SYBASE®

Performance and Tuning Guide

Sybase® IQ

12.7

DOCUMENT ID: DC00169-01-1270-01

LAST REVISED: June 2006

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Subject

This book presents performance and tuning recommendations.

Audience

This guide is for system and database administrators who need to understand performance issues. Familiarity with relational database systems and introductory user-level experience with Sybase IQ is assumed. Use this guide in conjunction with other manuals in the documentation set.

How to use this book

The following list shows which chapters fit a particular interest or need. To read about

- Structure SELECT statements, see Chapter 1, “Selecting Data from Database Tables.”
- Composing joins, see Chapter 2, “Joining Tables.”
- Optimizing queries, see Chapter 3, “Optimizing Queries and Deletions.”
- Adjusting memory, disk I/O and CPUs, see Chapter 5, “Managing System Resources.”
- Performance, see Chapter 6, “Monitoring and Tuning Performance.”
- Windows performance, see Chapter 7, “Tuning Servers on Windows Systems.”

Related documents

The Sybase IQ document set consists of these documents:

- *Introduction to Sybase IQ* – contains information and exercises for users unfamiliar with Sybase IQ and with the Sybase Central™ database management tool.
- *New Features in Sybase IQ 12.7* – includes a brief description of new features in Sybase IQ.
- *Sybase IQ Performance and Tuning Guide* – describes query optimization, design, and tuning issues for very large databases.
- *Sybase IQ System Administration Guide* – describes administrative concepts, procedures and performance tuning recommendations supported by Sybase IQ, including how to manage the IQ Store.

- *Sybase IQ Troubleshooting and Recovery Guide* – Shows how to solve problems and perform system recovery and database repair.
- *Sybase IQ Error Messages* – refers to Sybase IQ error messages referenced by SQLCode, SQLState, and Sybase error code, and SQL preprocessor errors and warnings.
- *Sybase IQ Utility Guide* – contains Sybase IQ utility program reference material, such as available syntax, parameters, and options.
- *Large Objects Management in Sybase IQ* – describes storage and retrieval of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) within the Sybase IQ data repository. You need a separate license to install this product option.
- *Sybase IQ Installation and Configuration Guide* – contains platform-specific instructions on installing Sybase IQ, migrating to a new version of Sybase IQ, and configuring Sybase IQ for a particular platform.
- *Sybase IQ Release Bulletin* – contains last-minute changes to the product and documentation.
- *Encrypted Columns in Sybase IQ* – describes the use of user encrypted columns within the Sybase IQ data repository. You need a separate license to install this product option.

Sybase IQ and Adaptive Server Anywhere

Because Sybase IQ is an extension of Adaptive Server® Anywhere, a component of SQL Anywhere® Studio, Sybase IQ supports many of the same features as Adaptive Server Anywhere. The Sybase IQ documentation set refers you to SQL Anywhere Studio documentation where appropriate.

Documentation for Adaptive Server Anywhere:

- *Adaptive Server Anywhere Programming Guide* – Intended for application developers writing programs that directly access the ODBC, Embedded SQL™, or Open Client™ interfaces, this book describes how to develop applications for Adaptive Server Anywhere.
- *Adaptive Server Anywhere Database Administration Guide* – Intended for all users, this book covers material related to running, managing, and configuring databases and database servers.

- *Adaptive Server Anywhere SQL Reference Manual* – Intended for all users, this book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.

You can also refer to the Adaptive Server Anywhere documentation in the SQL Anywhere Studio 9.0.2 collection on the Sybase Product Manuals Web site. To access this site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Other sources of information

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/.
- Infocenter is an online version of SyBooks that you can view using a standard Web browser. To access the Infocenter Web site, go to Sybooks Online Help at http://infocenter.sybase.com/help/index.jsp.

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 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.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
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.

Syntax conventions

This documentation uses the following syntax conventions in syntax descriptions:

- **Keywords** SQL keywords are shown in UPPER CASE. However, SQL keywords are case insensitive, so you can enter keywords in any case you wish; SELECT is the same as Select which is the same as select.
- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.
- **Continuation** Lines beginning with … are a continuation of the statements from the previous line.
- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots). One or more list elements are allowed. If more than one is specified, they must be separated by commas.
- **Optional portions** Optional portions of a statement are enclosed by square brackets. For example:

  ```
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  It indicates that the *savepoint-name* is optional. The square brackets should not be typed.
- **Options** When none or only one of a list of items must be chosen, the items are separated by vertical bars and the list enclosed in square brackets. For example:

  ```
  [ ASC | DESC ]
  ```

  It indicates that you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces. For example:

  ```
  QUOTES { ON | OFF }
  ```

  It indicates that exactly one of ON or OFF must be provided. The braces should not be typed.

Typographic conventions

Table 1 lists the typographic conventions used in this documentation.

Table 1: Typographic conventions

Item	Description
`Code`	SQL and program code is displayed in a mono-spaced (fixed-width) font.
`User entry`	Text entered by the user is shown in bold serif type.
emphasis	Emphasized words are shown in italic.
file names	File names are shown in italic.
database objects	Names of database objects, such as tables and procedures, are shown in bold, san-serif type in print, and in italic online.

The sample database

Sybase IQ includes a sample database used by many of the examples in the Sybase IQ documentation.

The sample database represents a small company. It contains internal information about the company (employees, departments, and financial data), as well as product information (products), sales information (sales orders, customers, and contacts), and financial information (fin_code, fin_data).

The sample database is held in a file named *asiqdemo.db*, located in the directory *$ASDIR/demo* on UNIX systems and *%ASDIR%\demo* on Windows systems.

Accessibility features

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.

Sybase IQ 12.7 and the HTML documentation have 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.

For information about accessibility support in the Sybase IQ plug-in for Sybase Central, see “Using accessibility features” in *Introduction to Sybase IQ*. The online help for this product, which you can navigate using a screen reader, also describes accessibility features, including Sybase Central keyboard shortcuts.

Configuring your accessibility tool
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 and see “Using screen readers” in *Introduction to Sybase IQ*.

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.

For a Section 508 compliance statement for Sybase IQ, go to Sybase Accessibility at http://www.sybase.com/products/accessibility.

If you need help

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

Selecting Data from Database Tables

About this chapter

This chapter reviews basic query construction and recommends refinements to take advantage of product design. In this tutorial, you will look at table contents, order query results, select columns and rows, and use search conditions to refine queries.

For advanced query performance recommendations, see Chapter 3, "Optimizing Queries and Deletions."

Contents

Prerequisites

If you use a graphical front-end tool instead of DBISQL to query your database, the tool may allow you to view the SQL syntax it generates. For example, in InfoMaker, you can view SQL statements by choosing the SQL Syntax button on the Table painter bar.

This tutorial introduces the SELECT statement used to retrieve information from databases. SELECT statements are commonly called queries, because they ask the database server about information in a database.

Note The SELECT statement is a versatile command. SELECT statements can become highly complex in applications retrieving very specific information from large databases. This tutorial uses only simple SELECT statements: later tutorials describe more advanced queries. For more information about the full syntax of the select statement, see the SELECT statement in Chapter 6, “SQL Statements,” in the *Sybase IQ Reference Manual*.

Ideally, you should be running Sybase IQ software on your computer while you read and work through the tutorial lessons.

This tutorial assumes that you have already started DBISQL and connected to the sample database. If you have not already done so, see Chapter 2, “Using Interactive SQL (dbisql)” in the *Sybase IQ Utility Guide*.

Viewing table information

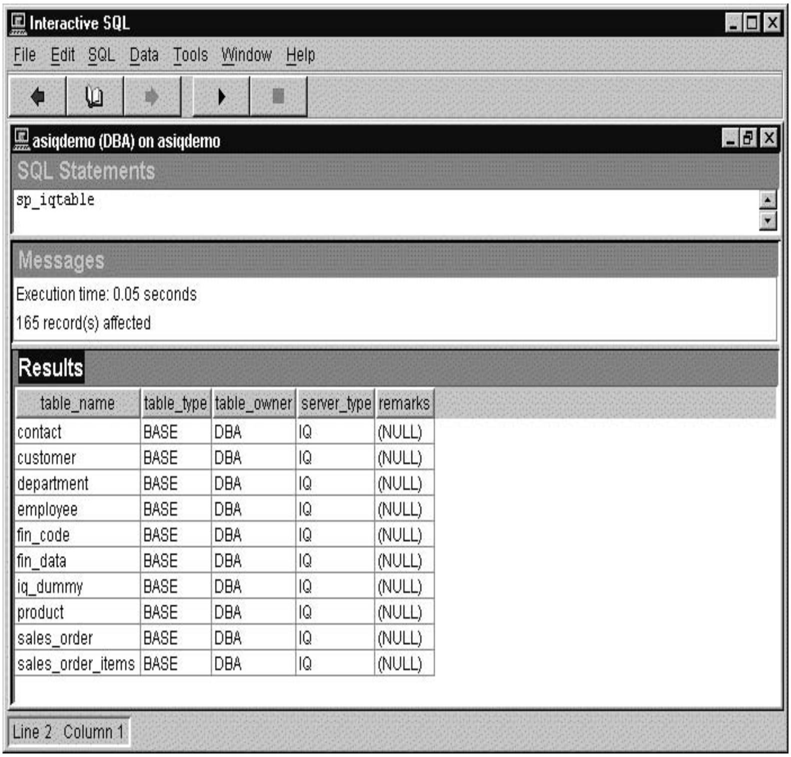
In this section, you will look at the data in the *employee* table.

The sample database you use in this tutorial is the same fictional company as in the previous chapter. The database contains information about employees, departments, sales orders, and so on. All the information is organized into tables.

Listing tables

In *Introduction to Sybase IQ*, you learned how to display a list of tables by opening the Tables folder in Sybase Central. You can also list user tables from interactive SQL using a system stored procedure, sp_iqtable. System stored procedures are system functions that are implemented as stored procedures in Sybase IQ.

In the SQL Statements window, type `sp_iqtable` to run the system stored procedure of the same name.



For complete details about this and other system stored procedures, see Chapter 10, “System Procedures,” in the *Sybase IQ Reference Manual*.

Using the SELECT statement

In this lesson, you view one of the tables in the database. The command used will look at everything in a table called employee.

Execute the command:

```
SELECT * FROM employee
```

The asterisk is a short form for all the columns in the table.

The SELECT statement retrieves all the rows and columns of the employee table, and the DBISQL Results window lists those that will fit:

emp_id	manager_id	emp_fname	emp_lname	dept_id
102	501	Fran	Whitney	100
105	501	Matthew	Cobb	100

emp_id	manager_id	emp_fname	emp_lname	dept_id
129	902	Philip	Chin	200
148	1293	Julie	Jordan	300
160	501	Robert	Breault	100

The employee table contains a number of **rows** organized into **columns**. Each column has a name, such as emp_lname or emp_id. There is a row for each employee of the company, and each row has a value in each column. For example, the employee with employee ID 102 is Fran Whitney, whose manager is employee ID 501.

You will also see some information in the DBISQL Messages window. This information is explained later.

Case sensitivity

The table name employee is shown starting with an upper case E, even though the real table name is all lower case. Sybase IQ databases can be created as case-sensitive (the default) or case-insensitive in their string comparisons, but are always case insensitive in their use of identifiers.

Note The examples in this book were created case-insensitive, using the CREATE DATABASE qualifier CASE IGNORE. The default is CASE RESPECT, which gives better performance.

For information on creating databases, see Chapter 5, "Working with Database Objects," *Sybase IQ System Administration Guide*.

You can type select or Select instead of SELECT. Sybase IQ allows you to type keywords in uppercase, lowercase, or any combination of the two. In this manual, uppercase letters are generally used for SQL keywords.

Manipulation of the DBISQL environment and use of DBISQL is specific to the operating system.

For information on how to scroll through data and manipulate the DBISQL environment, see Chapter 2, "Using Interactive SQL (dbisql)" in *Sybase IQ Utility Guide*.

Ordering query results

In this section, you will add an ORDER BY clause to the SELECT statement to display results in alphabetical or numerical order.

Unless otherwise requested, Sybase IQ displays the rows of a table in no particular order. Often it is useful to look at the rows in a table in a more meaningful sequence. For example, you might like to see employees in alphabetical order.

Listing employees in alphabetical order

The following example shows how adding an ORDER BY clause to the SELECT statement causes the results to be retrieved in alphabetical order.

```
SELECT * FROM employee ORDER BY emp_lname
```

emp_id	manager_id	emp_fname	emp_lname	dept_id
1751	1576	Alex	Ahmed	400
1013	703	Joseph	Barker	500
591	1576	Irene	Barletta	400
191	703	Jeannette	Bertrand	500
1336	1293	Janet	Bigelow	300

Notes

The order of the clauses is important. The ORDER BY clause must follow the FROM clause and the SELECT clause.

Note If you omit the FROM clause, or if all tables in the query are in the SYSTEM dbspace, the query is processed by Adaptive Server Anywhere instead of Sybase IQ and may behave differently, especially with respect to syntactic and semantic restrictions and the effects of option settings. See the Adaptive Server Anywhere documentation for rules that may apply to processing.

If you have a query that does not require a FROM clause, you can force the query to be processed by Sybase IQ by adding the clause "FROM iq_dummy," where iq_dummy is a one row, one column table that you create in your database.

Selecting columns and rows

Often, you are only interested in some of the columns in a table. For example, to make up birthday cards for employees you might want to see the emp_lname, dept_id, and birth_date columns.

Listing last name, department, and birth date of each employee

In this section, you will select each employee's birth date, last name, and department ID. Type the following:

```
SELECT emp_lname, dept_id, birth_date
FROM employee
```

emp_lname	dept_id	birth_date	...
Whitney	100	1958-06-05	...
Cobb	100	1960-12-04	...
Chin	200	1966-10-30	...
Jordan	300	1951-12-13	...
Breault	100	1947-05-13	...

Rearranging columns

The three columns appear in the order in which you typed them in the SELECT command. To rearrange the columns, simply change the order of the column names in the command. For example, to put the birth_date column on the left, use the following command:

```
SELECT birth_date, emp_lname, dept_id
FROM employee
```

Ordering rows

You can order rows and look at only certain columns at the same time as follows:

```
SELECT birth_date, emp_lname, dept_id
FROM employee
ORDER BY emp_lname
```

The asterisk in

```
SELECT * FROM employee
```

is a short form for all columns in the table.

Using search conditions

In this section you will learn procedures for comparing dates, using compound search conditions in the WHERE clause, pattern matching, and search condition shortcuts.

Sometimes you will not want to see information on all the employees in the employee table. Adding a WHERE clause to the SELECT statement allows only some rows to be selected from a table.

For example, suppose you would like to look at the employees with first name John.

❖ **List all employees named John:**

- Type the following:

```
SELECT *
FROM employee
WHERE emp_fname = 'John'
```

emp_id	manager_id	emp_fname	emp_lname	dept_id
318	1576	John	Crow	400
862	501	John	Sheffield	100
1483	1293	John	Letiecq	300

Apostrophes and case-sensitivity

- The apostrophes (single quotes) around the name 'John' are required. They indicate that John is a character string. Quotation marks (double quotes) have a different meaning. Quotation marks can be used to make otherwise invalid strings valid for column names and other identifiers.
- The sample database is not case sensitive, so you would get the same results whether you searched for ' 'JOHN', 'john', or 'John'.

Again, you can combine what you have learned:

```
SELECT emp_fname, emp_lname, birth_date
FROM employee
WHERE emp_fname = 'John'
ORDER BY birth_date
```

Notes

- How you order clauses is important. The FROM clause comes first, followed by the WHERE clause, and then the ORDER BY clause. If you type the clauses in a different order, you will get a syntax error.

- You do not need to split the statement into several lines. You can enter the statement into the SQL Statements window in any format. If you use more than the number of lines that fit on the screen, the text scrolls in the SQL Statements window.

Comparing dates in queries

Sometimes you will not know exactly what value you are looking for, or you would like to see a set of values. You can use comparisons in the WHERE clause to select a set of rows that satisfy the search condition.

Listing employees born before March 3, 1964

The following example shows the use of a date inequality search condition. Type the following:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date < 'March 3, 1964'
```

emp_lname	birth_date
Whitney	1958-06-05 00:00:00.000
Cobb	1960-12-04 00:00:00.000
Jordan	1951-12-13 00:00:00.000
Breault	1947-05-13 00:00:00.000
Espinoza	1939-12-14 00:00:00.000
Dill	1963-07-19 00:00:00.000

Sybase IQ knows that the birth_date column contains a date, and converts *'March 3, 1964'* to a date automatically.

Compound search conditions in the WHERE clause

So far, you have seen equal (=) and less than (<) as comparison operators. Sybase IQ also supports other comparison operators, such as greater than (>), greater than or equal (>=), less than or equal (<=), and not equal (<>).

These conditions can be combined using AND and OR to make more complicated search conditions.

Qualifying the list

To list all employees born before March 3, 1964, but exclude the employee named Whitney, type:

```
SELECT emp_lname, birth_date
```

```
FROM employee
WHERE birth_date < '1964-3-3'
AND emp_lname <> 'Whitney'
```

emp_lname	birth_date
Cobb	1960-12-04 00:00:00.000
Jordan	1951-12-13 00:00:00.000
Breault	1947-05-13 00:00:00.000
Espinoza	1939-12-14 00:00:00.000
Dill	1963-07-19 00:00:00.000
Francis	1954-09-12 00:00:00.000

Pattern matching in search conditions

Another useful way to look for things is to search for a pattern. In SQL, the word LIKE is used to search for patterns. The use of LIKE can be explained by example.

Listing employees whose surname begins with BR

Type the following:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE emp_lname LIKE 'br%'
```

emp_lname	emp_fname
Breault	Robert
Braun	Jane

The % in the search condition indicates that any number of other characters may follow the letters BR.

Qualifying the surname search

To list all employees whose surname begins with BR, followed by zero or more letters and a T, followed by zero or more letters, type:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE emp_lname LIKE 'BR%T%'
```

emp_lname	emp_fname
Breault	Robert

The first % sign matches the string “eaul”, while the second % sign matches the empty string (no characters).

Another special character that can be used with LIKE is the _ (underscore) character, which matches exactly one character.

The pattern BR_U% matches all names starting with BR and having U as the fourth letter. In Braun the _ matches the letter A and the% matches N.

Matching rows by sound

With the SOUNDEX function, you can match rows by sound, as well as by spelling. For example, suppose a phone message was left for a name that sounded like "Ms. Brown". Which employees in the company have names that sound like Brown?

Searching surnames by sound

To list employees with surnames that sound like Brown, type the following:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE SOUNDEX( emp_lname ) = SOUNDEX( 'Brown' )
```

emp_lname	emp_fname
Braun	Jane

Jane Braun is the only employee matching the search condition.

Shortcuts for typing search conditions

Using the short form BETWEEN

SQL has two short forms for typing in search conditions. The first, BETWEEN, is used when you are looking for a range of values. For example,

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date BETWEEN '1964-1-1'
AND '1965-3-31'
```

is equivalent to:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date >= '1964-1-1'
AND birth_date <= '1965-3-31'
```

Using the short form IN

The second short form, IN, may be used when looking for one of a number of values. The command

```
SELECT emp_lname, emp_id
FROM employee
```

```
WHERE emp_lname IN ('Yeung','Bucceri','Charlton')
```

means the same as:

```
SELECT emp_lname, emp_id
FROM employee
WHERE emp_lname = 'Yeung'
OR emp_lname = 'Bucceri'
OR emp_lname = 'Charlton'
```

Obtaining aggregate data

This section tells how to construct queries that give you aggregate information. Examples of aggregate information are:

- The total of all values in a column
- The number of entries in a column
- The average value of entries in a column

A first look at aggregate functions

Suppose you want to know how many employees there are. The following statement retrieves the number of rows in the employee table:

```
SELECT count( * )
FROM employee
```

count(*)
75

The result returned from this query is a table with only one column (with title count(*)) and one row, which contains the number of employees.

The following command is a slightly more complicated aggregate query:

```
SELECT   count( * ),
min( birth_date ),
max( birth_date )
FROM employee
```

count(*)	min(birth_date)	max(birth_date)
75	1936-01-02	1973-01-18

The result set from this query has three columns and only one row. The three columns contain the number of employees, the birth date of the oldest employee, and the birth date of the youngest employee.

COUNT, MIN, and MAX are called **aggregate functions**. Each of these functions summarizes information for an entire table. In total, there are seven aggregate functions: MIN, MAX, COUNT, AVG, SUM, STDDEV, and VARIANCE. All of the functions have either the name of a column or an expression as a parameter. As you have seen, COUNT also has an asterisk as its parameter.

Using aggregate functions to obtain grouped data

In addition to providing information about an entire table, aggregate functions can be used on groups of rows.

Using an aggregate function on groups of rows

To list the number of orders for which each sales representative is responsible, type:

```
SELECT sales_rep, count( * )
FROM sales_order
GROUP BY sales_rep
```

sales_rep	count(*)
129	57
195	50
299	114
467	56
667	54

The results of this query consist of one row for each sales_rep ID number, containing the sales_rep ID, and the number of rows in the sales_order table with that ID number.

Whenever GROUP BY is used, the resulting table has one row for each different value found in the GROUP BY column or columns.

Restricting groups

You have already seen how to restrict rows in a query using the WHERE clause. You can restrict GROUP BY clauses by using the HAVING keyword.

Restricting GROUP BY clauses

To list all sales reps with more than 55 orders, type:

```
SELECT sales_rep, count( * )
FROM  sales_order
GROUP BY sales_rep
HAVING count( * ) > 55
```

sales_rep	count(*)
129	57
299	114
467	56
1142	57

Note GROUP BY must always appear before HAVING. In the same manner, WHERE must appear before GROUP BY.

Using WHERE and GROUP BY

To list all sales reps with more than 55 orders and an ID of more than 1000, type:

```
SELECT sales_rep, count( * )
FROM sales_order
WHERE sales_rep > 1000
GROUP BY sales_rep
HAVING count( * ) > 55
```

The Sybase IQ query optimizer moves predicates from the HAVING clause to the WHERE clause, when doing so provides a performance gain. For example, if you specify:

```
GROUP BY sales_rep
HAVING count( *) > 55
 AND sales_rep > 1000
```

instead of the WHERE clause in the preceding example, the query optimizer moves the predicate to a WHERE clause.

Sybase IQ performs this optimization with simple conditions (nothing involving OR or IN). For this reason, when constructing queries with both a WHERE clause and a HAVING clause, you should be careful to put as many of the conditions as possible in the WHERE clause.

Improving subtotal calculation

If you have data that varies across dimensions such as date or place, you may need to determine how the data varies in each dimension. You can use the ROLLUP and CUBE operators to create multiple levels of subtotals and a grand total from a list of references to grouping columns. The subtotals "roll up" from the most detailed level to the grand total. For example, if you are analyzing sales data, you can compute an overall average and the average sales by year using the same query.

Using ROLLUP

To select total car sales by year, model and color:

```
SELECT year, model, color, sum(sales)
FROM sales_tab
GROUP BY ROLLUP (year, model, color);
```

year	model	color	sales
1990	Chevrolet	red	5
1990	Chevrolet	white	87
1990	Chevrolet	blue	62
1990	Chevrolet	NULL	154
1990	Ford	blue	64
1990	Ford	red	62
1990	Ford	white	63
1990	Ford	NULL	189
1990	NULL	NULL	343
1991	Chevrolet	blue	54
1991	Chevrolet	red	95
1991	Chevrolet	white	49
1991	Chevrolet	NULL	198
1991	Ford	blue	52
1991	Ford	red	55
1991	Ford	white	9
1991	Ford	NULL	116
1991	NULL	NULL	314
NULL	NULL	NULL	657

When processing this query, Sybase IQ groups the data first by all three specified grouping expressions (year, model, color), then for all grouping expressions except the last one (color). In the fifth row, NULL indicates the ROLLUP value for the color column, in other words, the total number of sales of that model in all colors. 343 represents the total sales of all models and colors in 1990 and 314 is the total for 1991. The last row represents total sales on all years, all models and all colors.

ROLLUP requires an ordered list of grouping expressions as arguments. When listing groups that contain other groups, list the larger group first (such as state before city.)

You can use ROLLUP with the aggregate functions: SUM, COUNT, AVG, MIN, MAX, STDDEV, and VARIANCE. ROLLUP does not support COUNT DISTINCT and SUM DISTINCT, however.

Using CUBE

The following query uses data from a census, including the state (geographic location), gender, education level, and income of people. You can use the CUBE extension of the GROUP BY clause, if you want to compute the average income in the entire census of state, gender, and education and compute the average income in all possible combinations of the columns state, gender, and education, while making only a single pass through the census data in the table census. For example, use the CUBE operator if you want to compute the average income of all females in all states, or compute the average income of all people in the census according to their education and geographic location.

When CUBE calculates a group, CUBE puts a NULL value in the column(s) whose group is calculated. The distinction is difficult between the type of group each row represents and whether the NULL is a NULL stored in the database or a NULL resulting from CUBE. The GROUPING function solves this problem by returning 1, if the designated column has been merged to a higher level group.

The following query illustrates the use of the GROUPING function with GROUP BY CUBE.

```
SELECT
CASE GROUPING ( state ) WHEN 1 THEN 'ALL' ELSE state END
AS c_state,
CASE GROUPING ( gender ) WHEN 1 THEN 'ALL' ELSE gender
END AS c_gender,
CASE GROUPING ( education ) WHEN 1 THEN 'ALL' ELSE
education END AS c_education,
COUNT(*), CAST (ROUND ( AVG ( income ), 2 ) AS NUMERIC
(18,2)) AS average
FROM census
```

```
GROUP BY CUBE (state, gender, education);
```

The results of this query are shown below. Note that the NULLs generated by CUBE to indicate a subtotal row are replaced with ALL in the subtotal rows, as specified in the query.

c_state	c_gender	c_education	count(*)	average
MA	f	BA	3	48333.33
MA	f	HS	2	40000.00
MA	f	MS	1	45000.00
MA	f	ALL	6	45000.00
MA	m	BA	4	55000.00
MA	m	HS	1	55000.00
MA	m	MS	3	85000.00
MA	m	ALL	8	66250.00
MA	ALL	ALL	14	57142.86
NH	f	HS	2	50000.00
NH	f	MS	1	85000.00
NH	f	ALL	3	61666.67
NH	m	BA	3	55000.00
NH	m	MS	1	49000.00
NH	m	ALL	4	53500.00
NH	ALL	ALL	7	57000.00
ALL	ALL	ALL	21	57095.24
ALL	ALL	BA	10	53000.00
ALL	ALL	MS	6	72333.33
ALL	ALL	HS	5	47000.00
ALL	f	ALL	9	50555.56
ALL	m	ALL	12	62000.00
ALL	f	BA	3	48333.33
ALL	m	HS	1	55000.00
ALL	m	MS	4	76000.00
ALL	m	BA	7	55000.00
ALL	f	MS	2	65000.00
ALL	f	HS	4	45000.00
NH	ALL	HS	2	50000.00
NH	ALL	MS	2	67000.00
MA	ALL	MS	4	75000.00
MA	ALL	HS	3	45000.00

c_state	c_gender	c_education	count(*)	average
MA	ALL	BA	7	52142.86
NH	ALL	BA	3	55000.00

Data warehouse administrators find ROLLUP and CUBE particularly useful for operations like:

- Subtotaling on a hierarchical dimension like geography or time, for example year/month/day or country/state/city
- Populating summary tables

ROLLUP and CUBE allow you to use one query to compute data using multiple levels of grouping, instead of a separate query for each level.

See the SELECT statement in Chapter 6, “SQL Statements,”*Sybase IQ Reference Manual*, for more information on the ROLLUP and CUBE operators.

Obtaining analytical data

This section tells how to construct queries that give you analytical information. There are two types of analytical functions: rank and inverse distribution. The rank analytical functions rank items in a group, compute distribution, and divide a result set into a number of groupings. The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data.

The rank analytical functions are RANK, DENSE_RANK, PERCENT_RANK, and NTILE. The inverse distribution analytical functions are PERCENTILE_CONT and PERCENTILE_DISC.

Suppose you want to determine the sale status of car dealers. The NTILE function divides the dealers into four groups based on the number of cars each dealer sold. The dealers with ntile = 1 are in the top 25% for car sales.

```
SELECT dealer_name, sales,
NTILE(4) OVER ( ORDER BY sales DESC )
FROM carSales;

dealer_name         sales       ntile
Boston              1000        1
Worcester           950         1
```

```
Providence          950          1
SF                  940          1
Lowell              900          2
Seattle             900          2
Natick              870          2
New Haven           850          2
Portland            800          3
Houston             780          3
Hartford            780          3
Dublin              750          3
Austin              650          4
Dallas              640          4
Dover               600          4
```

To find the top 10% of car dealers by sales, you specify NTILE(10) in the example SELECT statement. Similarly, to find the top 50% of car dealers by sales, specify NTILE(2).

NTILE is a rank analytical function that distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket. You can divide a result set into tenths (deciles), fourths (quartiles), and other numbers of groupings.

The rank analytical functions require an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. Note that this ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for the SELECT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

Similarly, the inverse distribution functions require a WITHIN GROUP (ORDER BY) clause. The ORDER BY specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is *not* an ORDER BY for the SELECT. The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result.

For more details on the analytical functions, see the section“Analytical functions” in Chapter 5, “SQL Functions” of the *Sybase IQ Reference Manual* For information on individual analytical functions, see the section for each function in the “SQL Functions” chapter.

Eliminating duplicate rows

Result tables from SELECT statements can contain duplicate rows. You can use the DISTINCT keyword to eliminate the duplicates. For example, the following command returns many duplicate rows:

```
SELECT city, state FROM employee
```

To list only unique combinations of city and state, use this command:

```
SELECT DISTINCT city, state FROM employee
```

Note The ROLLUP and CUBE operators do not support the DISTINCT keyword.

This chapter provides an overview of single-table SELECT statements. For more information about single-table SELECT statements, see Chapter 5, "Working with Database Objects," in the *Sybase IQ System Administration Guide*, Chapter 3, "SQL Language Elements," in the *Sybase IQ Reference Manual*, and "SELECT statement" in Chapter 6, "SQL Statements," in the *Sybase IQ Reference Manual*.

Advanced uses of the SELECT statement are described in the next chapter.

CHAPTER 2

Joining Tables

About this chapter

This chapter explains how to look at information in more than one table and describes various types of joins. You will complete tutorial tasks on joining tables.

Contents

Joining tables with the cross product

One of the tables in the sample database is fin_data, which lists the financial data for the company. Each data record has a code column that tells its department and whether it is an expense or revenue record. There are 84 rows in the fin_data table.

You can get information from two tables at the same time by listing both tables, separated by a comma, in the FROM clause of a SELECT query.

Example

The following dbisql SELECT command lists all the data in the fin_code and fin_data tables:

```
SELECT *
FROM fin_code, fin_data
```

The results of this query, displayed in the dbisql data window, match every row in the fin_code table with every row in the fin_data table.This join is called a full cross product, also known as a cartesian product. Each row consists of all columns from the fin_code table followed by all columns from the fin_data table.

The cross product join is a simple starting point for understanding joins, but not very useful in itself. Subsequent sections in this chapter tell how to construct more selective joins, which you can think of as applying restrictions to the cross product table.

Restricting a join

To make a cross product join useful, you need to include only rows that satisfy some condition in the result. That condition, called the join condition, compares one column from one table to one column in the other table, using a comparison operator (=, =>, <, etc.). You thus eliminate some of the rows from the cross product result.

For example, to make the join in the preceding section useful, you could insist that the sales_rep in the sales_order table be the same as the one in the employee table in every row of the result. Then each row contains information about an order and the sales representative responsible for it.

Example 1

To do this, add a WHERE clause to the previous query to show the list of employees and their course registrations:

```
SELECT *
```

```
FROM sales_order, employee
WHERE sales_order.sales_rep = employee.emp_id
```

The table name is given as a prefix to identify the columns. Although not strictly required in this case, using the table name prefix clarifies the statement, and is required when two tables have a column with the same name. A table name used in this context is called a **qualifier**.

The results of this query contain only 648 rows (one for each row in the sales_order table). Of the original 48,600 rows in the join, only 648 of them have the employee number equal in the two tables.

Example 2

The following query is a modified version that fetches only some of the columns and orders the results.

```
SELECT employee.emp_lname, sales_order.id,
sales_order.order_date
FROM sales_order, employee
WHERE sales_order.sales_rep = employee.emp_id
ORDER BY employee.emp_lname
```

If there are many tables in a SELECT command, you may need to type several qualifier names. You can reduce typing by using a correlation name.

Correlation names

A **correlation name** is an alias for a particular instance of a table. This alias is valid only within a single statement. Correlation names are created by putting a short form for a table name immediately after the table name, separated by the keyword AS. You then *must* use the short form as a qualifier instead of the corresponding table name.

```
SELECT E.emp_lname, S.id, S.order_date
FROM sales_order AS S, employee AS E
WHERE S.sales_rep = E.emp_id
ORDER BY E.emp_lname
```

Here, two correlation names S and E are created for the sales_order and employee tables.

Note A table name or correlation name is only needed to resolve ambiguity if two columns of different tables have the same name. If you have created a correlation name, you must use it instead of the full table name, but if you have not created a correlation name, use the full table name.

How tables are related

To construct other types of joins, you must first understand how the information in one table is related to that in another.

The primary key for a table identifies each row in the table. Tables are related to each other using a foreign key.

This section shows how primary and foreign keys together let you construct queries from more than one table.

Rows are identified by a primary key

Every table in the asiqdemo database has a primary key. (It is a good idea to have a primary key for each table.) A primary key is one or more columns that uniquely identify a row in the table. For example, an employee number uniquely identifies an employee—emp_id is the primary key of the employee table.

The sales_order_items table is an example of a table with two columns that make up the primary key. The order ID by itself does not uniquely identify a row in the sales_order_items table because there can be several items in an order. Also, the line_id number does not uniquely identify a row in the sales_order_items table. Both the order ID name and line_id are required to uniquely identify a row in the sales_order_items table. The primary key of the table is both columns taken together.

Tables are related by a foreign key

Several tables in the asiqdemo database refer to other tables in the database. For example, in the sales_order table, the sales_rep column indicates which employee is responsible for an order. Only enough information to uniquely identify an employee is kept in the sales_order table. The sales_rep column in the sales_order table is a foreign key to the employee table.

Foreign key

A foreign key is one or more columns that contain candidate key values from another table. (For more about candidate keys, see Chapter 5, "Working with Database Objects" in *Sybase IQ System Administration Guide*.) Each foreign key relationship in the employee database is represented graphically by an arrow between two tables. You can see these arrows in the diagram of the Sample Database, Figure 1-1 on page 12, in *Introduction to Sybase IQ*. The arrow starts at the foreign key side of the relationship and points to the candidate key side of the relationship.

Join operators

Many common joins are between two tables related by a foreign key. The most common join restricts foreign key values to be equal to primary key values. The example you have already seen restricts foreign key values in the sales_order table to be equal to the candidate key values in the *employee* table.

```
SELECT emp_lname, id, order_date
FROM sales_order, employee
WHERE sales_order.sales_rep = employee.emp_id
```

The query can be more simply expressed using a KEY JOIN.

Joining tables using key joins

Key joins are an easy way to join tables related by a foreign key. For example:

```
SELECT emp_lname, id, order_date
FROM sales_order
KEY JOIN employee
```

gives the same results as a query with a WHERE clause that equates the two employee ID number columns:

```
SELECT emp_lname, id, order_date
FROM sales_order, employee
WHERE sales_order.sales_rep = employee.emp_id
```

The join operator (KEY JOIN) is just a short cut for typing the WHERE clause; the two queries are identical.

In the diagram of the asiqdemo database, in *Introduction to Sybase IQ*, foreign keys are represented by lines between tables. Anywhere that two tables are joined by a line in the diagram, you can use the KEY JOIN operator. Remember that your application must enforce foreign keys in order to ensure expected results from queries based on key joins.

Joining two or more tables

Two or more tables can be joined using join operators. The following query uses four tables to list the total value of the orders placed by each customer. It connects the four tables customer, sales_order, sales_order_items and product single foreign-key relationships between each pair of these tables.

```
SELECT company_name,
CAST( SUM(sales_order_items.quantity *
product.unit_price) AS INTEGER) AS value
FROM customer
KEY JOIN sales_order
KEY JOIN sales_order_items
KEY JOIN product
GROUP BY company_name
```

company_name	value
McManus Inc.	3,156
Salt & Peppers.	4,980
The Real Deal	1,884
Totos Active Wear	2,496
The Ristuccia Center	4,596
...	

The CAST function used in this query converts the data type of an expression. In this example the sum that is returned as an integer is converted to a value.

Joining tables using natural joins

The NATURAL JOIN operator joins two tables based on common column names. In other words, Sybase IQ generates a WHERE clause that equates the common columns from each table.

Example

For example, for the following query:

```
SELECT emp_lname, dept_name
FROM employee
NATURAL JOIN department
```

the database server looks at the two tables and determines that the only column name they have in common is dept_id. The following ON phrase is internally generated and used to perform the join:

```
FROM employee JOIN department
...
ON employee.dept_id = department.dept_id
```

Errors using NATURAL JOIN

This join operator can cause problems by equating columns you may not intend to be equated. For example, the following query generates unwanted results:

```
SELECT *
FROM sales_order
NATURAL JOIN customer
```

The result of this query has no rows.

The database server internally generates the following ON phrase:

```
FROM sales_order JOIN customer
    ON sales_order.id = customer.id
```

The id column in the sales_order table is an ID number for the order. The id column in the customer table is an ID number for the customer. None of them matched. Of course, even if a match were found, it would be a meaningless one.

You should be careful using join operators. Always remember that the join operator just saves you from typing the WHERE clause for an unenforced foreign key or common column names. Be mindful of the WHERE clause, or you may create queries that give results other than what you intend.

Ad hoc joins vs. using join indexes

If you have defined join indexes on the join columns referenced in your query, Sybase IQ will automatically use them to make the query process faster. (For information about defining join indexes, see Chapter 6, “Using Sybase IQ Indexes,”in the *Sybase IQ System Administration Guide*.)

Any join that does not use join indexes is known as an **ad hoc join**. If several tables are referenced by the query, and not all of them have join indexes defined, Sybase IQ will use the join indexes for those tables that have them in combination with an ad hoc join with the rest of the tables.

Because you cannot create join indexes for all possible joins, ad hoc joins may sometimes be necessary. Thanks to optimizations in Sybase IQ, you may find that queries perform as well or better without join indexes.

Keep these rules in mind when creating join indexes:

- Only full outer joins are supported in the index. The query can be an inner, left outer, or right outer join if indexed.

  A full outer join is one where *all* rows from both the left and right specified tables are included in the result, with NULL returned for any column with no matching value in the corresponding column.

- The only comparison operator that may be used in the join predicate ON clause is EQUALS.
- You can use the NATURAL keyword instead of an ON clause, but you can only specify one pair of tables.
- Join index columns must have identical data type, precision, and scale.

Joins and data types

Join columns require like data types for optimal performance. Sybase IQ allows you to make an ad hoc join on any data types for which an implicit conversion exists. Unless join column data types are identical, however, performance can suffer to varying degrees, depending on the data types and the size of the tables. For example, while you can join an INT to a BIGINT column, this join prevents certain types of optimizations. The Sybase IQ index advisor identifies performance concerns for join columns whose data types differ.

For tables of implicit data type conversions, see Chapter 7, “Moving Data In and Out of Databases” in *Sybase IQ System Administration Guide*.

Support for joins between stores or databases

This section clarifies current support for joins between stores or between databases.

Joining tables within a Sybase IQ database

Any joins within a given Sybase IQ database are supported. This means that you can join any system or user tables in the Catalog Store with any tables in the IQ Store, in any order.

Joining Adaptive Server Enterprise and Sybase IQ tables

Joins of Sybase IQ tables with tables in an Adaptive Server Enterprise database are supported under the following conditions:

- The Sybase IQ database can be either the local database or the remote database.
- If a Sybase IQ table is to be used as a proxy table in ASE, the table name must be 30 characters or fewer.
- In order to join a local Adaptive Server Enterprise table with a remote Sybase IQ 12 table, the ASE version must be 11.9.2 or higher, and you must use the correct server class:
    - To connect from a front end of Adaptive Server Enterprise 12.5 or higher to a remote Sybase IQ 12.5 or higher, use the ASIQ server class, which was added in ASE 12.5.
    - To connect from a front end of Adaptive Server Enterprise 11.9.2 through 12.0 to a remote Sybase IQ 12.x (or Adaptive Server Anywhere 6.x or higher), you must use server class ASAnywhere.
- When you join a local Sybase IQ table with any remote table, the local table must appear first in the FROM clause, which means the local table is the outermost table in the join.

Joins between Sybase IQ and Adaptive Server Enterprise rely on Component Integration Services (CIS).

For more information on queries from Adaptive Server Enterprise databases to Sybase IQ, see *Component Integration Services Users's Guide* in the Adaptive Server Enterprise core documentation set.

For more information on queries from Sybase IQ to other databases, see “Querying remote and heterogeneous databases.”

Joining Adaptive Server Anywhere and Sybase IQ tables

The CHAR data type is incompatible between Adaptive Server Anywhere and Sybase IQ when the database is built with BLANK PADDING OFF. If you want to perform cross-database joins between Adaptive Server Anywhere and Sybase IQ tables using character data as the join key, use the CHAR data type with BLANK PADDING ON.

Note Sybase IQ CREATE DATABASE no longer supports BLANK PADDING OFF for new databases. This change has no effect on existing databases. You can test the state of existing databases using the BlankPadding database property:

```
select db_property ( 'BlankPadding' )
```

Sybase recommends that you change any existing columns affected by BLANK PADDING OFF, to ensure correct join results. Recreate join columns as CHAR data type, rather than VARCHAR. CHAR columns are always blank padded.

Querying remote and heterogeneous databases

This section summarizes how you use Sybase IQ with Component Integration Services (CIS). CIS allows you to query Adaptive Server Enterprise databases and remote databases or nonrelational data sources through Sybase IQ. CIS is installed as part of Sybase IQ.

Using CIS, you can access tables on remote servers as if the tables were local. CIS performs joins between tables in multiple remote, heterogeneous servers and transfers the contents of one table into a supported remote server.

To query a remote database or data source, you need to map its tables to local proxy tables. CIS presents proxy tables to a client application as if the data were stored locally. When you query the tables, CIS determines the actual server storage location.

❖ **To join remote databases:**

1. Create proxy tables, following the steps in the *Sybase IQ System Administration Guide*.
2. Map the remote tables to the proxy tables.

3 Reference the proxy tables in your SELECT statement, using the proxy database name as the qualifying name for each remote table. For example:

```
SELECT a.c_custkey, b.o_orderkey
FROM proxy_asiqdemo..cust2 a,
asiqdemo..orders b
WHERE a.c_custkey = b.o_custkey
```

For more information, see Chapter 16, "Accessing Remote Data" and Chapter 17, "Server Classes for Remote Data Access" in *Sybase IQ System Administration Guide*.

Replacing joins with subqueries

A join returns a result table constructed from data from multiple tables. You can also retrieve the same result table using a subquery. A subquery is simply a SELECT statement within another select statement. This is a useful tool in building more complex and informative queries.

For example, suppose you need a chronological list of orders and the company that placed them, but would like the company name instead of their customer ID. You can get this result using a join as follows:

Using a join

To list the order_id, order_date, and company_name for each order since the beginning of 1994, type:

```
SELECT  sales_order.id,
sales_order.order_date,
customer.company_name
FROM sales_order
KEY JOIN customer
WHERE order_date > '1994/01/01'
ORDER BY order_date
```

id	order_date	company_name
2473	1994-01-04	Peachtree Active Wear
2474	1994-01-04	Sampson & Sons
2036	1994-01-05	Hermanns
2475	1994-01-05	Salt & Peppers
2106	1994-01-05	Cinnamon Rainbows

Using an outer join

The join in previous sections of the tutorial is more fully called an **inner join**.

You specify an **outer join** explicitly. In this case, a GROUP BY clause is also required:

```
SELECT  company_name,
MAX( sales_order.id ),state
FROM customer
KEY LEFT OUTER JOIN sales_order
WHERE state = 'WA'
GROUP BY company_name, state
```

company_name	max(sales_order.id)	state
Custom Designs	2547	WA
Its a Hit!	(NULL)	WA

Using a subquery

To list order items for products low in stock, type:

```
SELECT *
FROM sales_order_items
WHERE prod_id IN
 ( SELECT id
FROM product
WHERE quantity < 20 )
ORDER BY ship_date DESC
```

id	line_id	prod_id	quantity	ship_date
2082	1	401	48	1994-07-09
2053	1	401	60	1994-06-30
2125	2	401	36	1994-06-28
2027	1	401	12	1994-06-17
2062	1	401	36	1994-06-17

The subquery in the statement is the phrase enclosed in parentheses:

```
(  SELECT id
FROM product
WHERE quantity < 20 )
```

By using a subquery, the search can be carried out in just one query, instead of using one query to find the list of low-stock products and a second to find orders for those products.

The subquery makes a list of all values in the id column in the product table satisfying the WHERE clause search condition.

Rephrasing the query

Consider what would happen if an order for ten tank tops were shipped so that the quantity column for tank tops contained the value 18. The query using the subquery would list all orders for both wool caps and tank tops. On the other hand, the first statement you used would have to be changed to the following:

```
SELECT *
FROM sales_order_items
WHERE prod_id IN ( 401, 300 )
ORDER BY ship_date DESC
```

The command using the subquery is an improvement because it still works even if data in the database is changed.

Remember the following notes about subqueries:

- Subqueries may also be useful in cases where you may have trouble constructing a join, such as queries that use the NOT EXISTS predicate.
- Subqueries can only return one column.
- Subqueries are allowed only as arguments of comparisons, IN, or EXISTS clauses.
- Subqueries cannot be used inside an outer join ON clause.

CHAPTER 3

Optimizing Queries and Deletions

About this chapter

This chapter offers query and deletion performance recommendations, including:

- Structuring queries for faster processing
- Using the query plans
- Setting query processing options
- Optimizing delete operations

Contents

Tips for structuring queries

Here are some hints for better query structure:

- In some cases, command statements that include subqueries can also be formulated as joins and may run faster.
- If you group on multiple columns in a GROUP BY clause, list the columns by descending order by number of unique values. This will give you the best query performance.
- Join indexes often cause join queries to execute faster than ad hoc joins, at the expense of using more disk space. However, when a join query does not reference the largest table in a multi-table join index, an ad hoc join usually outperforms the join index.
- You can improve performance by using an additional column to store frequently calculated results.

Impact on query performance of GROUP BY over a UNION ALL

To improve performance, very large tables are often segmented into several small tables and accessed using a UNION ALL in a view. For certain very specific queries using such a view with a GROUP BY, the Sybase IQ optimizer is able to enhance performance by pushing some GROUP BY operations into each arm of such a UNION ALL, performing the operations in parallel, then combining the results. This method, referred to as split GROUP BY, reduces the amount of data that is processed by the top level GROUP BY, and consequently reduces query processing time.

Only certain queries with a GROUP BY over a UNION ALL show a performance improvement. The following simple query, for example, benefits from the split GROUP BY:

```
CREATE VIEW vtable (v1 int, v2 char(4)) AS
SELECT a1, a2 FROM tableA
UNION ALL
SELECT b1, b2 FROM tableB;

SELECT COUNT(*), SUM(v1) FROM vtable GROUP BY v2;
```

When analyzing this query, the optimizer first performs COUNT(*) GROUP BY on tableA and COUNT(*) GROUP BY on tableB, then passes these results to the top level GROUP BY. The top level GROUP BY performs a SUM of the two COUNT(*) results, to produce the final query result. Note that the role of the top level GROUP BY changes: the aggregation used by the top level GROUP BY is SUM instead of COUNT.

Restrictions on split GROUP BY

There are some restrictions on the situations and queries that benefit from the split GROUP BY.

- The query may benefit from the split GROUP BY, if the query uses UNION ALL, rather than UNION. The following query uses GROUP BY with UNION, so it does *not* take advantage of the GROUP BY split:

```
CREATE VIEW viewA (va1 int, va2 int, va3 int,
va4 int) AS
SELECT b1, b2, b3, b4 FROM tableB
UNION
SELECT c1, c2, c3, c4 FROM tableC;

SELECT SUM(va1) FROM viewA GROUP BY va3;
```

- The query may benefit from the split GROUP BY, if an aggregation in the query does not contain DISTINCT. The following query uses SUM DISTINCT, so it does *not* take advantage of the split GROUP BY:

```
CREATE VIEW viewA (va1 int, va2 int, va3 int,
va4 int) AS
SELECT b1, b2, b3, b4 FROM tableB
UNION ALL
SELECT c1, c2, c3, c4 FROM tableC;

SELECT SUM(DISTINCT va1) FROM viewA GROUP BY va3;
```

- In order for the query to benefit from the split GROUP BY, you need enough memory in the temporary shared buffer cache to store the aggregation information and data used for processing the additional GROUP BY operators.

```
CREATE VIEW viewA (va1 int, va2 int, va3 int,
va4 int) AS
SELECT b1, b2, b3, b4 FROM tableB
UNION ALL
SELECT c1, c2, c3, c4 FROM tableC
UNION ALL
SELECT d1, d2, d3, d4 FROM tableD
UNION ALL
SELECT e1, e2, e3, e4 FROM tableE
```

```
UNION ALL
SELECT f1, f2, f3, f4 FROM tableF
UNION ALL
SELECT g1, g2, g3, g4 FROM tableG;

SELECT SUM(va1) FROM viewA GROUP BY va3;
```

In this example, the Sybase IQ optimizer splits the GROUP BY and inserts six GROUP BY operators into the query plan. Consequently, the query requires more temporary cache to store aggregation information and data. If the system cannot allocate enough cache, the optimizer does not split the GROUP BY.

You can use the TEMP_CACHE_MEMORY_MB database option to increase the size of the temporary cache, if memory is available. For information on setting buffer cache sizes, see "Determining the sizes of the buffer caches" on page 117 and "TEMP_CACHE_MEMORY_MB option" in the chapter "Database Options" of the *Sybase IQ Reference Manual*.

- In order for the query to benefit from split GROUP BY, the AGGREGATION_PREFERENCE database option should be set to its default value of 0. This value allows the Sybase IQ optimizer to determine the best algorithm to apply to the GROUP BY. The query does *not* benefit from split GROUP BY, if the value of AGGREGATION_PREFERENCE forces the Sybase IQ optimizer to choose a sort algorithm to process the GROUP BY. The option AGGREGATION_PREFERENCE can be used to override the optimizer's choice of algorithm for processing the GROUP BY and should not be set to 1 or 2 in this case.

Examples of split GROUP BY

In this example, a large table named tableA is segmented into four smaller tables: tabA1, tabA2, tabA3, and tabA4. The view unionTab is created using the four smaller tables and UNION ALL:

```
CREATE VIEW unionTab (v1 int, v2 int, v3 int, v4 int) AS
SELECT a, b, c, d FROM tabA1
UNION ALL
SELECT a, b, c, d FROM tabA2
UNION ALL
SELECT a, b, c, d FROM tabA3
UNION ALL
SELECT a, b, c, d FROM tabA4;
```

The Sybase IQ optimizer splits the GROUP BY operation in the following queries and improves query performance:

```
SELECT v1, v2, SUM(v3), COUNT(*) FROM unionTab
```

```
GROUP BY v1, v2;

SELECT v3, SUM(v1*v2) FROM unionTab
GROUP BY v3;
```

Conditions that cause processing by Adaptive Server Anywhere

Sybase IQ architecture includes a portion of the product that processes queries according to Adaptive Server Anywhere rules. This feature, called CIS (formerly OMNI) functional compensation, allows queries not directly supported by Sybase IQ semantics to be processed, but with a major performance cost.

CIS intercepts queries that:

- Reference a user-defined function
- Include a cross-database join or proxy table
- Include certain system functions
- Reference a Catalog Store table or a table created in the SYSTEM dbspace

For more information on differences between Sybase IQ and Adaptive Server Anywhere, see Appendix A, “Compatibility with Other Sybase Databases,” in *Sybase IQ Reference Manual*.

Planning queries

If you have created the right indexes, the Sybase IQ query optimizer can usually execute queries in the most efficient way—sometimes even if you have not used the most effective syntax. Proper query design is still important, however. When you plan your queries carefully, you can have a major impact on the speed and appropriateness of results.

Before it executes any query, the Sybase IQ query optimizer creates a query plan. Sybase IQ helps you evaluate queries by letting you examine and influence the query plan, using the options described in the sections that follow. For details of how to specify these options, see the *Sybase IQ Reference Manual*.

Note For all database options that accept integer values, Sybase IQ truncates any decimal *option-value* setting to an integer value. For example, the value 3.8 is truncated to 3.

Query evaluation options

The following options can help you evaluate the query plan. See the *Sybase IQ Reference Manual* for details of these options.

- INDEX_ADVISOR – When set ON, the index advisor prints index recommendations as part of the Sybase IQ query plan or as a separate message in the Sybase IQ message log file if query plans are not enabled. These messages begin with the string “Index Advisor:” and you can use that string to search and filter them from a Sybase IQ message file. This option outputs messages in OWNER.TABLE.COLUMN format and is OFF by default.

  See also the “sp_iqindexadvice procedure” in the *Sybase IQ Reference Manual.*

- INDEX_ADVISOR_MAX_ROWS – Used to limit the number of messages stored by the index advisor. Once the specified limit has been reached, the INDEX_ADVISOR will not store new advice. It will, however, continue to update count and timestamps for existing advice.

- NOEXEC – When set ON, Sybase IQ produces a query plan but does not execute the query, except when the EARLY_PREDICATE_EXECUTION option is ON.

- QUERY_DETAIL – When this option and either QUERY_PLAN or QUERY_PLAN_AS_HTML are both ON, Sybase IQ displays additional information about the query when producing its query plan. When QUERY_PLAN and QUERY_PLAN_AS_HTML are OFF, this option is ignored.

- QUERY_PLAN – When set ON (the default), Sybase IQ produces messages about queries. These include messages about using join indexes, about the join order, and about join algorithms for the queries.
- QUERY_PLAN_AFTER_RUN – When set ON, the query plan is printed after the query has finished running. This allows the plan to include additional information, such as the actual number of rows passed on from each node of the query. In order for this option to work, QUERY_PLAN must be ON. This option is OFF by default.
- QUERY_PLAN_AS_HTML – Produces a graphical query plan in HTML format for viewing in a Web browser. Hyperlinks between nodes make the HTML format much easier to use than the text format in the *.iqmsg* file. Use the QUERY_NAME option to include the query name in the file name for the query plan. This option is OFF by default.
- QUERY_PLAN_AS_HTML_DIRECTORY – When set ON and a directory is specified with QUERY_PLAN_AS_HTML_DIRECTORY, Sybase IQ writes the HTML query plans in the specified directory.
- QUERY_TIMING – Controls the collection of timing statistics on subqueries and some other repetitive functions in the query engine. Normally it should be OFF (the default) because for very short correlated subqueries the cost of timing every subquery execution can be very expensive in terms of performance.

Note Query plans can add a lot of text to your *.iqmsg* file. When QUERY_PLAN is ON, and especially if QUERY_DETAIL is ON, you will probably want to enable message log wrapping by setting IQMSG_LENGTH_MB to a positive value.

The query tree

The optimizer creates a query “tree” that represents the flow of data in the query. The query plan presents the query tree in text form in the *.iqmsg* file, and optionally in graphical form.

The query tree consists of nodes. Each node represents a stage of work. The lowest nodes on the tree are leaf nodes. Each leaf node represents a table or a prejoin index set in the query.

At the top of the plan is the root of the operator tree. Information flows up from the tables and through any operators representing joins, sorts, filters, stores, aggregation, and subqueries.

Using the HTML query plan

A good way to start using query plans is to set the QUERY_PLAN_AS_HTML option ON. This option places a graphical version of the query plan in the same directory as the *.iqmsg* file. You can view this file in most Web browsers.

In the HTML query plan, each node in the tree is a hyperlink to the details. Each box is hyperlinked to the tree above. You can click on any node to navigate quickly through the plan.

Controlling query processing

Any user can set limits on the amount of time spent processing a particular query. Users with DBA privileges can give certain users' queries priority over others, or change processing algorithms to influence the speed of query processing. See the *Sybase IQ Reference Manual* for details on the options described in this section.

Setting query time limits

By setting the MAX_QUERY_TIME option, a user can disallow long queries. If a query takes longer to execute than desired, Sybase IQ stops the query with an appropriate error.

Note Sybase IQ truncates all decimal *option-value* settings to integer values. For example, the value 3.8 is truncated to 3.

Setting query priority

Queries waiting in queue for processing are queued to run in order of the priority of the user who submitted the query, followed by the order in which the query was submitted. No queries are run from a lower priority queue until higher priority queries have all been executed.

The following options assign queries a processing priority by user.

- IQGOVERN_PRIORITY – Assigns a numeric priority (1, 2, or 3, with 1 being the highest) to queries waiting in the processing queue.
- IQGOVERN_MAX_PRIORITY – Allows the DBA to set an upper boundary on IQGOVERN_PRIORITY for a user or a group.
- IQ_GOVERN_PRIORITY_TIME – Allows high priority users to start if a high priority (priority 1) query has been waiting in the -iqgovern queue for more than a designated amount of time.

To check the priority of a query, check the IQGovernPriority attribute returned by the sp_iqcontext stored procedure.

Setting query optimization options

The following options affect query processing speed:

- AGGREGATION_PREFERENCE – Controls the choice of algorithms for processing an aggregate (GROUP BY, DISTINCT, SET functions). This option is designed primarily for internal use; do not use it unless you are an experienced database administrator.
- DEFAULT_HAVING_SELECTIVITY – Sets the selectivity for all HAVING predicates in a query, overriding optimizer estimates for the number of rows that will be filtered by the HAVING clause.
- DEFAULT_LIKE_MATCH_SELECTIVITY – Sets the default selectivity for generic LIKE predicates, for example, `LIKE '`***`string%string`***`'` where % is a wildcard character. The optimizer relies on this option when other selectivity information is not available and the match string does not start with a set of constant characters followed by a single wildcard.
- DEFAULT_LIKE_RANGE_SELECTIVITY – Sets the default selectivity for leading constant LIKE predicates, of the form `LIKE '`***`string%`***`'` where the match string is a set of constant characters followed by a single wildcard character (%). The optimizer relies on this option when other selectivity information is not available.

- EARLY_PREDICATE_EXECUTION – Controls whether simple local predicates are executed before join optimization. Under most circumstances, it should not be changed.
- ENABLED_ORDERED_PUSHDOWN_INSERTION – Controls how the query optimizer adds in the semijoin predicates for push-down joins selected by the join optimizer. Re-analyzes any intermediate joins that may be indirectly affected by those semijoins. Under most circumstances, it should not be changed.
- IN_SUBQUERY_PREFERENCE – Controls the choice of algorithms for processing IN subqueries. This option is designed primarily for internal use; do not use it unless you are an experienced database administrator.
- INDEX_PREFERENCE – Sets the index to use for query processing. The Sybase IQ optimizer normally chooses the best index available to process local WHERE clause predicates and other operations which can be done within an IQ index. This option is used to override the optimizer choice for testing purposes; under most circumstances it should not be changed.
- JOIN_PREFERENCE – Controls the choice of algorithms when processing joins. This option is designed primarily for internal use; do not use it unless you are an experienced database administrator.
- JOIN_SIMPLIFICATION_THRESHOLD – Controls the minimum number of tables being joined together before any join optimizer simplifications are applied. Normally you should not need to change this value.
- MAX_HASH_ROWS – Sets the maximum estimated number of rows the query optimizer will consider for a hash algorithm. The default is 1,250,000 rows. For example, if there is a join between two tables, and the estimated number of rows entering the join from both tables exceeds this option value, the optimizer will not consider a hash join. On systems with more than 50MB per user of TEMP_CACHE_MEMORY_MB, you may want to consider a higher value for this option.
- MAX_JOIN_ENUMERATION – Sets the maximum number of tables to be optimized for join order after optimizer simplifications have been applied. Normally you should not need to set this option.

Setting predicate hints

Sybase IQ supports a hint string that lets you specify per-predicate hints, such as selectivity, usefulness, index preference, and execution mode.

You can set selectivity in combination with three other query optimizations:

- Setting the equivalent of an index preference option
- Setting the usefulness (ordering the predicates)
- Delaying one or more predicates

Under normal circumstances, there are no advantages to delaying evaluation, which could slow the query. If you choose to, however, you can move any of the following four behaviors to later in the query:

- Before optimization
- At first “first fetch” time
- At second “first fetch” time (inside correlated subqueries or on the left side of a nested loop pushdown join only)
- Not using indexes at all (“horizontal processing”)

For syntax, parameters, and examples, see “User-supplied condition hint strings,” Chapter 3, “SQL Language Elements,” in the *Sybase IQ Reference Manual*.

Optimizing delete operations

Sybase IQ chooses one of three algorithms to process delete operations:

- **Small delete**

  Small delete provides optimal performance when rows are deleted from very few groups. It is typically selected when the delete is only 1 row or the delete has an equality predicate on the columns with an HG (High_Group) index. The small delete algorithm can randomly access the HG. Worst case I/O is proportional to the number of groups visited.

- **Mid delete**

  Mid delete provides optimal performance when rows are deleted from several groups, but the groups are sparse enough or few enough that not many HG pages are visited. The mid delete algorithm provides ordered access to the HG. Worst case I/O is bounded by the number of index pages. Mid delete has the added cost of sorting the records to delete.

- **Large delete**

Large delete provides optimal performance when rows are deleted from a large number of groups. The large delete scans the HG in order until all rows are deleted. Worst case I/O is bounded by the number of index pages. Large delete is parallel, but parallelism is limited by internal structure of the index and the distribution of group to deleted from. Range predicates on HG columns can be used to reduce the scan range of the large delete.

Delete costing

Prior to 12.6, the HG delete cost model considered only worst case I/O performance and therefore preferred large delete in most cases. The current cost model considers many factors including I/O costs, CPU costs, available resources, index metadata, parallelism, and predicates available from the query.

Specifying predicates on columns that have HG indexes greatly improves costing. In order for the HG costing to pick an algorithm other than Large delete, it must be able to determine the number of distinct values (groups) affected by deletions. Distinct count is initially assumed to be lesser of the number of index groups and the number of rows deleted. Predicates can provide an improved or even exact estimate of the distinct count.

Costing currently does not consider the effect of range predicates on the large delete. This can cause mid delete to be chosen in cases where large delete would be faster. You can force the large delete algorithm if needed in these cases, as described in the next section.

Using delete performance option

You can use the HG_DELETE_METHOD option to control HG delete performance.

The value of the parameter specified with the HG_DELETE_METHOD option forces the specified delete algorithm as follows:

- 1 = Small Delete
- 2 = Large Delete
- 3 = Mid Delete

CHAPTER 4

Using OLAP

About this chapter

OLAP (online analytical processing) is an efficient method of data analysis on information stored in a relational database. Using OLAP you can analyze data on different dimensions, acquire result sets with subtotaled rows, and organize data into multidimensional cubes, all in a single SQL query. You can also use filters to drill down into the data, returning result sets quickly. This chapter describes the SQL/OLAP functionality that Sybase IQ supports.

Note The tables shown in OLAP examples are available in the asiqdemo database.

Contents

About OLAP

Extensions to the ANSI SQL standard to include complex data analysis were introduced as an amendment to the 1999 SQL standard. Sybase IQ added portions of these SQL enhancements in previous releases. Sybase IQ 12.7, however, contains comprehensive support for these extensions.

These analytic functions, which offer the ability to perform complex data analysis within a single SQL statement, are facilitated by a category of software technology named On Line Analytical Processing (OLAP) whose functions include GROUP BY clause extensions and analytical functions as shown in the following list:

- GROUP BY clause extensions — CUBE and ROLLUP
- Analytical functions:
  - Simple aggregates — AVG, COUNT, MAX, MIN, and SUM, STDDEV and VARIANCE

    Note All simple aggregate functions, except Grouping(), can be used with an OLAP windowed function.

  - Window functions:
    - Windowing aggregates — AVG, COUNT, MAX, MIN, and SUM
    - Ranking functions — RANK, DENSE_RANK, PERCENT_RANK, and NTILE
    - Statistical functions — STDDEV, STDDEV_SAMP, STDDEV_POP, VARIANCE, VAR_SAMP, and VAR_POP
    - Distribution functions — PERCENTILE_CONT and PERCENTILE_DISC
  - Numeric functions — WIDTH_BUCKET, CEIL, and LN, EXP, POWER, SQRT, and FLOOR

Some database products provide a separate OLAP module that requires you to move data from the database into the OLAP module before analyzing it. By contrast, Sybase IQ builds OLAP features into the database itself, making deployment and integration with other database features, such as stored procedures, easy and seamless.

OLAP benefits

OLAP functions, when combined with the GROUPING, CUBE and ROLLUP extensions, provide two primary benefits. First, they let you perform multidimensional data analysis, data mining, time series analyses, trend analysis, cost allocations, goal seeking, ad hoc multidimensional structural changes, non-procedural modeling, and exception alerting, often with a single SQL statement. Second, the window and reporting aggregate functions use a relational operator, called a **window** that can be executed more efficiently than semantically equivalent queries that use self-joins or correlated subqueries. The result sets you obtain using OLAP can have subtotal rows and can be organized into multidimensional cubes. See "Windowing" on page 67.

Moving averages and moving sums can be calculated over various intervals; aggregations and ranks can be reset as selected column values change; and complex ratios can be expressed in simple terms. Within the scope of a single query expression, you can define several different OLAP functions, each with its own partitioning rules.

Understanding OLAP evaluation

OLAP evaluation can be conceptualized as several phases of query execution that contribute to the final result. You can identify OLAP phases of execution by the relevant clause in the query. For example, if a SQL query specification contains window functions, the WHERE, JOIN, GROUP BY, and HAVING clauses are processed first. Partitions are created after the groups defined in the GROUP BY clause and before the evaluation of the final SELECT list in the query's ORDER BY clause.

For the purpose of grouping, all NULL values are considered to be in the same group, even though NULL values are not equal to one another.

The HAVING clause acts as a filter, much like the WHERE clause, on the results of the GROUP BY clause.

Consider the semantics of a simple query specification involving the SQL statements and clauses, SELECT, FROM, WHERE, GROUP BY, and HAVING from the ANSI SQL standard:

1. The query produces a set of rows that satisfy the table expressions present in the FROM clause.
2. Predicates from the WHERE clause are applied to rows from the table. Rows that fail to satisfy the WHERE clause conditions (do not equal true) are rejected.

3 Except for aggregate functions, expressions from the SELECT list and in the list and GROUP BY clause are evaluated for every remaining row.

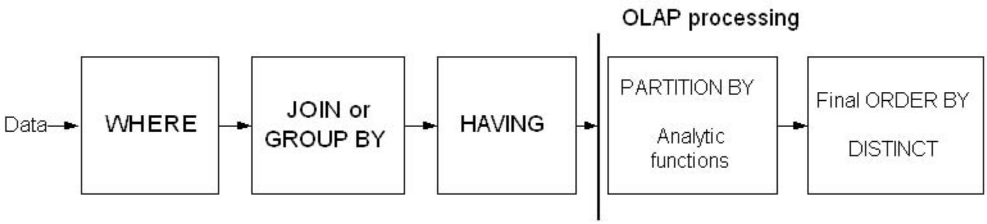
4 The resulting rows are grouped together based on distinct values of the expressions in the GROUP BY clause, treating NULL as a special value in each domain. The expressions in the GROUP BY clause serve as partition keys if a PARTITION BY clause is present.

5 For each partition, the aggregate functions present in the SELECT list or HAVING clause are evaluated. Once aggregated, individual table rows are no longer present in the intermediate result set. The new result set consists of the GROUP BY expressions and the values of the aggregate functions computed for each partition.

6 Conditions from the HAVING clause are applied to result groups. Groups are eliminated that do not satisfy the HAVING clause.

7 Results are partitioned on boundaries defined in the PARTITION BY clause. OLAP windows functions (rank and aggregates) are computed for result windows.

Figure 4-1: Semantic phases of execution



See “Grammar rule 2” on page 108. See also “BNF grammar for OLAP functions” on page 108 for details on OLAP syntax.

GROUP BY clause extensions

Extensions to the GROUP BY clause let application developers write complex SQL statements that

- Partition the input rows in multiple dimensions and combine multiple subsets of result groups.

- Create a "data cube," providing a sparse, multi-dimensional result set for data mining analyses.
- Create a result set that includes the original groups, and optionally includes a subtotal and grand-total row.

OLAP Grouping() operations, such as ROLLUP and CUBE, can be conceptualized as prefixes and subtotal rows.

Prefixes

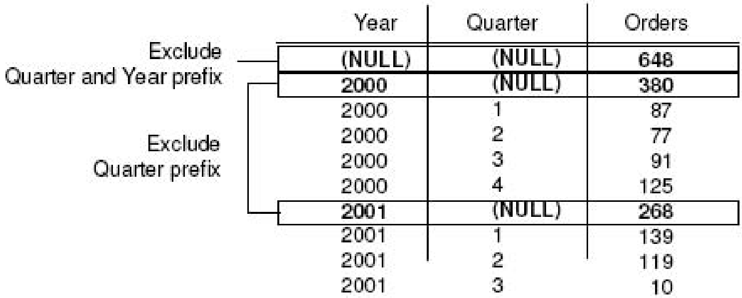
A list of **prefixes** is constructed for any query that contains a GROUP BY clause. A prefix is a subset of the items in the GROUP BY clause and is constructed by excluding one or more of the rightmost items from those in the query's GROUP BY clause. The remaining columns are called the **prefix columns**.

ROLLUP example 1 In the following ROLLUP example query, the GROUP BY list includes two variables, *Year* and *Quarter*:

```
SELECT year (order_date) Year, quarter(order_date)
   Quarter, COUNT(*) Orders
FROM alt_sales_order
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

The query's two prefixes are:

- Exclude Quarter – The set of prefix columns contains the single column Year.
- Exclude both Quarter and Year – There are no prefix columns.



Note The GROUP BY list contains the same number of prefixes as items.

Group by ROLLUP and CUBE

Two important syntactic shortcuts exist to concisely specify common grouping for prefixes. The first of these patterns is called ROLLUP, and the second is called CUBE.

Group by ROLLUP

The ROLLUP operator requires an ordered list of grouping expressions to be supplied as arguments, as in the following syntax.

SELECT … [**GROUPING** (*column-name*) …] …
GROUP BY [*expression* [, …]
| **ROLLUP** (*expression* [, …])]

GROUPING takes a column name as a parameter and returns a Boolean value as listed in Table 4-1.

Table 4-1: Values returned by GROUPING with the ROLLUP operator

If the value of the result is	GROUPING returns
NULL created by a ROLLUP operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
Not created by a ROLLUP operation	0 (FALSE)
A stored NULL	0 (FALSE)

ROLLUP first calculates the standard aggregate values specified in the GROUP BY clause. Then ROLLUP moves from right to left through the list of grouping columns and creates progressively higher-level subtotals. A grand total is created at the end. If *n* is the number of grouping columns, then ROLLUP creates *n*+1 levels of subtotals.

This SQL syntax...	Defines the following sets...
`GROUP BY ROLLUP (A, B, C);`	(A, B, C)
	(A, B)
	(A)
	( )

ROLLUP and subtotal rows

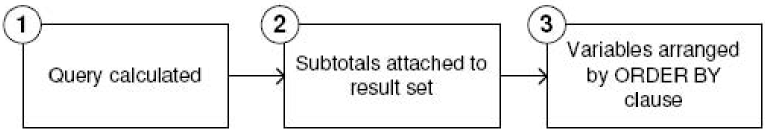
ROLLUP is equivalent to a UNION of a set of GROUP BY queries. The result sets of the following queries are identical. The result set of GROUP BY (A, B) consists of subtotals over all those rows in which A and B are held constant. To make a union possible, column C is assigned NULL.

This ROLLUP query...	Is equivalent to this query without ROLLUP...
`SELECT A, B, C, SUM( D ) FROM T1 GROUP BY ROLLUP (A, B, C);`	`SELECT * FROM ( ( SELECT A, B, C, SUM( D ) GROUP BY A, B, C ) UNION ALL ( SELECT A, B, NULL, SUM( D ) GROUP BY A, B ) UNION ALL ( SELECT A, NULL, NULL, SUM( D ) GROUP BY A ) UNION ALL ( SELECT NULL, NULL, NULL, SUM( D ) ) )`

Subtotal rows can help you analyze data, especially if there are large amounts of data, different dimensions to the data, data contained in different tables, or even different databases altogether. For example, a sales manager might find reports on sales figures broken down by sales representative, region, and quarter to be useful in understanding patterns in sales. Subtotals for the data give the sales manager a picture of overall sales from different perspectives. Analyzing this data is easier when summary information is provided based on the criteria that the sales manager wants to compare.

With OLAP, the procedure for analyzing and computing row and column subtotals is invisible to users. Figure 4-2 shows conceptually how Sybase IQ creates subtotals:

Figure 4-2: Subtotals



1 This step yields an intermediate result set that has not yet considered the ROLLUP.

2 Subtotals are evaluated and attached to the result set.

3 The rows are arranged according to the ORDER BY clause in the query.

NULL values and subtotal rows

When rows in the input to a GROUP BY operation contain NULL, there is the possibility of confusion between subtotal rows added by the ROLLUP or CUBE operations and rows that contain NULL values that are part of the original input data.

The Grouping() function distinguishes subtotal rows from others by taking a column in the GROUP BY list as its argument, and returning 1 if the column is NULL because the row is a subtotal row, and 0 otherwise.

The following example includes Grouping() columns in the result set. Rows are highlighted that contain NULL as a result of the input data, not because they are subtotal rows. The Grouping() columns are highlighted. The query is an outer join between the employee table and the sales_order table. The query selects female employees who live in either Texas, New York, or California. NULL appears in the columns corresponding to those female employees who are not sales representatives (and therefore have no sales).

```
SELECT employee.emp_id AS Employee, year(order_date) AS
   Year, COUNT(*) AS Orders, GROUPING(Employee) AS
   GE, GROUPING(Year) AS GY
   FROM employee LEFT OUTER JOIN alt_sales_order ON
   employee.emp_id = alt_sales_order.sales_rep
   WHERE employee.sex IN ('F') AND employee.state
   IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (Year, Employee)
ORDER BY Year, Employee
```

The following result set is from the query.

```
emp_id      year      Orders     GY     GE
------      ----      ------     --     --
NULL        NULL           1      1      0
NULL        NULL         165      1      1
1090        NULL           1      0      0
NULL        2000          98      1      0
 667        2000          34      0      0
 949        2000          31      0      0
1142        2000          33      0      0
NULL        2001          66      1      0
 667        2001          20      0      0
949         2001          22      0      0
1142        2001          24      0      0
```

For each prefix, a **subtotal row** is constructed that corresponds to all rows in which the prefix columns have the same value.

To demonstrate ROLLUP results, examine the example query again:

```
SELECT year (order_date) AS Year, quarter
   (order_date) AS Quarter, COUNT (*) Orders
FROM sales_order
   GROUP BY ROLLUP (Year, Quarter)
   ORDER BY Year, Quarter
```

In this query, the prefix containing the Year column leads to a summary row for Year=2000 and a summary row for Year=2001. A single summary row for the prefix has no columns, which is a subtotal over all rows in the intermediate result set.

The value of each column in a subtotal row is as follows:

- Column included in the prefix – The value of the column. For example, in the preceding query, the value of the Year column for the subtotal over rows with Year=2000 is 2000.
- Column excluded from the prefix – NULL. For example, the Quarter column has a value of NULL for the subtotal rows generated by the prefix consisting of the Year column.
- Aggregate function – An aggregate over the values of the excluded columns.

  Subtotal values are computed over the rows in the underlying data, not over the aggregated rows. In many cases, such as SUM or COUNT, the result is the same, but the distinction is important in the case of statistical functions such as AVG, STDDEV, and VARIANCE, for which the result differs.

Restrictions on the ROLLUP operator are:

- The ROLLUP operator supports all of the aggregate functions available to the GROUP BY clause except COUNT DISTINCT and SUM DISTINCT.
- ROLLUP can only be used in the SELECT statement; you cannot use ROLLUP in a subquery.
- A grouping specification that combines multiple ROLLUP, CUBE, and GROUP BY columns in the same GROUP BY clause is not currently supported.
- Constant expressions as GROUP BY keys are not supported.

For the general format of an expression, see “Expressions,” “SQL Language Elements,” in the *Sybase IQ Reference Manual*.

ROLLUP example 2 The following example illustrates the use of ROLLUP and GROUPING and displays a set of mask columns created by GROUPING. The digits 0 and 1 displayed in columns S, N, and C are the values returned by GROUPING to represent the value of the ROLLUP result. A program can analyze the results of this query by using a mask of “011” to identify subtotal rows and “111” to identify the row of overall totals.

```
SELECT size, name, color, SUM(quantity),
```

```
    GROUPING(size) AS S,
    GROUPING(name) AS N,
    GROUPING(color) AS C
FROM product
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1
or C=1)
ORDER BY size, name, color;
```

The following are the results from the above query:

```
size               name          color      SUM  S  N  C
----               -----         ------     ---  -  -  -
(NULL)             (NULL)        (NULL)     496  1  1  1
Large              (NULL)        (NULL)      71  0  1  1
Large              Sweatshirt    (NULL)      71  0  0  1
Medium             (NULL)        (NULL)     134  0  1  1
Medium             Shorts        (NULL)      80  0  0  1
Medium             Tee Shirt     (NULL)      54  0  0  1
One size fits all  (NULL)        (NULL)     263  0  1  1
One size fits all  Baseball Cap  (NULL)     124  0  0  1
One size fits all  Tee Shirt     (NULL)      75  0  0  1
One size fits all  Visor         (NULL)      64  0  0  1
Small              (NULL)        (NULL)      28  0  1  1
Small              Tee Shirt     (NULL)      28  0  1  1
```

ROLLUP example 3 The following example illustrates the use of GROUPING to distinguish stored NULL values and "NULL" values created by the ROLLUP operation. Stored NULL values are then displayed as [NULL] in column prod_id, and "NULL" values created by ROLLUP are replaced with ALL in column PROD_IDS, as specified in the query.

```
SELECT year(ship_date) AS Year, prod_id, SUM(quantity)
    AS OSum, CASE WHEN GROUPING(Year) = 1 THEN 'ALL' ELSE
    CAST(Year AS char(8)) END, CASE WHEN
    GROUPING(prod_id) = 1 THEN 'ALL' ELSE CAST(prod_id
    as char(8)) END
FROM alt_sales_order_items
GROUP BY ROLLUP(Year, prod_id) HAVING OSum > 36
ORDER BY Year, prod_id;
```

The following are the results from the above query:

```
ship_date     prod_id   SUM    SHIP_DATES   PROD_IDS
---------     -------   ---    ----------   --------
NULL           NULL    28359   ALL          ALL
2000           NULL    17642   2000         ALL
2000           300      1476   2000         300
```

```
2000          301    1440   2000          301
2000          302    1152   2000          302
2000          400    1946   2000          400
2000          401    1596   2000          401
2000          500    1704   2000          500
2000          501    1572   2000          501
2000          600    2124   2000          600
2000          601    1932   2000          601
2000          700    2700   2000          700
2001          NULL  10717   2001          ALL
2001          300     888   2001          300
2001          301     948   2001          301
2001          302     996   2001          302
2001          400    1332   2001          400
2001          401    1105   2001          401
2001          500     948   2001          500
2001          501     936   2001          501
2001          600     936   2001          600
2001          601     792   2001          601
2001          700    1836   2001          700
```

ROLLUP example 4 The next example query returns data that summarizes the number of sales orders by year and quarter.

```
SELECT year(order_date) AS Year, quarter(order_date)
   AS Quarter, COUNT(*) AS Orders
   FROM alt_sales_order
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

The following figure illustrates the query results with subtotal rows highlighted in the result set. Each subtotal row contains a NULL value in the column or columns over which the subtotal is computed.

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**2000**	**(NULL)**	**380**
3	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	235
2	**2001**	**(NULL)**	**268**
3	2001	1	139
	2001	2	119
	2001	3	10

Row [1] represents the total number of orders across both years (2000, 2001) and all quarters. This row contains NULL in both the Year and Quarter columns and is the row where all columns were excluded from the prefix.

Note Every ROLLUP operation returns a result set with one row where NULL appears in each column except for the aggregate column. This row represents the summary of each column to the aggregate function. For example, if SUM were the aggregate function in question, this row would represent the grand total of all values.

Row [2] represent the total number of orders in the years 2000 and 2001, respectively. Both rows contain NULL in the Quarter column because the values in that column are rolled up to give a subtotal for Year. The number of rows like this in your result set depends on the number of variables that appear in your ROLLUP query.

The remaining rows marked [3] provide summary information by giving the total number of orders for each quarter in both years.

ROLLUP example 5 This example of the ROLLUP operation returns a slightly more complicated result set, which summarizes the number of sales orders by year, quarter, and region. In this example, only the first and second quarters and two selected regions (Canada and the Eastern region) are examined.

```
SELECT year(order_date) AS Year, quarter(order_date)
   AS Quarter, region, COUNT(*) AS Orders
   FROM alt_sales_order WHERE region IN ('Canada',
   'Eastern') AND quarter IN (1, 2)
GROUP BY ROLLUP (Year, Quarter, Region)
ORDER BY Year, Quarter, Region
```

The following figure illustrates the result set from the above query. Each subtotal row contains a NULL in the column or columns over which the subtotal is computed.

Year	Quarter	Region	Orders
(NULL)	**(NULL)**	**(NULL)**	**183**
2000	**(NULL)**	**(NULL)**	**68**
2000	1	(NULL)	36
2000	1	Canada	3
2000	1	Eastern	33
2000	2	(NULL)	32
2000	2	Canada	3
2000	2	Eastern	29
2001	**(NULL)**	**(NULL)**	**115**
2001	1	(NULL)	57
2001	1	Canada	11
2001	1	Eastern	46
2001	2	(NULL)	58
2001	2	Canada	4
2001	2	Eastern	54

Row [1] is an aggregate over all rows and contains NULL in the Year, Quarter, and Region columns. The value in the Orders column of this row represents the total number of orders in Canada and the Eastern region in quarters 1 and 2 in the years 2000 and 2001.

The rows marked [2] represent the total number of sales orders in each year (2000) and (2001) in quarters 1 and 2 in Canada and the Eastern region. The values of these rows [2] are equal to the grand total represented in row [1].

The rows marked [3] provide data about the total number of orders for the given year and quarter by region.

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	**1**	**(NULL)**	**36**
2000	1	Canada	3
2000	1	Eastern	33
2000	**2**	**(NULL)**	**32**
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	**1**	**(NULL)**	**57**
2001	1	Canada	11
2001	1	Eastern	46
2001	**2**	**(NULL)**	**58**
2001	2	Canada	4
2001	2	Eastern	54

The rows marked [4] provide data about the total number of orders for each year, each quarter, and each region in the result set.

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	1	(NULL)	36
2000	1	Canada	3
2000	1	Eastern	33
2000	2	(NULL)	32
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	1	(NULL)	57
2001	1	Canada	11
2001	1	Eastern	46
2001	2	(NULL)	58
2001	2	Canada	4
2001	2	Eastern	54

Group by CUBE

The CUBE operator in the GROUP BY clause analyzes data by forming the data into groups in more than one dimension (grouping expression). CUBE requires an ordered list of dimensions as arguments and enables the SELECT statement to calculate subtotals for all possible combinations of the group of dimensions that you specify in the query and generates a result set that shows aggregates for all combinations of values in selected columns.

CUBE syntax:

SELECT … [**GROUPING** (*column-name*) …] …
GROUP BY [*expression* [,…]
| **CUBE** (*expression* [,…])]

GROUPING takes a column name as a parameter and returns a Boolean value as listed in Table 4-2.

Table 4-2: Values returned by GROUPING with the CUBE operator

If the value of the result is	GROUPING returns
NULL created by a CUBE operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
Not created by a CUBE operation	0 (FALSE)
A stored NULL	0 (FALSE)

CUBE is particularly useful when your dimensions are not a part of the same hierarchy.

This SQL syntax...	Defines the following sets...
`GROUP BY CUBE (A, B, C);`	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ( )

Restrictions on the CUBE operator are:

- The CUBE operator supports all of the aggregate functions available to the GROUP BY clause, but CUBE is currently not supported with COUNT DISTINCT or SUM DISTINCT.

- CUBE is currently not supported with the inverse distribution analytical functions, PERCENTILE_CONT and PERCENTILE_DISC.
- CUBE can only be used in the SELECT statement; you cannot use CUBE in a SELECT subquery.
- A GROUPING specification that combines ROLLUP, CUBE, and GROUP BY columns in the same GROUP BY clause is not currently supported.
- Constant expressions as GROUP BY keys are not supported.

Note Performance of CUBE will diminish if the size of the cube exceeds the size of the temp cache.

GROUPING can be used with the CUBE operator to distinguish between stored NULL values and “NULL” values in query results created by CUBE.

See the examples in the description of the ROLLUP operator for illustrations of the use of the GROUPING function to interpret results.

All CUBE operations return result sets with at least one row where NULL appears in each column except for the aggregate columns. This row represents the summary of each column to the aggregate function.

CUBE example 1 The following queries use data from a census, including the state (geographic location), gender, education level, and income of people. The first query contains a GROUP BY clause that organizes the results of the query into groups of rows, according to the values of the columns state, gender, and education in the table census and computes the average income and the total counts of each group. This query uses only the GROUP BY clause without the CUBE operator to group the rows.

```
SELECT state, sex as gender, dept_id, COUNT(*),
   CAST(ROUND(AVG(salary),2) AS NUMERIC(18,2))
   AS average
FROM employee WHERE state IN ('MA' , 'CA')
GROUP BY state, sex, dept_id
ORDER BY 1,2;
```

The following are the results from the above query:

```
state  gender  dept_id  count(*)  avg salary
-----  ------  -------  --------  ----------
CA       F       200       2        58650.00
CA       M       200       1        39300.00
MA       F       500       4        29950.00
MA       F       400       8        41959.88
```

```
MA        F         300       7          59685.71
MA        F         200       3          60451.00
MA        F         100       6          58243.42
MA        M         300       2          58850.00
MA        M         500       5          36793.96
MA        M         400       8          45321.47
MA        M         100      13          58563.59
MA        M         200       8          46810.63
```

Use the CUBE extension of the GROUP BY clause, if you want to compute the average income in the entire census of state, gender, and education and compute the average income in all possible combinations of the columns state, gender, and education, while making only a single pass through the census data. For example, use the CUBE operator if you want to compute the average income of all females in all states, or compute the average income of all people in the census according to their education and geographic location.

When CUBE calculates a group, a NULL value is generated for the columns whose group is calculated. The GROUPING function must be used to distinguish whether a NULL is a NULL stored in the database or a NULL resulting from CUBE. The GROUPING function returns 1 if the designated column has been merged to a higher level group.

CUBE example 2 The following query illustrates the use of the GROUPING function with GROUP BY CUBE.

```
SELECT case grouping(state) WHEN 1 THEN 'ALL' ELSE state
   END AS c_state, case grouping(sex) WHEN 1 THEN 'ALL'
   ELSE sex end AS c_gender, case grouping(dept_id)
   WHEN 1 THEN 'ALL' ELSE cast(dept_id as char(4)) end
   AS c_dept, COUNT(*), CAST(ROUND(AVG(salary),2) AS
   NUMERIC(18,2))AS AVERAGE
FROM employee WHERE state IN ('MA' , 'CA')
GROUP BY CUBE(state, sex, dept_id)
ORDER BY 1,2,3;
```

The results of this query are shown below. Note that the NULLs generated by CUBE to indicate a subtotal row are replaced with ALL in the subtotal rows, as specified in the query.

```
state     sex     dept_id    count   avg salary
-----     ---     -------    -----   ----------
ALL       ALL     100         19       58462.48
ALL       ALL     200         14       50888.43
ALL       ALL     300          9       59500.00
ALL       ALL     400         16       43640.67
ALL       ALL     500          9       33752.20
```

```
ALL        ALL     ALL        67      50160.38
ALL        F       100         6      58243.42
ALL        F       200         5      59730.60
ALL        F       300         7      59685.71
ALL        F       400         8      41959.88
ALL        F       500         4      29950.00
ALL        F       ALL        30      50713.08
ALL        M       100        13      58563.59
ALL        M       200         9      45976.11
ALL        M       300         2      58850.00
ALL        M       400         8      45321.47
ALL        M       500         5      36793.96
ALL        M       ALL        37      49712.25
CA         ALL     200         3      52200.00
CA         ALL     ALL         3      52200.00
CA         F       200         2      58650.00
CA         F       ALL         2      58650.00
CA         M       200         1      39300.00
CA         M       ALL         1      39300.00
MA         ALL     100        19      58462.48
MA         ALL     200        11      50530.73
MA         ALL     300         9      59500.00
MA         ALL     400        16      43640.67
MA         ALL     500         9      33752.20
MA         ALL     ALL        64      50064.78
MA         F       100         6      58243.42
MA         F       200         3      60451.00
MA         F       300         7      59685.71
MA         F       400         8      41959.88
MA         F       500         4      29950.00
MA         F       ALL        28      50146.16
MA         M       100        13      58563.59
MA         M       200         8      46810.63
MA         M       300         2      58850.00
MA         M       400         8      45321.47
MA         M       500         5      36793.96
MA         M       ALL        36      50001.48
```

CUBE example 3 In this example, the query returns a result set that summarizes the total number of orders and then calculates subtotals for the number of orders by year and quarter.

Note As the number of variables that you want to compare increases, the cost of computing the cube increases exponentially.

```
SELECT year(order_date) AS Year, quarter(order_date)
   AS Quarter, COUNT(*) AS Orders
FROM alt_sales_order
GROUP BY CUBE(Year, Quarter)
ORDER BY Year, Quarter
```

The figure that follows represents the result set from the query. The subtotal rows are highlighted in the result set. Each subtotal row has a NULL in the column or columns over which the subtotal is computed.

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**(NULL)**	**1**	**226**
	(NULL)	**2**	**196**
	(NULL)	**3**	**101**
	(NULL)	**4**	**125**
3	**2000**	**(NULL)**	**380**
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
3	**2001**	**(NULL)**	**268**
	2001	1	139
	2001	2	119
	2001	3	10

The first highlighted row [1] represents the total number of orders across both years and all quarters. The value in the Orders column is the sum of the values in each of the rows marked [3]. It is also the sum of the four values in the rows marked [2].

The next set of highlighted rows [2] represents the total number of orders by quarter across both years. The two rows marked by [3] represent the total number of orders across all quarters for the years 2000 and 2001, respectively.

Analytical functions

Sybase IQ offers both simple and windowed aggregation functions that offer the ability to perform complex data analysis within a single SQL statement. These functions can be used to compute answers to queries such as "What is the quarterly moving average of the Dow Jones Industrial average" or "List all employees and their cumulative salaries for each department." Moving averages and cumulative sums can be calculated over various intervals, and aggregations and ranks can be partitioned such that aggregate calculation is reset when partition values change. Within the scope of a single query expression, you can define several different OLAP functions, each with its own arbitrary partitioning rules. Analytical functions can be broken into two categories

- Simple aggregate functions, such as AVG, COUNT, MAX, MIN, and SUM summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement.
- Unary statistical aggregate functions that take one argument include STDDEV, STDDEV_SAMP, STDDEV_POP, VARIANCE, VAR_SAMP, and VAR_POP().

Both the simple and unary categories of aggregates summarize data over a group of rows from the database and can be used with a window specification to compute a moving window over a result set as it is processed.

Note The aggregate functions AVG, SUM, STDDEV, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP, and VARIANCE do not support binary data types BINARY and VARBINARY.

Simple aggregate functions

Simple aggregate functions, such as AVG, COUNT, MAX, MIN, and SUM summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. These aggregates are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement.

Note With the exception of Grouping() functions, both the simple and unary aggregates can be used in a windowing function that incorporates a <window clause> in a SQL query specification (a **window**) that conceptually creates a moving window over a result set as it is processed. See “Windowing” on page 67.

For more information, see “Aggregate functions,” Chapter 5, “SQL Functions,” in the *Sybase IQ Reference Manual*.

Windowing

A major feature of the ANSI SQL extensions for OLAP is a construct called a **window**. This windowing extension let users divide result sets of a query (or a logical partition of a query) into groups of rows called partitions and determine subsets of rows to aggregate with respect to the current row.

You can use three classes of window functions with a window: ranking functions, the row numbering function, and window aggregate functions.

```
<WINDOWED TABLE FUNCTION TYPE> ::=
   <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
   | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
   | <WINDOW AGGREGATE FUNCTION>
```

See also “Grammar rule 6” on page 109.

Windowing extensions specify a window function type over a window name or specification and are applied to partitioned result sets within the scope of a single query expression. A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER clause:

```
olap_function() OVER (PARTITION BY col1, col2...)
```

Windowing operations let you establish information such as the ranking of each row within its partition, the distribution of values in rows within a partition, and similar operations. Windowing also lets you compute moving averages and sums on your data, enhancing the ability to evaluate your data and its impact on your operations.

A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER() clause:

```
OLAP_FUNCTION() OVER (PARTITION BY col1, col2...)
```

An OLAP window's three essential parts

The OLAP windows comprise three essential aspects: window partitioning, window ordering, and window framing. Each has a significant impact on the specific rows of data visible in a window at any point in time. Meanwhile, the OLAP OVER clause differentiates OLAP functions from other analytic or reporting functions with three distinct capabilities:

- Defining window partitions (PARTITION BY clause). See "Window partitioning" on page 69.
- Ordering rows within partitions (ORDER BY clause). See "Window ordering" on page 70.
- Defining window frames (ROWS/RANGE specification). "Window framing" on page 71.

A name can be specified for an OLAP window specification. This name can be used to specify multiple windows functions to avoid redundant window definitions. In this usage, the keyword, WINDOW, is followed by at least one window definition, separated by commas. A window definition includes the name by which the window is known in the query and the details from the windows specification, which lets you to define window partitioning, ordering, and framing:

```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>

<WINDOW DEFINITION LIST> ::=
    <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
    } . . . ]

<WINDOW DEFINITION> ::=
    <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>

<WINDOW SPECIFICATION DETAILS> ::=
    [ <EXISTING WINDOW NAME> ]
    [ <WINDOW PARTITION CLAUSE> ]
    [ <WINDOW ORDER CLAUSE> ]
    [ <WINDOW FRAME CLAUSE> ]
```

For each row in a window partition, users can define a window frame, which may vary the specific range of rows used to perform any computation on the current row of the partition. The current row provides the reference point for determining the start and end points of the window frame.

Window specifications can be based on either a physical number of rows using a window specification that defines a window frame unit of ROWS or a logical interval of a numeric value, using a window specification that defines a window frame unit of RANGE. See "Window framing" on page 71 for details.

Within OLAP windowing operations, you can use the following functional categories:

- "Ranking functions" on page 82
- "Windowing aggregate functions" on page 87
- "Statistical aggregate functions" on page 89
- "Distribution functions" on page 90

Window partitioning

Window partitioning is the division of user-specified result sets (input rows) using a PARTITION BY clause. A partition is defined by one or more value expressions separated by commas. Partitioned data is also implicitly sorted and the default sort order is ascending (ASC).

```
<WINDOW PARTITION CLAUSE> ::=
   PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

If a window partition clause is not specified, then the input is treated as single partition.

Note The term, partition, as used with analytic functions, refers only to dividing the set of result rows using a PARTITION BY clause.

A window partition can be defined based on an arbitrary expression. Also, because window partitioning occurs after GROUPING (if a GROUP BY clause is specified), the result of any aggregate function, such as SUM, AVG, and VARIANCE, can be used in a partitioning expression. Therefore, partitions provide another opportunity to perform grouping and ordering operations *in addition to* the GROUP BY and ORDER BY clauses; for example, you can construct queries that compute aggregate functions over aggregate functions, such as the maximum SUM of a particular quantity.

You can specify a PARTITION BY clause, even it there is no GROUP BY clause.

Window ordering

Window ordering is the arrangement of results (rows) within each window partition using a window order clause, which contains one or more value expressions separated by commas. If a window order clause is not specified, the input rows could be processed in an arbitrary order.

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

The OLAP window order clause is different from the ORDER BY clause that can be appended to a non-windowed query expression. See “Grammar rule 31” on page 111.

The ORDER BY clause in an OLAP function, for example, typically defines the expressions for sorting rows within window partitions; however, you can use the ORDER BY clause without a PARTITION BY clause, in which case the sort specification ensures that the OLAP function is applied to a meaningful (and intended) ordering of the intermediate result set.

An order specification is a prerequisite for the ranking family of OLAP functions; it is the ORDER BY clause, not an argument to the function itself, that identifies the measures for the ranking values. In the case of OLAP aggregates, the ORDER BY clause is not required in general, but it is a prerequisite to defining a window frame. (See “Window framing” on page 71.) This is because the partitioned rows must be sorted before the appropriate aggregate values can be computed for each frame.

The ORDER BY clause includes semantics for defining ascending and descending sorts, as well as rules for the treatment of NULL values. By default, OLAP functions assume an ascending order, where the lowest measured value is ranked 1.

Although this behavior is consistent with the default behavior of the ORDER BY clause that ends a SELECT statement, it is counter-intuitive for most sequential calculations. OLAP calculations often require a descending order, where the highest measured value is ranked 1; this requirement must be explicitly stated in the ORDER BY clause with the DESC keyword.

Note Ranking functions require a <window order clause> because they are defined only over sorted input. As with an <order by clause> in a <query specification>, the default sort sequence is ascending.

The use of a <window frame unit> of RANGE also requires the existence of a <window order clause>. In the case of RANGE, the <window order clause> may only consist of a single expression. See “Window framing.”

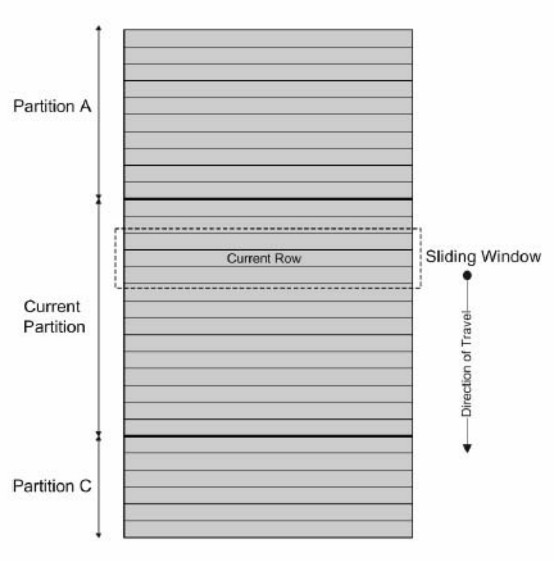
Window framing

For non-ranking aggregate OLAP functions, you can define a window frame with a window frame clause, which specifies the beginning and end of the window relative to the current row.

```
<WINDOW FRAME CLAUSE> ::=
   <WINDOW FRAME UNIT>
   <WINDOW FRAME EXTENT>
```

This OLAP function is computed with respect to the contents of a moving frame rather than the fixed contents of the whole partition. Depending on its definition, the partition has a start row and an end row, and the window frame slides from the starting point to the end of the partition.

Figure 4-3: Three-row moving window with partitioned input



UNBOUNDED PRECEEDING and FOLLOWING

Window frames can be defined by an unbounded aggregation group that either extends back to the beginning of the partition (UNBOUNDED PRECEDING) or extends to the end of the partition (UNBOUNDED FOLLOWING), or both.

UNBOUNDED PRECEDING includes all rows within the partition *preceding* the current row, which can be specified with either ROWS or RANGE. UNBOUNDED FOLLOWING includes all rows within the partition *following* the current row, which can be specified with either ROWS or RANGE. See "ROWS" on page 74 and "RANGE" on page 77.

The value FOLLOWING specifies either the range or number of rows following the current row. If ROWS is specified, then the value is a positive integer indicating a number of rows. If RANGE is specified, the window includes any rows that are less than the current row plus the specified numeric value. For the RANGE case, the data type of the windowed value must be comparable to the type of the sort key expression of the ORDER BY clause. There can be only one sort key expression, and the data type of the sort key expression must allow *addition*.

The value PREDCEEDING specifies either the range or number of rows preceding the current row. If ROWS is specified, then the value is a positive integer indicating a number of rows. If RANGE is specified, the window includes any rows that are less than the current row minus the specified numeric value. For the RANGE case, the data type of the windowed value must be comparable to the type of the sort key expression of the ORDER BY clause. There can be only one sort key expression, and the data type of the sort key expression must allow *subtraction*. This clause cannot be specified in second bound group if the first bound group is CURRENT ROW or value FOLLOWING.

The combination BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING provides an aggregate over an entire partition, without the need to construct a join to a grouped query. An aggregate over an entire partition is also known as a reporting aggregate.

CURRENT ROW concept

In physical aggregation groups, rows are included or excluded based on their position relative to the current row, by counting adjacent rows. The current row is simply a reference to the next row in a query's intermediate results. As the current row advances, the window is reevaluated based on the new set of rows that lie within the window. There is no requirement that the current row be included in a window.

If a window frame clause is not specified, the default window frame depends on whether or not a window order clause is specified:

- If the window specification contains a window order clause, the window's start point is UNBOUNDED PRECEDING, and the end point is CURRENT ROW, thus defining a varying-size window suitable for computing cumulative values.
- If the window specification does not contain a window order clause, the window's start point is UNBOUNDED PRECEDING, and the end point is UNBOUNDED FOLLOWING, thus defining a window of fixed size, regardless of the current row.

Note A window frame clause cannot be used with a ranking function.

You can also define a window by specifying a window frame unit that is row-based (rows specification) or value-based (range specification).

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE

<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW
FRAME BETWEEN>
```

When a window frame extent specifies BETWEEN, it explicitly provides the beginning and end of a window frame.

If the window frame extent specifies only one of these two values then the other value defaults to CURRENT ROW.

ROWS

The window frame unit, ROWS, defines a window in the specified number of rows before or after the current row, which serves as the reference point that determines the start and end of a window. Each analytical calculation is based on the current row within a partition. To produce determinative results for a window expressed in rows, the ordering expression should be unique.

The reference point for all window frames is the current row. The SQL/OLAP syntax provides mechanisms for defining a row-based window frame as any number of rows preceding or following the current row or preceding and following the current row.

The following list illustrates common examples of a window frame unit:

- *Rows Between Unbounded Preceding and Current Row* – specifies a window whose start point is the beginning of each partition and the end point is the current row and is often used to construct windows that compute cumulative results, such as cumulative sums.
- *Rows Between Unbounded Preceding and Unbounded Following* – specifies a fixed window, regardless of the current row, over the entire partition. The value of a window aggregate function is, therefore, identical in each row of the partition.
- *Rows Between 1 Preceding and 1 Following* – specifies a fixed-sized moving window over three adjacent rows, one each before and after the current row. You can use this window frame unit to compute, for example, a 3-day or 3-month moving average. See Figure 4-3 on page 72.

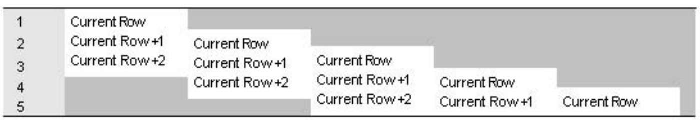
Be aware of non-meaningful results that could be generated using ROWS due to gaps in the windowed values. If the set of values is not continuous, consider using RANGE instead of ROWS, because a window definition based on RANGE automatically handle adjacent rows with duplicate values and will not include other rows when there are gaps in the range.

Note In the case of a moving window, it is assumed that rows containing NULL values exist before the first row, and after the last row, in the input. What this means is that with a 3-row moving window, the computation for the last row in the input—the current row—will include the immediately preceding row and a NULL value.

- *Rows Between Current Row and Current Row* – restricts the window to the current row only.
- *Rows Between 1 Preceding and 1 Preceding* – specifies a single row window consisting only of the preceding row, with respect to the current row. In combination with another window function that computes a value based on the current row only, this construction makes it possible to easily compute deltas, or differences in value, between adjacent rows. See “Computing deltas between adjacent rows” on page 79.

Row-based window frames In the example in Figure 4-4, rows [1] through [5] represent a partition; each row becomes the current row as the OLAP window frame slides forward. The frame is defined as Between Current Row And 2 Following, so each frame includes a maximum of three rows and a minimum of one row. When the frame reaches the end of the partition, only the current row is included. The shaded areas indicate which rows are excluded from the frame at each step in Figure 4-4.

Figure 4-4: Row-based window frames

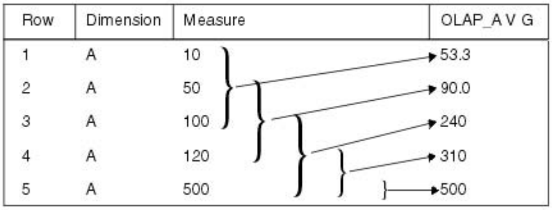


The window frame in Figure 4-4 imposes the following rules:

- When row [1] is the current row, rows [4] and [5] are excluded.
- When row [2] is the current row, rows [5] and [1] are excluded.
- When row [3] is the current row, rows [1] and [2] are excluded.
- When row [4] is the current row, rows [1], [2], and [3] are excluded.

- When row [5] is the current row, rows [1], [2], [3], and [4] are excluded.

The following diagram applies these rules to a specific set of values, showing the OLAP AVG function that would be calculated for each row. The sliding calculations produce a moving average with an interval of three rows or fewer, depending on which row is the current row:



The following example demonstrates a sliding window:

```
SELECT dimension, measure,
    AVG(measure) OVER(partition BY dimension
        ORDER BY measure
        ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
        AS olap_avg
FROM ...
```

The averages are computed as follows:

- Row [1] = (10 + 50 + 100)/3
- Row [2] = (50+ 100 + 120)/3
- Row [3] = (100 + 120 + 500)/3
- Row [4] = (120 + 500 + NULL)/3
- Row [5] = (500 + NULL + NULL)/3

Similar calculations would be computed for all subsequent partitions in the result set (such as, B, C, and so on).

If there are no rows in the current window, the result is NULL, except for COUNT.

RANGE

Range-based window frames The previous example, Row-based window frames, demonstrates one among many row-based window frame definitions. The SQL/OLAP syntax also supports another kind of window frame whose limits are defined in terms of a value-based—or range-based—set of rows, rather than a specific sequence of rows.

Value-based window frames define rows within a window partition that contain a specific range of numeric values. The OLAP function's ORDER BY clause defines the numeric column to which the range specification is applied, relative to the current row's value for that column. The range specification uses the same syntax as the rows specification, but the syntax is interpreted in a different way.

The window frame unit, RANGE, defines a window frame whose contents are determined by finding rows in which the ordering column has values within the specified range of value relative to the current row. This is called a logical offset of a window frame, which you can specify with constants, such as "3 preceding," or any expression that can be evaluated to a numeric constant. When using a window defined with RANGE, there can be only a single numeric expression in the ORDER BY clause.

For example, a frame could be defined as the set of rows with *year* values some number of years preceding or following the current row's year:

```
ORDER BY year ASC range BETWEEN CURRENT ROW and 1
PRECEDING
```

In the above example query, 1 preceding means the current row's *year* value minus 1.

This kind of range specification is inclusive. If the current row's *year* value is 2000, all rows in the window partition with year values 2000 and 1999 qualify for the frame, regardless of the physical position of those rows in the partition. The rules for including and excluding value-based rows are quite different from the rules applied to row-based frames, which depend entirely on the physical sequence of rows.

Put in the context of an OLAP AVG() calculation, the following partial result set further demonstrates the concept of a value-based window frame. Again, the frame consists of rows that:

- Have the same year as the current row

- Have the same year as the current row minus 1

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

The following query demonstrates a range-based window definition:

```
SELECT dimension, year, measure,
   AVG(measure) OVER(PARTITION BY dimension
      ORDER BY year ASC
      range BETWEEN CURRENT ROW and 1 PRECEDING)
      as olap_avg
FROM ...
```

The averages are computed as follows:

- Row [1] = 1999; rows [2] through [5] are excluded; AVG = 10,000/1
- Row [2] = 2001; rows [1], [4], and [5] are excluded; AVG = 6,000/2
- Row [3] = 2001; rows [1], [4], and [5] are excluded; AVG = 6,000/2
- Row [4] = 2002; row [1] is excluded; AVG = 21,000/4
- Row [5] = 2002; row [1] is excluded; AVG = 21,000/4

Ascending and descending order for value-based frames The ORDER BY clause for an OLAP function with a value-based window frame not only identifies the numeric column on which the range specification is based; it also declares the sort order for the ORDER BY values. The following specification is subject to the sort order that precedes it (ASC or DESC):

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

The specification *n* FOLLOWING means:

- Plus *n* if the partition is sorted in default ascending order (ASC)
- Minus *n* if the partition is sorted in descending order (DESC)

For example, assume that the year column contains four distinct values, from 1999 to 2002. The following table shows the default ascending order of these values on the left and the descending order on the right:

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

If the current row is 1999 and the frame is specified as follows, rows that contain the values 1999 and 1998 (which doesn't exist in the table) are included in the frame:

```
ORDER BY year ASC range BETWEEN CURRENT ROW and 1
FOLLOWING
```

Note The sort order of the ORDER BY values is a critical part of the test for qualifying rows in a value-based frame; the numeric values alone do not determine exclusion or inclusion.

Using an unbounded window The following query produces a result set consisting of all of the products accompanied by the total quantity of all products:

```
SELECT id, description, quantity,
   SUM(quantity) OVER () AS total
FROM product;
```

Computing deltas between adjacent rows Using two windows—one over the current row and the other over the previous row—provides a direct way of computing deltas, or changes, between adjacent rows. See the following query example and results.

```
SELECT emp_id, emp_lname, SUM(salary) OVER (ORDER BY
   birth_date rows between current row and current row)
   AS curr, SUM(salary) OVER (ORDER BY birth_date rows
   between 1 preceding and 1 preceding) AS prev, (curr
   -prev) as delta
FROM employee WHERE state IN ('MA', 'AZ') AND dept_id
   =100
ORDER BY emp_id, emp_lname;
```

The following are the results from the query:

emp_id	emp_lname	curr	prev	delta
102	Whitney	45700.000	64500.000	-18800.000
105	Cobb	62000.000	68400.000	-6400.000
160	Breault	57490.000	96300.000	-38810.000
243	Shishov	72995.000	59840.000	13155.000
247	Driscoll	48023.690	87900.000	-39876.310
249	Guevara	42998.000	48023.690	-5025.690
266	Gowda	59840.000	57490.000	2350.000
278	Melkisetian	48500.000	74500.000	-26000.000

```
 316    Pastor       74500.000  62000.000    12500.000
 445    Lull         87900.000  67890.000    20010.000
 453    Rabkin       64500.000  42998.000    21502.000
 479    Siperstein   39875.500  42500.000    -2624.500
 501    Scott        96300.000  54900.000    41400.000
 529    Sullivan     67890.000  72995.000    -5105.000
 582    Samuels      37400.000  39875.500    -2475.500
 604    Wang         68400.000  45700.000    22700.000
 839    Marshall     42500.000  48500.000    -6000.000
1157    Soo          39075.000  37400.000     1675.000
1250    Diaz         54900.000
```

Although the window function SUM() is used, the sum contains only the salary value of either the current or previous row because of how the window is specified. Also, the prev value of the first row in the result is NULL because it has no predecessor; therefore, the delta is NULL as well.

In each of the examples above, the function used with the OVER() clause is the SUM() aggregate function.

Explicit and in-line window clauses

SQL OLAP provides two ways of specifying a window in a query:

- The explicit window clause lets you define a window that follows a HAVING clause. You reference windows defined with those window clauses by specifying their names when you invoke an OLAP function, such as

  ```
  SUM ( ...) OVER w2
  ```

- The in-line window specification lets you define a window in the SELECT list of a query expression. This capability lets you define your windows in a window clause that follows the HAVING clause and then reference them by name from your window function invocations, or to define them along with the function invocations.

> **Note** If you use an in-line window specification, you cannot name the window. Two or more window function invocations in a single SELECT list that use identical windows must either reference a named window defined in a window clause or they must define their in-line windows redundantly.

Window function example The following example shows a window function. The query returns a result set that partitions the data by department and then provides a cumulative summary of employees' salaries starting with the employee who has been at the company the longest. The result set includes only those employees who reside in Massachusetts. The column Sum_Salary provides the cumulative total of employees' salaries.

```
SELECT dept_id, emp_lname, start_date, salary,
   SUM(salary) OVER (PARTITION BY dept_id ORDER BY
   start_date rows between unbounded preceding and
   current row) AS sum_salary
FROM employee
WHERE state IN ('MA') AND dept_id IN (100, 200)
ORDER BY dept_id;
```

The following result set is partitioned by department.

dept_id	emp_lname	start_date	salary	sum_salary
100	Whitney	1984-08-28	45700.000	45700.000
100	Cobb	1985-01-01	62000.000	107700.000
100	Breault	1985-06-17	57490.000	165190.000
100	Shishov	1986-06-07	72995.000	238185.000
100	Driscoll	1986-07-01	48023.690	286208.690
100	Guevara	1986-10-14	42998.000	329206.690
100	Gowda	1986-11-30	59840.000	389046.690
100	Melkisetian	1986-12-06	48500.000	437546.690
100	Pastor	1987-04-26	74500.000	512046.690
100	Lull	1987-06-15	87900.000	599946.690
100	Rabkin	1987-06-15	64500.000	664446.690
100	Siperstein	1987-07-23	39875.500	704322.190
100	Scott	1987-08-04	96300.000	800622.190
100	Sullivan	1988-02-03	67890.000	868512.190

```
100       Samuels     1988-03-23   37400.000    905912.190
100       Wang        1988-09-29   68400.000    974312.190
100       Marshall    1989-04-20   42500.000   1016812.190
100       Soo         1990-07-31   39075.000   1055887.190
100       Diaz        1990-08-19   54900.000   1110787.190
200       Dill        1985-12-06   54800.000     54800.000
200       Powell      1988-10-14   54600.000    109400.000
200       Poitras     1988-11-28   46200.000    155600.000
200       Singer      1989-06-01   34892.000    190492.000
200       Kelly       1989-10-01   87500.000    277992.000
200       Martel      1989-10-16   55700.000    333692.000
200       Sterling    1990-04-29   64900.000    398592.000
200       Chao        1990-05-13   33890.000    432482.000
200       Preston     1990-07-11   37803.000    470285.000
200       Goggin      1990-08-05   37900.000    508185.000
200       Pickett     1993-08-12   47653.000    555838.000
```

Ranking functions

Ranking functions let you compile a list of values from the data set in ranked order, as well as compose single-statement SQL queries that answer questions such as, “Name the top 10 products shipped this year by total sales,” or “Give the top 5% of salespersons who sold orders to at least 15 different companies.” The functions include RANK(), DENSE_RANK(), PERCENT_RANK(), and NTILE() with a PARTITION BY clause. See “Ranking functions” on page 82.

SQL/OLAP defines four functions that are categorized as ranking functions:

```
<RANK FUNCTION TYPE> ::=
   RANK | DENSE RANK | PERCENT RANK | NTILE
```

Ranking functions let you compute a rank value for each row in a result set based on the order specified in the query. For example, a sales manager might need to identify the top or bottom sales people in the company, the highest- or lowest-performing sales region, or the best- or worst-selling products. Ranking functions can provide this information.

RANK() function

The RANK function returns a number that indicates the rank of the current row among the rows in the row's partition, as defined by the ORDER BY clause. The first row in a partition has a rank of 1, and the last rank in a partition containing 25 rows is 25. RANK is specified as a syntax transformation, which means that an implementation can choose to actually transform RANK into its equivalent, or it can merely return a result equivalent to the result that transformation would return.

In the following example, ws1 indicates the window specification that defines the window named w1.

```
RANK() OVER ws
```

is equivalent to

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

The transformation of the RANK function uses logical aggregation (RANGE). As a result, two or more records that are tied—or have equal values in the ordering column—will have the same rank.The next group in the partition that has a different value will have a rank that is more than one greater than the rank of the tied rows. For example, if there are rows whose ordering column values are 10, 20, 20, 20, 30, the rank of the first row is 1 and the rank of the second row is 2. The rank of the third and fourth row is also 2, but the rank of the fifth row is 5. There are no rows whose rank is 3 or 4. This algorithm is sometimes known as sparse ranking.

See also "RANK function [Analytical]," Chapter 5, "SQL Functions," in the *Sybase IQ Reference Manual*.

DENSE_RANK() function

While RANK returns duplicate values in the ranking sequence when there are ties between values, DENSE_RANK returns ranking values without gaps. The values for rows with ties are still equal, but the ranking of the rows represents the positions of the clusters of rows having equal values in the ordering column, rather than the positions of the individual rows. As in the RANK example, where rows ordering column values are 10, 20, 20, 20, 30, the rank of the first row is still 1 and the rank of the second row is still 2, as are the ranks of the third and fourth rows. The last row, however, is 3, not 5.

DENSE_RANK is computed through a syntax transformation, as well.

```
DENSE_RANK() OVER ws
```

is equivalent to

```
COUNT ( DISTINCT ROW ( expr_1, . . ., expr_n ) )
  OVER ( ws RANGE UNBOUNDED PRECEDING )
```

In the above example, *expr_1* through *expr_n* represent the list of value expressions in the sort specification list of window w1.

See also "DENSE_RANK function [Analytical]," Chapter 5, "SQL Functions," in the *Sybase IQ Reference Manual*.

PERCENT_RANK() function

The PERCENT_RANK function calculates a percentage for the rank, rather than a fractional amount, and returns a decimal value between 0 and 1. In other words, PERCENT_RANK returns the relative rank of a row, which is a number that indicates the relative position of the current row within the window partition in which it appears. For example, in a partition that contains 10 rows having different values in the ordering columns, the third row would be given a PERCENT_RANK value of 0.222 …, because you have covered 2/9 (22.222...%) of rows following the first row of the partition. PERCENT_RANK of a row is defined as one less than the RANK of the row divided by one less than then umber of rows in the partition, as seen in the following example (where "ANT" stands for an approximate numeric type, such as REAL or DOUBLE PRECISION).

```
PERCENT_RANK() OVER ws
```

is equivalent to

```
CASE
    WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
    PRECEDING AND UNBOUNDED FOLLOWING ) = 1
    THEN CAST (0 AS ANT)
    ELSE
      ( CAST ( RANK () OVER ( ws ) AS ANT ) -1 /
      ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
      PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
END
```

See also Chapter , "PERCENT_RANK function [Analytical]," Chapter 5, "SQL Functions," in the *Sybase IQ Reference Manual*.

Ranking examples

Ranking example 1 The SQL query that follows finds the male and female employees from Utah and ranks them in descending order according to salary.

```
SELECT emp_lname, salary, sex, RANK() OVER (ORDER BY
   salary DESC) AS Rank
FROM employee WHERE state IN ('MA') AND dept_id =100
ORDER BY salary DESC;
```

The following are the results from the above query:

```
emp_lname      salary        sex    rank
---------      --------      ---    ----
Scott         96300.000      M         1
Lull          87900.000      M         2
Pastor        74500.000      F         3
Shishov       72995.000      F         4
Wang          68400.000      M         5
Sullivan      67890.000      F         6
Rabkin        64500.000      M         7
Cobb          62000.000      M         8
Gowda         59840.000      M         9
Breault       57490.000      M        10
Diaz          54900.000      M        11
Melkisetian   48500.000      F        12
Driscoll      48023.690      M        13
Whitney       45700.000      F        14
Guevara       42998.000      M        15
Marshall      42500.000      M        16
Siperstein    39875.500      F        17
Soo           39075.000      M        18
Samuels       37400.000      M        19
```

Ranking example 2 Using the query from Ranking example 1, you can change the data by partitioning it by gender. The following example ranks employees in descending order by salary and partitions by gender:

```
SELECT emp_lname, salary, sex, RANK() OVER (PARTITION
   BY sex ORDER BY salary DESC) AS RANK
FROM employee WHERE state IN ('MA', 'AZ') AND dept_id
   IN (100, 200)
ORDER BY sex, salary DESC;
```

The following are the results from the above query:

```
emp_lname            salary       sex      rank
---------         ---------       ---      ----
Kelly             87500.000       F           1
Pastor            74500.000       F           2
Shishov           72995.000       F           3
```

```
Sullivan          67890.000     F          4
Melkisetian       48500.000     F          5
Pickett           47653.000     F          6
Poitras           46200.000     F          7
Whitney           45700.000     F          8
Siperstein        39875.500     F          9
Scott             96300.000     M          1
Lull              87900.000     M          2
Wang              68400.000     M          3
Sterling          64900.000     M          4
Rabkin            64500.000     M          5
Cobb              62000.000     M          6
Gowda             59840.000     M          7
Breault           57490.000     M          8
Martel            55700.000     M          9
Diaz              54900.000     M         10
Dill              54800.000     M         11
Powell            54600.000     M         12
Driscoll          48023.690     M         13
Guevara           42998.000     M         14
Marshall          42500.000     M         15
Soo               39075.000     M         16
Goggin            37900.000     M         17
Preston           37803.000     M         18
Samuels           37400.000     M         19
Singer            34892.000     M         20
Chao              33890.000     M         21
```

Ranking example 3 This example takes a list of female employees in California and Texas and ranks them in descending order according to salary. The PERCENT_RANK function is used to provide a cumulative total in descending order.

```
SELECT emp_lname, salary, sex, CAST(PERCENT_RANK() OVER
   (ORDER BY salary DESC) AS numeric (4, 2)) AS RANK
FROM employee WHERE state IN ('CA', 'TX') AND sex ='F'
ORDER BY salary DESC;
```

The following are the results from the above query:

emp_lname	salary	sex	percent
Savarino	72300.000	F	0.00
Smith	51411.000	F	0.33
Clark	45000.000	F	0.66
Garcia	39800.000	F	1.00

Ranking example 4 You can use the PERCENT_RANK function to find the top or bottom percentiles in the data set. In this example, the query returns male employees whose salary is in the top five percent of the data set.

```
SELECT * FROM (SELECT emp_lname, salary, sex,
   CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM employee WHERE state IN ('MA') AND sex ='F' ) AS
   DT where percent > 0.5
ORDER BY salary DESC;
```

The following are the results from the above query:

emp_lname	salary	sex	percent
Whitney	45700.000	F	0.51
Barletta	45450.000	F	0.55
Higgins	43700.000	F	0.59
Siperstein	39875.500	F	0.62
Coe	36500.000	F	0.66
Espinoza	36490.000	F	0.70
Wetherby	35745.000	F	0.74
Braun	34300.000	F	0.77
Butterfield	34011.000	F	0.81
Bigelow	31200.000	F	0.85
Bertrand	29800.000	F	0.88
Lambert	29384.000	F	0.92
Kuo	28200.000	F	0.96
Romero	27500.000	F	1.00

Windowing aggregate functions

Windowing aggregate functions let you manipulate multiple levels of aggregation in the same query. For example, listing all quarters in which expenses are less than the average. Aggregate functions, including the simple aggregate functions AVG, COUNT, MAX, MIN, and SUM, can be used to place results—possibly computed at different levels in the statement—on the same row. This placement provides a means to compare aggregate values with detail rows within a group, avoiding the need for a join or a correlated subquery.

These functions also let you compare non-aggregate values to aggregate values. For example, a salesperson might need to compile a list of all customers who ordered more than the average number of a product in a specified year, or a manager might want to compare an employee's salary against the average salary of the department.

If a query specifies DISTINCT in the SELECT statement, then the DISTINCT operation is applied after the window operator. (A window operator is computed after processing the GROUP BY clause and before the evaluation of the SELECT list items and a query's ORDER BY clause.).

Windowing aggregate example 1 In this example, the query returns a result set, partitioned by year, that shows a list of the products that sold higher-than-average sales.

```
SELECT * FROM (SELECT year(order_date) AS Y, prod_id,
   SUM(quantity) AS Q,  CAST(AVG(SUM(quantity)) OVER
   (PARTITION BY Y) AS numeric (8, 2)) AS Average
FROM alt_sales_order S, alt_sales_order_items O
WHERE S.id = O.id
GROUP BY Y, O.prod_id ) AS derived_table
   WHERE Q > Average
ORDER BY Y, prod_id;
```

The following are the results from the query:

```
Year      prod_id        Q      Average
----      -------     ----      -------
2000          400     2030      1787.00
2000          600     2124      1787.00
2000          601     1932      1787.00
2000          700     2700      1787.00
2001          400     1248      1048.90
2001          401     1057      1048.90
2001          700     1836      1048.90
```

For the year 1993, the average number of orders was 1,787. Four products (700, 601, 600, and 400) sold higher than that amount. In 1994, the average number of orders was 1,048 and three products exceeded that amount.

Windowing aggregate example 2 In this example, the query returns a result set that shows the employees whose salary is one standard deviation greater than the average salary of their department. Standard deviation is a measure of how much the data varies from the mean.

```
SELECT * FROM (SELECT emp_lname AS E_name, dept_id AS
   Dept, CAST(salary AS numeric(10,2) ) AS Sal,
   CAST(AVG(Sal) OVER(PARTITION BY dept_id) AS
   numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
   OVER(PARTITION BY dept_id) AS numeric(10,2)) AS
   STD_DEV
FROM employee
```

```
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
   Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

The results of this query are as follows:. Every department has at least one employee whose salary significantly deviates from the mean.

Employee	Dept	Salary	Average	Std_Dev
Lull	100	87900.00	58736.28	16829.59
Sheffield	100	87900.00	58736.28	16829.59
Scott	100	96300.00	58736.28	16829.59
Sterling	200	64900.00	48390.94	13869.59
Savarino	200	72300.00	48390.94	13869.59
Kelly	200	87500.00	48390.94	13869.59
Shea	300	138948.00	59500.00	30752.39
Blaikie	400	54900.00	43640.67	11194.02
Morris	400	61300.00	43640.67	11194.02
Evans	400	68940.00	43640.67	11194.02
Martinez	500	55500.80	33752.20	9084.49

Employee Scott earns $96,300.00 while the departmental average is $58,736.28. The standard deviation for that department is 16,829.00, which means that salaries less than $75,565.88 (58736.28 + 16829.60 = 75565.88) fall within one standard deviation of the mean. At $96,300.00, employee Scott is well above that figure.

Statistical aggregate functions

The ANSI SQL/OLAP extensions provide a number of additional aggregate functions that permit statistical analysis of numeric data. This support includes functions to compute variance, standard deviation, correlation, and linear regression.

Standard deviation and variance

The SQL/OLAP general set functions that take one argument include STDDEV, STDDEV_POP, STDDEV_SAMP, VARIANCE, VAR_POP, and VAR_SAMP.

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
   <BASIC AGGREGATE FUNCTION TYPE>
   | STDDEV | STDDEV_POP | STDDEV_SAMP
   | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

- STDDEV_POP – Computes the population standard deviation of the provided value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.
- STDDEV_SAMP – Computes the population standard deviation of the provided value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.
- VAR_POP – Computes the population variance of value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression from the mean of value expression, divided by the number of rows (remaining) in the group or partition.
- VAR_SAMP – Computes the sample variance of value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression, divided by one less than the number of rows (remaining) in the group or partition.

These functions, including STDDEV and VARIANCE, are true aggregate functions in that they can compute values for a partition of rows as determined by the query's ORDER BY clause. As with other basic aggregate functions such as MAX or MIN, their computation ignores NULL values in the input. Also, regardless of the domain of the expression being analyzed, all variance and standard deviation computation uses IEEE double-precision floating point. If the input to any variance or standard deviation function is the empty set, then each function returns NULL as its result. If VAR_SAMP is computed for a single row, it returns NULL, while VAR_POP returns the value 0.

Distribution functions

SQL/OLAP defines several functions that deal with ordered sets. The two inverse distribution functions are named PERCENTILE_CONT and PERCENTILE_DISC. These analytical functions take a percentile value as the function argument and operate on a group of data specified in the WITHIN GROUP clause or operate on the entire data set.

These functions return one value per group. For PERCENTILE_DISC (discrete), the data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. For PERCENTILE_CONT (continuous), the data type of the results is either numeric, if the ORDER BY item in the WITHIN GROUP clause is a numeric, or double, if the ORDER BY item is an integer or floating point.

The inverse distribution analytical functions require a WITHIN GROUP (ORDER BY) clause. For example:

PERCENTILE_CONT (*expression1*)
WITHIN GROUP (**ORDER BY** *expression2* [**ASC** | **DESC**])

The value of *expression1* must be a constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a "wrong argument for percentile" error is returned. If the argument value is less than 0, or greater than 1, then a "data value out of range" error is returned.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is *not* an ORDER BY for the SELECT statement.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result.

The value *expression2* is a sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Inverse distribution analytical functions are allowed in a subquery, a HAVING clause, a view, or a union. The inverse distribution functions can be used anywhere the simple non analytical aggregate functions are used. The inverse distribution functions ignore the NULL value in the data set.

PERCENTILE_CONT example This example uses the PERCENTILE_CONT function to determine the 10th percentile value for car sales in a region using the following data set:

```
sales       region       dealer_name
-----       ---------    -----------
900         Northeast    Boston
800         Northeast    Worcester
800         Northeast    Providence
```

```
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

In the following example query, the SELECT statement contains the PERCENTILE_CONT function:

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the SELECT statement lists the 10th percentile value for car sales in a region:

region	percentile_cont
Northeast	840
Northwest	740
South	470

PERCENTILE_DISC example This example uses the PERCENTILE_DISC function to determine the 10th percentile value for car sales in a region, using the following data set:

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin

```
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

In the following query, the SELECT statement contains the PERCENTILE_DISC function:

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP
    (ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the SELECT statement lists the 10th percentile value for car sales in each region:

```
region           percentile_cont
---------        ---------------
Northeast        900
Northwest        800
South            500
```

For more information about the distribution functions, see "PERCENTILE_CONT function [Analytical]" and "PERCENTILE_DISC function [Analytical]," Chapter 5, "SQL Functions," in the *Sybase IQ Reference Manual*.

Numeric functions

OLAP numeric functions supported by Sybase IQ include CEILING (CEIL is an alias), EXP (EXPONENTIAL is an alias), FLOOR, LN (LOG is an alias), SQRT, and WIDTH_BUCKET.

```
<numeric value function> :: =
  <natural logarithm>
|  <exponential function>
|  <power function>
|  <square root>
|  <floor function>
|  <ceiling function>
|  <width bucket function>
```

The syntax for each supported numeric value function is shown in Table 4-3.

Table 4-3: Numeric value functions and syntax

Numeric value function	Syntax
Natural logarithm	LENGTH (*string-expression*)
Exponential function	EXP (*numeric-expression*)
Power function	POWER (*numeric-expression1, numeric-expression2*)
Square root	SQRT (*numeric-expression*)
Floor function	FLOOR (*numeric-expression*)
Ceiling function	CEILING (*numeric-expression*)
Width bucket function	WIDTH_BUCKET (*expression, min_value, max_value, num_buckets*)

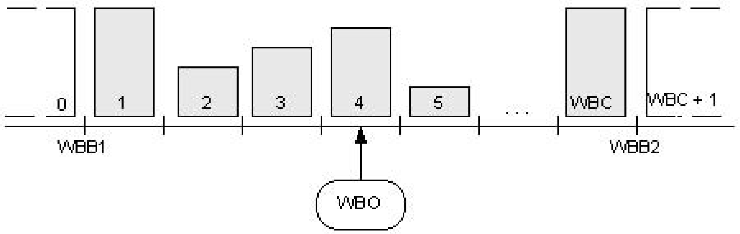
The semantics of the numeric value functions are:

- LN: Returns the natural logarithm of the argument value. Raises an error condition if the argument value is zero or negative. LN is a synonym for LOG.
- EXP: Returns the value computed by raising the value of *e* (the base of natural logarithms) to the power specified by the value of the argument.
- POWER: Returns the value computed by raising the value of the first argument to the power specified by the value of the second argument. If the first argument is zero and the second is zero, returns one. If the first argument is zero and the second is positive, returns zero. If the first argument is zero and the second argument is negative, raises an exception. If the first argument is negative and the second is not an integer, raises an exception.
- SQRT: Returns the square root of the argument value, defined by syntax transformation to "POWER (*expression, 0.5*)."
- FLOOR: Returns the integer value nearest to positive infinity that is not greater than the value of the argument.
- CEILING: Returns the integer value nearest to negative infinity that is not less than the value of the argument. CEIL is a synonym for CEILING.

WIDTH_BUCKET function

The WIDTH_BUCKET function is somewhat more complicated than the other numeric value functions. It accepts four arguments: "live value," two range boundaries, and the number of equal-sized (or as nearly so as possible) partitions into which the range indicated by the boundaries is to be divided. WIDTH_BUCKET returns a number indicating the partition into which the live value should be placed, based on its value as a percentage of the difference between the higher range boundary and the lower boundary. The first partition is partition number one.

In order to avoid errors when the live value is outside the range of boundaries, live values that are less than the smaller range boundary are placed into an additional first bucket, bucket zero, and live values that are greater than the larger range boundary are placed into an additional last bucket, bucket N+1.



For example, WIDTH_BUCKET (*14*, *5*, *30*, *5*) returns 2 because:

- (30-5)/5 is 5, so the range is divided into 5 partitions, each 5 units wide.
- The first bucket represents values from 0.00% to 19.999 …%; the second represents values from 20.00% to 39.999 …%; and the fifth bucket represents values from 80.00% to 100.00%.
- The bucket chosen is determined by computing (5*(14-5)/(30-5)) + 1 — one more than the number of buckets times the ratio of the offset of the specified value from the lower value to the range of possible values, which is (5*0/25) + 1, which is 2.8. This value is the range of values for bucket number 2 (2.0 through 2.999 …), so bucket number 2 is chosen.

WIDTH_BUCKET example

The following example creates a ten-bucket histogram on the credit_limit column for customers in Massachusetts in the sample table and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

Note This example is for illustration purposes only and was not generated using the asiqdemo database.

```
SELECT customer_id, cust_last_name, credit_limit,
   WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
   Group"
   FROM customers WHERE territory = 'MA'
   ORDER BY "Credit Group";
```

CUSTOMER_ID	CUST_LAST_NAME	CREDIT_LIMIT	Credit Group
825	Dreyfuss	500	1
826	Barkin	500	1
853	Palin	400	1
827	Siegel	500	1
843	Oates	700	2
844	Julius	700	2
835	Eastwood	1200	3
840	Elliott	1400	3
842	Stern	1400	3
841	Boyer	1400	3
837	Stanton	1200	3
836	Berenger	1200	3
848	Olmos	1800	4
849	Kaurusmdki	1800	4
828	Minnelli	2300	5
829	Hunter	2300	5
852	Tanner	2300	5
851	Brown	2300	5
850	Finney	2300	5
830	Dutt	3500	7
831	Bel Geddes	3500	7
832	Spacek	3500	7
838	Nicholson	3500	7
839	Johnson	3500	7
833	Moranis	3500	7
834	Idle	3500	7
845	Fawcett	5000	11
846	Brando	5000	11
847	Streep	5000	11

When the bounds are reversed, the buckets are open-closed intervals. For example: WIDTH_BUCKET (*credit_limit*, *5000*, *0*, *5*). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

See also

"BIT_LENGTH function [String]," "EXP function [Numeric]," "FLOOR function [Numeric]," "POWER function [Numeric]," "SQRT function [Numeric]," and "WIDTH_BUCKET function [Numerical]," Chapter 5, "SQL Functions," in the *Sybase IQ Reference Manual*.

OLAP rules and restrictions

OLAP functions can be used

Within SQL queries, OLAP functions can be used under the following conditions:

- In the SELECT list
- In expressions
- As arguments of scalar functions
- In the final ORDER BY clause (by using aliases or positional references to OLAP functions elsewhere in the query)

OLAP functions cannot be used

OLAP functions *cannot* be used under the following conditions:

- In subqueries
- In the search condition of a WHERE clause
- As arguments for SET (aggregate) functions. For example, the following expression is not valid:

```
SUM(RANK() OVER(ORDER BY dollars))
```

- A windowed aggregate cannot be an argument to argument to another unless the inner one was generated within a view or derived table. The same applies to ranking functions.
- Window aggregate and RANK functions are not allowed in a HAVING clause.
- Window aggregate functions should not specify DISTINCT.
- Window function cannot be nested inside of other window functions.

- Inverse distribution functions are not supported with the OVER clause.
- Outer references are not allowed in a window definition clause.
- Correlation references are allowed within OLAP functions, but correlated column aliases are not allowed.

Columns referenced by an OLAP function must be grouping columns or aggregate functions from the same query block in which the OLAP function and the GROUP BY clause appear. OLAP processing occurs after the grouping and aggregation operations and before the final ORDER BY clause is applied; therefore, it must be possible to derive the OLAP expressions from those intermediate results. If there is no GROUP BY clause in a query block, OLAP functions can reference other columns in the select list.

Sybase IQ limitations

The following are the Sybase IQ limitations with SQL OLAP functions:

- User-defined functions in a window frame definition are not supported.
- The constants used in a window frame definition must be unsigned numeric value and should not exceed the value of maximum BIG INT 2^{63-1}.
- Window aggregate functions and RANK functions cannot be used in DELETE and UPDATE statements.
- Window aggregate and RANK functions are not allowed in subqueries.
- CUME_DIST is currently not supported.
- Grouping sets are currently not supported.
- Correlation and linear regression functions are currently not supported.

Additional OLAP examples

This section provides additional examples using the OLAP functions.

Both start and end points of a window may vary as intermediate result rows are processed. For example, computing a cumulative sum involves a window with the start point fixed at the first row of each partition and an end point that slides along the rows of the partition to include the current row. See Figure 4-3 on page 72.

As another example, both the start and end points of the window can be variable yet define a constant number of rows for the entire partition. Such a construction lets users compose queries that compute moving averages; for example, a SQL query that returns a moving three-day average stock price.

Example: Window functions in queries

Consider the following query which lists all products shipped in July and August 2005 and the cumulative shipped quantity by shipping date:

```
SELECT p.id, p.description, s.quantity, s.ship_date,
   SUM(s.quantity) OVER (PARTITION BY prod_id ORDER BY
   s.ship_date rows between unbounded preceding and
   current row)
FROM alt_sales_order_items s JOIN product p on
(s.prod_id =
   p.id) WHERE s.ship_date BETWEEN '2001-05-01' and
   '2001-08-31' AND  s.quantity > 40
ORDER BY p.id;
```

The following are the results from the above query:

```
ID   description    quantity   ship_date   sum quantity
---  -----------    --------   ---------   ------------
302  Crew Neck            60  2001-07-02             60
400  Cotton Cap           60  2001-05-26             60
400  Cotton Cap           48  2001-07-05            108
401  Wool cap             48  2001-06-02             48
401  Wool cap             60  2001-06-30            108
401  Wool cap             48  2001-07-09            156
500  Cloth Visor          48  2001-06-21             48
501  Plastic Visor        60  2001-05-03             60
501  Plastic Visor        48  2001-05-18            108
501  Plastic Visor        48  2001-05-25            156
501  Plastic Visor        60  2001-07-07            216
601  Zipped Sweatshirt    60  2001-07-19             60
700  Cotton Shorts        72  2001-05-18             72
700  Cotton Shorts        48  2001-05-31            120
```

In this example, the computation of the SUM window function occurs after the join of the two tables and the application of the query's WHERE clause. The query uses an in-line window specification that specifies that the input rows from the join is processed as follows:

1 Partition (group) the input rows based on the value of the prod_id attribute.

2 Within each partition, sort the rows by the ship_date attribute.

3 For each row in the partition, evaluate the SUM() function over the quantity attribute, using a sliding window consisting of the first (sorted) row of each partition, up to and including the current row. See Figure 4-3.

An alternative construction for the query is to specify the window separate from the functions that use it. This is useful when more than one window function is specified that are based on the same window. In the case of the query using window functions, a construction that uses the window clause (declaring a window identified by cumulative) is as follows:

```
SELECT p.id, p.description, s.quantity, s.ship_date,
    SUM(s.quantity) OVER(cumulative
                         ROWS BETWEEN UNBOUNDED PRECEDING
                                  and CURRENT ROW
                         ) AS cumulative qty
FROM sales_order_items s JOIN product p On (s.prod_id =
  p.id)
WHERE s.ship_date BETWEEN '2005-07-01' and '2005-08-31'
Window cumulative as (PARTITION BY s.prod_id ORDER BY
  s.ship date)
ORDER BY p.id
```

Note how the window clause appears before the ORDER BY clause in the query specification. When using a window clause, the following restrictions apply:

- The in-line window specification cannot contain a PARTITION BY clause.
- The window specified within the window clause cannot contain a window frame clause. For example, from "Grammar rule 32" on page 111:

```
<WINDOW FRAME CLAUSE> ::=
    <WINDOW FRAME UNIT>
    <WINDOW FRAME EXTENT>
```

- Either the in-line window specification, or the window specification specified in the window clause, can contain a window order clause, but not both. For example, from "Grammar rule 31" on page 111:

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

Example: Window with multiple functions

It is possible to define a single (named) window and compute multiple function results over it, as the following example demonstrates.

```
SELECT p.id, p.description, s.quantity, s.ship_date,
   SUM(s.quantity) OVER ws1, MIN(s.quantity) OVER ws1
FROM sales_order_items s JOIN product p ON (s.prod_id =
   p.id) WHERE s.ship_date BETWEEN '1994-05-01' AND
   '1994-08-31' AND  s.quantity > 40 window ws1 AS
   (PARTITION BY prod_id ORDER BY ship_date rows
   between unbounded preceding and current row)
ORDER BY p.id;
```

The following are the results from the above query:

ID	description	quantity	ship_date	sum	min
302	Crew Neck	60	1994-07-02	60	60
400	Cotton Cap	60	1994-05-26	60	60
400	Cotton Cap	48	1994-07-05	108	48
401	Wool cap	48	1994-06-02	48	48
401	Wool cap	60	1994-06-30	108	48
401	Wool cap	48	1994-07-09	156	48
500	Cloth Visor	48	1994-06-21	48	48
501	Plastic Visor	60	1994-05-03	60	60
501	Plastic Visor	48	1994-05-18	108	48
501	Plastic Visor	48	1994-05-25	156	48
501	Plastic Visor	60	1994-07-07	216	48
601	Zipped Sweatshirt	60	1994-07-19	60	60
700	Cotton Shorts	72	1994-05-18	72	72
700	Cotton Shorts	48	1994-05-31	120	48

Example: Calculate cumulative sum

This query calculates a cumulative sum of salary per department and ORDER BY start_date.

```
SELECT dept_id, start_date, name, salary,
   SUM(salary) OVER (PARTITION BY dept_id ORDER BY
   start_date ROWS BETWEEN UNBOUNDED PRECEDING AND
   CURRENT ROW)
FROM emp1
ORDER BY dept_id, start_date;
```

The following are the results from the above query:

dept_id	start_date	name	salary	sum(salary)
100	1996-01-01	Anna	18000	18000
100	1997-01-01	Mike	28000	46000
100	1998-01-01	Scott	29000	75000

```
100      1998-02-01    Antonia   22000        97000
100      1998-03-12    Adam      25000       122000
100      1998-12-01    Amy       18000       140000
200      1998-01-01    Jeff      18000        18000
200      1998-01-20    Tim       29000        47000
200      1998-02-01    Jim       22000        69000
200      1999-01-10    Tom       28000        97000
300      1998-03-12    Sandy     55000        55000
300      1998-12-01    Lisa      38000        93000
300      1999-01-10    Peter     48000       141000
```

Example: Calculate moving average

This query generates the moving average of sales in three consecutive months. The size of the window frame is three rows: two preceding rows plus the current row. The window slides from the beginning to the end of the partition.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	avg(sales)
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00
20	1	20	20.00
20	2	30	25.00
20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00

Example: ORDER BY results

In this example, the top ORDER BY clause of a query is applied to the final results of a window function. The ORDER BY in a window clause is applied to the input data of a window function.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
    (PARTITION BY prod_id ORDER BY month_num ROWS
    BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	avg(sales)
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00
20	1	20	20.00
20	2	30	25.00
20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00

Example: Multiple aggregate functions in a query

This example calculates aggregate values against different windows in a query.

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
    (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
    CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
    PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
    prod_id
ORDER BY month_num)
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	CAvg	CSum
-------	---------	-----	----	----
10	1	100	110.00	100
10	2	120	106.66	220
10	3	100	116.66	320
10	4	130	116.66	450
10	5	120	120.00	570
10	6	110	115.00	680
20	1	20	25.00	20
20	2	30	25.00	50
20	3	25	28.33	75
20	4	30	28.66	105
20	5	31	27.00	136
20	6	20	25.50	156
30	1	10	10.50	10
30	2	11	11.00	21
30	3	12	8.00	33
30	4	1	6.50	34

Example: Window frame comparing ROWS and RANGE

This query compares ROWS and RANGE. The data contain duplicate ROWS per the ORDER BY clause.

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
    (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
       Range_sum, SUM(sales) OVER
    (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
       Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
    month_num)
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	Range_sum	Row_sum
-------	---------	-----	----------	-------
10	1	100	250	100
10	1	150	250	250
10	2	120	370	370
10	3	100	470	370
10	4	130	350	350
10	5	120	381	350
10	5	31	381	281
10	6	110	391	261
20	1	20	20	20

20	2	30	50	50
20	3	25	75	75
20	4	30	85	85
20	5	31	86	86
20	6	20	81	81
30	1	10	10	10
30	2	11	21	21
30	3	12	33	33
30	4	1	25	24
30	4	1	25	14

Example: Window frame excludes current row

In this example, you can define the window frame to exclude the current row. The query calculates the sum over four rows, excluding the current row.

```
SELECT prod_id, month_num, sales, sum(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN 6 PRECEDING AND 2 PRECEDING)
FROM sale
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	sum(sales)
10	1	100	(NULL)
10	1	150	(NULL)
10	2	120	(NULL)
10	3	100	250
10	4	130	370
10	5	120	470
10	5	31	470
10	6	110	600
20	1	20	(NULL)
20	2	30	(NULL)
20	3	25	20
20	4	30	50
20	5	31	75
20	6	20	105
30	1	10	(NULL)
30	2	11	(NULL)
30	3	12	10
30	4	1	21
30	4	1	21

Example: Default window frame for ROW

This query illustrates the default window frame for ROW.

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	sum(sales)
10	1	100	350
10	1	150	350
10	2	120	381
10	3	100	391
10	4	130	261
10	5	120	110
10	5	31	110
10	6	110	(NULL)
20	1	20	85
20	2	30	86
20	3	25	81
20	4	30	51
20	5	31	20
20	6	20	(NULL)
30	1	10	25
30	2	11	14
30	3	12	2
30	4	1	NULL)
30	4	1	(NULL)

Example: Unbounded preceding and unbounded following

In this example, the window frame can include all rows in the partition. The query calculates max(sales) sale over the entire partition (no duplicate rows in a month).

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	max(sales)
10	1	100	680
10	2	120	680
10	3	100	680
10	4	130	680
10	5	120	680
10	6	110	680
20	1	20	156
20	2	30	156
20	3	25	156
20	4	30	156
20	5	31	156
20	6	20	156
30	1	10	34
30	2	11	34
30	3	12	34
30	4	1	34

The query in this example is equivalent to:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

Example: Default window frame for RANGE

This query illustrates the default window frame for RANGE:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

The following are the results from the above query:

prod_id	month_num	sales	max(sales)
10	1	100	250
10	1	150	250
10	2	120	370
10	3	100	470
10	4	130	600
10	5	120	751
10	5	31	751

10	6	110	861
20	1	20	20
20	2	30	50
20	3	25	75
20	4	30	105
20	5	31	136
20	6	20	156
30	1	10	10
30	2	11	21
30	3	12	33
30	4	1	35
30	4	1	35

The query in this example is equivalent to:

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sale
ORDER BY prod_id, month_num;
```

BNF grammar for OLAP functions

The following Backus-Naur Form grammar outlines the specific syntactic support for the various ANSI SQL analytic functions, many of which are implemented in Sybase IQ.

Grammar rule 1

```
<SELECT LIST EXPRESSION> ::=
   <EXPRESSION>
   | <GROUP BY EXPRESSION>
   | <AGGREGATE FUNCTION>
   | <GROUPING FUNCTION>
   | <TABLE COLUMN>
   | <WINDOWED TABLE FUNCTION>
```

Grammar rule 2

```
<QUERY SPECIFICATION> ::=
   <FROM CLAUSE>
   [ <WHERE CLAUSE> ]
   [ <GROUP BY CLAUSE> ]
   [ <HAVING CLAUSE> ]
   [ <WINDOW CLAUSE> ]
[ <ORDER BY CLAUSE> ]
```

Grammar rule 3

```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

Grammar rule 4

<GROUPING FUNCTION> ::=
GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
<RIGHT PAREN>

Grammar rule 5

<WINDOWED TABLE FUNCTION> ::=
<WINDOWED TABLE FUNCTION TYPE> **OVER** <WINDOW NAME OR SPECIFICATION>

Grammar rule 6

<WINDOWED TABLE FUNCTION TYPE> ::=
<RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
| **ROW_NUMBER** <LEFT PAREN> <RIGHT PAREN>
| <WINDOW AGGREGATE FUNCTION>

Grammar rule 7

<RANK FUNCTION TYPE> ::=
RANK | **DENSE RANK** | **PERCENT RANK** | **CUME_DIST**

Grammar rule 8

<WINDOW AGGREGATE FUNCTION> ::=
<SIMPLE WINDOW AGGREGATE FUNCTION>
| <STATISTICAL AGGREGATE FUNCTION>

Grammar rule 9

<AGGREGATE FUNCTION> ::=
<DISTINCT AGGREGATE FUNCTION>
| <SIMPLE AGGREGATE FUNCTION>
| <STATISTICAL AGGREGATE FUNCTION>

Grammar rule 10

<DISTINCT AGGREGATE FUNCTION> ::=
<BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
<**DISTINCT**> <EXPRESSION> <RIGHT PAREN>
| **LIST** <LEFT PAREN> **DISTINCT** <EXPRESSION>
[<COMMA> <DELIMITER>]
[<ORDER SPECIFICATION>] <RIGHT PAREN>

Grammar rule 11

<BASIC AGGREGATE FUNCTION TYPE> ::=
SUM | **MAX** | **MIN** | **AVG** | **COUNT**

Grammar rule 12

<SIMPLE AGGREGATE FUNCTION> ::=
<SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
<EXPRESSION> <RIGHT PAREN>
| **LIST** <LEFT PAREN> <EXPRESSION> [<COMMA>
<DELIMITER>]
[<ORDER SPECIFICATION>] <RIGHT PAREN>

Grammar rule 13

<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW AGGREGATE FUNCTION TYPE>

Grammar rule 14

<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
<EXPRESSION> <RIGHT PAREN>
| GROUPING FUNCTION

Grammar rule 15

<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=

<BASIC AGGREGATE FUNCTION TYPE>
| **STDDEV** | **STDDEV_POP** | **STDDEV_SAMP**
| **VARIANCE** | **VARIANCE_POP** | **VARIANCE_SAMP**

Grammar rule 16

<STATISTICAL AGGREGATE FUNCTION> ::=
<STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>
<DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT
EXPRESSION> <RIGHT PAREN>

Grammar rule 17

<STATISTICAL AGGREGATE FUNCTION TYPE> ::=
CORR | **COVAR_POP** | **COVAR_SAMP** | **REGR_R2** |
REGR_INTERCEPT | **REGR_COUNT** | **REGR_SLOPE** |
REGR_SXX | **REGR_SXY** | **REGR_SYY** | **REGR_AVGY** |
REGR_AVGX

Grammar rule 18

<WINDOW NAME OR SPECIFICATION> ::=
<WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>

Grammar rule 19

<WINDOW NAME> ::= <IDENTIFIER>

Grammar rule 20

<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW
SPECIFICATION>

Grammar rule 21

<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>

Grammar rule 22

<WINDOW DEFINITION LIST> ::=
<WINDOW DEFINITION> [{ <COMMA> <WINDOW DEFINITION>
} . . .]

Grammar rule 23

<WINDOW DEFINITION> ::=
<NEW WINDOW NAME> **AS** <WINDOW SPECIFICATION>

Grammar rule 24

<NEW WINDOW NAME> ::= <WINDOW NAME>

Grammar rule 25

<WINDOW SPECIFICATION> ::=
<LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT
PAREN>

Grammar rule 26

<WINDOW SPECIFICATION DETAILS> ::=
[<EXISTING WINDOW NAME>]
[<WINDOW PARTITION CLAUSE>]
[<WINDOW ORDER CLAUSE>]
[<WINDOW FRAME CLAUSE>]

Grammar rule 27

<EXISTING WINDOW NAME> ::= <WINDOW NAME>

Grammar rule 28

<WINDOW PARTITION CLAUSE> ::=
PARTITION BY <WINDOW PARTITION EXPRESSION LIST>

Grammar rule 29

<WINDOW PARTITION EXPRESSION LIST> ::=
<WINDOW PARTITION EXPRESSION>
[{ <COMMA> <WINDOW PARTITION EXPRESSION> } . . .]

Grammar rule 30

<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>

Grammar rule 31

<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>

Grammar rule 32

<WINDOW FRAME CLAUSE> ::=
<WINDOW FRAME UNIT>
<WINDOW FRAME EXTENT>

Grammar rule 33

<WINDOW FRAME UNIT> ::= **ROWS** | **RANGE**

Grammar rule 34

<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME BETWEEN>

Grammar rule 35

<WINDOW FRAME START> ::=
UNBOUNDED PRECEDING
| <WINDOW FRAME PRECEDING>
| **CURRENT ROW**

Grammar rule 36

<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE SPECIFICATION> **PRECEDING**

Grammar rule 37

<WINDOW FRAME BETWEEN> ::=
BETWEEN <WINDOW FRAME BOUND 1> **AND** <WINDOW FRAME BOUND 2>

Grammar rule 38

<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>

Grammar rule 39

<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>

Grammar rule 40

<WINDOW FRAME BOUND> ::=
<WINDOW FRAME START>
| **UNBOUNDED FOLLOWING**
| <WINDOW FRAME FOLLOWING>

Grammar rule 41

<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE SPECIFICATION> **FOLLOWING**

Grammar rule 42

<GROUP BY EXPRESSION> ::= <EXPRESSION>

Grammar rule 43

<SIMPLE GROUP BY TERM> ::=
<GROUP BY EXPRESSION>
| <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>
| <LEFT PAREN> <RIGHT PAREN>

Grammar rule 44

<SIMPLE GROUP BY TERM LIST> ::=
<SIMPLE GROUP BY TERM> [{ <COMMA> <SIMPLE GROUP BY TERM> } . . .]

Grammar rule 45

<COMPOSITE GROUP BY TERM> ::=
<LEFT PAREN> <SIMPLE GROUP BY TERM>
[{ <COMMA> <SIMPLE GROUP BY TERM> } . . .]
<RIGHT PAREN>

Grammar rule 46

<ROLLUP TERM> ::= **ROLLUP** <COMPOSITE GROUP BY TERM>

Grammar rule 47

<CUBE TERM> ::= **CUBE** <COMPOSITE GROUP BY TERM>

Grammar rule 48

<GROUP BY TERM> ::=
<SIMPLE GROUP BY TERM>
| <COMPOSITE GROUP BY TERM>
| <ROLLUP TERM>
| <CUBE TERM>

Grammar rule 49

<GROUP BY TERM LIST> ::=
<GROUP BY TERM> [{ <COMMA> <GROUP BY TERM> } . . .]

Grammar rule 50

<GROUP BY CLAUSE> ::= **GROUP BY** <GROUPING SPECIFICATION>

Grammar rule 51

<GROUPING SPECIFICATION> ::=
<GROUP BY TERM LIST>
| <SIMPLE GROUP BY TERM LIST> **WITH ROLLUP**
| <SIMPLE GROUP BY TERM LIST> **WITH CUBE**
| <GROUPING SETS SPECIFICATION>

Grammar rule 52

<GROUPING SETS SPECIFICATION> ::=
GROUPING SETS <LEFT PAREN> <GROUP BY TERM LIST>
<RIGHT PAREN>

Grammar rule 53

<ORDER SPECIFICATION> ::= **ORDER BY** <SORT SPECIFICATION LIST>

CHAPTER 5 Managing System Resources

About this chapter

This chapter describes the way Sybase IQ uses memory, disk I/O, and CPUs, and the relationships among these factors. It also explains how the DBA can tune performance by adjusting resource usage.

The suggestions in this chapter are generic. You need to adjust them to suit your hardware and software configuration. Recommendations for each platform are in its *Sybase IQ Installation and Configuration Guide*.

Contents

Introduction to performance terms

Performance is the measure of efficiency of a computerized business application, or of multiple applications running in the same environment. It is usually measured in *response time* and *throughput*.

Response time is the time it takes for a single task to complete. It is affected by:

- Reducing contention and wait times, particularly disk I/O wait times
- Using faster components
- Reducing the amount of time the resources are needed (increasing concurrency)

Throughput refers to the volume of work completed in a fixed time period. Throughput is commonly measured in transactions per second (tps), but can be measured per minute, per hour, per day, and so on.

Designing for performance

Most gains in performance derive from good database design, thorough query analysis, and appropriate indexing. The largest performance gains can be realized by establishing a good design and by choosing the correct indexing strategy.

Other considerations, such as hardware and network analysis, can locate bottlenecks in your installation.

For more information, see Chapter 3, “Optimizing Queries and Deletions.”

Overview of memory use

Sybase IQ uses memory for several purposes:

- Buffers for data read from disk to resolve queries
- Buffers for data read from disk when loading from flat files
- Overhead for managing connections, transactions, buffers, and database objects

The sections that follow explain how the operating system supports Sybase IQ use of memory, how Sybase IQ allocates memory for various purposes, how you can adjust the memory allocations for better performance, and what you may need to do to configure the operating system so that enough memory is available for Sybase IQ.

Paging increases available memory

When there is not enough memory on your system, performance can degrade severely. If this is the case, you need to find a way to make more memory available. Like any RDBMS software, Sybase IQ requires a lot of memory. The more memory you can allocate to Sybase IQ, the better.

However, there is always a fixed limit to the amount of memory in a system, so sometimes operating systems can have only part of the data in memory and the rest on disk. When the operating system must go out to disk and retrieve any data before a memory request can be satisfied, it is called *paging* or *swapping*. The primary objective of good memory management is to avoid or minimize paging or swapping.

The most frequently used operating system files are *swap files*. When memory is exhausted, the operating system swaps pages of memory to disk to make room for new data. When the pages that were swapped are called again, other pages are swapped, and the required memory pages are brought back. This is very time-consuming for users with high disk usage rates. In general, try to organize memory to avoid swapping and, thus, to minimize use of operating system files. See “Platform-specific memory options” on page 130 for information on configuring memory to minimize swapping.

To make the maximum use of your physical memory, Sybase IQ uses buffer caches for *all* reads and writes to your databases.

Note Your swap space on disk must be at least large enough to accommodate all of your physical memory.

Utilities to monitor swapping

You can use the UNIX vmstat command, the UNIX sar command, or the Windows Task Manager, to get statistics on the number of running processes and the number of page-outs and swaps. Use this information to find out if the system is paging excessively. Then make any necessary adjustments. You may want to put your swap files on special fast disks.

For examples of vmstat output, see "Monitoring paging on UNIX systems."

Server memory

Sybase IQ allocates memory for various purposes from a single memory pool, called *server memory*. Server memory includes all of the memory allocated for managing buffers, transactions, databases, and servers.

At the operating system level, Sybase IQ server memory consists of heap memory. For the most part, you do not need to be concerned with whether memory used by Sybase IQ is heap memory or shared memory. All memory allocation is handled automatically. However, you may need to make sure that your operating system kernel is correctly configured to use shared memory before you run Sybase IQ. See the *Sybase IQ Installation and Configuration Guide* for your platform for details.

Managing memory for multiplexes

Each server in the multiplex can be on its own host or share a host with other servers. Two or more servers on the same system consume no more CPU time than would a single combined server handling the same workload, but separate servers might need more physical memory than a single combined server, because the memory used by each server is not shared by any other server.

Memory for loads, inserts, updates, synchronizations, and deletions

To avoid overallocating the physical memory on the machine, you can set the LOAD_MEMORY_MB database option for operations where loads occur. In addition to LOAD operations, this option affects INSERT, UPDATE, SYNCHRONIZE and DELETE operations. The LOAD_MEMORY_MB option sets an upper bound (in MB) on the amount of heap memory subsequent loads can use. For information on loads and buffer cache use, see "Memory requirements for loads" on page 119. For details of the LOAD_MEMORY_MB option, see Chapter 2, "Database Options," in the *Sybase IQ Reference Manual*.

Killing processes affects shared memory

Warning! Killing processes on UNIX systems may result in semaphores or shared memory being left behind instead of being cleaned up automatically. The correct way to shut down a Sybase IQ server on UNIX is the stop_asiq utility, described in "Stopping the database server" in Chapter 2, "Running Sybase IQ," *Sybase IQ System Administration Guide*. For information on using the ipcs and ipcrm to clean up after an abnormal exit, see Chapter 1, "Troubleshooting Hints," in *Sybase IQ Troubleshooting and Recovery Guide*.

Managing buffer caches

Sybase IQ needs more memory for buffer caches than for any other purpose. Sybase IQ has two buffer caches, one for the IQ Store and one for the Temporary Store. It uses these two buffer caches for all database I/O operations—for paging, for insertions into the database, and for backup and restore. Data is stored in one of the two caches whenever it is in memory. All user connections share these buffer caches. Sybase IQ keeps track of which data is associated with each connection.

Read the sections that follow for in-depth information on managing buffer caches:

- For information on how to calculate your memory requirements, see "Determining the sizes of the buffer caches."
- For information on how to set buffer cache sizes once you know what they should be, see "Setting buffer cache sizes."
- For an example of how to determine appropriate buffer cache sizes, see Table 5-1 on page 123.

Determining the sizes of the buffer caches

The buffer cache sizes you specify for the IQ Store and Temporary Store will vary based on several factors. The default values (16MB for the main and 12MB for the temporary cache) are too low for most databases. The actual values required for your application depend on:

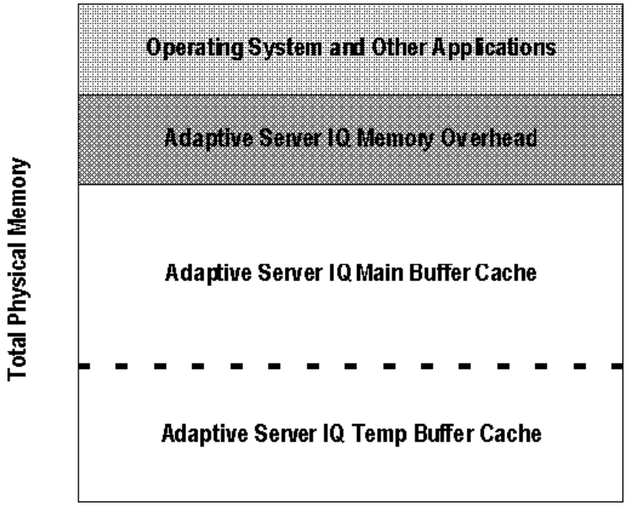
- The total amount of physical memory on your system
- How much of this memory Sybase IQ, the operating system, and other applications need to do their tasks

- Whether you are doing loads, queries, or both

Read the next several sections for guidelines in determining the best settings for your site, through the example in Table 5-1 on page 123.

The following diagram shows the relationship between the buffer caches and other memory consumption.

Figure 5-1: Buffer caches in relation to physical memory



The following sections describe each part in more detail and provide guidelines to help you determine how much memory each part requires.

Operating system and other applications

This amount of memory will vary for different platforms and how the system is used. For example, UNIX file systems do more file buffering than UNIX raw partitions, so the operating system has a higher memory requirement. As a minimum, you can assume that UNIX systems use 60MB or more, while Windows systems use 30MB or more.

In addition, other applications that run in conjunction with Sybase IQ (such as query tools) have their own memory needs. See your application and operating system documentation for information on their memory requirements.

Sybase IQ memory overhead

After determining how much physical memory the operating system and other applications use, you can calculate how much of the remaining memory Sybase IQ requires to do its tasks. The factors that affect this overhead are described in the following sections.

Raw partitions versus file systems

For UNIX systems, databases using file systems rather than raw partitions may require another 30% of the remaining memory to handle file buffering by the operating system. On Windows, file system caching should be disabled by setting OS_FILE_CACHE_BUFFERING = 'OFF' (the default for new databases). For more information, see the *Sybase IQ Installation and Configuration Guide* for your platform.

Multiuser database access

For multiuser queries of a database, Sybase IQ needs about 10MB per "active" user. Active users are defined as users who simultaneously access or query the database. For example, 30 users may be connected to Sybase IQ, but only 10 or so may be actively using a database at any one time.

Memory requirements for loads

Sybase IQ also requires a portion of memory separate from the buffer caches to perform load operations, synchronization, and deletions. This memory is used for buffering I/O for flat files. Sybase IQ uses memory to buffer a read from disk. The size of this read equals the BLOCK FACTOR multiplied by the size of the input record. BLOCK FACTOR is an option of the LOAD TABLE command. With the default value of 10,000, an input row of data of 200 bytes results in 2MB total that Sybase IQ uses for buffering I/O. Memory requirements for a load are determined by the number and width of columns, not the number of rows.

This memory is required only when loading from flat files, using INSERT..LOCATION, or INSERT..SELECT. A relatively small amount of memory is needed for deletions and updates.

Memory for thread stacks

Processing threads require a small amount of memory. The more Sybase IQ processing threads you use, the more memory needed. The -iqmt server switch controls the number of threads for Sybase IQ.

If you have a large number of users, the memory needed for Catalog Store processing threads also increases, although it is still relatively small. The -gn switch controls Catalog Store processing threads.

The total number of threads (-iqmt plus -gn) must not exceed the number allowed for your platform. For details, see Chapter 1, "Running the Database Server," in *Sybase IQ Utility Guide*.

Other memory use

All commands and transactions use some memory. The following operations are the most significant memory users in addition to those discussed previously:

Backup. The amount of virtual memory used for backup is a function of the IQ PAGE SIZE specified when the database was created. It is approximately 2 * number of CPUs * 20 * (IQ PAGE SIZE/16). On some platforms you may be able to improve backup performance by adjusting BLOCK FACTOR in the BACKUP command, but increasing BLOCK FACTOR also increases the amount of memory used. See "Increasing memory used during backup" in Chapter 14, "Data Backup, Recovery, and Archiving," in *Sybase IQ System Administration Guide*.

Database validation and repair. When you check an entire database, the sp_iqcheckdb procedure opens all Sybase IQ tables, their respective fields, and indexes before initiating any processing. Depending on the number of Sybase IQ tables and the cumulative number of columns and indexes in those tables, sp_iqcheckdb may require very little or a large amount of virtual memory. To limit the amount of memory needed, use the sp_iqcheckdb options to check or repair a single index or table.

Dropping leaked blocks. The drop leaks operation also needs to open all Sybase IQ tables, files, and indexes, so it uses as much virtual memory as sp_iqcheckdb uses when checking an entire database. It uses the Sybase IQ temp buffer cache to keep track of blocks used.

Sybase IQ main and temp buffer caches

After determining how much overhead memory Sybase IQ needs, you must decide how to split what is left between your main Sybase IQ and temp buffer caches. The dashed line dividing the two areas in Figure 5-1 indicates that this split may change from one database to another based on several factors.

Unlike most other databases, the general rule of thumb for Sybase IQ is a split of 40% for the main buffer cache and 60% for temp buffer cache. This rule of thumb, however, is only a start. While some operations, such as queries with large sort-merge joins or inserts involving HG indexes, may require a temp buffer cache larger than main, other applications might have different needs. Sybase IQ supports memory allocation ratios from 30/70 to 70/30, main to temp.

Note These guidelines assume you have one active database on your system at a time (that is, any Sybase IQ users are accessing only one database). If you have more than one active database, you need to further split the remaining memory among the databases you expect to use.

Sybase strongly recommends that you start with the general guidelines presented here and watch the performance of Sybase IQ by using its monitor tool (described in “Monitoring the buffer caches” on page 170) and any specific tools described in the *Sybase IQ Installation and Configuration Guide* for your platform.

Buffer caches and physical memory

The total memory used for Sybase IQ main and temporary buffer caches, plus Sybase IQ memory overhead, and memory used for the operating system and other applications, must not exceed the physical memory on your system.

If you set buffer cache sizes higher than your system will accommodate, Sybase IQ cannot open the database. Specify the server startup options -iqmc (main cache size) and -iqtc (temp cache size) to open the database and reset the cache sizes from their default values, which are only 16MB for the main cache and 12MB for the temporary cache.

Note On some UNIX platforms, you may need to set other server switches to make more memory available for buffer caches. See “Platform-specific memory options” on page 130 for more information.

Other considerations

Sybase IQ buffer cache sizes may differ from one database to the next based on how you use it. For maximum performance, you need to change the settings between inserting, querying the database, and mixed use. In a mixed-use environment, however, it is not always feasible to require all users to exit the database so that you can reset buffer cache options. In those cases, you may need to favor either load or query performance. When possible, define the cache sizes before doing any work in the database.

The buffer cache and memory overhead guidelines also may differ between platforms. See your *Sybase IQ Installation and Configuration Guide* for any other issues.

Example of setting buffer cache sizes

The following table lists factors that may consume memory for your system and shows an example of how much remains for your main and temp buffer caches. This example assumes that the system has 1GB of physical memory, no other significant applications on the hardware other than running Sybase IQ, and only one active database at a time. The table gives separate figures for the primary type of database access: queries or inserts.

Table 5-1: Example of memory (in MB) available for buffer caches

Memory use	Amount used	Memory available: Queries	Memory available: Inserts
Total amount of physical memory available (approximate in MB)		1000	1000
Operating system use assuming a minimum amount for a UNIX system	100[a]	900	900
Overhead for number of active users: approximately 30 connected users but only about 10 active at 10MB each	100	825	
Overhead for inserts from flat files assuming a 200-byte record size and default settings	97		828
Memory remaining for the main and temp buffer caches		675	828
iqmc (Main_Cache_Memory_MB) setting: 40% of memory remaining for buffer caches		405	497
iqtc (Temp_Cache_Memory_MB) setting: 60% of memory remaining for buffer caches		270	331

[a]Minimum operating system use for Windows is 30MB

As shown in the table, you should have one set of values for your buffer caches when primarily inserting into the database, another set when primarily querying the database, each differing from a typical mixed load of inserting and querying. To change the cache sizes, see "Setting buffer cache sizes." Remember that the cache size options do not take effect until you stop and restart the database.

Setting buffer cache sizes

By default, Sybase IQ sets the size of the main and temporary buffer caches to 16MB and 12MB respectively. Most applications will require much higher values (limited by the total amount of physical memory). See the preceding sections to determine the right settings for your system.

Once you know what settings you need, use the options described in Table 5-2 to set buffer cache sizes. You may also use the options described in Table 5-3 to make more memory available for buffer caches.

Table 5-2: Settings that change buffer cache sizes

Method	When to use it	How long the setting is effective	For more information, see
-iqmc and -iqtc server switches	Recommended method. Sets cache sizes when the database and server are not running. Allows cache sizes >4GB. Especially useful for 64-bit platforms, or if cache size database options are set larger than your system can accommodate.	From the time the server is started until it is stopped	"Setting buffer cache size server switches" on page 124
MAIN_CACHE_MEMORY_MB and TEMP_CACHE_MEMORY_MB database options	Set buffer cache sizes up to 4GB. Database must be open to set these values. This method is not recommended.	From the next time the database is restarted until you reset these options, or override them with server switches	"Database Options" chapter of *Sybase IQ Reference Manual*

Table 5-3: Settings that affect memory available for buffer caches

Method	When to use it	How long the setting is effective	For more information, see
-iqwmem server switch	Use on some UNIX platforms to provide additional memory for use as buffer caches. They do not actually set the cache sizes.	From the time the server is started until it is stopped	"Platform-specific memory options" on page 130
LOAD_MEMORY_MB database option	Indirectly affects buffer cache size, by controlling the memory that can be used for loads. On some platforms, allowing unlimited memory for loads means less memory is available for buffer caches.	Immediately until you reset the option	"Memory for loads, inserts, updates, synchronizations, and deletions" on page 116

Setting buffer cache size server switches

Setting the buffer cache sizes using the server startup options -iqmc and -iqtc has two important advantages:

- It allows you to set cache sizes greater than 4GB. These larger cache sizes are not allowed on 32-bit platforms, but are recommended on 64-bit platforms.
- It allows you to modify cache sizes if you have set the database options larger than your system can accommodate, so that you cannot open the database.

Whether you use the database options or the server options, you must restart the server to change buffer cache sizes. The -iqmc and -iqtc server startup options only remain in effect while the server is running, so you need to include them every time you restart the server.

Specifying page size

When you create a database, you set its page size. This parameter, in conjunction with the size of the buffer cache, determines memory use and disk I/O throughput for that database.

Note The page size cannot be changed and determines the upper size limit on some database objects.

Setting the page size

Sybase IQ swaps data in and out of memory in units of **pages**. When you create a database, you specify a separate page size for the Catalog Store and the IQ Store. The Temporary Store has the same page size as the IQ Store.

For Sybase IQ page size recommendations for the best performance, see “Choosing an IQ page size”in Chapter 5, “Working with Database Objects,”, *Sybase IQ System Administration Guide*.

Because the Catalog Store accounts for only a tiny fraction of I/O, the page size for the Catalog Store has no real impact on performance. The default value of 4096 bytes should be adequate.

The IQ page size determines two other performance factors, the default I/O transfer block size, and the maximum data compression for your database. These factors are discussed in the sections that follow.

Block size

All I/O occurs in units of **blocks**. The size of these blocks is set when you create a Sybase IQ database; you cannot change it without recreating the database. By default, the IQ page size determines the I/O transfer block size. For example, the default IQ page size of 128KB results in a default block size of 8192 bytes. In general, Sybase IQ uses this ratio of default block size to page size, but it considers other factors also.

The default block size should result in an optimal balance of I/O transfer rate and disk space usage for most systems. It does favor saving space over performance, however. If the default block size does not work well for you, you can set it to any power of two between 4096 and 32,768, subject to the constraints that there can be no fewer than two and no more than 16 blocks in a page. You may want to set the block size explicitly in certain cases:

- For a raw disk installation that uses a disk array, larger blocks may give better performance at the expense of disk space.
- For a file system installation, to optimize performance over disk space, the IQ block size should be greater than or equal to the operating system's native block size, if there is one. You may get better I/O rates if your IQ block size matches your file system's block size.

Table 5-4 shows the default block size for each IQ page size.

Table 5-4: Default block sizes

IQ page size (KB)	***Default block size (bytes)***
64	4096
128 (default for new databases)	8192
256	16384
512	32768

Data compression

Sybase IQ compresses all data when storing it on disk. Data compression both reduces disk space requirements and contributes to performance. The amount of compression is determined automatically, based on the IQ page size.

Saving memory

If your machine does not have enough memory, to save memory you can try the following adjustments.

Decrease buffer cache settings

You may be able to save memory by decreasing buffer cache sizes. Keep in mind that if you decrease the buffer caches too much, you could make your data loads or queries inefficient or incomplete due to insufficient buffers.

Decrease memory used for loads

You can set the LOAD_MEMORY_MB option to limit the amount of heap memory used for loads and other similar operations. See ““Memory for loads, inserts, updates, synchronizations, and deletions” on page 116.

Adjust blocking factor for loads

Use BLOCK FACTOR to reduce I/O when loading from a flat file. The BLOCK FACTOR option of the LOAD command specifies the blocking factor, or number of records per block, that were used when the input file was created. The default BLOCK FACTOR is 10,000.

The syntax for this load option is as follows:

BLOCK FACTOR = *integer*

Use the following guideline to determine BLOCK FACTOR:

```
record size * BLOCK FACTOR = memory required
```

You need extra memory for this option, in addition to the memory for the buffers. If you have a lot of memory available, or if no other users are active concurrently, increasing the value of BLOCK FACTOR can improve load performance.

Optimizing for large numbers of users

Sybase IQ handles up to 200 user connections on 32-bit platforms (Linux and Windows 2000.2003/XP), and up to 1000 user connections on 64-bit platforms (Sun Solaris, HP-UX and Itanium, and AIX). To support this greater number of users on 64-bit systems, you may need to adjust both operating system parameters and start_asiq server parameters. For recommendations, see the *Sybase IQ Installation and Configuration Guide* as well as the sections that follow.

Sybase IQ command line option changes for large numbers of users

The following start_asiq switches affect operations with large numbers of users:

-gm *#_connections_to_support*

-iqgovern *#_ ACTIVE_ queries_to_support*

-gn *#_Catalog_Store_front_end_threads*

-c *Catalog_Store_cache_size*

-ch *size*

-cl *size*

-gm

This is the total number of connections the server will support. Statistically, some of these are expected to be connected and idle while others are connected and actively using the database.

-iqgovern

Although 1000 users can be connected to Sybase IQ, for best throughput you should ensure that far fewer users are allowed to query at once, so that each of them has sufficient resources to be productive. The -iqgovern value places a ceiling on the maximum number of queries to execute at once. If more users than the -iqgovern limit have submitted queries, new queries will be queued until one of the active queries is finished.

The optimal value for -iqgovern depends on the nature of your queries, number of CPUs, and size of the Sybase IQ buffer cache. The default value is 2**numCPU* + 10. With a large number of connected users, you may find that setting this option to 2**numCPU* + 4 provides better throughput.

-gn

The correct value for -gn depends on the value of -gm. The start_asiq utility calculates -gn and sets it appropriately. Setting -gn too low can prevent the server from operating correctly. Setting -gn above 480 is not recommended.

-c

The Catalog Store buffer cache is also the general memory pool for the Catalog Store. To specify in MB, use the form -c nM, for example, -c 64M. Sybase recommends these values:

Table 5-5: Catalog buffer cache settings

For this many users	On these platforms	Set -c to this minimum value or higher
up to 1000	64-bit only	64MB
up to 200	64-bit	48MB (start_asiq default for 64-bit); larger numbers of users may benefit from 64MB
up to 200	32-bit	32MB (start_asiq default for 32-bit)

In some cases the standard Catalog cache size may be too small, for example, to accommodate certain queries that need a lot of parsing. In these cases, you may find it helpful to set -cl and -ch. For example, on 32-bit platforms, try these settings

```
-cl 128M
-ch 256M
```

Do not use -c in the same configuration file or command line with -ch or -cl. For related information, see the -ch cache-size option.

Warning! To control Catalog Store cache size explicitly, you must do *either* of the following, but not both, in your configuration file (*.cfg*) or on the UNIX command line for server startup:

- Set the -c parameter
- Set specific upper and lower limits for the Catalog Store cache size using the -ch and -cl parameters

Specifying different combinations of the parameters above can produce unexpected results.

-iqmt

You do not need to set the -iqmt option. If -iqmt is set too low for the number of specified connections, the number of threads will be increased to handle the number of requested connections. That is, -gm overrides -iqmt. However, if the number of Sybase IQ threads is elevated by means of the -iqmt option then that factor needs to be used in setting limits, as described in “Setting operating system parameters for large numbers of users.”

Increasing Sybase IQ temporary space for large numbers of users

You may need to increase your temporary dbspace to accommodate more users.

Relative priorities of new and existing connections

If Sybase IQ is very busy handling already connected users, it may be slow to respond to new connection requests. In extreme cases (such as test scripts that fire off hundreds of connections in a loop while the server is busy with inserts) new connections may have to wait up to a minute before becoming active or may even time out their connection request. In this situation, the server may appear to be down when it is merely very busy. A user getting this behavior should try to connect again.

Platform-specific memory options

On all platforms, Sybase IQ uses memory for four primary purposes:

- Main buffer cache
- Temporary buffer cache
- Sybase IQ memory overhead (including thread stacks)
- Load buffers

See Figure 5-1 on page 118 for a diagram of Sybase IQ memory use.

On all 64-bit platforms, the total amount of usable memory is effectively unlimited. The only limit is the system's virtual memory.

For performance tuning hints on HP-UX systems, see the *Sybase IQ Installation and Configuration Guide* for that platform.

On 32-bit platforms restrictions apply; see the following table for details.

Table 5-6: Total available memory on 32-bit platforms

Platform	***Total memory available***
RedHat Linux 2.1	About 1.7GB available to Sybase IQ
RedHat Linux 3.0	About 2.7GB available to Sybase IQ
Windows 2000/2003/XP[a]	2.75GB available to Sybase IQ

[a]You need Windows 2000 Advanced Server or Datacenter Server, Windows Server 2003 Standard, Enterprise or Datacenter Edition, or Windows XP Professional to get this much memory, and you must set the /3GB switch. Without the switch, the limit is 2GB. This amount is the total memory available to the process. Total size of buffer caches must not exceed 2GB on Windows servers, even with the /3GB setting. For details, see the *Sybase IQ Installation and Configuration Guide for Windows*.

Due to the virtual memory usage pattern within the Sybase IQ server, virtual memory fragmentation could cause excessive process growth on Windows platforms. To reduce the likelihood of this situation, Sybase IQ supports the use of Microsoft's low-fragmentation heap (LFH) on Windows XP and Windows Server 2003.

For more performance tuning hints on Windows platforms, see Chapter 7, "Tuning Servers on Windows Systems."

For UNIX systems only, Sybase IQ provides two command-line options that can help you manage memory.

Wired memory pool

On HP and Sun platforms, you can designate a specified amount of memory as "wired" memory. Wired memory is shared memory that is locked into physical memory. The kernel cannot page this memory out of physical memory.

Wired memory may improve Sybase IQ performance when other applications are running on the same machine at the same time. Dedicating wired memory to Sybase IQ, however, makes it unavailable to other applications on the machine.

To create a pool of "wired" memory on these UNIX platforms only, specify the -iqwmem command-line switch, indicating the number of MB of wired memory. (You must be user root to set -iqwmem, except on Sun.) On 64-bit platforms, the only upper limit on -iqwmem is the physical memory on the machine.

For example, on a machine with 14GB of memory, you may be able to set aside 10GB of wired memory. To do so, you specify:

```
-iqwmem 10000
```

Warning! Use this switch only if you have enough memory to dedicate the amount you specify for this purpose. Otherwise, you can cause serious performance degradation.

Note For this version:

- On Sun Solaris, -iqwmem always provides wired memory.
- On HP, -iqwmem provides wired memory if you start the server as root. It provides unwired memory if you are not root when you start the server. This behavior may change in a future version.

Impact of other applications and databases

Remember, the memory used for the server comes out of a pool of memory used by all applications and databases. If you try to run multiple servers or multiple databases on the same machine at the same time, or if you have other applications running, you may need to reduce the amount of memory your server requests.

The server log reports how much memory you actually get:

```
Created 1073741824 byte segement id 51205 Attached at
80000000

Created 184549376 byte segement id 6151 Attached at
C3576000
```

You can also issue the UNIX command `ipcs -mb` to see the actual number of segments.

Troubleshooting HP memory issues

If you have memory issues on HP-UX, check the value of the maxdsiz_64bit kernel parameter. This parameter restricts the amount of virtual memory available to Sybase IQ on 64-bit HP processors. See your *Sybase IQ Installation and Configuration Guide* for the recommended value.

Controlling file system buffering

On Solaris UFS and Windows file systems only, you can control whether file system buffering is turned on or off. Turning off file system buffering saves a data copy from the file system buffer cache to the main IQ buffer cache. Usually, doing so reduces paging, and therefore improves performance. Be aware of one exception: If the IQ page size for the database is less than the file system's block size (typically only in the case in testing situations) turning off file system buffering may *decrease* performance, especially during multiuser operation.

File system buffering is turned off by default for newly created Sybase IQ databases.

To disable file system buffering for existing databases, issue the following statement:

```
SET OPTION "PUBLIC".OS_FILE_CACHE_BUFFERING = OFF
```

You can only set this option for the PUBLIC group. You must shut down the database and restart it for the change to take effect.

Note Solaris does not have a kernel parameter to constrain the size of its file system buffer cache. Over time, the file system buffer cache grows and displaces the IQ buffer cache pages, leading to excess operating system paging activity and reduced Sybase IQ performance.

Windows can bias the paging algorithms to favor applications at the expense of the file system. This bias is recommended for Sybase IQ performance. See Chapter 7, “Tuning Servers on Windows Systems” for details.

Other ways to get more memory

In certain environments, you may be able to adjust other options to make more memory available to Sybase IQ.

Options for Java-enabled databases

The JAVA_HEAP_SIZE option of the SET OPTION command sets the maximum size (in bytes) of that part of the memory that is allocated to Java applications on a per connection basis. Per connection memory allocations typically consist of the user's working set of allocated Java variables and Java application stack space. While a Java application is executing on a connection, the per connection allocations come out of the fixed cache of the database server, so it is important that a run-away Java application is prevented from using up too much memory.

The JAVA_NAMESPACE_SIZE option of the SET OPTION command sets the maximum size (in bytes) of that part of the memory that is allocated to Java applications on a per database basis. Per database memory allocations include Java class definitions. As class definitions are effectively read-only, they are shared among connections. Consequently, their allocations come right out of the fixed cache, and this option sets a limit on the size of these allocations.

The process threading model

Sybase IQ uses operating system kernel threads for best performance. Threads can be found at the user level and at the kernel level. Lightweight processes are underlying threads of control that are supported by the kernel. The operating system decides which lightweight processes (LWPs) should run on which processor and when. It has no knowledge about what the user threads are, but does know if they are waiting or able to run.

The operating system kernel schedules LWPs onto CPU resources. It uses their scheduling classes and priorities. Each LWP is independently dispatched by the kernel, performs independent system calls, incurs independent page faults, and runs in parallel on a multiprocessor system.

A single, highly threaded process serves all Sybase IQ users. Sybase IQ assigns varying numbers of kernel threads to each user connection, based on the type of processing being done by that connection, the total number of threads available, and the various option settings.

Insufficient threads error

When you do not have enough server threads to initiate the query you have issued, you get the error:

```
Not enough server threads available for this query
```

This condition may well be temporary. When some other query finishes, threads are made available and the query may succeed the next time you issue it. If the condition persists, you may need to restart the server and specify more Sybase IQ threads, as described in the next section.

Sybase IQ options for managing thread usage

Sybase IQ offers the following options to help you manage thread usage.

- To set the maximum number of threads available for Sybase IQ use, set the server startup option -iqmt. The default value is calculated from the number of connections and the number of CPUs and is usually adequate.
- To set the stack size of the internal execution threads in the server, set the server startup option -iqtss. The default value is generally sufficient, but may be increased if complex queries return an error indicating that the depth of the stack exceeded this limit. For details about -iqmt and -iqtss, see Chapter 1, “Running the Database Server,” *Sybase IQ Utility Guide*.
- To set the maximum number of threads a single user will use, issue the command SET OPTION MAX_IQ_THREADS_PER_CONNECTION. This can be used to control the amount of resources a particular operation consumes. For example, the DBA can set this option before issuing an INSERT or LOAD command.

Balancing I/O

This section explains the importance of balancing I/O on your system. It explains how to use disk striping and how to locate files on separate disks to gain better performance. Controlling the size of the message log file is also discussed.

Raw I/O (on UNIX operating systems)

Most UNIX file systems divide disks into fixed size partitions. *Partitions* are physical subsets of the disk that are accessed separately by the operating system. Disk partitions are typically accessed in two modes: file system mode (through the UFS file system) or raw mode. Raw mode (sometimes called character mode) does unbuffered I/O, generally making a data transfer to or from the device with every read or write system call. The UFS mode is a UNIX file system and a buffered I/O system which collects data in a buffer until it can transfer an entire buffer at a time.

When you create a database or a dbspace, you can place it on either a raw device or a file system file. Sybase IQ determines automatically from the pathname you specify whether it is a raw partition or a file system file. Raw partitions can be any size.

For more information, see the section “Working with database objects” in Chapter 5, “Working with Database Objects” of the *Sybase IQ System Administration Guide*.

Using disk striping

Traditional file management systems allow you to locate individual files on specific disks. Consequently, all file operations occur against a single disk drive. Some operating systems allow you to create logical devices or volumes that span multiple disk drives. Once a file fills the first disk drive, it is automatically continued onto the next drive in the logical volume. This feature increases the maximum file size and concentrates activity on a single disk until it is full.

However, there is another way. *Disk striping* is a generic method of spreading data from a single file across multiple disk drives. This method allows successive disk blocks to be located on striped disk drives. Striping combines one or more physical disks (or disk partitions) into a single logical disk. Striped disks split I/O transfers across the component physical devices, performing them in parallel. They achieve significant performance gains over single disks.

Disk striping lets you locate blocks on different disks. The first block is located on the first drive. The second block is located on the second drive, and so on. When all the drives have been used, the process cycles back and uses additional blocks on the drives. The net effect of disk striping is the random distribution of data across multiple disk drives. Random operations against files stored on striped disks tend to keep all of the drives in the striped set equally busy, thereby maximizing the total number of disk operations per second. This is a very effective technique in a database environment.

You can use disk striping either as provided by your operating system and hardware, or Sybase IQ internal disk striping.

Setting up disk striping on UNIX

UNIX systems offering striped disks provide utilities for configuring physical disks into striped devices. See your UNIX system documentation for details.

Setting up disk striping on Windows

On Windows systems, use hardware disk striping via an appropriate SCSI-2 disk controller. If your machine does not support hardware striping, but you have multiple disks available for your databases, you can use Windows striping to spread disk I/O across multiple disks. Set up Windows striping using the Disk Management.

Recommendations for disk striping

Here are some general rules on disk striping:

- For maximum performance, the individual disks in a striped file system should be spread out across several disk controllers. But be careful not to saturate a disk controller with too many disks. Typically, most SCSI machines can handle 2–3 disks per controller. See your hardware documentation for more information.
- Do not put disks on the same controller as slower devices, such as tape drives or CD-ROMs. This slows down the disk controller.
- Allocate 4 disks per server CPU in the stripe.
- The individual disks must be identical devices. This means they must be the same size, have the same format, and often be the same brand. If the layouts differ, the size of the smallest one is often used and other disk space is wasted. Also, the speed of the slowest disk is often used.

- In general, disks used for file striping should not be used for any other purpose. For example, do not use a file striped disk as a swap partition.
- Never use the disk containing the root file system as part of a striped device.

In general, you should use disk striping whenever possible.

Note For the best results when loading data, dump the data to a flat file located on a striped disk and then read the data into Sybase IQ with the LOAD TABLE command.

Internal striping

Sybase IQ stores its information in a series of dbspaces—files or raw partitions of a device—in blocks. Assuming that disk striping is in use, Sybase IQ spreads data across all dbspaces that have space available. This approach lets you take advantage of multiple disk spindles at once, and provides the speed of parallel disk writes.

Disk striping option

This section explains how you can use the option Sybase IQ provides to do disk striping, without using third party software. If you already have a disk striping solution through third party software and hardware, you should use that method instead.

Turning disk striping on or off

The syntax you use to turn disk striping on or off is:

SET OPTION "PUBLIC".DISK_STRIPING = { ON | OFF }

The default for the DISK_STRIPING option is ON for all platforms. When disk striping is ON, incoming data is spread across all dbspaces with space available. When disk striping is OFF, dbspaces (disk segments) are filled up from the front on the logical file, filling one disk segment at a time.

You must restart the Sybase IQ server in order for a change to the value of the DISK_STRIPING option to take effect.

The database options MAIN_DISK_KB_PER_STRIPE and TEMP_DISK_KB_PER_STRIPE define the number of kilobytes (KB) to write to each dbspace before the disk striping algorithm moves to the next stripe for the IQ Main Store and the IQ Temporary Store, respectively. The default value for these options is 1, which rounds up to one page.

You can drop a dbspace using the DROP DBSPACE command when disk striping is on. Before dropping the dbspace, however, you must relocate all of the data in the dbspace using the sp_iqrelocate stored procedure. Because disk striping spreads data across multiple dbspaces, the relocation process may require the relocation of many tables and indexes. Use the sp_iqdbspaceinfo and sp_iqdbspace stored procedures to determine which tables and indexes reside on a dbspace.

Using multiple dbspaces

Using multiple dbspaces allows your Sybase IQ and temporary data to be broken down into multiple operating system files or partitions. These files can then be spread across multiple disks.

Like disk striping, randomness can be created by placing successive database files across multiple drives. You can create additional segments for your Sybase IQ and temporary data with the CREATE DBSPACE command.

When to create dbspaces

When possible, allocate all dbspaces when you create a database.

If you add dbspaces later, Sybase IQ stripes new data across both old and new dbspaces. Striping may even out, or it may remain unbalanced, depending on the type of updates you have. The number of pages that are “turned over” due to versioning has a major impact on whether striping is rebalanced.

Strategic file locations

Performance related to randomly accessed files can be improved by increasing the number of disk drives devoted to those files, and therefore, the number of operations per second performed against those files. Random files include those for the IQ Store, the Temporary Store, the Catalog Store, programs (including the Sybase IQ executables, user and stored procedures, and applications), and operating system files.

Conversely, performance related to sequentially accessed files can be improved by locating these files on dedicated disk drives, thereby eliminating contention from other processes. Sequential files include the transaction log and message log files.

To avoid disk bottlenecks, follow these suggestions:

- Keep random disk I/O away from sequential disk I/O.
- Isolate Sybase IQ database I/O from I/O for proxy tables in other databases, such as Adaptive Server Enterprise.
- Place the transaction log and message log on separate disks from the IQ Store, Catalog Store, and Temporary Store, and from any proxy databases such Adaptive Server Enterprise.
- Place the database file, temporary dbspace, and transaction log file on the same physical machine as the database server.

The transaction log file

The transaction log file contains information that allows Sybase IQ to recover from a system failure. The default filename extension for this file is *.log*.

To move or rename the transaction log file, use the Transaction Log utility (dblog). For syntax and details, see Chapter 3, "Database Administration Utilities,"*Sybase IQ Utility Guide*.

Warning! The Sybase IQ transaction log file is different from most relational database transaction log files. If for some reason you lose your database files, then you lose your database (unless it is the log file that is lost). However, if you have an appropriate backup, then you can reload the database.

Truncating the transaction log

Sybase IQ records in the transaction log the information necessary to recover from a system failure. Although the information logged is small for each committed transaction, the transaction log continues to grow in size. In systems with a high number of transactions that change data, over a period of time the log can grow to be very large.

Log truncation generally requires the Sybase IQ servers involved to be taken off line. When to truncate the log is really up to the DBA responsible for supporting the Sybase IQ systems, and depends on the growth profile of the log file and the operational procedures at the site. The log truncation procedure should be scheduled at least once a month or more frequently if the log file is exceeding 100MB.

Table 5-7shows methods for truncating transaction logs in Sybase IQ.

Table 5-7: Truncating transaction logs

If your database is …	Use this method …	For details, see …
Non-multiplex	The –m switch, which causes the transaction log to be truncated after each checkpoint for all databases	"Truncating the transaction log of a non-multiplex database"
Multiplex	The DELETE_OLD_LOGS database option	"Truncating the transaction log of a multiplex database"
Running	The dbbackup command line utility	Backup utility (dbbackup) in *Sybase IQ Utility Guide*.

Be sure to use the appropriate method. Sybase IQ database replication inherently relies on transaction log information. For this reason, only the DELETE_OLD_LOGS option should be used for a multiplex database (see "Truncating the transaction log for a multiplex database."). Also, the transaction log provides Sybase support with valuable information for problem diagnosis and reproduction. Both methods should include archiving the existing log (keeping a copy of the log), in case Sybase support needs the log for further diagnostic work.

Truncating the transaction log for a non-multiplex database

Use the –m server startup switch to truncate the transaction log of a non-multiplex database. Note that leaving the –m server startup switch permanently set is *not* recommended. This switch should only be used to start Sybase IQ for a transaction log truncation. How this is done is up to the DBA, but the following procedure provides a suggestion.

❖ Truncating the transaction log of a non-multiplex database

1. Create a copy of the server switches *.cfg* file with a name identifying the file as the log truncation configuration setting and edit this copy of the file to add the –m switch.
2. Perform normal full backup procedures, including making copies of the *.db* and *.log* files.
3. Shut down Sybase IQ. Verify that 'CloseDatabase' was written in the *iq.msg* file.
4. Restart Sybase IQ with the configuration file containing the –m option. Note that no user access or transactions should be allowed at this time.
5. Shut down Sybase IQ and restart using the configuration file without the –m option set.

Truncating the transaction log for a multiplex database

❖ Truncating the transaction log of a multiplex database

1. Back up the database from the write server, if you have not already done so.
2. Set the DELETE_OLD_LOGS option on the write server:

   ```
   SET OPTION Public.Delete_Old_Logs='On'
   ```

3. Stop the write server's dbremote and restart it with the -x command line switch. (Create a special version of the *start_dbremote.bat* script, found in the write server's database directory, to do this.) This truncates the log at the write server. For example:

   ```
   cd \Server01\mpxdb\cmd /c
   start dbremote -q -v -x -o
   "d:\Server01\mpxdb\dbremote.log" -c
   "uid=DBA;pwd=SQL;eng=Server01;dbf=
   d:\Server01\mpxdb\mpxdb;
   links=tcpip{port=1704;host=FIONA-PC}"
   ```

4. Clear the DELETE_OLD_LOGS option on the write server:

   ```
   SET OPTION Public.Delete_Old_Logs='Off'
   ```

Note The query server transaction log is always truncated during synchronization, no matter when the write server log was last truncated.

The message log

A message log file exists for each database. The default name of this file is *dbname.iqmsg*, although you can specify a different name when you create the database. The message log file is actually created when the first user connects to a database.

By default, Sybase IQ logs all messages in the message log file, including error, status, and insert notification messages. You can turn off notification messages in the LOAD and INSERT statements.

At some sites the message log file tends to grow rapidly, due to the number of insertions, LOAD option and NOTIFY_MODULUS database option settings, or certain other conditions. Sybase IQ lets you limit the size of this file by wrapping the message log.

When you enable message log wrapping, as soon as the file reaches the maximum size specified in the IQMSG_LENGTH_MB database option, new messages are written starting at the beginning of the file. Existing messages are overwritten, line-by-line.

When wrapping is enabled, the tag `<next msg insertion place>` tells you where new messages are being placed. Additional tags at the beginning and end of the file remind you that log wrapping is enabled, and that the last message in the file may not be the most recent one.

To enable message log wrapping and set the maximum log file size, see “IQMSG_LENGTH_MB option” in *Sybase IQ Reference Manual*.

Working space for inserting, deleting, and synchronizing

When you insert or delete data, and when you synchronize join indexes, Sybase IQ needs some working space in the IQ Store. This space is reclaimed for other purposes when the transaction that needs it commits.

Ordinarily, as long as you maintain a reasonable percentage of free space in your IQ Store, you will have enough free space. However, for certain deletions, depending on the size of the data and its distribution among database pages, you may need a large amount of working space. In the case where you are deleting a major portion of your database, and the data is distributed sparsely across many pages, you could temporarily double the size of your database.

Setting reserved space options

Two database options, MAIN_RESERVED_DBSPACE_MB and TEMP_RESERVED_DBSPACE_MB, control the amount of space Sybase IQ reserves for certain operations. For more information see “Reserving space to handle out-of-space conditions”in *Sybase IQ System Administration Guide*.

Options for tuning resource use

The number of concurrent users of a Sybase IQ database, the queries they run, and the processing threads and memory available to them, can have a dramatic impact on performance, memory use, and disk I/O. Sybase IQ provides several options for adjusting resource use to accommodate varying numbers of users and types of queries. These may be:

- SET OPTION command options that affect only the current database.
- Command-line options that affect an entire database server.
- Connection parameters that affect the current connection only.

For more information on all of these options, including parameters, when the options take effect, and whether you can set them for both a single connection and the PUBLIC group, see the *Sybase IQ Reference Manual*.

For information specific to optimizing tables, see "Optimizing storage and query performance," *Sybase IQ System Administration Guide*

Restricting concurrent queries

The -iqgovern command-line option lets you control the number of concurrent queries on a server. This is not the same as the number of connections, which is controlled by your license.

The -iqgovern switch optimizes paging of buffer data out to disk, so that memory is used most effectively. The default value of -iqgovern is (2 x the number of CPUs) + 4.

Setting the number of CPUS available

The -iqnumbercpus switch on the Sybase IQ startup command lets you specify the number of CPUs available. This switch is recommended only:

- On machines with Intel® CPUs and hyperthreading enabled
- On machines where an operating system utility has been used to restrict Sybase IQ to a subset of the CPUs within the machine

For details, see "Setting the number of CPUs" in the *Sybase IQ System Administration Guide*.

Limiting a query's temporary dbspace use

The QUERY_TEMP_SPACE_LIMIT option of the SET command lets you restrict the amount of temporary dbspace available to any one query. By default, a query can use 2000MB of temporary dbspace.

When you issue a query, Sybase IQ estimates the temporary space needed to resolve the query. If the total estimated temporary result space for sorts, hashes, and row stores exceeds the current QUERY_TEMP_SPACE_LIMIT setting, the query is rejected, and you receive a message such as:

```
Query rejected because it exceeds total space resource
limit
```

If this option is set to 0, there is no limit, and no queries are rejected based on their temporary space requirements.

Limiting queries by rows returned

The QUERY_ROWS_RETURNED_LIMIT option of the SET command tells the query optimizer to reject queries that might otherwise consume too many resources. If the query optimizer estimates that the result set from a query will exceed the value of this option, it rejects the query with the message:

```
Query rejected because it exceed resource:
Query_Rows_Returned_Limit
```

If you use this option, set it so that it only rejects queries that consume vast resources.

Forcing cursors to be non-scrolling

When you use scrolling cursors with no host variable declared, Sybase IQ creates a temporary store node where query results are buffered. This storage is separate from the Temporary Store buffer cache. If you are retrieving very large numbers (millions) of rows, this store node can require a lot of memory.

You can eliminate this temporary store node by forcing all cursors to be non-scrolling. To do so, set the FORCE_NO_SCROLL_CURSORS option to ON. You may want to use this option to save on temporary storage requirements if you are retrieving very large numbers (millions) of rows. The option takes effect immediately for all new queries submitted.

If scrolling cursors are never used in your application, you should make this a permanent PUBLIC option. It will use less memory and make a big improvement in query performance.

Limiting the number of cursors

The MAX_CURSOR_COUNT option specifies a resource governor to limit the maximum number of cursors that a connection can use at once. The default is 50. Setting this option to 0 allows an unlimited number of cursors.

Limiting the number of statements

The MAX_STATEMENT_COUNT option specifies a resource governor to limit the maximum number of prepared statements that a connection can use at once.

Prefetching cache pages

The SET option PREFETCH_BUFFER_LIMIT defines the number of cache pages available to Sybase IQ for use in prefetching (the read ahead of database pages). This option has a default value of 0. Set this option only if advised to do so by Sybase Technical Support. For more information, see "PREFETCH_BUFFER_LIMIT option" in the *Sybase IQ Reference Manual*.

The SET option BT_PREFETCH_MAX_MISS determines whether to continue prefetching pages for a given query. If queries using HG indexes run more slowly than expected, try gradually increasing the value of this option. For more information, see "BT_PREFETCH_MAX_MISS option" in the *Sybase IQ Reference Manual*.

Optimizing for typical usage

Sybase IQ tracks the number of open cursors and allocates memory accordingly. In certain circumstances, USER_RESOURCE_RESERVATION option can be set to adjust the minimum number of current cursors that thinks is currently using the product and hence allocate memory from the temporary cache more sparingly.

This option should only be set after careful analysis shows it is actually required. Contact Sybase Technical Support with details if you need to set this option.

Controlling the number of prefetched rows

Prefetching is used to improve performance on cursors that only fetch relative 1 or relative 0. Two connection parameters let you change cursor prefetch defaults. PrefetchRows (PROWS) sets the number of rows prefetched; PrefetchBuffer (PBUF) sets the memory available to this connection for storing prefetched rows. Increasing the number of rows you prefetch may improve performance under certain conditions:

- The application fetches many rows (several hundred or more) with very few absolute fetches.
- The application fetches rows at a high rate, and the client and server are on the same machine or connected by a fast network.
- Client/server communication is over a slow network, such as a dial-up link or wide area network.

Other ways to improve resource use

This section describes several ways to adjust your system for maximum performance or better use of disk space.

Managing disk space in multiplex databases

Sybase IQ cannot drop old versions of tables while any user on any server might be in a transaction that might need the old versions. Sybase IQ may therefore consume a very large amount of disk space when table updates and queries occur simultaneously in a multiplex database. The amount of space consumed depends on the nature of the data and indexes and the update rate.

You can free disk blocks by allowing the write server to drop obsolete versions no longer required by queries. All users on all servers should commit their current transactions periodically to allow recovery of old table versions. The servers may stay up and are fully available. The dbremote processes must all continue to run to forward the latest information about the use of table versions on each query server to the write server.

Load balancing among query servers

You may be able to use the IQ Network Client to balance the query load among multiplex query servers. This method requires an intermediate system that is able to dispatch the client connection to a machine in a pool, depending on the workload of the machine.

To use this method, on the client system you create a special ODBC DSN, with the IP address and port number of this intermediate load balancing system, a generic server name, and the VerifyServerName connection parameter set to NO. When a client connects using this DSN, the load balancer establishes the connection to the machine it determines is least loaded.

For details on how to define an ODBC DSN for use in query server load balancing, see “VerifyServerName parameter [Verify]” in Chapter 4, “Connection and Communication Parameters” of the *Sybase IQ System Administration Guide*.

Restricting database access

For better query performance, set the database to read-only, if possible, or schedule significant updates for low usage hours. Sybase IQ allows multiple query users to read from a table while you are inserting or deleting from that table. However, performance can degrade during concurrent updates to the database.

Disk caching

Disk cache is memory used by the operating system to store copies of disk blocks temporarily. All file system based disk reads and writes usually pass through a disk cache. From an application's standpoint, all reads and writes involving disk caches are equivalent to actual disk operations.

Operating systems use two different methods to allocate memory to disk cache: fixed and dynamic. A preset amount of memory is used in a fixed allocation; usually a 10–15 percent memory allocation is set aside. The operating system usually manages this workspace using a LRU (least recently used) algorithm. For a dynamic allocation, the operating system determines the disk cache allocation as it is running. The goal is to keep as much memory in active use as possible, balancing the demand for real memory against the need for data from disk.

Indexing tips

The following sections give some tips for selecting and managing indexes. See Chapter 6, "Using Sybase IQ Indexes," in the *Sybase IQ System Administration Guide* for more information on these topics.

Choosing the right index type

It is important to choose the correct index type for your column data. Sybase IQ provides some indexes automatically—a default index on all columns that optimizes projections, and an HG index for UNIQUE and PRIMARY KEYS and FOREIGN KEYS. While these indexes are useful for some purposes, you need other indexes to process certain queries as quickly as possible. Sybase IQ chooses the best index type for you when there are multiple index types for a column.

The Sybase IQ query optimizer features an index advisor that generates messages when the optimizer would benefit from an additional index on one or more columns in your query. To activate the index advisor, set the INDEX_ADVISOR option ON. Messages print as part of a query plan or as a separate message in the message log (*.iqmsg*) if query plans are not enabled, and output is in OWNER.TABLE.COLUMN format. For details, see INDEX_ADVISOR option in "Database Options," *Sybase IQ Reference Manual*.

You should create either an LF or HG index in addition to the default index on LF or HG on grouping columns referenced by the WHERE clause in a join query. Sybase IQ cannot guarantee that its query optimizer will produce the best execution plan if some columns referenced in the WHERE clause lack either an LF or HG index. Non-aggregated columns referenced in the HAVING clause should also have the LF or HG index in addition to the default index. For example:

```
SELECT c.name, SUM(l.price * (1 - l.discount))
FROM customer c, orders o, lineitem l
WHERE c.custkey = o.custkey
    AND o.orderkey = l.orderkey
    AND o.orderdate >= "1994-01-01"
    AND o.orderdate < "1995-01-01"
GROUP by c.name
HAVING c.name NOT LIKE "I%"
    AND SUM(l.price * (1 - l.discount)) > 0.50
ORDER BY 2 desc
```

In addition to the default index, all columns in this example beside l.price and l.discount should have an LF or HG index.

Using join indexes

Users frequently need to see the data from more than one table at once. This data can be joined at query time, or in advance by creating a join index. Sometimes you can improve query performance by creating a join index for columns that are joined in a consistent way.

Because join indexes require substantial time and space to load, you should create them only for joins needed on a regular basis. Sybase IQ join indexes support one-to-many and one-to-one join relationships.

Allowing enough disk space for deletions

When you delete data rows, Sybase IQ creates a version page for each database page that contains any of the data being deleted. The versions are retained until the delete transaction commits. For this reason, you may need to add disk space when you delete data. See “Overlapping versions and deletions” on page 394 for details.

Managing database size and structure

This section offers ideas on improving your database design and managing your data.

Managing the size of your database

The size of your database depends largely on the indexes you create, and the quantity of data you maintain. You achieve faster query processing by creating all of the indexes you need for the types of queries your users issue. However, if you find that some tables or indexes are not needed, you can drop them. By doing so, you free up disk space, increase the speed of loads and backups, and reduce the amount of archive storage you need for backups.

To control the quantity of data stored in a given table, consider how best to eliminate data rows you no longer need. If your database contains data that originated in an Adaptive Server Anywhere database, you may be able to eradicate unneeded data by simply replaying Anywhere deletions; command syntax is compatible. You can do the same with data from an Adaptive Server Enterprise database, because Sybase IQ provides Transact-SQL compatibility.

Controlling index fragmentation

Internal index fragmentation occurs when index pages are not being used to their maximum volume.

Row fragmentation can occur when rows are deleted. If you delete an entire page of rows, that page is freed, but if some rows on a page are unused, unused space remains on the disk.

DML operations (INSERT, UPDATE, DELETE) that act on tables cause index fragmentation. Two stored procedures report fragmentation:

- sp_iqrowdensity reports row fragmentation at the default index level. See “sp_iqrowdensity procedure.”
- sp_iqindexfragmentation reports internal fragmentation within supplemental indexes. See “sp_iqindexfragmentation procedure.”

The database administrator may create other indexes to supplement the default index on a column. These indexes can use more space than needed when rows are deleted from a table.

Neither procedure recommends further action. The database administrator must examine the information reported and determine whether to take further action, such as recreating, reorganizing, or rebuilding indexes.

Minimizing catalog file growth

Growth of the catalog files is normal and varies depending on the application and catalog content. The size of the *.DB* file does not affect performance, and free pages within the *.DB* file are reused as needed. To minimize catalog file growth:

- Avoid using IN SYSTEM on CREATE TABLE statements.
- Issue COMMIT statements after running system stored procedures.
- Issue COMMIT statements during long-running transactions.

Denormalizing for performance

Once you have created your database in normalized form, you may perform benchmarks and decide to intentionally back away from normalization to improve performance. Denormalizing:

- Can be done with tables or columns
- Assumes prior normalization
- Requires a knowledge of how the data is being used

Good reasons to denormalize are:

- All queries require access to the “full” set of joined data
- Computational complexity of derived columns require storage for selects

Denormalization has risks

Denormalization can be successfully performed only with thorough knowledge of the application and should be performed only if performance issues indicate that it is needed. One of the things to consider when you denormalize is the amount of effort it will then take to keep your data up-to-date with changes.

This is a good example of the differences between decision support applications, which frequently need summaries of large amounts of data, and transaction processing needs, which perform discrete data modifications. Denormalization usually favors some processing, at a cost to others.

Whatever form of denormalization you choose, it has the potential for data integrity problems which must be carefully documented and addressed in application design.

Disadvantages of denormalization

Denormalization has these disadvantages:

- Denormalization usually speeds retrieval but can slow updates. This is not a real concern in a DSS environment.
- Denormalization is always application-specific and needs to be re-evaluated if the application changes.
- Denormalization can increase the size of tables. This is not a problem in Sybase IQ, because you can optimize the storage of column data. For details, see the IQ UNIQUE column constraint in CREATE TABLE statement and "MINIMIZE_STORAGE option" in *Sybase IQ Reference Manual*.
- In some instances, denormalization simplifies coding; in others, it makes it more complex.

Performance benefits of denormalization

Denormalization can improve performance by:

- Minimizing the need for joins
- Precomputing aggregate values, that is, computing them at data modification time, rather than at select time
- Reducing the number of tables, in some cases

Deciding to denormalize

When deciding whether to denormalize, you need to analyze the data access requirements of the applications in your environment and their actual performance characteristics. Some of the issues to examine when considering denormalization include:

- What are the critical queries, and what is the expected response time?
- What tables or columns do they use? How many rows per access?
- What is the usual sort order?
- What are concurrency expectations?
- How big are the most frequently accessed tables?
- Do any processes compute summaries?
- Should you create join indexes to gain performance?

Using UNION ALL views for faster loads

To minimize load times for very large tables, you can use a UNION ALL view. Sybase IQ lets you partition tables by splitting the data into several separate base tables (for example, by date). You load data into these smaller tables. You then join the tables back together into a logical whole by means of a UNION ALL view, which you can then query.

UNION ALL views are simple to administer. If the data is partitioned by month, for example, you can drop an entire month's worth of data by deleting a table and updating the UNION ALL view definition appropriately. You can have many view definitions for a year, a quarter, and so on, without adding extra date range predicates.

To create a UNION ALL view, choose a logical means of dividing a base table into separate physical tables. The most common division is by month.

For example, to create a view including all months for the first quarter, enter:

```
CREATE VIEW
SELECT * JANUARY
UNION ALL
SELECT * FEBRUARY
UNION ALL
SELECT * MARCH
```

```
UNION ALL
```

Each month, you can load data into a single base table—JANUARY, FEBRUARY, or MARCH in this example. Next month, load data into a new table with the same columns, and the same index types.

For syntax details, see UNION operation in the *Sybase IQ Reference Manual*.

Note You cannot perform an INSERT...SELECT into a UNION ALL view.

Optimizing queries that reference UNION ALL views

All partitions in a UNION ALL view must have a complete set of indexes defined for optimization to work.

Queries with DISTINCT will tend to run more slowly using a UNION ALL view than a base table.

Sybase IQ includes patented optimizations for UNION ALL views, including:

- Split group by over union all view
- Push-down join into union all view

Should you need to adjust performance for queries that reference UNION ALL views, you might want to set the Join_Preference database option, which affects joins between UNION ALL views. For details of these options, see Chapter 2, “Database Options,” in the *Sybase IQ Reference Manual.*

A UNION can be treated as a partitioned table only if it satisfies all of the following constraints:

- It contains only one or more UNION ALL.
- Each arm of the UNION has only one table in its FROM clause, and that table is a physical base table.
- No arm of the UNION has a DISTINCT, a RANK, an aggregate function, or a GROUP BY clause.
- Each item in the SELECT clause within each arm of the UNION is a column.
- The sequence of data types for the columns in the SELECT list of the first UNION arm is identical to the sequence in each subsequent arm of the UNION.

See also “SELECT statement,” in the *Sybase IQ Reference Manual*.

Network performance

The following sections offer suggestions for solving some network performance issues.

Improving large data transfers

Large data transfers simultaneously decrease overall throughput and increase the average response time. Here are some suggestions to improve performance during these transfers:

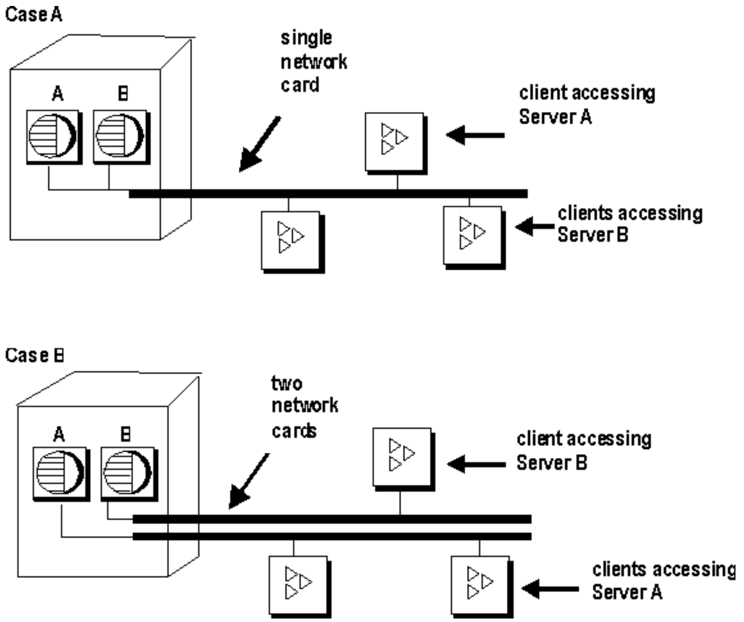
- Perform large transfers during off-hour periods, if possible.
- Limit the number of concurrent queries during large transfers.
- Do not run queries and insertions concurrently during large transfers.
- Use stored procedures to reduce total traffic.
- Use row buffering to move large batches through the network.
- If large transfers are common, consider installing better network hardware that is suitable for such transfers. For example:
    - Token ring–responds better during heavy utilization periods than ethernet hardware.
    - Fiber optic–provides very high bandwidth, but is usually too expensive to use throughout the entire network.
    - Separate network–can be used to handle network traffic between the highest volume workstations and the server.

Isolate heavy network users

In case A in Figure 12-4, clients accessing two different database servers use one network card. That means that clients accessing Servers A and B have to compete over the network and past the network card. In the case B, clients accessing Server A use a different network card than clients accessing Server B.

It would be even better to put your database servers on different machines. You may also want to put heavy users of different databases on different machines.

Figure 5-2: Isolating heavy network users



Put small amounts of data in small packets

If you send small amounts of data over the network, keep the default network packet size small (default is 512 bytes). The -p server startup option lets you specify a maximum packet size. Your client application may also let you set the packet size.

Put large amounts of data in large packets

If most of your applications send and receive large amounts of data, increase default network packet size. This will result in fewer (but larger) transfers.

Process at the server level

Filter as much data as possible at the server level.

CHAPTER 6 Monitoring and Tuning Performance

About this chapter

This chapter describes tools you use to monitor Sybase IQ performance. Use these tools to determine whether your system is making optimal use of available resources. To understand how Sybase IQ uses memory, process threads, and disk, and to learn about options you can set to control resource use, see Chapter 5, “Managing System Resources.” See also the sections on performance implications and tuning in other chapters of this guide for more tuning hints.

Contents

Viewing the Sybase IQ environment

The first step in tuning Sybase IQ performance is to look at your environment. You have various options:

- Use system monitoring tools (each system and site has different tools in place).
- Use one of the stored procedures that displays information about Sybase IQ. See the next section for more information.
- Determine appropriateness of index types. See Chapter 6, "Using Sybase IQ Indexes" in *Sybase IQ System Administration Guide* for more information about choosing index types.
- For on-screen information, look at your insert and delete notification messages. Chapter 7, "Moving Data In and Out of Databases" in *Sybase IQ System Administration Guide* gives more information about these messages.
- Look at the Sybase IQ message file, called *dbname.iqmsg* by default.
- Use the performance monitor in Sybase Central.
- Use procedure profiling to track execution times for stored procedures, functions, and events

Getting information using stored procedures

Sybase IQ offers several stored procedures that display information about your database:

- sp_iqconnection displays statistics about user connections and versions
- sp_iqcontext displays information about what statements are executing
- sp_iqcheckdb checks the validity of your current database
- sp_iqdbstatistics reports results of the most recent sp_iqcheckdb
- sp_iqdbsize gives the size of the current database
- sp_iqspaceinfo displays space usage by each object in the database
- sp_iqstatus displays miscellaneous status information about the database.
- sp_iqtablesize gives the size of the table you specify.
- sp_iqgroupsize lists the members of the specified group.

See *Sybase IQ Reference Manual* for syntax details and examples of all Sybase IQ stored procedures.

Using the Sybase Central performance monitor

You can monitor server statistics using Sybase Central as follows.

❖ **Monitoring performance in Sybase Central**

1 Select a server.

2 On the Statistics tab, right-click the name on the and choose Add to Performance Monitor.

3 Click the Performance Monitor tab. Sybase Central only tracks the difference from one snapshot to the next, so some selected statistics may show no activity in the Performance Monitor.

   For a description of each statistic, right-click its name on the Statistics tab, and choose Properties. You may also graph that statistic on the Performance Monitor by clicking the check box on the Properties tab, choosing Apply, and OK.

Profiling database procedures

Procedure profiling shows you how long it takes your stored procedures, functions, events, system triggers, and triggers to execute. You can also view the execution time for each line of a procedure. Using the database profiling information, you can determine which procedures can be fine-tuned to increase performance within your database.

When profiling is enabled, Sybase IQ monitors which stored procedures, functions, events, system triggers, and triggers are used, keeping track of how long it takes to execute them, and how many times each one is called.

Profiling information is stored in memory by the server and can be viewed in Sybase Central via the Profile tab or in Interactive SQL. Once profiling is enabled, the database gathers profiling information until you disable profiling or until the server is shut down.

For more information about obtaining profiling information in Interactive SQL, see “Viewing procedure profiling information in Interactive SQL” on page 168.

Enabling procedure profiling

Procedure profiling tracks the usage of procedures and triggers by all connections. You can enable profiling in either Sybase Central or Interactive SQL. You must have DBA authority to enable and use procedure profiling.

❖ **To enable profiling (Sybase Central)**

1 Connect to your database as a user with DBA authority.

2 Select the database in the left pane.

3 From the File menu, choose Properties.

The Database property sheet appears.

4 On the Profiling tab, select Enable Profiling on This Database.

5 Click OK to close the property sheet.

Note

You can also right click your database in Sybase Central to enable profiling. From the popup menu, choose Profiling > Start Profiling.

❖ **To enable profiling (SQL)**

1 Connect to your database as a user with DBA authority.

2 Call the sa_server_option stored procedure with the ON setting.

For example, enter:

```
CALL sa_server_option ( 'procedure_profiling', 'ON')
```

If necessary, you can see what procedures a specific user is using, without preventing other connections from using the database. This is useful if the connection already exists, or if multiple users connect with the same userid.

❖ **To filter procedure profiling by user**

1 Connect to the database as a user with DBA authority.

2 Call the following procedure:

```
CALL sa_server_option
('ProfileFilterUser','userid')
```

The value of *userid* is the name of the user being monitored.

Resetting procedure profiling

When you reset profiling, the database clears the old information and immediately starts collecting new information about procedures, functions, events, and triggers.

The following sections assume that you are already connected to your database as a user with DBA authority and that procedure profiling is enabled.

❖ **To reset profiling (Sybase Central)**

1. Select the database in the left pane.
2. From the File menu, choose Properties.

   The Database property sheet appears.
3. On the Profiling tab, click Reset Now.
4. Click OK to close the property sheet.

Note

You can also right click your database in Sybase Central to reset profiling. From the popup menu, click Profiling > Reset Profiling Information.

❖ **To reset profiling (SQL)**

- Call the sa_server_option stored procedure with the RESET setting.

  For example, enter:

  ```
  CALL sa_server_option ('procedure_profiling',
  'RESET')
  ```

Disabling procedure profiling

Once you are finished with the profiling information, you can either disable profiling or you can clear profiling. If you disable profiling, the database stops collecting profiling information and the information that it has collected to that point remains on the Profile tab in Sybase Central. If you clear profiling, the database turns profiling off and removes all the profiling data from the Profile tab in Sybase Central.

❖ **To disable profiling (Sybase Central)**

1. Select the database in the left pane.
2. From the File menu, choose Properties.

   The Database property sheet appears.
3. On the Profiling tab, clear the Enable Profiling on This Database option.

4 Click OK to close the property sheet.

Note

You can also right click your database in Sybase Central to disable profiling. From the popup menu, choose Profiling > Stop Profiling.

❖ To disable profiling (SQL)

- Call the sa_server_option stored procedure with the OFF setting.

  For example, enter:

```
CALL sa_server_option ('procedure_profiling',
'OFF')
```

❖ To clear profiling (Sybase Central)

1. Select the database in the left pane.
2. From the File menu, choose Properties.

   The Database property sheet appears.
3. On the Profiling tab, click Clear Now.

   You can only clear profiling if profiling is enabled.
4. Click OK to close the property sheet.

Note

You can also right click your database in Sybase Central to clear profiling. From the popup menu, select Profiling > Clear Profiling Information.

❖ To clear profiling (SQL)

- Call the sa_server_option stored procedure with the CLEAR setting.

  For example, enter:

```
CALL sa_server_option ('procedure_profiling',
'CLEAR')
```

Viewing procedure profiling information in Sybase Central

Procedure profiling provides you with different information depending whether you choose to look at information for your entire database, a specific type of object, or a particular procedure. The information can be displayed in the following ways:

- details for all profiled objects within the database
- details for all stored procedures and functions
- details for all events

- details for all triggers
- details for all system triggers
- details for individual profiled objects

You must be connected to your database and have profiling enabled to view profiling information.

When you view profiling information for your entire database, the following columns appear:

- **Name** Lists the name of the object.
- **Owner** Lists the owner of the object.
- **Table** Lists which table a trigger belongs to (this column only appears on the database Profile tab).
- **Event** Shows the type of trigger for system triggers. This can be Update or Delete.
- **Type** Lists the type of object, for example, a procedure.
- **# Exes.** Lists the number times each object has been called.
- **#msecs.** Lists the total execution time for each object.

These columns provide a summary of the profiling information for all of the procedures that have been executed within the database. One procedure can call other procedures, so there may be more items listed than those users call specifically.

❖ To view summary profiling information for stored procedures and functions

1. Select the Procedures & Functions folder in the left pane.
2. Click the Profile tab in the right pane.

   Profiling information about all the stored procedures and functions within your database appears on the Profile tab.

❖ To view summary profiling information for events

1. Open the Events folder in the left pane.

   A list of all the events in your database appears on the Events tab in the right pane.
2. Click the Profile tab in the right pane.

Profiling information about all of the events within your database appears on the Profile tab.

❖ To view summary profiling information for triggers

1. Open the Triggers folder in the left pane.

   A list of all the triggers in your database appears on the Triggers tab.

2. Click the Profile tab in the right pane.

   Profiling information about all of the triggers in your database appears on the Profile tab.

❖ To view summary profiling information for system triggers

1. Open the System Triggers folder in the left pane.

   A list of all the triggers in your database appears on the System Triggers tab.

2. Click the Profile tab in the right pane.

   Profiling information about all of the system triggers in your database appears on the Profile tab.

Viewing profiling information for a specific procedure

Sybase IQ provides procedure profiling information about individual stored procedures, functions, events, and triggers. Sybase Central displays different information about individual procedures than it does about all of the stored procedures, functions, events, or triggers within a database.

When you look at the profiling information for a specific procedure, the following columns appear:

- **Calls** Lists the number of times the object has been called.
- **Milliseconds** Lists the total execution time for each object.
- **Line** Lists the line number beside each line of the procedure.
- **Source** Displays the SQL procedure, line by line.

The procedure is broken down line by line and you can examine it to see which lines have longer execution times and therefore might benefit from changes to improve the procedure's performance. You must be connected to the database, have profiling enabled, and have DBA authority to access procedure profiling information.

❖ **To view the profiling information for a stored procedure or function**

1. Expand the database in the left pane.
2. Select the Procedures and Functions folder in the left pane.

   A list of all the stored procedures and functions within your database appears on the Procedures & Functions tab in the right pane.
3. Click the stored procedure or function you want to profile in the left pane.
4. Click the Profile tab in the right pane.

   Profiling information about the specific stored procedure or function appears on the Profile tab in the right pane.

❖ **To view profiling information for an event**

1. Expand the database in the left pane.
2. Select the Events folder in the left pane.

   A list of all the events within your database appears on the Events tab in the right pane.
3. Click the event you want to profile in the left pane.
4. Click the Profile tab in the right pane.

   Profiling information about the specific event appears on the Profile tab in the right pane.

❖ **To view profiling information for triggers**

1. Expand the database in the left pane.
2. Open the Triggers folder in the left pane.

   A list of all the triggers appears on the Triggers tab in the right pane.
3. Select the trigger you want to profile in the right pane.
4. Click the Profile tab in the right pane.

   Profiling information about the specific trigger appears on the Profile tab in the right pane.

❖ **To view profiling information for system triggers**

1. Expand the database in the left pane.
2. Open the System Triggers folder in the left pane.

A list of all the system triggers appears on the System Triggers tab in the right pane.

3. Select the system trigger you want to profile in the right pane.
4. Click the Profile tab in the right pane.

   Profiling information about the specific system trigger appears on the Profile tab in the right pane.

Viewing procedure profiling information in Interactive SQL

You can use stored procedures to view procedure profiling information. The profiling information is the same whether you view it in Sybase Central or in Interactive SQL.

The sa_procedure_profile_summary stored procedure provides information about all of the procedures within the database. You can use this procedure to view the profiling data for stored procedures, functions, events, system triggers, and triggers within the same result set. The following parameters restrict the rows the procedure returns.

- **p_object_name** Specifies the name of an object to profile.
- **p_owner_name** Specifies the owner whose objects you want to profile.
- **p_table_name** Specifies table to profile triggers.
- **p_object_type** Specifies the type of object to profile. You can choose from the following five options. Choosing one of these values restricts the result set to only objects of the specified type.
  - **P** Stored procedure
  - **F** Function
  - **T** Trigger
  - **E** Event
  - **S** System trigger
- **p_ordering** Specifies the sort order of the result set.

Keep in mind that there may be more items listed than those called specifically by users because one procedure can call another procedure.

The following sections assume that you are already connected to your database as a user with DBA authority and that you have procedure profiling enabled.

❖ To view summary profiling information for all procedures

1. Execute the sa_procedure_profile_summary stored procedure.

   For example, enter:

   ```
   CALL sa_procedure_profile_summary
   ```

2. From the SQL menu, choose Execute.

   A result set with information about all of the procedures in your database appears on the Results pane.

For more information about the sa_procedure_profile_summary stored procedure, see *Adaptive Server Anywhere SQL Reference*.

Viewing profiling information for a specific procedure in Interactive SQL

The sa_procedure_profile stored procedure provides information about individual lines within specific procedures. The result set includes the line number, execution time, and percentage of total execution time for lines within procedures. You can use the following parameters to restrict the rows the procedure returns:

- **p_object_name** Specifies the name of an object to profile.
- **p_owner_name** Specifies the owner whose objects you want to profile.
- **p_table_name** Specifies which table to profile triggers.

If you do not include any parameters in your query, the procedure returns profiling information for all the procedures that have been called.

❖ To view profiling information for specific lines within procedures

1. Execute the sa_procedure_profile stored procedure.

   For example, enter:

   ```
   CALL sa_procedure_profile
   ```

2. From the SQL menu, choose Execute.

   A result set with profiling information for individual procedure lines appears in the Results pane.

For more information about the sa_procedure_profile stored procedure, see *Adaptive Server Anywhere SQL Reference*.

Monitoring the buffer caches

Sybase IQ provides a tool to monitor the performance of the buffer caches. This monitor collects statistics on the buffer cache, memory, and I/O functions taking place within Sybase IQ, and stores them in a log file.

Buffer cache performance is a key factor in overall performance of Sybase IQ. Using the information the monitor provides, you can fine tune the amount of memory you allocate to the main and temp buffer caches. If one cache is performing significantly more I/O than the other, reallocate some of the memory appropriately. Reallocate in small amounts such as 10 to 50MB and on an iterative basis. After reallocating, rerun the workload and monitor the changes in performance.

Starting the buffer cache monitor

You run the Sybase IQ buffer cache monitor from DBISQL. Each time you start the monitor it runs as a separate kernel thread within Sybase IQ.

Use this syntax to start the monitor:

IQ UTILITIES { MAIN | PRIVATE }
INTO *dummy_table_name*
START MONITOR '*monitor_options* […]'

MAIN starts monitoring of the main buffer cache, for all tables in the IQ Store of the database you are connected to.

PRIVATE starts monitoring of the temp buffer cache, for all tables in the Temporary Store of the database you are connected to.

You need to issue a separate command to monitor each buffer cache. *You must keep each of these sessions open while the monitor collects results; a monitor run stops when you close its connection.* A connection can run up to a maximum of two monitor runs, one for the main and one for the temp buffer cache.

dummy_table_name can be any Sybase IQ base or temporary table. The table name is required for syntactic compatibility with other IQ UTILITIES commands. It is best to have a table that you use only for monitoring.

To control the directory placement of monitor output files, set the MONITOR_OUTPUT_DIRECTORY option. If this option is not set, the monitor sends output to the same directory as the database. All monitor output files are used for the duration of the monitor runs. They remain after a monitor run has stopped.

Either declare a temporary table for use in monitoring, or create a permanent dummy table when you create a new database, before creating any multiplex query servers. These solutions avoid DDL changes, so that data stays up on query servers during production runs.

Tip
To simplify monitor use, create a stored procedure to declare the dummy table, specify its output location, and start the monitor.

'*monitor_options*' can include one or more of the following values:

- -summary displays summary information for both the main and temp buffer caches. If you do not specify any monitor options, you receive a summary report. The fields displayed are as described for the other options, plus the following:
    - *Users*: Number of users connected to the buffer cache
    - *IO*: Combined physical reads and writes by the buffer cache
- -cache displays activity in detail for the main or temp buffer cache. Critical fields are Finds, HR%, and BWaits. The fields displayed are:
    - *Finds*: Find requests to the buffer cache. If the Finds value suddenly drops to zero and remains zero, the server is deadlocked. When the server has any activity, the Finds value is expected to be non-zero.
    - *Creats*: Requests to create a page within the database
    - *Dests*: Requests to destroy a page within the database
    - *Dirty*: Number of times the buffer was dirtied (modified)
    - *HR%*: Hit rate, the percentage of above satisfied by the buffer cache without requesting any I/O. The higher the Hit Rate the better, usually 90% - 100% if you set the cache large enough. For a large query, Hit Rate may be low at first, but increase as prefetching starts to work.
    - *BWaits*: Find requests forced to wait for a busy page (page frame contention). Usually it is low, but is some special cases it may be high. For example, if identical queries are started at the same time, both need the same page, so the second request must wait for the first to get that page from disk.

  - *ReReads*: Approximate number of times the same portion of the store needed to be reread into the cache within the same transaction. Should always be low, but a high number is not important for Sybase IQ 12.4.2 and above.
  - *FMiss*: False misses, number of times the buffer cache needed multiple lookups to find a page in memory. This number should be 0 or very small. If the value is high, it is likely that a rollback occurred, and certain operations needed to be repeated
  - *Cloned*: Number of buffers that Sybase IQ needed to make a new version for a writer, while it had to retain the previous version for concurrent readers. A page only clones if other users are looking at that page.
  - *Reads/Writes*: Physical reads and writes performed by the buffer cache
  - *PF/PFRead*: Prefetch requests and reads done for prefetch.
  - *GDirty*: Number of times the LRU buffer was grabbed dirty and Sybase IQ had to write it out before using it. This value should not be greater than 0 for a long period. If it is, you may need to increase the number of sweeper threads or move the wash marker.
  - *Pin%*: Percentage of pages in the buffer cache in use and locked.
  - *Dirty%*: Percentage of buffer blocks that were modified. Try not to let this value exceed 85-90%; otherwise, GDirty will become greater than 0.
- -cache_by_type produces the same results as -cache, but broken down by IQ page type. (An exception is the Bwaits column, which shows a total only.) This format is most useful when you need to supply information to Sybase Technical Support.
- -file_suffix *suffix* creates a monitor output file named <dbname>.<connid>-<main_or_temp>-<suffix>. If you do not specify a suffix, it defaults to *iqmon*.
- -io displays main or temp (private) buffer cache I/O rates and compression ratios during the specified interval. These counters represent all activity for the server; the information is not broken out by device. The fields displayed are:
  - *Reads*: Physical reads performed by the buffer cache

    - *Lrd(KB)*: Logical kilobytes read in (page size multiplied by the number of requests)
    - *Prd(KB)*: Physical kilobytes read in
    - *Rratio*: Compression ratio of logical to physical data read in, a measure of the efficiency of the compression to disk for reads
    - *Writes*: Physical writes performed by the buffer cache
    - *Lwrt(KB)*: Logical kilobytes written
    - *Pwrt(KB)*: Physical kilobytes written
    - *Wratio*: Compression ratio of logical to physical data written

- -bufalloc displays information on the main or temp buffer allocator, which reserves space in the buffer cache for objects like sorts, hashes, and bitmaps.

    - *OU*: User_Resource_Reservation option setting (formerly Optimize_For_This_Many_Users)
    - *AU*: Current number of active users
    - *MaxBuf*: Number buffers under control of the buffer allocator
    - *Avail*: Number of currently available buffers for pin quota allocation
    - *AvPF*: Number of currently available buffers for prefetch quota allocation
    - *Slots*: Number of currently registered objects using buffer cache quota
    - *PinUser*: Number of objects (for example, hash, sort, and B-tree objects) using pin quota
    - *PFUsr*: Number of objects using prefetch quota
    - *Posted*: Number of objects that are pre-planned users of quota
    - *UnPost*: Number of objects that are ad hoc quota users
    - *Locks*: Number of mutex locks taken on the buffer allocator
    - *Waits*: Number of times a thread had to wait for the lock

- -contention displays many key buffer cache and memory manager locks. These lock and mutex counters show the activity within the buffer cache and heap memory and how quickly these locks were resolved. Watch the timeout numbers. If system time exceeds 20%, it indicates a problem.

Note Due to operating system improvements, Sybase IQ no longer uses spin locks. As a result, the woTO, Loops, and TOs statistics are rarely used.

- *AU*: Current number of active users
- *LRULks*: Number times the LRU was locked (repeated for the temp cache)
- *woTO*: Number times lock was granted without timeout (repeated for the temp cache)
- *Loops*: Number times Sybase IQ retried before lock was granted (repeated for the temp cache)
- *TOs*: Number of times Sybase IQ timed out and had to wait for the lock (repeated for the temp cache)
- *BWaits*: Number of “Busy Waits” for a buffer in the cache (repeated for the temp cache)
- *IOLock*: Number of times Sybase IQ locked the compressed I/O pool (repeated for the temp cache); can be ignored
- *IOWait*: Number of times Sybase IQ had to wait for the lock on the compressed I/O pool (repeated for the temp cache); can be ignored
- *HTLock*: Number of times Sybase IQ locked the block maps hash table (repeated for the temp cache)
- *HTWait*: Number of times Sybase IQ had to wait for the block maps hash table (repeated for the temp cache); HTLock and HTWait indicate how many block maps you are using
- *FLLock*: Number of times Sybase IQ had to lock the free list (repeated for the temp cache)
- *FLWait*: Number of times Sybase IQ had to wait for the lock on the free list (repeated for the temp cache)
- *MemLks*: Number of times Sybase IQ took the memory manager (heap) lock

    - *MemWts*: Number of times Sybase IQ had to wait for the memory manager lock

- -threads displays counter used by the processing thread manager. Values are server-wide (i.e., it does not matter whether you select this option for main or private). They represent new events since the last page of the report.
    - *cpus*: Number of CPUs Sybase IQ is using; this may be less than the number on the system
    - *Limit*: Maximum number of threads Sybase IQ can use
    - *NTeams*: Number of thread teams currently in use
    - *MaxTms*: Largest number of teams that has ever been in use
    - *NThrds*: Current number of existing threads
    - *Resrvd*: Number of threads reserved for system (connection) use
    - *Free*: Number of threads available for assignment. Monitor this value— if it is very low, it indicates thread starvation
    - *Locks*: Number of locks taken on the thread manager
    - *Waits*: Number of times Sybase IQ had to wait for the lock on the thread manager

Note When an object or query needs work, Sybase IQ allocates a group of processing threads called a thread team. Useful options in adjusting thread use include database options MAX_IQ_THREADS_PER_CONNECTION and MAX_IQ_THREADS_PER_TEAM, and the server option -iqmt which specifies the number of threads Sybase IQ can use.

- -interval specifies the reporting interval in seconds. The default is every 60 seconds. The minimum is every 2 seconds. You can usually get useful results by running the monitor at the default interval during a query or time of day with performance problems. A very short interval may not give meaningful results. The interval should be proportional to the job time; one minute is generally more than enough.

  The first display shows counters from the start of the server. Subsequent displays show the difference from the previous display.

- -append | - truncate Append to existing output file or truncate existing output file, respectively. Truncate is the default.

- -debug is used mainly to supply information to Sybase Technical Support. It displays all the information available to the performance monitor, whether or not there is a standard display mode that covers the same information. The top of the page is an array of statistics broken down by disk block type. This is followed by other buffer cache statistics, memory manager statistics, thread manager statistics, free list statistics, CPU utilization, and finally buffer allocator statistics. The buffer allocator statistics are then broken down by client type (hash, sort, and so on) and a histogram of the most recent buffer allocations is displayed. Note that memory allocations indicate how much is allocated since the last page of the report.

Note The interval, with two exceptions, applies to each line of output, not to each page. The exceptions are -cache_by_type and -debug, where a new page begins for each display.

Checking results while the monitor runs

On UNIX systems, you can watch monitor output as queries are running.

For example, you could start the monitor using the following command:

```
iq utilities main into monitor_tab
start monitor "-cache -interval 2 -file_suffix iqmon"
```

This command sends output to an ASCII file with the name *dbname.conn#-[main|temp]-iqmon*. So, for the database asiqdemo, results would be sent to *asiqdemo.2-main-iqmon*.

To watch results, issue the following command at the system prompt:

```
$ tail -f asiqdemo.2-main-iqmon
```

Stopping the buffer cache monitor

The command you use to stop a monitor run is similar to the one you use to start it, except that you do not need to specify any options. Use this syntax to stop the Sybase IQ buffer cache monitor:

IQ UTILITIES { MAIN | PRIVATE }
INTO *dummy_table_name* **STOP MONITOR**

Note In order for certain option settings to take effect you must restart the database. If the monitor is running you need to shut it down so that the database can be restarted.

Examining and saving monitor results

The monitor stores results in an ordinary text file. This file defaults to:

- *dbname.connection#-main-iqmon* for main buffer cache results
- *dbname.connection#-temp-iqmon* for temp buffer cache results

The prefix *dbname.connection#* represents your database name and connection number. If you see more than one connection number and are uncertain which is yours, you can run the Catalog stored procedure sa_conn_info. This procedure displays the connection number, user ID, and other information for each active connection to the database.

You can use the -file_suffix parameter on the IQ UTILITIES command to change the suffix *iqmon* to a suffix of your choice.

To see the results of a monitor run, use a text editor or any other method you would normally use to display or print a file.

When you run the monitor again from the same database and connection number, by default it overwrites the previous results. If you need to save the results of a monitor run, copy the file to another location before starting the monitor again from the same database or use the -append option.

Examples of monitor results

This section shows sample results using different monitor options.

The -summary option produces results like the following. Note that it shows both main and temp buffer cache statistics, no matter which you request in the IQ UTILITIES command:

```
Sybase Adaptive Server IQ Performance Monitor

                   ------------------------------------------
```

```
                                Version 3.2


Options string for Main cache: "-summary -interval 5"
                                                           Summary

                                                    2004-07-16 13:53:24

Active|                Main Cache                   |           Temp Cache

 Users|  Finds   HR%  Reads/Writes  GDirty  Pin% Dirty% InUse%|  Finds   HR%
Reads/Writes  GDirty  Pin% Dirty% InUse%

     0     286  99.3      2/34          0   0.0    1.6   26.2     608  99.7
2/47           0   0.0    3.6   20.0

     1    2621  99.4     16/155          0   5.6    8.7   81.7    4121  99.6
16/163          0  11.4   23.2   67.3

     1    2646  99.8      6/48          0   1.6   13.5  100.0    3388  99.8
6/70           1   4.1   40.9   94.5

     1    2684  99.9      7/78          0   5.6   14.3  100.0    3497  99.9
8/103          1  10.9   42.3   99.1

     1    1993  99.9     17/22          0   4.0   31.0  100.0    3342  98.7
122/149          0   8.2   41.4   91.4

     1    2479  99.9     32/110          0   5.6   13.5  100.0    3370  99.8
55/112          0  11.4   45.5   95.9

     1    3273 100.0      0/0           0   5.6   23.8  100.0    3951 100.0
0/108          1  13.6   49.1  100.0

     1    2512  99.9      2/0           0   1.6   31.0  100.0    3916  98.9
88/173          0   5.5   48.6  100.0

     1    1264  99.9     66/131          0   4.0   45.2  100.0    4317  98.9
378/305          0   6.4   40.0   77.3

     1    2122  99.8     30/125          0   5.6   12.7   99.2    3122  99.7
67/127          0  12.3   40.0   90.5

     1    3370 100.0      2/0           0   5.6   23.0  100.0    4034 100.0
2/98           2  13.2   46.4   98.2

     1    2981  99.9      2/0           0   5.6   31.7  100.0    3715  99.9
2/110          0  14.1   53.2  100.0

     1    3351  99.6     13/3           0   5.6   39.7  100.0    4131  99.7
13/123          0  14.1   57.7  100.0

     1    3286  99.6     13/13          0   5.6   40.5  100.0    4135  99.6
```

```
15/139          0  12.3   55.9   97.7

     1     296 100.0      0/0           0   1.6   41.3  100.0    3646  96.9
366/320          0   7.3   53.2  100.0

     1    1230  99.4     71/129         0   6.3   58.7  100.0    4221  98.9
390/297          0   9.5   59.1   91.8

     1    1900 100.0    125/279         0   4.0   50.0  100.0    4102 100.0
344/279          0   7.7   38.6   72.3


                Sybase Adaptive Server IQ Performance Monitor
                ---------------------------------------------
                               Shutting Down


     0     422  98.8     16/99          0   0.0    0.8   99.2     853  98.9
34/101          0   0.0    1.8   59.1
```

The -cache option produces results like the following, which are for the temp buffer cache.

```
Options string for Temp cache: "-cache -interval 10"
                           Temp Shared Buffer Cache
                            2001-02-18 17:43:55
    Finds Creats Dests Dirty   HR% BWaits ReReads FMiss Cloned Reads/    PF/
GDirty Pin% Dirty%
                                                                  Writes PFRead
Tm:   640     82    57    84  99.4      0       4     0      0    4/0    0/0
0  0.0   2.8
Tm:  1139    109    83   109 100.0      0       0     0      0    0/0    0/0
0  0.0   5.5
Tm:  6794    754   749   754 100.0      0       0     0      0    0/0    0/0
0  0.0   6.1
Tm: 10759   1646  1646  1646 100.0      0       0     0      0    0/0    0/0
0  0.0   6.1
```

The -io option produces results like the following, which are for the main buffer cache:

```
Options string for main cache: "-IO -interval 5"
                          Main Buffer Cache
                         2001-02-18 13:58:48
                Input                           Output
      Reads Lrd(KB) Prd(KB)     Rratio  Writes  Lwrt(KB)  Pwrt(KB)  Wratio
Mn:      10      40      34       1.18      14        56        23    2.43
Mn:       0       0       0       0.00      21        84        34    2.43
```

```
Mn:         0        0        0       0.00        7         28         11     2.43
Mn:         0        0        0       0.00       22         88         35     2.48
Mn:         0        0        0       0.00       63        252        100     2.51
Mn:         0        0        0       0.00       54        216         93     2.32
Mn:         0        0        0       0.00       64        256        101     2.52
Mn:         0        0        0       0.00       62        248         94     2.62
Mn:         0        0        0       0.00       73        292        110     2.65
Mn:         0        0        0       0.00      105        420        121     3.47
```

The -buffalloc option produces results like the following.

```
 Options string for Main cache: "-bufalloc -file_suffix bufalloc-iqmon -append
-interval 10"
                              Buffer Allocation
2001-02-18 10:58:39

OU/AU MaxBuf  Avail AvPF  Slots PinUsr  PFUsr Posted UnPost Quota Locks  Waits
1/0    1592   1592    20      0      0      0      0      0     0     1      0

1/1    1592   1592    20      0      0      0      0      0     0     1      0

1/1    1592   1592    20      0      0      0      0      0     0     1      0
```

Note The actual -contention output shows Main Cache, Temp Cache, and Memory Manager on the same line. Because this format is very wide, each of these sets of columns is shown separately here.

The -contention results for the main cache are:

```
Options string for Main cache:
"-contention -file_suffix contention-iqmon -append -interval 10"
                       Contention
                       2001-02-18 10:57:03


                      Main Cache                                             |
AU |LRULks woTO  Loops    TOs BWaits IOLock IOWait HTLock HTWait FLLock FLWait

 0     66     0      0      0     0      1      0     5      0      4   0
 1   2958     0      0      0     0    160      0  1117      0      6   0
 1   1513     0      0      0     1    378      0     2      0      8   0
 1    370     0      0      0     0     94      0     2      0     10   0
 1    156     0      0      0     0     46      0     2      0     12   0
 1    885     0      0      0     0    248      0     2      0     14   0
 1   1223     0      0      0     0    332      1     2      0     16   0
 1    346     0      0      0     0     66      0     2      0     18   0
```

The -contention results for the temp cache are:

```
                        Temp Cache
|LRULks   woTO  Loops    TOs BWaits IOLock IOWait HTLock HTWait FLLock FLWait
70         0      0      0      0      1      0      4      0      5      0
466        0      0      0      0      2      0     15      0     12      0
963        0      0      0      0      2      0      8      0     20      1
1186       0      0      0      0      2      0      2      0     23      1
357        0      0      0      0      2      0      2      0     25      1
444        0      0      0      0      2      0      3      0     29      0
884        0      0      0      0      2      0      2      0     31      1
1573       0      0      0      0      2      0      5      0     37      1
```

The results for the memory manager are:

```
| Memory Mgr
MemLks MemWts
55483    13
5705      0
2048      0
186       4
2         0
137       0
22        0
203       3
```

The results of the -threads option look like the following:

```
 Options string for Main cache: "-threads -file_suffix threads-iqmon -append -
interval 10"


                                    Threads

                              2001-02-18 10:59:24



  CPUs  Limit NTeams MaxTms NThrds  Resrvd Free  Locks Waits

  10    100      4     12    100      13    68    106   590

  10    100      6     12    100      12    63      4     6

  10    100      6     12    100      12    63      0     0

  10    100      7     12    100      12    62      1     1

  10    100      7     12    100      12    62      0     0
```

10	100	7	12	100	12	58	1	5
10	100	7	12	100	12	58	0	0

Buffer cache structure

Sybase IQ automatically calculates the number of cache partitions for the buffer cache according to the number of CPUs on your system. If load or query performance in a multi-CPU configuration is slower than expected, you may be able to improve it by changing the value of the CACHE_PARTITIONS database option. For details, see CACHE_PARTITIONS option in *Sybase IQ Reference Manual*.

As buffers approach the Least Recently Used (LRU) end of the cache, they pass a wash marker. Sybase IQ writes the oldest pages—those past the wash marker—out to disk so that the cache space they occupy can be reused. A team of Sybase IQ processing threads, called sweeper threads, sweeps (writes) out the oldest buffers.

When Sybase IQ needs to read a page of data into the cache, it grabs the LRU buffer. If the buffer is still "dirty" (modified) it must first be written to disk. The Gdirty column in the monitor -cache report shows the number of times the LRU buffer was grabbed dirty and Sybase IQ had to write it out before using it.

Usually Sybase IQ is able to keep the Gdirty value at 0. If this value is greater than 0 for more than brief periods, you may need to adjust one of the database options that control the number of sweeper threads and the wash marker. See "SWEEPER_THREADS_PERCENT option" or "WASH_AREA_BUFFERS_PERCENT option" in Chapter 2, "Database Options," *Sybase IQ Reference Manual*.

Avoiding buffer manager thrashing

Operating system paging affects queries that need buffers which exceed the free memory available. Some of this paging is necessary, especially as you allocate more and more physical memory to your buffer caches. However, if you overallocate the physical memory to your buffer caches, the operating system paging occurs much more frequently, and it can cause your entire system to thrash. The reverse is true as well: Sybase IQ thrashes if you do not allocate enough memory to your buffer caches.

Buffer manager thrashing occurs when the operating system chooses less optimum buffers to page out to disk, which forces the buffer manager to make extra reads from disk to bring those buffers back to memory. Since Sybase IQ knows which buffers are the best candidates to flush out to disk, you want to avoid this operating system interference by reducing the overall number of page outs.

When you set buffer sizes, keep in mind the following trade-off:

- If the Sybase IQ buffer cache is too large, the operating system is forced to page as Sybase IQ tries to use all of that memory.
- If the Sybase IQ buffer cache is too small, then Sybase IQ thrashes because it cannot fit enough of the query data into the cache.

If you are experiencing dramatic performance problems, you should monitor paging to determine if thrashing is a problem. If so, then reset your buffer sizes as described in “Managing buffer caches”.

If you monitor paging and determine that thrashing is a problem, you can also limit the amount of thrashing during the execution of a statement which includes a query that involves hash algorithms. Adjusting the HASH_THRASHING_PERCENT database option controls the percentage of hard disk I/Os allowed before the statement is rolled back and an error is returned.

The default value of HASH_THRASHING_PERCENT is 10%. Increasing HASH_THRASHING_PERCENT permits more paging to disk before a rollback and decreasing HASH_THRASHING_PERCENT permits less paging before a rollback.

Queries involving hash algorithms that executed in earlier versions of Sybase IQ may now be rolled back when the default HASH_THRASHING_PERCENT limit is reached. Sybase IQ reports the error `Hash insert thrashing detected` or `Hash find thrashing detected`. Take one or more of the following actions to provide the query with the resources required for execution:

- Relax the paging restriction by increasing the value of HASH_THRASHING_PERCENT.
- Increase the size of the temporary cache (DBA only). Keep in mind that increasing the size of the temporary cache reduces the size of the main cache.
- Attempt to identify and alleviate why Sybase IQ is misestimating one or more hash sizes for this statement. For example, check that all columns that need an LF or HG index have one. Also consider if a multicolumn index is appropriate.
- Decrease the value of the database option HASH_PINNABLE_CACHE_PERCENT.

For more information on these database options, see the sections “HASH_THRASHING_PERCENT option” and “HASH_PINNABLE_CACHE_PERCENT option” in Chapter 2, “Database Options” of the *Sybase IQ Reference Manual*.

To identify possible problems with a query, generate a query plan by running the query with the temporary database options QUERY_PLAN = 'ON' and QUERY_DETAIL = 'ON', then examine the estimates in the query plan. The generated query plan is in the message log file.

Monitoring paging on Windows systems

Windows provides the System Monitor to help you monitor paging. To access it, select the object Logical Disk, the instance of the disk containing the file *PAGEFILE.SYS*, and the counter Disk Transfers/Sec. This should be on a separate disk from your database files. You can also monitor the Object Memory and the counter Pages/Sec. However, this value is the sum of all memory faults which includes both soft and hard faults.

Monitoring paging on UNIX systems

UNIX provides a system command, vmstat, to help you monitor system activity such as paging. The abbreviated command syntax is:

vmstat *interval count*

The *interval* is the time between rows of output, and *count* is the number times a row of output is displayed. For more information about vmstat (including its options and field descriptions), see your operating system's documentation. Here is an example:

```
>  vmstat 2 3
procs    memory             page             disk     faults     cpu
r b w swap    free   re mf pi po fr de sr s0 s1 sd  in   sy  cs us sy id

0 0 0 3312376 31840 0  8  0  0  0  0  0   0  0  0 297  201 472 82  4 14
0 0 0 3312376 31484 2  3  0  0  0  0  0   0  0  0 260  169 597 80  3 17
0 0 0 3312368 31116 0  8  0  0  0  0  0   0  0  0 205 1202 396 67  4 29
```

The above output shows a steady Sybase IQ querying state where the physical memory of the machine has not been overallocated. Little to no system page faulting is occurring. These next set of examples show vmstat output that indicates a problem. (The output shown omits some of the above fields to fit better on the page.)

```
procs     memory            page                faults       cpu
r b w swap    free    re   mf  pi po fr de sr  in    sy   cs us sy  id

0 0 0 217348  272784  0  148  11  3  9  0  2 251 1835  601  6  3  91
0 0 0 3487124 205572  0    5   0  0  0  0  0  86  131  133  0  1  99
0 0 0 3487124 205572  0    5   0  0  0  0  0  71  162  121  0  0 100
0 0 0 3483912 204500  0  425  36  0  0  0  0 169  642  355  2  2  96
0 0 0 3482740 203372  0   17   6  0  0  0  0 158  370  210  1  3  97
0 0 0 3482676 203300  0    4  10  0  0  0  0 160 1344  225  1  2  97
0 0 0 3343272 199964  1 2123  36  0  0  0  0 213  131  399  7  8  85
0 0 0 3343264 185096  0  194  84  0  0  0  0 283  796  732  1  6  93
0 0 0 3342988 183972  0   17  58  0  0  0  0 276 1051  746  2  4  94
0 0 0 3342860 183632  0  119 314  0  0  0  0 203 1660  529  3  4  94
0 0 0 3342748 182316  2  109 184  0  0  0  0 187  620  488  4  2  95
0 0 0 3342312 181104  2  147  96  0  0  0  0 115  256  260  9  2  89
0 0 0 3340748 179180  0  899  26  0  0  0  0 163  836  531  4  4  92
0 0 0 3328704 167224  0 2993   6  0  0  0  0  82 2195  222  4  7  89
```

The first line of the above output provides a summary of the system activity since the machine was started. The first three lines show that there is approximately 200MB of free physical memory and that the machine is idle. The fourth line corresponds to Sybase IQ starting up for the first time. Beginning at the eighth line, the amount of free memory starts to reduce rapidly. This corresponds to the Sybase IQ buffer caches being allocated and database pages being read in from disk (note that CPU usage has increased). At this time there is little user CPU time as no queries have begun.

```
procs     memory                page             faults      cpu
r  b w swap    free    re mf   pi    po fr de sr in     sy  cs    us sy id

 7 0 0 3247636 58920  0 1880 1664 0  0  0  0  1131  442 1668 80 18  3
18 0 0 3246568 43732  0  709 1696 0  0  0  0  1084  223 1308 90 10  1
12 0 0 3246604 37004  0  358  656 0  0  0  0   600  236  722 95  5  0
15 0 0 3246628 32156  0  356 1606 0  0  0  0  1141  226 1317 91  9  0
19 0 0 3246612 26748  0  273 1248 0  0  0  0   950  394 1180 92  7  0
```

The above output is from slightly later when the query is underway. This is evident from the user mode CPU level (*us* field). The buffer cache is not yet full as page-in faults (*pi* field or KB paged in) are still occurring and the amount of free memory is still going down.

```
procs     memory            page                         faults        cpu
r  b w swap    free   re mf pi    po   fr    de sr   in    sy    cs    us sy id

21 0 0 3246608 22100 0 201 1600   0     0 0     0 1208 1257 1413 88 12  0
18 0 0 3246608 17196 0 370 1520   0   464 0   139  988  209 1155 91  8  0
11 0 0 3251116 16664 0 483 2064 138 2408 0   760 1315  218 1488 88 12  0
30 0 0 3251112 15764 0 475 2480 310 4450 0 1432 1498  199 1717 87 13  0
```

The above output is from even later. On the third line of the output it shows that the system has reached its threshold for the amount of free memory it can maintain. At this point, page-outs (*po* field or KB paged out) occur and the level of system mode CPU (*sy* field) increases accordingly. This situation results because physical memory is overallocated: the Sybase IQ buffer caches are too big for the machine. To resolve this problem, reduce the size of one or both of the buffer caches.

Buffer cache monitor checklist

The following table summarizes the most common items to look for in monitor results, and suggests actions you may need to take if behavior is outside the normal range. The Statistic column lists the name you see in the standard monitor reports; if this statistic appears differently in the debug report, the debug statistic is also listed.

Remember that for any monitor statistic, a temporary anomaly may occur while the system changes state, such as when a new query is starting.

Table 6-1: Buffer cache monitor checklist

Statistic	Normal behavior	Behavior that needs adjusting	Recommended action
HR% (Cache hit rate)	Above 90%. For individual internal data structures like garray, barray, bitmap (bm), hash object, sort object, variable-length btree (btreev), fixed-length btree (btreef), bit vector (bv), dbext, dbid, vdo, store, checkpoint block (ckpt), the hit rate should be above 90% while a query runs. It may be below 90% at first. Once prefetch starts working (PF or PrefetchReqs > 0), the hit rate should gradually grow to above 90%.	Hit rate below 90% after prefetch is working. **Note** Some objects do not do prefetching, so their hit rate may be low normally.	Try rebalancing the cache sizes of main versus temp by adjusting -iqmc and -iqtc. Also try increasing the number of prefetch threads by adjusting PREFETCH_THREADS_PERCENT option.
Gdirty (Grabbed Dirty)	0 in a system with a modest cache size (< 10GB).	GDirty > 0 **Note** Sweeper threads are activated only when the number of dirty pages reaches a certain percentage of the wash area. If GDirty/GrabbedDirty is above 0 and the I/O rate (Writes) is low, the system may simply be lightly loaded, and no action is necessary.	Adjust SWEEPER_THREADS_PERCENT option (default 10%) or WASH_AREA_BUFFERS_PERCENT option (default 20%) to increase the size of the wash area.
BWaits (Buffer Busy Waits)	0	Persistently > 0, indicating that multiple jobs are colliding over the same buffers.	If the I/O rate (Writes) is high, Busy Waits may be caused by cache thrashing. Check Hit Rate in the cache report to see if you need to rebalance main versus temp cache. If a batch job is starting a number of nearly identical queries at the same time, try staggering the start times.

Statistic	Normal behavior	Behavior that needs adjusting	Recommended action
LRU Waits (LRUNum TimeOuts percentage in debug report)	20% or less	> 20%, which indicates a serious contention problem.	Check the operating system patch level and other environment settings. This problem tends to be an O.S. issue.
IOWait (IONumWaits)	10% or lower	> 10%	Check for disk errors or I/O retries
FLWait (FLMutexWaits)	20% or lower	> 20%	Check the dbspace configuration: Is the database almost out of space? Is DISK_STRIPING ON? Does sp_iqcheckdb report fragmentation greater than 15%?
HTWait (BmapHTNumWaits) MemWts (MemNtimes Waited) (PFMgrCond VarWaits)	10% or lower	> 10%	Contact Sybase Technical Support.

Statistic	Normal behavior	Behavior that needs adjusting	Recommended action
CPU time (CPU Sys Seconds, CPU Total Seconds, in debug report)	CPU Sys Seconds < 20%	CPU Sys Seconds > 20% If CPU Total Seconds also reports LOW utilization, and there are enough jobs that the system is busy, the cache may be thrashing or parallelism may be lost.	Adjust -iqgovern to reduce allowed total number of concurrent queries. Check Hit Rate and I/O Rates in the cache report for cache thrashing. Also check if hash object is thrashing by looking at the hit rate of the has object in cache_by_type (or debug) report: is it <90% while the I/O rate (Writes) is high? Check query plans for attempted parallelism. Were enough threads available? Does the system have a very large number of CPUs? Strategies such as multiplex configuration may be necessary.
InUse% (Buffers in use)	At or near 100% except during startup	Less than about 100%	The buffer cache may be too large. Try rebalancing the cache sizes of main versus temp by adjusting -iqmc and -iqtc.
Pin% (Pinned buffers)	< 90%	> 90 to 95%, indicating system is dangerously close to an Out of Buffers condition, which would cause transactions to roll back	Try rebalancing the cache sizes of main versus temp. If rebalancing buffer cache sizes is not possible, try reducing -iqgovern to limit the number of jobs running concurrently.

Statistic	Normal behavior	Behavior that needs adjusting	Recommended action
Free threads (ThrNumFree)	Free > Resrvd	If the number of free threads drops to the reserved count, the system may be thread starved.	Try one of the following: Increase the number of threads by setting -iqmt. Reduce thread-related options: MAX_IQ_THREADS_PER_CONNECTION, MAX_IQ_THREADS_PER_TEAM, PARALLEL_GBH_UNITS (for queries using Group By hash). Restrict query engine resource allocations by setting USER_RESOURCE_RESERVATION. Limit the number of jobs by setting -iqgovern.
FlOutOfSpace (debug only)	0, indicating that the free list for this store is not full; unallocated pages are available	1, indicating that this store (main or temporary) is fully allocated	Add more dbspace to that store

System utilities to monitor CPU use

Use these operating system utilities to monitor CPU usage while using Sybase IQ.

On UNIX systems use:

- ps command
- vmstat command (see example in the previous section)
- sar command (UNIX SystemV)

On Windows systems use:

- System Monitor
- Task Manager

CHAPTER 7

Tuning Servers on Windows Systems

About this chapter

This chapter provides performance and tuning guidelines specific to running Sybase IQ on Windows systems. Use this chapter in conjunction with Chapter 5, “Managing System Resources.”

Contents

General performance guidelines

The following are general guidelines that apply to both loading and querying data. The recommended minimum amount of memory (RAM) for running Sybase IQ under Windows is 512MB. While Sybase IQ will function correctly with smaller memory configurations, performance may be compromised.

Maximizing throughput

If you are running on Windows, make sure the Network Services Server option "Maximize Throughput for Network Applications" is enabled. Enabling this option ensures that the NTFS cache does not steal memory from Sybase IQ and cause excessive page faulting, particularly in memory-constrained environments.

❖ **To enable Maximize Throughput for Network Applications:**

1. From the Control Panel, double click on the Network Services Server icon.
2. Select the Services tab and double click on the Server network service.
3. Click on the option "Maximize Throughput for Network Applications".
4. Click OK twice and reboot the machine.

Preventing memory over allocation

Excessive system page faulting results from overallocating the physical memory (RAM) of the machine. Excessive page faults severely degrade the performance of Sybase IQ. By carefully allocating Sybase IQ buffers, monitoring the virtual address space of the Sybase IQ process(es), and monitoring available physical memory, you can prevent memory overallocation. This section offers guidelines for monitoring Sybase IQ use of your machine's physical memory.

Monitoring physical memory

The amount of physical memory available to applications (Sybase IQ) is displayed under Physical Memory (K). If the Available value is consistently below 5000 it is possible the physical memory for the machine is overallocated. This is because at the 5000(K) mark, Windows produces page faults in order to maintain a minimum of 5MB of free memory.

To monitor physical memory, from the Task Manager applet, select the Performance tab.

File systems

The Windows file system supports compression at the file, directory and volume level. *You should disable Windows file system compression for all disks and volumes on which you store* Sybase IQ *databases.* This is because Sybase IQ provides built-in compression. The file system compression will be unable to reduce the database size further, but may add CPU overhead when performing reads or writes.

Monitoring performance

Your primary tool for monitoring performance of Sybase IQ is the Sybase IQ performance monitor, described in Chapter 6, "Monitoring and Tuning Performance." However, you can also use operating system monitoring tools.

Windows provides two tools for monitoring the performance of your system.

Windows Task Manager has two windows that provide an easy-to-read overview of current system performance. To start the Task Manager type CTRL-ALT-DEL and select the Task Manager button. By selecting the Processes tab you can see a list of all the currently running processes on the system. To customize the columns displayed, select View > Select Columns. The CPU Usage, Memory Usage and Virtual Memory Size columns help you identify CPU or memory bottlenecks. The Performance tab allows you to see various counts and a history of the machine performance.

The System Monitor provides a more detailed analysis of your machine's performance. It allows you to monitor individual counters to various system and application objects, including processors, processes, disks and the network.

Monitoring virtual address space and working set

The virtual address space of a process is the total size of the process. The working set of a process is the amount of physical memory currently allocated to the process. In most cases, in order to avoid excessive system page faulting the virtual address space for the Sybase IQ process(es) should be less than the physical memory of the machine.

Due to the virtual memory usage pattern within the Sybase IQ server, virtual memory fragmentation could cause excessive process growth on Windows platforms. To reduce the likelihood of this situation, Sybase IQ supports the use of Microsoft's low-fragmentation heap (LFH) on Windows XP and Windows Server 2003.

❖ **To monitor virtual address space and working set:**

1. Start the Performance Monitor.
2. Click on the + icon and select the Process object.
3. Select the first Sybase IQ instance.
4. Select the counters Virtual Bytes and Working Set.

Monitoring page faults

From the Windows Performance Monitor select the Sybase IQ process as described above. Select the counter Page Faults/sec. This counter includes both the “soft” and “hard” page faults. Hard page faults are the page faults resulting in disk I/O. Soft page faults in general are not a performance issue.

To determine the number of hard page faults, select the object LogicalDisk and the instance of where the file *pagefile.sys* is located (this should be on a separate volume from the Sybase IQ database). Select the counter Disk Transfers/sec. This value when compared with the Page Faults/sec value will give an indication of the percentage of page faults that are hard page faults. Ideally there should be little to no I/O activity to the page file. In small memory configurations, however, paging is likely to occur.

Sustained hard page fault rates above 20 per second indicate that the physical memory of the machine has been overallocated.

Using the NTFS cache

With the Network Services Server option “Maximize Throughput for Network Applications” enabled, use of the NTFS and its associated cache can improve the performance of Sybase IQ for both inserts and queries. This is largely due to the NTFS being able to store significantly more data than the Sybase IQ buffer cache, with the same amount of physical memory, and the performance of the Intel Pentium to decompress Sybase IQ pages. As a result, when you use Sybase IQ on Windows platforms, you should reduce the size of the Sybase IQ buffer caches from their normal recommended settings.

The Sybase IQ main and temp buffer caches store Sybase IQ data (pages) in uncompressed form. As a result, a Sybase IQ buffer cache of 100MB can store 100MB worth of data. Conversely, the NTFS cache manages Sybase IQ data in its compressed form. Therefore, if the compression ratio were 2:1, 100MB of NTFS cache is potentially storing 200MB of Sybase IQ data. As a result, the NTFS cache is likely to sustain a higher cache hit rate which can lead to a reduction in I/O. The savings in I/O outweigh the computational overhead needed to decompress data as it moves from the NTFS cache to the Sybase IQ buffer caches.

Tuning inserts and queries

This section provides additional guidelines for tuning inserts and queries on Windows platforms.

Characteristics of well-tuned insert operations

A well-tuned Sybase IQ insert operation exhibits certain characteristics. You can observe these characteristics from the Windows Task Manager and Windows Performance Monitor.

- Insert operations are generally CPU-bound. All CPUs within the system should be running at close to 100%, with 95% or higher of the CPU being executed in user mode. You can see this easily by clicking on the Performance tab of the Windows Task Manager with the View-Show Kernel Times option set.

- Physical memory should not be overallocated and in particular, the virtual address space for the Sybase IQ process should be less than the physical memory (RAM) for the machine. On machines with large amounts of physical memory, that is, 512MB to 2GB, this will not be a problem. On machines with small amounts of memory, that is, 256MB or less, see the additional guidelines in the following section.
- Hard page faults (I/O to the volume containing *pagefile.sys*) should be low and ideally close to 0 (zero).
- I/O operations to the IQ Store should be steady and within the I/O capacity of the disk subsystem.

Sybase IQ uses the Windows CreateFile option (for both creating and opening a file) that specifies a file is to be read for sequential access. This option is used on the files specified in the LOAD TABLE command. As a result, load performance is improved through read ahead and reduced NTFS Cache memory utilization.

Load performance can be further improved, sometimes significantly, by setting the size of the main and temporary Sybase IQ buffer caches considerably smaller than the calculated recommended values in “Sybase IQ main and temp buffer caches” on page 120. The reasons for this performance improvement are described in “Using the NTFS cache” on page 195. You can set the main and temporary Sybase IQ buffer caches as much as 50% smaller than the calculated recommended values.

Tuning for queries

You may also improve query performance by reducing the size of the main and temp buffer caches as described in the previous paragraph. Be careful in a multiuser environment, however, because reducing the size of the temp buffer cache affects the various page pinning and sort algorithms which, in turn, may degrade performance. See Chapter 3, “Optimizing Queries and Deletions,” for details about query plans, structure, and options.

Tuning backup operations

Windows supports only fixed-length I/O devices. This means that each read or write to tape must be the same size as the one that preceded it and the one that follows. If any read/write operation exceeds the capacity of the hardware device, the operation fails. For backup and restore operations, this means that your backup (or restore) fails unless all of your writes (or reads) are the size the hardware is configured for.

The Sybase IQ defaults are designed to make your read and write operations as efficient as possible on each platform. However, if you override the default block size when you create a Sybase IQ database, you need to adjust the block factor when you back up that database.

For any backup or restore:

block size x *block factor* ≈ *I/O size*

To adjust the block factor on a Windows system, you must know the maximum physical block size that can be handled by your tape device. This information usually is not documented by the drive manufacturer. To determine the value, typically 64KB, you need to write a small applet using WIN32 API calls. You must then use the block size of the database and the BLOCK FACTOR option of the BACKUP command to optimize backup performance. For complete syntax and usage, see the *Sybase IQ Reference Manual*.

The closer to the maximum block size you can make each I/O operation, the better your backup performance will be. Use an integral BLOCK FACTOR that when multiplied by the block size yields as close to the drive's block size as possible.

Keep in mind that Sybase IQ adds some extra data to each block as it is written, for data integrity. So, if your database block size is 8192, and the maximum block size handled by the tape device is 128KB, you cannot use a block factor of 16, even though 8192 * 16 = 128KB. You have to account for the extra data added on each I/O operation by Sybase IQ and use a BLOCK FACTOR of 15. Note that 15 is the default block factor on Windows for the default database block size and the default IQ page size of 128KB.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W