

loading simulated statistics 269, 581

overwriting statistics with **create index** 122

**optimized report**

**dbcc indexalloc** 235, 236

**dbcc tablealloc** 238

@@**options** global variable 581

or keyword

**where** 634

order

of arguments in translated strings 463
ascending sort 456, 522

of column list and insert data 406
of columns (fixed- and variable-length) 459
columns and row aggregates 71
of creating indexes 118
of date parts 546
descending sort 456, 522
error message arguments 463
of evaluation 597
of names in a group 401
of null values 458
of parameters in **create procedure** 347, 350
for unbinding a rule 153

**order by** clause 456–461

**compute by** and 70, 458, 522

**select** and 521

order of commands 377, 496
original identity, resuming an (**setuser** command) 588

output

**dbcc** 245
zer-length string 465

**output** Interactive SQL command 656

**output** option

**create procedure** 131, 347

**execute** 347

return parameter 347

overflow errors

**set arithabort** and 541

overhead

triggers 212

override. *See with override* option

overwriting triggers 212, 309

owners. *See* Database Owners; database object owners

ownership

*See also* permissions; **setuser** command

of command and object permissions 375
of rules 154
of stored procedures 140, 162
of triggers 219
of views 229

P

padding, data and blanks 409

page splits 31, 111, 171

pages, data
See also index pages; table pages
chain of 34, 48–49
exents and 119, 183
 extents and dbcc tablealloc 238
extents reported by dbcc indexalloc 236
multibyte characters and 235
statistics io and 560
pages, OAM (object allocation map)
dbcc indexalloc report on 236
dbcc report on table 238
pages, overflow
descending scans and 461
pages, ratio of filled to empty 24
pair, mirrored 289
parallel keyword, select command 517
@@parallel_degree global variable 581
set parallel_degree and 553
parallel_degree option, set command 553
parameters
grant dbcc 363
revoke dbcc 492
parameters Interactive SQL command 661
parameters, procedure
datatypes 130
defauls 130
execute and 347
naming 130
not part of transactions 351
ways to supply 347, 350
parentheses ()
in SQL statements xiv
parseonly option, set 543
partial characters, reading 477
partition clause, alter table command 34
partitioned tables and alter table 34
partitioning tables 24
passthrough mode
connect to command 76
passwd keyword in alter role 19
passwords
adding to roles 19
adding to user-defined roles 21
changing for user-defined roles 22
dropping from roles 19
dropping from user-defined roles 21
roles and 19
user-defined roles and 149, 558
path name
DLL and extended stored procedures 131
mirror device 276
remote dump device 428
percent sign (%)
error message placeholder 463
literal in error messages 465
performance
select into and 531
showplan and diagnostics 543
sort_resources and diagnostics 544
triggers and 212
writetext during dump database 641
permissions
assigned by database owner 361
assigning 361
changing with setuser command 375–377
creating with create schema 156–157
for creating triggers 219, 378, 497
grant 360–389
grant dbcc 388
“public” group 375–377
revoke command 489–503
revoke dbcc 502
physical reads (statistics io) 560
physname option
disk init 270
disk reinit 281
placeholders
print message 463
plan
create procedure and 131
set showplan on and 543
set sort_resources on and 544
plans
creating with create plan 127
pointers
text or image column 475
pointers, device. See segments
pound sign (#) temporary table name prefix 165
precedence
order-sensitive commands and 377, 496
rule binding 154
of user-defined return values 488
Index

preference, uppercase letter sort order 459
prefetch keyword
delete 261
select 518
set 555
update 603
prepare transaction command 462
primary key constraint
alter table 29
create table 169
primary keys 192
updating 208
primary option, disk unmirror 289
print command 463–465
local variables and 252
using raiserror or 465
printing user-defined messages 463–465
privileges. See permissions.
procedure groups 300, 347
procedure option
create existing table 95
procedure plan, create procedure and 131
procedures. See stored procedures; system procedures.
process logical name. See logical device name.
process_limit_action option, set 556
processes (server tasks)
See also servers
ID number 416
infected, waitfor errexit 629
killing 416–417
sp_who report on 416
processexit keyword, waitfor 628
protection system
command and object permissions 375
hierarchy of roles, groups and users 381
stored procedures 140
user-defined roles 150
proxy option, set 556
granting 363, 388
revoke 491
proxy tables
mapping to remote tables 95
mapping to remote tables with create proxy_table 145
mapping to remote tables with create table 200
“public” group 381, 498
See also groups
grant and 362
permissions 375–377
revoke and 492
public keyword
grant 362
revoke 492

Q
queries
compilation without execution 543
execution settings 538–585
keywords list 551
syntax check (set parseonly) 543
trigger firing by 211
union 596–599
views and 224
with/without group by and having 394
query analysis
set noexec 543
set statistics io 560
set statistics time 560
query plans
set showplan on and 543
query processing
set options for 538
question marks (?) for partial characters 477
quiesce database
encryption 468
quiesce database command 466–469
quotation marks (“ ”) literal specification of 635
quoted_identifier option, set 557

R
raiserror command 470–474
compared to print 474
local variables and 252
restricted_select_list parameter 471–472
using print or 465
range with set rowcount 558
Index

692

ratio of filled to empty pages 24
read Interactive SQL command 662
read-only cursors 257
readpast option
delete command 261
isolation levels and 534
readtext command 475
select command 516
update command 603
writetext command 639
readtext command 475–479
rebuild option, reorg command 483
rebuild_text option, dbcc 236
rebuilding
automatic, of nonclustered index 118
indexes 237
system tables 236, 239
text and image data 236
reclaim_space option, reorg command 484
recompilation
create procedure with recompile option 131, 135
execute with recompile option 348
stored procedures 135
reconfigure command 480
recovery
dump transaction and 340
to specified time in transaction log 439
time and checkpoint 61
recovery of master database 320
after using create database 86
after using disk init 274
re-creating
indexes 237
procedures 138
tables 306
text and image data 236
recursions, limited 216
reducing storage fragmentation 24
reference information
Transact-SQL commands 1–6
references constraint
alter table 32
create table 172
referencing, object. See dependencies, database object.
referential integrity constraints 24, 193, 318
create table and 190
cross-database 195, 307
referential integrity, triggers for 206–220
regulations in sort order ties 459–460
reindex option, dbcc 237
reinitializing, disk reinit and 281–284
remIRRoring. See disk mirroring.
remote procedure calls 529
execute and 351
rollback and 505
remote procedures, defining 99
remote servers 529
cOnstraints for 28, 33
remove java command 481–482
remove option, disk unmirror 289
removing. See dropping; deleting.
renaming
identity of object owner 375
stored procedures 135
triggers 213
views 224
reorg command 483–485
repairing a damaged database 234
repeatable reads isolation level 523
repeated execution. See while loop.
replace keyword, alter table 34
reports
sp_who 416
types of dbcc 238
reserved return status values 487
reservepagegap option
alter table 31, 45
create index 111, 122
create table 173, 199
select into 516
restarting while loops 78
restarts, Server
after using disk refit 280
before using create database 84
using dataserver utility 278, 286
restoring
See also recovery
a damaged master database 280
damaged master databases 280, 281
database with load database 418–432
results
See also output
of aggregate operations 393

cursor result set 257, 353

order by and sorting 456–461

resume option, reorg 484

retain option, disk unmirror 289

retaindays option

dump database 315

dump transaction 334

retrieving

error message text 463

return command 486–488

return parameters

output keyword 131, 347

return status

stored procedure 346, 486

revoke command 489–503

object and command permissions 375

public group and 492

revoke dbcc

described 489

eamples 493

parameters 502

syntax 489

uses 498

revoking

create trigger permission 219, 378, 497

default permissions from system tables 384

role privileges using with override 302

role option

grant 363

revoke 492

set command 558

roles

adding passwords to 19

creating (user-defined) 149

dropping passwords from 19

granting 380

mutually exclusive 19

permissions and 381

stored procedure permissions and 380

turning on and off with set role 558

roles, system

revoking 492

roles, user-defined

limitations 150

revoking 492

turning on and off 558

rollback command 504–505

begin transaction and 59

commit and 65

triggers and 213, 216

rollback transaction command. See rollback command.

rollback trigger command 213, 506

rollback work command. See rollback command.

rolling back processes

checkpoint and 61

parameter values and 351

dow aggregates

compute and 66

dow length 40

dow size 40

@@rowcount global variable 581

set nocount and 581

triggers and 214

rowcount option, set 558

rows, table

See also select command

aggregate functions applied to 393

comparison order of 459

create index and duplication of 108, 112

deleting unlocked 260

deleting with truncate table 594

displaying command-affected 543

grouping 390

insert 408

rowcount setting 558

scalar aggregates applied to 393

selecting unlocked 532, 533

update 602

updating unlocked 602

ways to group 393

rules

binding 154

column definition conflict with 154

creating new 152–155

default violation of 90

dropping user-defined 304

insert and 409

naming user-created 152

running a procedure with execute 346
Index

S

save transaction command 507–508
savepoints
See also checkpoint process
rollback and 504
setting using save transaction 508
scalar aggregates

  group by and 393
  @ scan_parallel_degree global variable 581
  set scan_parallel_degree and 558
scan_parallel_degree option, set 558
scans
  cursor 257
  number of (statistics io) 560
search conditions
  group by and having query 391, 395
  select 520
  where clause 630–636
secondary option, disk unmirror 289
security
See also permissions
  command and object permissions 375
  views and 224
seed values and set identity_insert 548
segments
See also database devices; log segment; space allocation
  changing table locking schemes 55
  clustered indexes on 118
  creating indexes on 32, 113, 118, 171, 174, 514
  dbcc checktable report on 233
  dbcc indexalloc report on 235
  mapping to a new device 11
  names of 32, 35, 171, 174, 514, 515
  number of named 85
  placing objects on 113
  separation of table and index 118, 188
select command 509–537
selecting
  unlocked rows 532
selecting unlocked rows 532
self_recursion option, set 216, 543
sentence order and numbered placeholders 463
separation, physical
  of table and index segments 118, 188
  of transaction log device 277, 286
sequence. See order by clause; sort order
serial option, disk mirror 276
server process ID number. See processes (server tasks).
servers
See also processes (server tasks); remote servers
  capacity for databases 85
services
  creating new 158–162
  dropping 305
session authorization option, set 558
  revoking 363, 388, 491
set command 538–585
See also individual set options
default settings 572
  inside a stored procedure 139
  inside a trigger 213
lock wait 550
  roles and 558
statistics simulate 560
strict_dtm_enforcement 560
transaction isolation level 563
  within update 603

group by and having clauses 390
insert and 409
local variables and 252
select * syntax features 535
size of text data to be returned with 561
top n and 512
variables and 251
select into command 513–531
  not allowed with compute 71, 522
select into/bulkcopy/pllsort database option
  select into and 530
  transaction log dumping and 337
select list 471–472, 512–513
  order by and 521
  union statements 598
select option, create view 221
selecting
  unlocked rows 533
  selecting unlocked rows 532
self_recursion option, set 216, 543
sentence order and numbered placeholders 463
separation, physical
  of table and index segments 118, 188
  of transaction log device 277, 286
sequence. See order by clause; sort order
serial option, disk mirror 276
server process ID number. See processes (server tasks).
servers
See also processes (server tasks); remote servers
  capacity for databases 85
services
  creating new 158–162
  dropping 305
session authorization option, set 558
  revoking 363, 388, 491
set command 538–585
See also individual set options
default settings 572
  inside a stored procedure 139
  inside a trigger 213
lock wait 550
  roles and 558
statistics simulate 560
strict_dtm_enforcement 560
transaction isolation level 563
  within update 603

694

Adaptive Server Enterprise
set connection Interactive SQL command 664
set option Interactive SQL command 665
set options
  exportable 584
setting identity gap 198
setuser command 588–589
  user impersonation using 375
severity levels, error, in user-defined messages 473
share option, lock table 445
shared keyword
  select 519
shared statistics, deleting 268
showplan option, set 543
shutdown command 590–592
side option, disk unmirror 289
size
  columns in table 40
  compiled stored procedure 135
  composite index 109
  database extension 8
  estimation of a compiled stored procedure 135
  image data to be returned with writetext 640
  new database 81
readtext data 475, 477
recompiled stored procedures 135
row 40
set textsize function 561
tables 183
text data to be returned with select 561
text data to be returned with writetext 640
transaction log device 86
size limit
  columns allowed per table 183
  print command 464
  tables per database 183
size option
  disk init 271, 282
skip rcindex option, dbcc 232
sort operations (order by), sorting plan for 544
sort order
  See also order
  ascending 456
  choices and effects 458
  descending 456
  group by and having and 401
  groups of names 401
order by and 458
rebuilding indexes after changing 237
specifying index with alter table 44
specifying index with create index 119
specifying index with create table 189
sort_resources option, set 544
sp_bindefault system procedure
  create default and 89
sp_bindrule system procedure
  create rule and 153
sp_dboption system procedure
  checkpoints and 61
spdepends system procedure 187
sp_transactions system procedure 234
sp_unbindefault system procedure 294
sp_unbindrule system procedure
  create rule and 153
  drop rule and 304
space
  See also size; space allocation
  adding to database 7–12
  for a clustered index 30, 110, 118, 170
  clustered indexes and max_rows_per_page 31, 111
  database storage 30, 110, 118, 170
  dbcc checktable reporting free 233
  extents 118, 183
  extents for indexes 236
  for index pages 30, 110, 170
  max_rows_per_page and 31, 111, 171
  new database 81
  for recompiled stored procedures 135
  required for alter table...lock 55
  required for reorg rebuild 485
  retrieving inactive log 332
  running out of 332
  for stored procedures 134
  used on the log segment 233, 332
space allocation
  dbcc commands for checking 231–235
  log device 86
  pages 238
  table 183, 231
space management properties
  create index and 122
  create table and 198
Index

space reclamation
  reorg reclaim_space for 483
spaces, character
  update of 608
speed (server)
  create database for load 86
  create index with sorted_data 112
  dump transaction compared to dump database 340
  execute 350
  truncate table compared to delete 594
  writetext compared to dbwritetext and dbmoretext 641
SQL derived tables
  create view command and 223
  creating views from 227
SQL standards
  set options for 584
  set session authorization and 558
SQLJ stored procedures
  creating 141–144
  square brackets [ ] in SQL statements xiv
standby_access option
  dump transaction 336
  online database 452
start logging Interactive SQL command 666
starting servers
  disk mirroring of master device and 278
  disk remirroring of master device and 286
startserver utility command
  See also Utility Programs manual
  disk mirror and 278
  disk remirror and 286
statements
  create trigger 207
  in create procedure 131
statistics
  deleting table and column with delete statistics 268
  generating for unindexed columns 617, 621
  simulated, loading 269, 581
  update all statistics on 614–615
statistics clause, create index command 113
statistics io option, set 560
statistics simulate option, set command 560
statistics subquerycache option, set 560
statistics time option, set 560
status
  stored procedures execution 351
stop logging Interactive SQL command 667
stopping
  procedures. See return command
  stopping servers 590
storage fragmentation, reducing 24
stored procedure triggers. See triggers
stored procedures
  creating 129–140
  dropping 129, 300–301
  dropping groups 300
  executing 346
  grouping 129, 347
  naming 129
  nesting 135, 351
parseonly not used with 543
permissions granted 361
permissions revoked 490
renaming 135
return status 137–138, 346, 351, 486
set commands in 538
storage maximums 134
strict dtm enforcement configuration parameter 560
strict_dtm_enforcement option, set command 560
string_rtruncation option, set 561
  insert and 409
  update and 608
strings
  print message 463
  truncating 409, 608
stripe on option
  dump database 314
  dump transaction 334
  load database 421
  load transaction 435
structure
  See also order
  clustered and nonclustered index 108
subgroups, summary values for 70
subqueries
  order by and 458
  union prohibited in 599
summary values
  generation with compute 70
suspect indexes. See reindex option, dbcc.
Index

Reference Manual: Commands

suspect partitions, in cross-platform dump and load 426
suspending databases 466
switch option, set 561
syb_identity keyword
select and 531
sybsecurity database, dropping 293
symbols in SQL statements xiii, xiv
synonyms
chars for characters, readtext 476
out for output 131, 347
tran, transaction, and work, commit command 64
tran, transaction, and work, rollback command 504
syntax
check using set parseonly 543
grant dbcc 360
revoke dbcc 489
syntax conventions, Transact-SQL xiii
syscolumns table 232
syscomments table
default definitions in 90
procedure definitions in 139
rule definitions in 154
trigger definitions in 218, 227
sysconfigures table
database size parameter 84
sysdevices table
disk init and 273
mirror names in 289
sysindexes table
composite indexes and 124
syslogs table
See also recovery; transaction logs
put on a separate device 278, 286
running dbcc checktable on 233
sysmessages table
raiserror and 470
sysobjects table
trigger IDs and 218
sysprocedures table
trigger execution plans in 218
sysprotects table
grant/revoke statements and 378, 496
sp_changegroup and 381

sysservers table
Backup Server and 321, 342
load database and 428
sysstatistics table, removing statistics with delete statistics 268
system activities
setting query-processing options for 538–585
shutdown 590
system databases, dumping 320
system Interactive SQL command 668
system logical name. See logical device name
system messages
See also error messages; messages
language setting for 546
individual procedure names
Reference Manual
Procedures
system procedures
create procedure (SQLJ) and 141–144
create procedure and 129–140
dropping user-defined 300–301
See also create procedure (SQLJ) command/ 141
See also individual procedure names/ 129
system roles
revoking 492
stored procedures and 380
system segment and alter database 11
system tables
See also tables; individual table names
affected by drop table 306
affected by drop view 310
dbcc checkcatalog and 232
default definitions in 90
fixing allocation errors found in 236, 239
lock table prohibited on 446
rebuilding of 236, 239
rule information in 153
triggers and 212
systransactions table 234
sysusermessages table
raiserror and 470

T
table option in create table 174
Index

table pages
  allocation with dbcc tablealloc 238

**tablealloc** option, dbcc 238

**table** option, creating
  allowed in a from clause 517
  changing 24–57
  creating duplicate 531
  creating new 163–205, 513
  creating with create schema 156–157
  creating with identity column 198

**dbcc checkdb** and 232
  dividing, with **group by** and **having** clauses 390–402
  dropping 306–308
  external 145
  index location 298, 617, 621
  migration to a clustered index 118, 188
  with no data 531
  object allocation maps of 238
  partitioning 24, 34, 48–49
  permissions on 361
  permissions revoked 490
  proxy 95
  single-group 394
  Transact-SQL extension effects and querying 395
  unpartitioning 24, 34
  **update statistics** on 625–626

tape labels
  **listonly** option to **load database** 422
  **listonly** option to **load transaction** 436

tempdb database
  adding objects to 188
  sysobjects table and 188
  systypes table and 188

tempdbs
  create database usage 85
  dbcc pravailabletempdbs and 236

temporary tables
  create procedure and 139
  create table and 165, 187
  identifier prefix (#) 165
  lock table prohibited on 446
  naming 188

text datatype
  initializing with update 608

length of data returned 528, 561

**order by** not allowed 458

storage on separate device 475
**textsize** setting 561
triggers and 212
text pointer values and **readtext** 475
**textptr** function 475, 477
@@textsize global variable 581
**readtext** and 477
set **textsize** and 561
**textsize** option, set 561

thresholds
  database dumps and 320
  transaction log dumps and 341

ties, regulations for sort order 459–460
time interval
  See also timing
  automatic checkpoint 61
  elapsed execution (**statistics time**) 560
  **reorg** 484
  for running a trigger 212
  **waitfor** 628
time option
  **reorg** 484
  **waitfor** 628
timesteps, order of transaction log dumps 427
timing
  See also time interval
  automatic checkpoint 61

to option
  dump database 312
  dump transaction 332
  **revoke** 496
totals, obtaining with **compute** command 458

tracefile option, set 561
@@tranchained global variable 581
**transaction isolation level** option, set 563

transaction isolation levels
  **readpast** option and 534

transaction logs
  See also **dump transaction** command; **syslogs** table
  backing up 311
  of deleted rows 264
  dump database and 311
  dumping 330
  inactive space 332
  loading 433–444
  master database 320, 339
Index

Reference Manual: Commands

placing on separate segment 339
purging 320
on a separate device 274, 277, 286, 337
space extension 11
space, monitoring 341
syslogs table \texttt{trunc log on chkpt} 337
\texttt{writetext with log} and 639
\texttt{transactional_rpc} option, \texttt{set} 563
transactions
  \textit{See also} batch processing; \texttt{rollback} command;
  user-defined transactions
\texttt{begin} 59
canceling. \textit{See} \texttt{rollback} command.
chained 65
committing in prepare state 246
\texttt{dump transaction} command 330–345
ending with \texttt{commit} 64
\texttt{fetch} and 354
isolation levels 563
parameters not part of 351
preparing 462
\texttt{save transaction} and 507–508
\texttt{update} iteration within given 607
Transact-SQL commands
  \texttt{executing} 346
  extensions for 395
  summary table 1–6
translation of arguments 463
trigger tables 213
triggers
  creating 206–220, 378, 497
  \texttt{delete} and 264
dropping 309
enabling self-recursion 216
\texttt{insert} and 409
nesting 215–216
nested, and \texttt{rollback trigger} 506
\texttt{@@nestlevel} and 216
on \texttt{image} columns 212
on \texttt{text} columns 212
\texttt{parseonly} not used with 543
recursion 216
renaming 213
\texttt{rollback} in 213, 505
\texttt{rolling back} 506
\texttt{@@rowcount} and 214
self-recursion 216
\texttt{set} commands in 538
stored procedures and 216
system tables and 212
time interval 212
\texttt{truncate table} command and 595
\texttt{update} and 605
\texttt{truncate table} command 594–595
\texttt{delete} triggers and 213
faster than \texttt{delete} command 263
\texttt{truncate_only} option, \texttt{dump transaction} 332, 339
truncation
datatypes with no length specified 130
default values 90
\texttt{insert} and 409
log, prohibited on mixed device 82
\texttt{set string_rtruncation} and 561
spaces to a single space 608
transaction log 330
\textbf{U}
unbinding
  defaults 90, 294
  rules 304
unconditional branching to a user-defined label 359
undoing changes. \textit{See} \texttt{rollback} command.
\texttt{union} command, changes 597
\texttt{union} operator 596–599
  maximum number of tables 597
  restrictions on use 599
unique constraints 192
\texttt{unique} keyword
  \texttt{alter table} 29
  \texttt{create index} 108
  \texttt{create table} 169
unload option
  \texttt{dump database} 314
  \texttt{dump transaction} 334
  \texttt{load database} 421
  \texttt{load transaction} 436
unloading compressed backups 420, 434
unmirroring devices. \textit{See} disk mirroring
\texttt{unmount}
  encryption 601
Index

See also mount
   See also quiesce database
unpartition clause, alter table 34
unpartitioning
   tables 24
updateable cursors 257
update all statistics command 614–615
update and delete using worktables 263, 605
update command 602–613
   ignore_dup_key and 112
   ignore_dup_row and 120
insert and 408
readpast option 603
triggers and 212
triggers and if update 214
views and 226, 611
update index statistics command 616–618
update statistics command 619–624
   create index and 117
   locking during 622
   scan type 622
   sort requirements 622
update table statistics command 625–626
update, cannot use with scrollable cursor 609
updating
   data in views 225
   “dirty” pages 61–62
   ignore_dup_key and 112
   primary keys 208
   trigger firing by 216
   unlocked rows 602
writetext 639
uppercase letter preference 459
us_english language, weekdays setting 573
usage
   grant dbcc 379
   revoke dbcc 498
use command 627
user errors. See errors; severity levels.
user groups. See groups; “public” group.
user keyword
   alter table 27
   create table 166
user permissions. See database owners; permissions
   user-defined procedures
      creating 129–140
executing 346
user-defined roles
   adding passwords to 19
   conflicting 21
   creating 149
   revoking 492
   system procedures and 380
   turning on and off 558
user-defined SQLJ procedures
   creating 141–144
user-defined transactions
   See also transactions
      begin transaction 59
      ending with commit 64
users
   guest permissions 381
   impersonating (setuser) 375
   system procedure permissions and 377
using option, writetext 476, 477
using...values option, update statistics command 616, 620

V
values
   IDENTITTY columns 411
   procedure parameter or argument 347
values option, insert 406
varchar datatype
   spaces in and insert 409
variable-length columns
   empty strings in 409
   stored order of 459
variables
   assigning as part of a select list 513
   in update statements 606
   local 251–252
   in print messages 463
   return values and 350
vdevno option
   disk init 270, 281
vector aggregates
   group by and 393
   @@version global variable 464
views
See also database objects; multitable views allowed in a from clause 517
changes to underlying tables of 225
check option and 610–611
creating 221–229
creating with create schema 156–157
dropping 310
inserting data through 412
permissions on 361
permissions revoked 490
readtext and 477
renaming 224
update and 226, 610–611
updating restrictions 611
with check option 226, 412–413
violation of domain or integrity rules 409
virtual device number 270, 281
volume names, database dumps 324

W
wait option, lock table command 445
wait option, shutdown 590
waitfor command 628–629
waiting for shutdown 590
weekday date value
names and numbers 546
where clause 630–636
aggregate functions not permitted in delete 635
group by clause and 395
having and 635
repeating a 398
where current of clause
delete 262
update 604
while keyword 637–638
continue and 78
exiting loop with break 60
loops 637
with check option option
create view 222
views and 227
with consumers clause, create index 111

with consumers option, update statistics command 617, 620
with dbid keyword
create database command 82
with default_location keyword
create database command 82
with grant option option, grant 362
with keyword
rollback trigger 506
set role command 558
with log option, writetext 639
with no recovery 430
with no_error option, set char_convert 544
with no_log option, dump transaction 332
with no_truncate option, dump transaction 335
with nowait option, shutdown 590
with override keyword
alter database 8
create database command 82
with override option 302
with recompile option
create procedure 131
execute 348
with resume option, reorg 484
with standby_access option
dump transaction 336
with statistics clause, create index command 113
with time option, reorg 484
with truncate_only option, dump transaction 332, 339
with wait option, shutdown 590
work session, set options for 538–585
worktables in update and delete commands 263, 605
write operations
logging text or image 639
writes option, disk mirror 276
writetext command 639–641
triggers and 213

X
X/Open XA 234
Index

Z
zero-length string output 465