SYBASE®

User's Guide

Risk Analytics Platform

3.0

DOCUMENT ID: DC00245-01-0300-01

LAST REVISED: September 2006

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

Contents

About This Book

Audience

Risk Analytics User's Guide is intended for Sybase® Professional Services, customer IT support, database and application development staff, and other technical personnel who need to set up and run Sybase Risk Analytics Platform. Familiarity with Sybase Adaptive Server® Enterprise, Sybase IQ, data warehousing, and other related topics is assumed.

How to use this book

Before following the instructions in this book to set up and run Risk Analytics Platform, be sure to complete the installation and configuration instructions in the *Risk Analytics Platform Installation and Configuration Guide*.

Related documents

Refer to the following documents for more information:

- *Release Bulletin Risk Analytics Platform*
- *Risk Analytics Platform Installation and Configuration Guide*
- *Risk Analytics Platform Administration Guide*
- Sybase IQ 12.6 product documentation
- Adaptive Server Enterprise 15.0 product documentation
- OpenSwitch 15.0 product documentation
- PowerDesigner® 11.1 product documentation
- Replication Server 12.6 product documentation
- White paper titled Time Series in finance: the array database approach at http://cs.nyu.edu/shasha/papers/jagtalk.html
- White paper titled FinTime --- a financial time series benchmark at http://www.cs.nyu.edu/cs/faculty/shasha/fintime.html

---

Note This product includes software developed by The Apache Software Foundation at http://www.apache.org/.

Other sources of information

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

- The Getting Started CD contains the release bulletin, installation and configuration guide, administration guide, and user's guide in PDF format. 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 Sybase Infocenter Web site is an online version of the product manuals that you can access using a standard Web browser.

  To access the Infocenter Web site, go to Sybooks Online Help at http://infocenter.sybase.com/help/index.jsp
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.

   Note The Product Family for Risk Analytics Platform 3.0 is Sybase IQ.
4. Click a Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

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

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.

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.

The Risk Analytics Platform 3.0 documentation complies 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 Chapter 1, “Introducing Sybase IQ” in *Introduction to Sybase IQ*. The online help for Sybase IQ, which you can navigate using a screen reader, also describes accessibility features, including Sybase Central keyboard shortcuts.

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

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

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

Sybase Risk Analytics Platform

About this Chapter

This chapter provides an overview of Sybase Risk Analytics Platform.

Contents

Overview

The Sybase Risk Analytics Platform (RAP) is a consolidated trading and risk data repository and data services platform for customers in the Capital Markets and Investment Management sectors. By consolidating market data from vendor feeds, historical time series data, real-time trades and quotes (TAQ) data, and reference data in one repository, the Risk Analytics Platform eliminates or reduces intraday and overnight batch processing and supports model-driven quantitative trading and real-time portfolio decisions by presenting subsets of data to various applications.

Risk Analytics infrastructure

The Risk Analytics Platform stores large amounts of historical, reference, and real-time data from corporate sources, market data vendors, and securities exchanges for fast access by automated trading applications and various user communities with analytic needs in customer organizations. The real-time data streams can be inserted directly into the cache database. Both the real-time data and scheduled downloads from market data vendors can be loaded into the repository very quickly using loading utilities at very frequent intervals (within seconds of delivery) to keep the data current. RAP 3.0 includes a set of load scripts to help automate the data loading process for real-time data that has been batched and transformed into a flat file format. The VLDBServer database can also be used in conjunction with Replication Server (not included in the RAP product), so that transactional updates can be applied to the RAP data.

RAP incorporates a data model designed for multi-asset portfolio trading applications and includes a tool for managing physical data models. The scripts for configuring the repository for high performance, test data, and performance tuning procedures are also included.

RAP version 3.0 incorporates an in-memory cache database. This feature provides real-time access to streaming market data.

Why RAP?

Due to the increase in TAQ data volume and data flow rates, conventional relational database management systems (RDBMS) cannot meet the real-time query requirements for automated securities trading and real-time market analysis. Sybase Risk Analytics Platform 3.0 is designed to meet this challenge.

RAP is designed to consolidate risk data, reference data, and real-time and historical trade data in one repository for presenting the data to different application environments in real-time. The product has a RAM-resident cache database, which is configured to capture streaming data, and a disk-based repository for storing historical data. The capacity of the historical repository reaches into the petabyte range. The VLDB repository is a column-based data store and is capable of scaling to a high number of concurrent user connections.

VLDBServer database

The VLDBServer database uses a vectorial representation of data and supports multi-user and multi-application workloads by scaling up as well as scaling out in multi-processor, clustered configurations. RAP captures real-time data flow in both the cache database and the repository and maintains a cached-copy and a disk-based copy; the repository stores the union of both the historical and intra-day data as one complete time series. The latency of the disk-based repository relative to the cache is kept in seconds.

RAP supports different applications, such as pre-trade analysis, post-trade analysis, quantitative modeling, and scenario-based back testing against a shared database, and distributes the query workload across the cache and the repository based on the time-criticality of the data access requirements of each user application.

The design is based on the requirements of high performance and concurrent retrievals by a large user population, as opposed to the design needs of a high rate of concurrent updates as in the case of RDBMS. The internal data structure and the way data is served to user applications renders the VLDBServer particularly suitable for storing large amounts of time series data. The next-generation automated trading infrastructure in institutional trading and prime brokerage firms is characterized by a limited number of inbound data streams (that represent market data delivery channels) and a high number of concurrent reader processes that access indexed columns to retrieve large data sets for analysis. The architecture of the Risk Analytics Platform meets these requirements.

RAPCache database

RAP stores current market data in an in-memory cache database. This allows for extremely fast load and retrieval times. In addition, using this data store for current market data provides a separate memory space for the specialized use of traders who need up-to-the-second information, but may not need the full historical data held in the VLDBServer database.

For those who need both current and historical data to complete their analyses, the repository contains all data. Quantitative analysts can access both real-time and historical data for trend analysis.

Data model

The Risk Analytics Platform uses a data model designed to support institutional trading and prime brokerage business processes. The Risk Analytics data model includes two major submodels, which focus on two specific business areas: **Instrument** and **Market Data**.

Instrument

The Instrument submodel contains data structures that represent financial instruments. The Instrument table stores information common to all financial instruments, including instrument name, trading symbol, issue date, issuer rating, trading currency, and exchange.

Corresponding tables store details about each type of instrument. Thus, stock-related information is stored in the Stock (STOCK) table; information related to bonds is stored in the Bond (BOND) table, and so on.

Market Data

The Market Data submodel contains data structures that represent historical and real-time data. The submodel includes several areas corresponding to different financial instruments; each area includes one or more tables storing historical or real-time (intraday) data.

Note
For more information about the data model included with Sybase Risk Analytics Platform, see Chapter 2, “Data Model.”

Sample queries

Risk Analytics Platform includes SQL scripts that allow you to run queries against the sample data. You can run the scripts against the sample data to evaluate the retrieval performance for historical or real-time (intraday) data.

The historical data sets are built using end-of-day trading data files from market data vendors and represent several months or years of data. The real-time TAQ data is the tick-by-tick bid/ask quotes and trade prices for each trading day. The historical time series queries analyze securities trades over long time periods to identify trends. Real-time queries analyze the intraday dynamics of securities trading and seek to identify arbitrage opportunities and optimal trading strategies. The interday queries span both real-time and historical information and can provide an alert for a change to statistical models that occurs during the course of the trading day.

Note
For more information about the sample queries included with Sybase Risk Analytics Platform, see Chapter 3, “Sample Queries.”

CHAPTER 2

Data Model

About this Chapter

This chapter describes the Risk Analytics Platform data model and the tables in the model.

Contents

Overview

The Sybase RAP data model supports the storage and fast retrieval of historical and real-time market data, reference information on financial instruments, and information on market indexes used as benchmarks in performance analysis.

The RAP package includes two similar physical data models stored in PowerDesigner: one for the VLDBServer database and another for the RAPCache database. The only difference between the models lies in the indexes that are database-specific and must be defined individually for each target database. All information provided below is applicable to each of the models that, for the purpose of this description, are both referred to as RAP data model, or model.

You can open and view the data models using PowerDesigner 11.1, which is a component of Risk Analytics Platform.

Model description

This section describes the RAP data model.

- The RAP model includes three diagrams (submodels): RAP, Instrument, and Market Data. Detailed descriptions of each submodel are provided below. No packages are defined in the model.
- The entire model contains 51 tables and 271 columns; each table is shown in at least one submodel. All tables and columns have detailed descriptions.
- The data types are assigned to table columns via domains; there are 16 domains in the model.
- All primary keys and foreign key constraints are named in accordance with standard naming conventions, that is, PK_<table name> and FK*nn*_<table name> for a primary key and foreign key constraint, respectively (where *nn* is the constraint number).

  In a similar manner, model indexes are named as XK*nn*_<table name> (where *nn* is the index number). Domain names are defined as SYB_<domain name>.
- Only indexes needed to support the sample queries are included in the model. The VLDBServer data model contains 64 indexes; the RAPCache data model contains 8 indexes. More indexes can be added at the customer site to meet the specific requirements related to the data load and/or query performance.

Submodels

This section describes the Risk Analytics Platform submodels.

RAP submodel

The RAP submodel is a default diagram that is shown to the user when the model is opened. The RAP submodel provides the model name, version, and copyright information.

Instrument submodel

The Instrument submodel contains data structures that represent financial instruments. The Instrument (INSTRUMENT) table stores information common to all financial instruments.

- Detailed information on each type of instrument represented in the model (that is, stocks, bonds, mutual funds, Exchange-Traded Funds (ETF), and options) is stored in corresponding tables. Thus, stock-related information is stored in the Stock (STOCK) table, bond-related information is stored in the Bond (BOND) table, and so on.
- The association table Instrument Exchange (INSTR_EXCHANGE) is used to specify exchanges where an instrument is traded. The Instrument Rating (INSTR_RATING) table stores information on the ratings assigned to an instrument by rating agencies (for example, Moody's Investor Service, Standard & Poor's Corporation, Fitch Ratings.).
- Information on market indexes is stored in the Market Index (MARKET_INDEX) table. The Index Composition (INDEX_CMPST) table stores information on an index composition and is used to specify all instruments included in an index. The Instrument Benchmark (INSTR_BENCHMK) table specifies a market index (or indexes) used as a benchmark for a given financial instrument.
- The association tables Underlying Index (ULYING_INDEX) and Underlying Instrument (ULYING_INSTR) are used, respectively, to specify an underlying index or an underlying financial instrument for an option.

Market Data submodel

The Market Data submodel represents data structures that store historical and real-time data. The submodel contains several areas, including the historical data and quotes and trades data for different financial instruments.

- Stock History (STOCK_HISTORY) stores historical data (one record per trading date) including open/close price, high/low price, and trading volume.
- Stock Quote (STOCK_QUOTE) stores intraday quote data including bid/ask price and size.
- Stock Trade (STOCK_TRADE) stores intraday trade data including trade price and size.
- Split Event (SPLIT_EVENT) stores data on stock splits (event date and split factor).
- Dividend Event (DIVIDEND_EVENT) stores data on dividend payment events (event date and dividend amount).

Note
The stock-related tables listed above are also used to store the data of Exchange-Traded Funds.

Separate tables with a similar structure are used to store historical and intraday data related to bonds and options. Mutual funds data contains only historical data stored in the Mutual Fund History (MUTL_FUND_HIST) table.

Index History (INDEX_HISTORY) and Index Intraday (INDEX_INTRADAY) tables are used to store historical and intraday data on market indexes.

Data model tables

The following table lists all data model tables, the code names, and descriptions:

Table name	Code	Description text
Currency	CURRENCY	This table contains a list of world currencies based on International Organization for Standards (ISO) publication 4217 (e.g., US Dollar, Hong Kong Dollar, etc).
Instrument	INSTRUMENT	This table stores the financial instruments data that is common to all types of instruments (e.g., trading symbol, name, date of issue, etc). Data that is specific to a particular type of instruments (stock, bond, option, mutual fund, etc) is stored in separate data structures; e.g., stock-specific data is stored in the Stock (STOCK) table.
Major Industry Classification	MAJOR_IDST_CLS	This table contains a list of definitions used to characterize an industry of a company (e.g., Technology, Energy, Healthcare, etc). Definitions are based on the Standard Industry Classification (SIC).
Stock History	STOCK_HISTORY	This table stores the stocks historical data, one record per each trading date. The data includes stocks daily prices (open/close, high/low) and trade volume (number of shares traded).
Exchange	EXCHANGE	This table stores a list of exchanges where financial instruments are listed and traded (e.g., New York Stock Exchange, NASDAQ, etc).
Instrument Type	INSTR_TYPE	This table stores a list of definitions used to specify a type of a financial instrument (e.g., stock, bond, option, mutual fund, ETF, etc).
Dividend Event	DIVIDEND_EVENT	This table stores information on a dividend payment event when a shareholder receives a certain payment for each share of stock in his/her possession. The dividend amount is commonly defined as a certain percentage of a share price but can be also specified as a monetary amount. Monetary or Percentage Indicator (MOP_INDICATOR) column indicates how the dividend amount is defined.

Table name	Code	Description text
Split Event	SPLIT_EVENT	This table stores information on a stock split event when the number of outstanding shares of a company's stock is increased and the price per share is simultaneously decreased so that proportionate equity of each shareholder remains the same. The split is characterized by a split factor; a factor of 0.5 indicates that the number of shares is increased two times and that the share price is decreased two times. In a less common reverse split, the number of shares is decreased and the price per share is increased in a similar manner; a split factor of 2 indicates that the number of shares is decreased two times and that the share price is increased two times.
Stock Trade	STOCK_TRADE	This table stores the stocks real-time (intraday) trade data. Each trade record includes a transactions price and size (i.e., a number of shares traded).
Stock Quote	STOCK_QUOTE	This table stores the stocks real-time (intraday) quote data. Each quote record includes a bid/ask price and corresponding size values (i.e., a number of shares offered at bid/ask price).
Mutual Fund	MUTUAL_FUND	This table stores the mutual funds data including a fund type (stocks, bonds, hybrid), fund family (e.g., Fidelity), investment objective (e.g., grows and income), expenses, sale load, etc. Funds attributes that are common to all types of financial instruments (trading symbol, name, currency, etc) are stored in the Instrument (INSTRUMENT) table.
Fund Type	FUND_TYPE	This table stores a list of definitions used to characterize a mutual fund based on a type of its financial instruments - stock fund (stocks), bond fund (bonds), hybrid fund (stocks and bonds), etc.
Option Instrument	OPTION_INSTR	This table stores the options data including an option type (put or call), option category (a type of an underlier), strike price, etc. Option attributes that are common to all types of the financial instruments (i.e., trading symbol, name, currency, etc) are stored in the Instrument (INSTRUMENT) table.
Option Type	OPTION_TYPE	This table stores a list of definitions used to specify a type of an option contract - put or call.

Table name	Code	Description text
Rating Score	RATING_SCORE	This table stores a list of scores that are assigned by rating agencies to issuers of financial instruments to characterize their creditworthiness. Thus, rating scores assigned by Standard & Poors range from AAA (premium) to D (default).
Secondary Industry Classification	SCND_IDST_CLS	This table stores a list of definitions that are used together with major industry classifications (see Major Industry Classification table) to further categorize an industry of a company. Thus, a company with a major classification Technology can be further categorized as Software, Hardware, etc. Definitions are based on the Standard Industry Classification (SIC).
Instrument Rating	INSTR_RATING	This association table is used to specify rating scores assigned to an issuer of a financial instrument by different rating agencies.
Fund Family	FUND_FAMILY	This table stores a list of mutual fund families (e.g., Fidelity, T. Rowe Price, Vanguard, etc). A fund family is a company offering many mutual funds, for various objectives.
Capitalization	CAPITALIZATION	This table contains a list of definitions that are used to specify a type of a market capitalization of a financial instruments issuer (e.g., Small-Cap, Medium, Large).
Share Series	SHARE_SERIES	This table stores a list of definitions used to specify a series (class) of mutual fund shares. Shares series indicates whether they carry commissions (sales load) and when these commissions must be paid. Thus, A shares carry a front-end load that must be paid when shares are bought; B shares carry back-end load that must be paid when shares are sold; C shares have no commissions but carry an ongoing fee (12-b fee) that is paid annually in addition to other fund-related expenses; etc.
Rating Agency	RATING_AGENCY	This table stores a list of agencies that collect information about the creditworthiness of issuers of financial instruments and assign to them a corresponding rating (credit score). Three major rating agencies are Moodys Investor Service, Standard & Poors Corporation and Fitch Ratings.

Table name	Code	Description text
Geographic Group	GEO_GROUP	This table contains a list of definitions used to group financial instruments by a geographical region of their issuers. Terms commonly used in US are: Domestic (US issuers), International (non-US issuers), Global (can include both domestic and international issuers), Europe (Europe-based issuers), etc.
Country	COUNTRY	This table contains a standard list of the world countries (e.g., USA, Japan, France, etc).
Instrument Benchmark	INSTR_BENCHMK	This association table specifies a market index that is used as a benchmark for a given financial instrument. More than one benchmark can be used for some instruments.
Index Composition	INDEX_CMPSTN	This association table is used to specify all financial instruments that constitute a market index. Thus, Dow Jones Industrial Average index is based on a stock valuation of the thirty major US corporations that are included in this index.
Stock	STOCK	This table stores the data on stocks, e.g., stock type (common stock, preferred stock, etc), dividend amount, number of shares outstanding, etc. Stocks attributes that are common to all types of financial instruments (trading symbol, name, currency, etc) are stored in the Instrument (INSTRUMENT) table.
Stock Type	STOCK_TYPE	This table stores a list of definitions used to specify a type of a stock, e.g., common stock, preferred stock, etc.
Stock Subtype	STOCK_SUBTYPE	This table stores a list of definitions that are used to categorize stocks of a particular type. Thus, a preferred stock can be categorized as cumulative, non-cumulative, participating and convertible.
Exchange Traded Fund	EXCH_TRD_FUND	ETF This table stores the Exchange Traded Funds (ETF) data. ETF attributes that are common to all types of financial instruments (trading symbol, name, currency, etc) are stored in the Instrument (INSTRUMENT) table.
Fund Category	FUND_CATEGORY	This table stores a list of definitions used to characterize an investment style of a mutual fund (e.g., Value, Sector, Growth, etc).
Investment Objective Type	INVST_OBJ_TYPE	This table stores a list of definitions used to characterize investment goals of a mutual fund (e.g., Capital Appreciation, Income, Income and Growth, etc).

Table name	Code	Description text
Bond	BOND	This table stores the bonds' data (e.g., bond type, maturity date, interest rate, etc). Bonds attributes that are common to all types of financial instruments (trading symbol, name, currency, etc) are stored in the Instrument (INSTRUMENT) table.
Bond Type	BOND_TYPE	This table stores a list of definitions used to specify a type of a bond (e.g., US Treasury, Municipal, Corporate, etc).
Bond Subtype	BOND_SUBTYPE	This table stores a list of definitions that are used to categorize bonds of a particular type. Thus, US Treasury issues can be categorized as Treasury Bonds, Zero-Coupon Bonds, Treasury Notes, etc.
Maturity Term Type	MTRTY_TERM_TYPE	This table stores a list of definitions used to specify a type of the bonds maturity term (e.g., short-term, intermediate, long-term, etc).
Payment Frequency Type	PYMT_FRQ_TYPE	This table stores a list of definitions used to specify a frequency of interest payments associated with a bond (annually, semi-annually, quarterly, etc).
Index History	INDEX_HISTORY	This table stores the index's historical data, one record per each trading date. The data includes the index's daily values (open/close, high/low) and trade volume.
Index Intraday	INDEX_INTRADAY	This table stores the index's real-time (intraday) data that shows its value movements during a trading day. Each data point includes an index value and trade volume.
Option History	OPTION_HISTORY	This table stores the options historical data, one record per each trading date. The data includes options daily price (open/close, high/low), trade volume (number of contracts traded), etc.
Option Quote	OPTION_QUOTE	This table stores the options real-time (intraday) quote data. Each quote record includes a bid/ask price, size (number of contracts offered at a bid/ask price), etc.
Option Trade	OPTION_TRADE	This table stores the options real-time (intraday) trade data. Each trade record includes a trade's price, size (number of contracts traded), etc.
Bond History	BOND_HISTORY	This table stores the bonds historical data, one record per each trading date. The data includes bonds daily price and yield values (open/close, high/low), trade volume (number of bonds traded), etc.
Bond Trade	BOND_TRADE	This table stores the bonds real-time (intraday) trade data. Each trade record includes a bonds price and yield and a transactions size (number of bonds traded).

Table name	Code	Description text
Bond Quote	BOND_QUOTE	This table stores the bonds real-time (intraday) quote data. Each quote record includes a yield, bid/ask price and size (i.e., a number of bonds offered at a bid/ask price).
Mutual Fund History	MUTL_FUND_HIST	This table stores the historical data for a mutual fund, one record per each trading date. The data includes a trade date and price.
Market Index	MARKET_INDEX	This table stores a list of market indexes (e.g., Dow Jones Industrial Average, S 500, NASDAQ Composite, etc) that are used in analysis of market trends, as benchmarks, etc.
Instrument Exchange	INSTR_EXCHANGE	This association table is used to specify an exchange where a given financial instrument is listed and traded. Note that some instruments can be listed on more than one exchange.
Fund Share	FUND_SHARE	This table stores the data on mutual fund shares of a particular series (class) including a sales load, fee (12-b fee), etc. Fund attributes that are common to all shares (fund type, family, investment objective type, etc) are stored in the Mutual Fund (MUTUAL_FUND) table.
Option Category	OPTION_CATEGORY	This table stores a list of definitions used to specify a category of an underlier that the characteristics of an option depend upon (e.g., a bond, stock, market index, currency, etc).
Underlying Instrument	ULYING_INSTR	This association table is used to define a financial instrument (underlier) that an option is based on (e.g., stock, bond, etc).
Underlying Index	ULYING_INDEX	This association table is used to define a market index (underlier) that an option is based on.

CHAPTER 3

Sample Queries

About this Chapter

This chapter describes the sample queries included with Risk Analytics Platform.

Contents

Overview

Risk Analytics Platform includes sample queries for historical market and TAQ data. These queries are packaged as SQL scripts and are located in subdirectories of the *$RAP30/Scripts/* directory. Throughout this chapter, the environment variable $RAP30 refers to the RAP 3.0 installation directory.

Appendix A, “SQL Scripts for Sample Queries” contains the SQL scripts for the Risk Analytics Platform sample TAQ, Interday, and Historical market data queries. The results returned by running the queries with the sample RAP data are also included in this appendix.

Note If you intend to run the queries from a client machine, your system administrator must copy the queries from the server to a target directory on the machine where the appropriate client tools are installed.

Running the sample queries

The TAQ data queries run on RAPCache and VLDBServer. The Interday and Historical market data queries are optimized to run on VLDBServer. You can run the TAQ query scripts against the cache database and the Interday and Historical market data query scripts against VLDBServer using Interactive SQL: isql for the RAPCache database and dbisql for VLDBServer.

RAPCache

Use Interactive SQL (isql) to run the TAQ data query scripts on your RAPCache database. isql sends Transact-SQL commands to Adaptive Server Enterprise, formatting the results and printing them to standard output. There is no maximum size for an isql statement. To use Transact-SQL directly from the operating system with the isql utility program, you must have an Adaptive Server Enterprise account or login.

For more information on running queries with Interactive SQL, see Chapter 2, "Using the isql Utility" in the *Adaptive Server Enterprise Utility Guide*.

VLDBServer

Use Interactive SQL Java (dbisql) to run the Interday and Historical market data query scripts on your VLDBServer database.

Interactive SQL (dbisql) is an application that allows you to type a SQL statement and send it to a database. Because interactions with databases use SQL statements, you can carry out any database operation from dbisql.

Which version of Interactive SQL should I use?

Although both Interactive SQL Classic (dbisqlc) and Interactive SQL Java (dbisql) are included, Sybase recommends that you use Interactive SQL Java to run the queries on VLDBServer.

The Force_No_Scroll_Cursors option can make a significant difference in reducing query execution time and should be set ON. If you use Interactive SQL Classic, when you scroll through the results you may get an error that says `"When Force_No_Scroll_Cursors=ON, scrolling cursor operations are not supported by Sybase IQ."` You can ignore this error.

For more information on running queries with Interactive SQL, see Chapter 2, "Using Interactive SQL (dbisql)" in the *Sybase IQ Utility Guide*.

TAQ data queries

Trades and Quotes (TAQ) data tables include real-time price quotes and trade prices that are updated frequently during a trading day. Queries against these tables use intraday price and quote fluctuations.

The tick queries are representative of the query workloads generated in pre-trade analysis and trade order generation. Although this is a partial list of possible queries, these queries constitute a reasonable sample test for use in performance and tuning analysis and also as a template to build a library of queries. The sample TAQ queries can also be modified to build a native T-SQL access layer to present data to computational applications.

The TAQ data queries are optimized to run on both the RAPCache database and the VLDBServer database.

Note The TAQ data queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede the query with a commit statement.

Script files

The TAQ data query scripts are located in subdirectories of the *$RAP30/Scripts* directory on the server. See the table below for a description of each script file.

Script name	Description
tick_qry1.sql	Get all ticks for a specified set of 100 securities for a specified three-hour time period on a specified trade date.
tick_qry2.sql	Determine the volume-weighted price of a security considering only the ticks in a specified three-hour interval.
tick_qry3.sql	Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.
tick_qry4.sql	Determine the top 10 most active stocks for a specified date, sorted by cumulative trade volume, by considering all trades.
tick_qry5.sql	Find the most active stocks in the COMPUTER industry (use SIC code).
tick_qry6.sql	Find the 10 stocks with the highest percentage spreads. Spread is the difference between the last ask-price and the last bid-price. Percentage spread is calculated as a percentage of the bid-point price (average of ask and bid price).

Note An additional SQL script setup_tick_qry3_last_price.sql should be run on the RAPCache database at the end of each trading day to capture the last price of that day. This data is referenced by tick_qry3.sql. For more information, see “setup_tick_qry3_last_price.sql” on page 46.

Tick query examples

Query description: tick_qry4 Determine the top 10 most active stocks for a specified date, sorted by cumulative trade volume, by considering all trades.

```
TRADING_SYMBOL  TRADESIZE
--------------- -----------
ASU                 2932300
BEG                 2929000
AYD                 2923600
AJA                 2884400
AVQ                 2874300
AAC                 2856400
AKC                 2854900
ACN                 2851800
AQL                 2834100
AFE                 2821600
```

Query description: tick_qry6 Find the 10 stocks with the highest percentage spreads. Spread is the difference between the last ask-price and the last bid-price. Percentage spread is calculated as a percentage of the bid-point price (average of ask and bid price).

```
TRADING_SYMBOL  PER
--------------- ----------------------------------
ACG                      0.027027027027027027027027
ADC                      0.018248175182481751824817
AFZ                      0.018018018018018018018018
BBN                      0.017654476670870113493064
ASS                      0.017431725740848343986054
AUZ                      0.016806722689075630252100
BGW                      0.016051364365971107544141
AKF                      0.015804597701149425287356
AUB                      0.012652889076339097427245
AZY                      0.012628255722178374112075
```

Interday queries

Interday queries reflect price quotes and trade prices across multiple trading days. These queries examine the tick data of the current day plus historical data and can provide an alert for changes to statistical models during the course of the trading day.

Interday queries are optimized for RAP, as these queries require data that spans several days and weeks. Use Interactive SQL Java (dbisql) to run these scripts against theVLDBServer.

For more information on running queries with Interactive SQL, see Chapter 2, "Using Interactive SQL (dbisql)" in the *Sybase IQ Utility Guide*.

Note The Interday queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede each query with a commit statement.

Script files

The Interday query scripts are located in the *$RAP30/Scripts/VLDB* directory on the server. See the table below for a description of each script file.

Script name	Description
interday_tick_qry1.sql	Determine the volume-weighted price of a security considering only the ticks in a specified three-day interval.
interday_tick_qry2.sql	Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.
interday_tick_qry3.sql	Find the most active stocks in the "COMPUTER" industry for the last three days.

Interday query examples

Query description: interday_tick_qry1 Determine the volume-weighted price of a security considering only the ticks in a specified three-day interval.

```
TRADING_SYMBOL   VOLUME_WEIGHTED_PRICE
AAA                   49.6641081572369
```

Query description: interday_tick_qry3 Find the most active stocks in the "COMPUTER" industry for last three days.

INSTRUMENT_ID	TRADING_SYMBOL	TRADESIZE	RANKING
289	ALD	554400	1
493	ASZ	538700	2
850	BGS	529400	3
346	ANI	519300	4
866	BHI	516700	5
394	APE	504600	6
44	ABS	502800	7
400	APK	492000	8
360	ANW	490500	9
560	AVO	488800	10
507	ATN	485800	11
886	BIC	484600	12
752	BCY	484400	13
616	AXS	477500	14
356	ANS	477200	15
487	AST	477000	16
80	ADC	475700	17
230	AIW	474200	18
980	BLS	466700	19
810	BFE	464400	20
588	AWQ	463400	21
771	BDR	463000	22
930	BJU	462800	23
460	ARS	462300	24
773	BDT	458400	25
...			

Historical market data queries

Historical market data queries compare price histories for different instruments over time. Historical market data does not change frequently, and updates typically occur at the end of the trading day.

Historical data queries are optimized for RAP, as they require the content of VLDBServer. You can run these sample queries against VLDBServer with Interactive SQL (dbisql).

For more information on running queries with Interactive SQL, see Chapter 2, "Using Interactive SQL (dbisql)" in the *Sybase IQ Utility Guide*.

Note The Historical market data queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede each query with a commit statement.

Script files

Historical data query scripts are located in the *$RAP30/Scripts/VLDB* directory on the server. See the table below for a description of each script file.

Script name	Description
hist_qry1.sql	Get the closing price of a set of 10 stocks for a 10-year period and group into weekly, monthly, and yearly aggregates. For each aggregate period, determine the low, high, and average closing price value. Output is sorted by TRADING_SYMBOL and trade date.
hist_qry2.sql	Adjust all prices and volumes (prices are multiplied by the split factor and volumes are divided by the split factor) for a set of 1000 stocks to reflect the split events during a specified 300-day period, assuming that events occur before the first trade of the split date. These are called split-adjusted prices and volumes.
hist_qry3.sql	For each stock in a specified list of 1000 stocks, find the differences between the daily high and daily low on the day of each split event during a specified period.
hist_qry4.sql	Calculate the value of the S&P 500 and Russell 2000 index for a specified day using unadjusted prices and the index composition of the two indexes on the specified day.

Script name	Description
hist_qry5.sql	Find the 21-day and 5-day moving average price for a specified list of 1000 stocks during a 6-month period. (Use split-adjusted prices.)
hist_qry6.sql	(Based on the previous query.) Find the points (specific days) when the 5-day moving average intersects the 21-day moving average for these stocks. Output is sorted by TRADING_SYMBOL and trade date.
hist_qry7.sql	Determine the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (That is, allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average, the complete allocation for that stock is invested and when the 20-day moving average crosses below the 5-month moving average, the entire position is sold. The trades are made on the closing price of the trading day.
hist_qry8.sql	Find the pair-wise coefficients of correlation in a set of 10 securities for a two-year period. Sort the securities by the coefficient of correlation, indicating the pair of securities corresponding to that row.
hist_qry9.sql	Determine the yearly dividends and annual yield (dividends/average closing price) for the past 3 years for all the stocks in the Russell 2000 index that did not split during that period. Use unadjusted prices since there were no splits to adjust for.

Historical market query examples

Query description: hist_qry1 Get the closing price of a set of 10 stocks for a 10-year period and group into weekly, monthly, and yearly aggregates. For each aggregate period, determine the low, high, and average closing price value. The output is sorted by TRADING_SYMBOL and trade date.

TRADING_SYMBOL	YEAR	MON	WEEK	MAX_PRICE	MIN_PRICE	AVG_PRICE
AAA	2005	2	7	26.24	25.48	25.925
AAA	2005	2	8	26.49	25.46	25.972
AAA	2005	2	9	27.28	26.48	26.746
AAA	2005	2	10	26.47	26.47	26.47
AAA	2005	2	(NULL)	27.28	25.46	26.25066667
AAA	2005	3	10	26.46	25.93	26.2625
AAA	2005	3	11	26.19	24.9	25.566
AAA	2005	3	12	26.16	25.38	25.792
AAA	2005	3	13	25.38	24.37	24.924
AAA	2005	3	14	25.61	25.11	25.36
AAA	2005	3	(NULL)	26.46	24.37	25.56086957
AAA	2005	4	14	25.87	25.87	25.87
AAA	2005	4	15	26.39	25.34	25.866
AAA	2005	4	16	26.1	25.07	25.634
AAA	2005	4	17	25.81	24.81	25.314
AAA	2005	4	18	27.94	26.59	27.284
AAA	2005	4	(NULL)	27.94	24.81	26.01714286
AAA	2005	5	19	27.1	26.55	26.828
AAA	2005	5	20	27.08	26.29	26.71
AAA	2005	5	21	27.08	26.54	26.862
AAA	2005	5	22	28.45	27.07	27.836
AAA	2005	5	23	27.88	27.6	27.74
AAA	2005	5	(NULL)	28.45	26.29	27.12090909
AAA	2005	6	23	27.05	26.24	26.69

...

Note The hist_qry1 query uses the ROLLUP operator. The NULL values for month and week are subtotals. A NULL value in the week column is a subtotal for the month. A NULL value in the month and week columns is a subtotal for the year.

Query description: hist_qry4 Calculate the value of the S&P 500 and Russell 2000 index for a specified day using unadjusted prices and the index composition of the two indexes on the specified day.

```
INDEX_NAME       AVERAGE_CLOSE_PRICE
Russell 2000        49.47026052
S&P 500             54.44644
```

Query description: hist_qry7 Determine the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (that is, allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average, the complete allocation for that stock is invested, and when the 20-day moving average crosses below the 5-month moving average, the entire position is sold. The trades are made on the closing price of the trading day.

```
STOCK_VALUE
289690.0039
```

CHAPTER 4

Generating DDL Scripts

About this Chapter

This chapter tells you how to generate the Data Definition Language (DDL) statements to create database objects from the Risk Analytics data models for the VLDBServer and RAPCache databases.

Contents

Overview

Risk Analytics Platform includes separate data models for the VLDBServer and RAPCache databases. Although these data models target different databases, they share an identical data structure.

Depending on your business environment, you may need to create additional tables or columns. If you modify the data model, the RAPCache database schema must match the VLDBServer schema, if both databases are targets of the same data loading process, or support the same queries, or both. After you make your changes, you can use PowerDesigner to produce a set of data definition language (DDL) statements directly from the data model. PowerDesigner saves the DDL in a SQL script that you can run to generate the tables and other objects for the target databases.

Note
You need to install Sybase PowerDesigner 11.1 on Windows before you generate DDL for the Risk Analytics databases. You can also use PowerDesigner to view and update the data model.

For information on installing PowerDesigner, see "Installing PowerDesigner PhysicalArchitect" in Chapter 2, "Installing RAP Core Components" of the *Risk Analytics Platform Installation and Configuration Guide*.

For information on using PowerDesigner, refer to the PowerDesigner 11.1 product documentation.

Risk Analytics Platform includes the DDL scripts you need to create database objects in both your RAPCache database (an Adaptive Server Enterprise database) and your VLDBServer database (a Sybase IQ database). The instructions in this chapter are optional, unless you customize the data models. In this case, the following instructions guide you through the PowerDesigner DDL generation process for both the VLDBServer and RAPCache databases.

Note
Your system administrator must copy the data models from the subdirectories of *$RAP30/Model* on the server to a target directory on the Windows machine where PowerDesigner is installed. As a convention, this document refers to the model directories on the target machine as *Model\VLDB* and *Model\RAPCache*. The environment variable $RAP30 refers to the RAP 3.0 installation directory.

Generating database schema with PowerDesigner

PowerDesigner includes all the resources you need to generate a set of data definition language (DDL) statements in SQL scripts directly from the Risk Analytics data models. You can run these scripts to generate a schema for your VLDBServer and RAPCache databases.

To generate DDL from a data model:

- Open the data model in PowerDesigner.
- Change the default database user.
- Generate the script that creates a schema for the new database.
- Log in to the database and run the script.

Procedures for the VLDBServer and the RAPCache databases vary. Refer to the appropriate section for specific instructions.

VLDBServer database

Follow these instructions to generate DDL for VLDBServer, which is a Sybase IQ database. The data model for VLDBServer is *RAP_IQ.pdm*, which is located in the *Model\VLDB* folder.

Changing the default database user

In the database, the user who creates an object (table, view, stored procedure, and so on) owns that object and is automatically granted all permissions on it. Risk Analytics data models ship with a default database owner named RAP_USER.

You may want to overwrite the default database owner with a name specific to your environment. Overwriting the default user name globally changes ownership of database objects from the default owner to the new owner.

1 Start PowerDesigner 11.1.

2 Click File | Open | Choose *RAP_IQ.pdm*.

3 Click Model | Users and Roles | Users.

4 In the Name column, change the default user (RAP_USER) to the new database user.

5 Click Apply.

Generating a DDL script

After you change the default database user, you can generate DDL directly from the data model. PowerDesigner saves the results in a SQL script that you use to generate the tables and other objects in the target database.

Note
Use the model *RAP_IQ.pdm* to generate DDL for the VLDBServer database. Do *not* use a different model by changing the target database to IQ, as this will result in the loss of index information.

1 Click Database | Generate Database.

2 On the Database Generation dialog, click the browse button and choose the directory where you want to store the script. Click OK.

3 In the File name box, type a name for the SQL script. You will use this script in the next procedure.

4 On the Tables & Views, Keys & Indexes, Database, and Options tabs, set options as listed in the following table:

On this tab...	In this panel...	Set these options...
Tables & Views	Tables	Create table
Keys & Indexes	Primary keys	Create primary key Inside Table
	Indexes	Create index
	Foreign keys	Create foreign key Outside Declarative Integrity
Database		Turn all options off
Options		Accept all defaults

Note If you select User-defined type in the Columns panel of the Tables & Views tab, you *must also* select Create data type in the User-defined data types panel of the Database tab. PowerDesigner generates correct data types in the DDL script, even if you do not select this pair of options.

5 Click the Selection tab.

The Selection tab includes two drop-down boxes: the drop-down box on the left is used to choose the RAP model to generate, and the drop-down box on the right is used to choose the database owner.

6 From the drop-down box on the right, choose the database owner.

7 On the Tables tab, click the Select All button, which is to the right of the database owner drop-down box.

8 On the Domains tab, choose the database owner, click the Select All button, click Apply, then click OK.

PowerDesigner checks the model for any errors, builds a result list, and generates the DDL. The Result dialog appears, which identifies the name and location of the generated file. You can click the Edit button on the Result dialog to view the generated script. Close the Result dialog.

The Result List dialog appears in the background and may include several warnings, for example, `"Existence of index"` and `"Existence of reference."` These warnings normally occur during generation.

9 Close the Result List dialog, then exit PowerDesigner.

- If PowerDesigner prompts you to save the current workspace, click No.
- If PowerDesigner prompts you to save the model, click Yes only if you want to save the modified model. Otherwise, click No.

Indexes in the VLDBServer database

As delivered, the *RAP_IQ.pdm* data model includes only those indexes that support the sample queries. Statements needed to create these indexes appear in the DDL scripts generated from the RAP IQ data model. Consequently, the indexes supplied with the model are created automatically when you run the corresponding DDL scripts.

Depending on site-specific issues such as limits on the available load time and the actual queries in the database, you may want to add or remove indexes from the RAP IQ data model. For detailed information on IQ indexes, refer to the Sybase IQ product documentation.

Executing the script

At this point, you can execute the DDL script in Interactive SQL and create database objects in the VLDBServer database.

1 Start the VLDB database server, if the server is not already running. To do this, change to the directory that contains the database files and use the following command format:

```
start_asiq -n server_name @config_file.cfg
database_name.db
```

Be sure to use the -n switch to name the server, either in the configuration file or on the command line when you start the server.

Note
If you specify -n *server_name* without a *database_name*, you connect to the default database on the current server. If you specify -n *database_name* without a *server_name*, you connect to the specified database on the current server.

2 Enter the following command at the operating system prompt to start Interactive SQL Java:

```
dbisql
```

3 Enter the correct User ID, Password, and server information in the dbisql Connect dialog box.

4 Open the generated DDL script for IQ and click the Execute SQL statement button on the Interactive SQL toolbar to execute the script.

RAPCache database

Follow these instructions to generate DDL for the RAPCache database, which is an Adaptive Server Enterprise database. The data model for RAPCache is *RAP_ASE.pdm* in the *Model\RAPCache* folder.

Changing the default database user

1 Start PowerDesigner 11.1.

2 Click File | Open | Choose *RAP_ASE.pdm*.

3 Click Model | Users and Roles | Users.

4 In the Name column, change the default user (RAP_USER) to the new database user.

5 Click Apply.

Generating a DDL script

After you change the default database user, you can generate DDL directly from the data model. PowerDesigner saves the results in a SQL script that you use to generate the tables and other objects in the target database.

Note
Use the model *RAP_ASE.pdm* to generate DDL for the RAPCache database. Do *not* use a different model by changing the target database to ASE, as this will result in the loss of index information.

1 Click Database | Generate Database.

2 On the Database Generation dialog, click the browse button, and choose the directory where you want to store the script. Click OK.

3 In the File name box, type a name for the SQL script. You will use this script in the next procedure.

4 On the Tables & Views, Keys & Indexes, Database, and Options tabs, set options as listed in the following table:

On this tab...	In this panel...	Set these options...
Tables & Views	Tables	Create table
	Columns	User Defined Type Check
Keys & Indexes	Primary keys	Create primary key Inside Table
	Indexes	Deselect Create index
	Foreign keys	Create foreign key Outside Declarative Integrity
Database	User-defined data type	Create data type
Options		Accept all defaults

5 Click the Selection tab.

The Selection tab includes two drop-down boxes: the drop-down box on the left is used to choose the RAP model to generate, and the drop-down box on the right is used to choose the database owner.

6 From the drop-down box on the right, choose the database owner.

7 On the Tables tab, click the Select All button, which is to the right of the database owner drop-down box.

8 On the Domains tab, choose the database owner, click the Select All button, click Apply, then click OK.

PowerDesigner checks the model for any errors, builds a result list, and generates the DDL. The Result dialog appears, which identifies the name and location of the generated file. You can click the Edit button on the Result dialog to view the generated script. Close the Result dialog.

The Result List dialog appears in the background and may include several warnings, for example, `"Existence of index"` and `"Existence of reference."` These warnings normally occur during generation.

9 Close the Result List dialog, then exit PowerDesigner.

- If PowerDesigner prompts you to save the current workspace, click No.
- If PowerDesigner prompts you to save the model, click Yes only if you want to save the modified model. Otherwise, click No.

Modifying the DDL script

The standard RAPCache database DDL script that ships with Risk Analytics includes configuration statements not found in DDL generated directly from the data model. If you generate DDL from the data model, you must edit the script and add the missing statements.

There are two ways to edit a custom script:

- Use an editor to copy the missing statements from the standard RAPCache DDL script into the script you generated from the data model. The standard RAPCache DDL script is *$RAP30/Model/RAPCache/RAP_Table.sql*.
- Open the script you generated from the data model and type the missing statements at the appropriate locations.

Editing the script

Add these statements immediately after the file header and before the first domain statement that begins on line 7. These edits modify the tempdb and model database size, change the database settings, and bind the cache to memory, preventing data from being paged-out to disk.

```
use master
go
alter database tempdb on master=100
go
alter database model on master=100
go
sp_dboption model, single, 'true'
go
use model
go
sp_bindcache c_log, model, syslogs
go
use master
go
sp_dboption model, single, 'true'
go
use model
go
sp_bindcache c_log, model, syslogs
go
use master
go
sp_dboption model, single, 'false'
```

```
go
use model
go
sp_logiosize '16K'
go
exec sp_addsegment s1 , model, master
exec sp_addsegment s2 , model, master
exec sp_addsegment s3 , model, master
exec sp_addsegment o1 , model, master
go
```

STOCK_QUOTE

The STOCK_QUOTE table stores real-time (intraday) quotes. To modify this table, comment out the constraint statement and add statements to reduce lock contention and partition the table to distribute I/O over different devices.

1 Locate the STOCK_QUOTE table; use double dashes to comment out the line that reads:

```
constraint PK_STOCK_QUOTE primary key
(INSTRUMENT_ID, QUOTE_DATE, QUOTE_SEQ_NBR) on o1
```

The line should now read:

```
-- constraint PK_STOCK_QUOTE primary key
(INSTRUMENT_ID, QUOTE_DATE, QUOTE_SEQ_NBR) on o1
```

2 Add these lines after the open parenthesis) on the next line:

```
)
lock datarows
partition by roundrobin (p1 on s1, p2 on s2, p3
on s3)
```

The entire edit looks like this:

```
-- constraint PK_STOCK_QUOTE primary key
(INSTRUMENT_ID, QUOTE_DATE, QUOTE_SEQ_NBR) on o1
)
lock datarows
partition by roundrobin (p1 on s1, p2 on s2, p3
on s3)
go
```

STOCK_TRADE

The STOCK_TRADE table stores real-time (intraday) trade data. To modify this table, comment out the constraint statement and add statements to reduce lock contention and partition the table to distribute I/O over different devices.

1 Locate the STOCK_TRADE table, use double dashes to comment out the line that reads:

```
constraint PK_STOCK_TRADE primary key
(INSTRUMENT_ID, TRADE_SEQ_NBR, TRADE_DATE) on o1
```

The line should now read:

```
-- constraint PK_STOCK_TRADE primary key
(INSTRUMENT_ID, TRADE_SEQ_NBR, TRADE_DATE) on o1
```

2 Add these lines after the open parenthesis) on the next line:

```
)
lock datarows
partition by roundrobin (p1 on s1, p2 on s2, p3
on s3)
```

The entire edit looks like this:

```
-- constraint PK_STOCK_TRADE primary key
(INSTRUMENT_ID, TRADE_SEQ_NBR, TRADE_DATE) on o1
)
lock datarows
partition by roundrobin (p1 on s1, p2 on s2, p3 on
s3)
go
```

ULYING_INSTR

ULYING_INSTR is an association table used to define a financial instrument. You need to add statements after this block to create a local index on quote time for the STOCK_QUOTE table and a local index on trade time for the STOCK_TRADE table.

1 Locate the ULYING_INSTR table and look for the block that reads:

```
create table <database owner>.ULYING_INSTR (
OPTION_INSTR_ID  SYB_ID not null,
INSTRUMENT_ID  SYB_ID not null,
constraint PK_ULYING_INSTR primary key
(OPTION_INSTR_ID, INSTRUMENT_ID)
)
go
```

2 Add the following lines after the `go` command:

```
create index STOCK_QUOTE_QUOTE_TIME on STOCK_QUOTE
(
QUOTE_TIME ASC
)

on o1
local index
go

create index STOCK_TRADE_TRADE_TIME on STOCK_TRADE
(
TRADE_TIME ASC
)
on o1
local index
go
```

The entire edit looks like this:

```
create table <database owner>.ULYING_INSTR (
OPTION_INSTR_ID  SYB_ID not null,
INSTRUMENT_ID  SYB_ID not null,
constraint PK_ULYING_INSTR primary key
(OPTION_INSTR_ID, INSTRUMENT_ID)
)
go

create index STOCK_QUOTE_QUOTE_TIME on STOCK_QUOTE
(
QUOTE_TIME ASC
)

on o1
local index
go

create index STOCK_TRADE_TRADE_TIME on STOCK_TRADE
(
TRADE_TIME ASC
)
on o1
local index
go
```

LAST_TRADE_PRICE

The LAST_TRADE_PRICE table stores information about the last trade prices of the trading day. The SQL statements that create the LAST_TRADE_PRICE table are not included in the DDL generated from the data model. Follow these instructions to add the necessary SQL statements to your custom script.

This table is used like a temporary table in the cache to support the tick_qry3 sample query. If you run this query against the VLDBServer database instead of RAPCache, you do not need this table, as you have access to historical data.

1 Locate the section of code that creates the MAJOR_IDST_CLS table. This section begins with the following lines:

```
/*==============================================*/
/* Table: MAJOR_IDST_CLS                        */
/*==============================================*/
```

2 Add the following lines *before* the create MAJOR_IDST_CLS table section:

```
/*==============================================*/
/* Table: LAST_TRADE_PRICE                      */
/*==============================================*/
create table LAST_TRADE_PRICE
(INSTRUMENT_ID    SYB_ID         not null,
TRADING_SYMBOL    SYB_CODE_VAR   not null,
TRADE_PRICE       SYB_MONEY      null,
TRADE_DATE        SYB_DATE       not null
)
create index LAST_TRADE_PRICE_INSTRUMENT_ID on
LAST_TRADE_PRICE (
INSTRUMENT_ID ASC
)
go
```

The entire edit looks like this:

```
/*==============================================*/
/* Table: LAST_TRADE_PRICE                      */
/*==============================================*/
create table LAST_TRADE_PRICE
(INSTRUMENT_ID    SYB_ID         not null,
TRADING_SYMBOL    SYB_CODE_VAR   not null,
TRADE_PRICE       SYB_MONEY      null,
TRADE_DATE        SYB_DATE       not null
)
create index LAST_TRADE_PRICE_INSTRUMENT_ID on
LAST_TRADE_PRICE (
INSTRUMENT_ID ASC
)
go
```

```
/*============================================*/
/* Table: MAJOR_IDST_CLS                      */
/*============================================*/
```

Executing the script

At this point, you can execute the DDL script in Interactive SQL and create database objects in the RAPCache database. These instructions apply to UNIX and Linux platforms.

1 At the operating system prompt, enter the command:

```
isql -Sserver_name -Uuser_name -Ppassword -iase_ddl.sql -ologfile
```

If the RAPCache server is not running, start the server as described in "Start the RAPCache server" in Chapter 2, "Installing RAP Core Components" of the *Risk Analytics Platform Installation and Configuration Guide*.

2 Check the log file for errors.

APPENDIX A

SQL Scripts for Sample Queries

About this Appendix

This appendix contains the SQL scripts for the Risk Analytics Platform sample TAQ, Interday, and Historical market data queries. The results returned by running the queries with the sample RAP data are also included. Script files are located in subdirectories of the *$RAP30/Scripts* directory. Throughout this appendix, the environment variable $RAP30 refers to the RAP 3.0 installation directory.

For more information about the RAP sample queries, refer to Chapter 3, “Sample Queries.”

Contents

TAQ data queries

TAQ data tables include real-time price quotes and trade prices that are updated frequently during a trading day. Queries against these tables use intraday price and quote fluctuations.

The tick queries are representative of the query workloads generated in pre-trade analysis and trade order generation. Although this is a partial list of possible queries, these queries constitute a reasonable sample test for use in performance and tuning analysis and also as a template for building a library of queries. The sample TAQ queries can also be modified to build a native T-SQL access layer to present data to computational applications.

The TAQ data queries are optimized to run with the Risk Analytics Platform sample data in both VLDBServer and RAPCache databases.

Note The TAQ data queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede the query with a commit statement.

Script name	Description
tick_qry1.sql	Get all ticks for a specified set of 100 securities for a specified three-hour time period on a specified trade date.
tick_qry2.sql	Determine the volume-weighted price of a security considering only the ticks in a specified three-hour interval.
tick_qry3.sql	Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.
tick_qry4.sql	Determine the top 10 most active stocks for a specified date, sorted by cumulative trade volume, by considering all trades.
tick_qry5.sql	Find the most active stocks in the COMPUTER industry (use SIC code).
tick_qry6.sql	Find the 10 stocks with the highest percentage spreads. Spread is the difference between the last ask-price and the last bid-price. Percentage spread is calculated as a percentage of the bid-point price (average of ask and bid price).

Note An additional SQL script setup_tick_qry3_last_price.sql should be run on RAPCache at the end of each trading day to capture the last price of that day. This data is referenced by tick_qry3.sql.

tick_qry1.sql

Get all ticks for a specified set of 100 securities for a specified three-hour time period on a specified trade date.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL   TRADE_DATE                    TRADE_TIME     TRADE_PRICE
AAO              2005-11-14      2005-11-14 09:00:02.000000          113.34
AAQ              2005-11-14      2005-11-14 09:00:03.000000           91.75
ABQ              2005-11-14      2005-11-14 09:00:10.000000           73.68
ACR              2005-11-14      2005-11-14 09:00:14.000000          109.62
ACE              2005-11-14      2005-11-14 09:00:17.000000           77.34
AAD              2005-11-14      2005-11-14 09:00:39.000000          106.18
ACI              2005-11-14      2005-11-14 09:00:49.000000           26.34
ABF              2005-11-14      2005-11-14 09:00:51.000000           78.68
ACQ              2005-11-14      2005-11-14 09:01:01.000000           78.28
ADF              2005-11-14      2005-11-14 09:01:25.000000           38.87
ABU              2005-11-14      2005-11-14 09:01:35.000000          113.75
AAR              2005-11-14      2005-11-14 09:01:43.000000           33.65
ACY              2005-11-14      2005-11-14 09:01:49.000000           76.71
ACD              2005-11-14      2005-11-14 09:01:55.000000           47.34
ACR              2005-11-14      2005-11-14 09:01:55.000000          109.62
ACR              2005-11-14      2005-11-14 09:01:57.000000          109.62
ACP              2005-11-14      2005-11-14 09:01:57.000000          101.28
ADS              2005-11-14      2005-11-14 09:02:02.000000           31.75
ADO              2005-11-14      2005-11-14 09:02:02.000000           95.21
AAC              2005-11-14      2005-11-14 09:02:19.000000          118.68
ACM              2005-11-14      2005-11-14 09:02:26.000000           80.25
ADL              2005-11-14      2005-11-14 09:02:29.000000          122
AAP              2005-11-14      2005-11-14 09:02:32.000000           53.28
AAQ              2005-11-14      2005-11-14 09:02:39.000000           91.75
ACK              2005-11-14      2005-11-14 09:02:53.000000           69.5
...
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and VLDBServer.

VLDBServer

```
-- Get all ticks for a specified set of 100 securities for a specified
-- three hour time period on a specified trade date.
-- This query is optimized to run IQ.

commit

;

SELECT TRADING_SYMBOL, TRADE_DATE, TRADE_TIME, TRADE_PRICE
FROM  STOCK_TRADE
WHERE TRADE_TIME BETWEEN '2005-11-14 9:00'
    AND '2005-11-14 12:00'
    AND TRADING_SYMBOL BETWEEN 'AAA' AND 'ADV'
    AND LENGTH(TRADING_SYMBOL) = 3
;
```

RAPCache

```
-- Get all ticks for a specified set of 100 securities for a specified
-- three hour time period on a specified trade date.

-- This query is optimized to run ASE.


SELECT TRADING_SYMBOL, TRADE_DATE, TRADE_TIME, TRADE_PRICE
FROM  STOCK_TRADE
WHERE TRADE_TIME BETWEEN '2005-11-14 9:00:00'
    AND '2005-11-14 12:00:00'
    AND TRADING_SYMBOL BETWEEN 'AAA' AND 'ADV'
    AND LEN(TRADING_SYMBOL) = 3

go
```

tick_qry2.sql

Determine the volume-weighted price of a security considering only the ticks in a specified three-hour interval.

Output

The following output displays the results of this query for the specified security ADV:

```
TRADING_SYMBOL    VOLUME_WEIGHTED_PRICE
ADV                          30.73768473
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and VLDBServer.

VLDBServer

```
-- Determine the volume weighted price of a security considering
-- only the ticks in a specified three hour interval.

-- This query will run on either the ASE or IQ platform.

commit
;

SELECT TRADING_SYMBOL,
SUM(TRADE_SIZE*TRADE_PRICE)/SUM(TRADE_SIZE) as VOLUME_WEIGHTED_PRICE
FROM STOCK_TRADE
WHERE TRADE_TIME BETWEEN '2005-11-14 12:00'
AND '2005-11-14 15:00'
AND TRADING_SYMBOL ='ADV'
GROUP BY TRADING_SYMBOL;
```

RAPCache

```
-- Determine the volume weighted price of a security considering
-- only the ticks in a specified three hour interval.

-- This query will run on either the ASE or IQ platform.


SELECT TRADING_SYMBOL,
SUM(TRADE_SIZE*TRADE_PRICE)/SUM(TRADE_SIZE) as VOLUME_WEIGHTED_PRICE
FROM STOCK_TRADE
WHERE TRADE_TIME BETWEEN '2005-11-14 12:00' AND '2005-11-14 15:00'
```

```
   AND TRADING_SYMBOL = 'ADV'
   GROUP BY TRADING_SYMBOL

   go
```

setup_tick_qry3_last_price.sql

This SQL script should be run on the RAPCache database at the end of each trading day to capture the last price of that day. This data is stored in the LAST_TRADE_PRICE table and is referenced by tick_qry3.sql. Before running the script, change the date in the two WHERE clauses to the date of the trading day for which the last trade price is to be captured.

Output

The following output displays the type of data inserted in the LAST_TRADE_TABLE table by this query:

```
INSTRUMENT_ID    TRADING_SYMBOL    TRADE_PRICE    TRADE_DATE
          768    BDO                     62.96    Nov 10 2005
          676    BAA                     37.96    Nov 10 2005
          419    AQD                     28.65    Nov 10 2005
           38    ABM                      8.03    Nov 10 2005
          986    BLY                     52.53    Nov 10 2005
          332    AMU                        17    Nov 10 2005
          200    AHS                     35.34    Nov 10 2005
          104    AEA                     19.81    Nov 10 2005
          823    BFR                     38.53    Nov 10 2005
          440    AQY                     41.21    Nov 10 2005
...
```

SQL

The following script contains the SQL statements for this query:

```
-- This code should be run at the end of each trading day to capture
-- the last price of that day. Before running the script, the date in
-- the two where clauses needs to be changed to that of the trading day
-- for which the last trade price is to be captured.

Insert LAST_TRADE_PRICE
Select INSTRUMENT_ID, TRADING_SYMBOL, TRADE_PRICE, TRADE_DATE
FROM  STOCK_TRADE st, (Select INSTRUMENT_ID AS idx, max(TRADE_TIME) AS maxtime,
  max(TRADE_SEQ_NBR) AS maxseq
```

```
    from STOCK_TRADE
    where TRADE_TIME between '2005-11-10 00:00:00' and '2005-11-10 23:59:59'
group by INSTRUMENT_ID) y
WHERE  st.TRADE_TIME between '2005-11-10 00:00:00' and '2005-11-10 23:59:59'
AND st.TRADE_TIME = maxtime and idx = st.INSTRUMENT_ID and st.TRADE_SEQ_NBR =
maxseq
go
```

tick_qry3.sql

Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.

Note The SQL script setup_tick_qry3_last_price.sql should be run at the end of each trading day on RAPCache database to capture data required by tick_qry3.sql.

Output

The following output displays the rows returned by this query:

```
INSTRUMENT_ID   TRADING_SYMBOL    PER_LOSER   LOSER_RANK
          443   ARB              -1.7355085             1
          620   AXW              -1.72143974            2
          173   AGR              -1.47387226            3
           99   ADV              -1.22580645            4
          863   BHF              -1.12903225            5
          805   BEZ              -1.03437785            6
          440   AQY              -1.0184937             7
          448   ARG              -0.91441111            8
          925   BJP              -0.90879584            9
          374   AOK              -0.79365079           10
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and VLDBServer.

VLDBServer

```
BEGIN
--Determine the top 10 percentage losers for the specified date on the
--specified exchanges sorted by percentage loss. The loss is calculated
--as a percentage of the last trade price of the previous day.

-- This query is optimized to run on  IQ.

commit
;

Select INSTRUMENT_ID, TRADING_SYMBOL, TRADE_PRICE, TRADE_DATE into
#temp_tick3a
FROM  DBA.STOCK_TRADE st, (Select INSTRUMENT_ID AS idx, max(TRADE_TIME) AS
maxtime
from DBA.STOCK_TRADE where TRADE_DATE = '2005-11-11'
group by INSTRUMENT_ID) y
WHERE  st.TRADE_DATE =  '2005-11-11'
AND st.TRADE_TIME = maxtime and idx = st.INSTRUMENT_ID
;


create variable prev_day date;
set prev_day = (Select MAX(TRADE_DATE) from STOCK_TRADE where TRADE_DATE <
'2005-11-11');

SELECT TOP 10 INSTRUMENT_ID,TRADING_SYMBOL, PER_LOSER, LOSER_RANK
FROM   (SELECT INSTRUMENT_ID, TRADING_SYMBOL, per_loser,
RANK() OVER (ORDER BY per_loser ASC) loser_rank
FROM   (SELECT t.INSTRUMENT_ID,t.TRADING_SYMBOL,  (t.mtp-y.mtp)*100/y.mtp
per_loser
FROM  (SELECT INSTRUMENT_ID,TRADING_SYMBOL,TRADE_PRICE  mtp
FROM #temp_tick3a) t,
(SELECT INSTRUMENT_ID,TRADING_SYMBOL,CLOSE_PRICE  mtp
FROM STOCK_HISTORY
WHERE TRADE_DATE = prev_day) y
WHERE  t.INSTRUMENT_ID=y.INSTRUMENT_ID
) a
) b
where PER_LOSER < 0
ORDER BY PER_LOSER;
```

```
drop variable prev_day;

END
```

RAPCache

```
--Determine the top 10 percentage losers for the specified date on the
--specified exchanges sorted by percentage loss. The loss is calculated
--as a percentage of the last trade price of the previous day.

-- This query is optimized to run on ASE.


set parallel_degree 1
go


-- current day
Select  INSTRUMENT_ID, TRADING_SYMBOL, TRADE_PRICE, TRADE_DATE into
#temp_tick3a
FROM  STOCK_TRADE st, (Select INSTRUMENT_ID AS idx, max(TRADE_TIME) AS
maxtime
from STOCK_TRADE
where TRADE_TIME between '2005-11-11 00:00:00' and '2005-11-11 23:59:59'
group by INSTRUMENT_ID) y
-- WHERE  st.TRADE_DATE =  '2005-11-11'
WHERE  st.TRADE_TIME between '2005-11-11 00:00:00' and '2005-11-11 23:59:59'
AND st.TRADE_TIME = maxtime and idx = st.INSTRUMENT_ID


set rowcount 10
SELECT  TRADING_SYMBOL, PER_LOSER
FROM   (SELECT TRADING_SYMBOL, PER_LOSER
FROM   (SELECT t.INSTRUMENT_ID, t.TRADING_SYMBOL,
   (t.mtp-y.mtp)*100/y.mtp PER_LOSER
FROM  (SELECT INSTRUMENT_ID,TRADING_SYMBOL, TRADE_DATE ,TRADE_PRICE  mtp
FROM #temp_tick3a) t,
(SELECT INSTRUMENT_ID, TRADING_SYMBOL, TRADE_DATE,TRADE_PRICE  mtp
FROM LAST_TRADE_PRICE) y
WHERE  t.INSTRUMENT_ID=y.INSTRUMENT_ID
AND    y.TRADE_DATE = '2005-11-10'
) a
) b
where PER_LOSER < 0
ORDER BY PER_LOSER ASC

set rowcount 0
```

```
drop table #temp_tick3a

go
```

tick_qry4.sql

Determine the top 10 most active stocks for a specified date, sorted by cumulative trade volume, by considering all trades.

Output

The following output displays the rows returned by this query:

```
TRADING_SYMBOL    TRADESIZE
ASU                 2932300
BEG                 2929000
AYD                 2923600
AJA                 2884400
AVQ                 2874300
AAC                 2856400
AKC                 2854900
ACN                 2851800
AQL                 2834100
AFE                 2821600
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and VLDBServer.

VLDBServer

```
-- Determine the top 10 most active stocks for a specified date
-- sorted by cumulative trade volume by considering all trades.


-- This query is optimized to run on IQ.

commit
;

SELECT TOP 10 TRADING_SYMBOL, sum(TRADE_SIZE) as TRADESIZE, DENSE_RANK ()
OVER (ORDER BY sum(TRADE_SIZE) DESC) as RANKING
    FROM STOCK_TRADE
```

```
WHERE  TRADE_DATE = '2005-11-14'
    GROUP BY TRADING_SYMBOL
    order by sum(TRADE_SIZE) DESC
;
```

RAPCache

```
-- Determine the top 10 most active stocks for a specified date
-- sorted by cumulative trade volume by considering all trades.

-- This query will run on either the ASE or IQ platform.

set  rowcount 10
go

SELECT TRADING_SYMBOL, sum(TRADE_SIZE) as TRADESIZE
    FROM STOCK_TRADE
WHERE
TRADE_TIME between '2005-11-14 00:00:00'  and '2005-11-14 23:59:59'
GROUP BY TRADING_SYMBOL
order by sum(TRADE_SIZE) DESC
go


set rowcount 0
go
```

tick_qry5.sql

Find the most active stocks in the COMPUTER industry (use SIC code).

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL    TRADESIZE    RANKING
ASZ                  249000          1
APE                  198300          2
BGS                  195400          3
BHI                  194200          4
BDR                  188400          5
AJB                  185700          6
BGW                  185100          7
AAZ                  182700          8
AEF                  182100          9
BFE                  181000         10
ADC                  180900         11
BCY                  180200         12
BIG                  177000         13
BLS                  176200         14
BJU                  175700         15
ANW                  174200         16
ANS                  171600         17
AJL                  169600         18
AOD                  168300         19
ALD                  166700         20
AXS                  166400         21
BIJ                  166300         22
ABS                  165900         23
AAV                  165500         24
BFG                  163200         25
...
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and VLDBServer.

VLDBServer

```
-- Find the most active stocks in the "COMPUTER" industry
-- for the current day.

-- This query is optimized to run on IQ.
```

```
commit
;

SELECT st.TRADING_SYMBOL, SUM(TRADE_SIZE) TRADESIZE,
DENSE_RANK() OVER (ORDER by SUM(TRADE_SIZE) DESC) as RANKING
FROM STOCK_TRADE st
inner join INSTRUMENT ii
on ii.INSTRUMENT_ID = st.INSTRUMENT_ID
inner join SCND_IDST_CLS sc
on ii.SCND_IDST_CLS_ID = sc.SCND_IDST_CLS_ID
and sc.SIC_NAME = 'COMPUTERS'
WHERE  st.TRADE_DATE = '2005-11-14'
GROUP BY
st.TRADING_SYMBOL
;
```

RAPCache

```
-- Find the most active stocks in the "COMPUTER" industry.


-- This query will run on either the ASE or IQ platform.
set forceplan on
go

SELECT st.TRADING_SYMBOL,SUM(TRADE_SIZE) as TRADESIZE
FROM STOCK_TRADE st
inner join INSTRUMENT ii
on ii.INSTRUMENT_ID = st.INSTRUMENT_ID
inner join SCND_IDST_CLS sc
on ii.SCND_IDST_CLS_ID = sc.SCND_IDST_CLS_ID
and sc.SIC_NAME = 'COMPUTERS'
WHERE
TRADE_TIME between '2005-11-14 00:00:00' and '2005-11-14 23:59:59'
GROUP BY st.TRADING_SYMBOL
order by SUM(TRADE_SIZE) DESC

go
```

tick_qry6.sql

Find the 10 stocks with the highest percentage spreads. Spread is the difference between the last ask-price and the last bid-price. Percentage spread is calculated as a percentage of the bid-point price (average of ask and bid price).

Output

The following output displays the rows returned by this query:

```
TRADING_SYMBOL          PER
ACG                     0.027027027027027027027027
ADC                     0.018248175182481751824817
AFZ                     0.018018018018018018018018
BBN                     0.017654476670870113493064
ASS                     0.017431725740848343986054
AUZ                     0.016806722689075630252100
BGW                     0.016051364365971107544141
AKF                     0.015804597701149425287356
AUB                     0.012652889076339097427245
AZY                     0.012628255722178374112075
```

SQL

The following script contains the SQL statements for this query. Note that there are separate scripts optimized for RAPCache and RAPCache.

VLDBServer

```
-- Find the 10 stocks with the highest percentage spreads.
-- Spread is the difference between the last ask-price and
-- the last bid-price.
-- Percentage spread is calculated as a percentage of the
-- bid-point price (average of ask and bid price).

-- This query is optimized to run on IQ.

commit
;
SELECT TOP 10 TRADING_SYMBOL, PER, RANK() OVER ( ORDER BY per DESC)
AS PER_RANK
FROM   (SELECT a.INSTRUMENT_ID, a.TRADING_SYMBOL,(ap-bp)*2/(ap+bp)
AS per
FROM   (SELECT INSTRUMENT_ID, TRADING_SYMBOL, BID_PRICE as bp
FROM STOCK_QUOTE st, (Select INSTRUMENT_ID AS idx, max(QUOTE_TIME)
AS maxtime
from STOCK_QUOTE where QUOTE_DATE = '2005-11-14'
```

```
AND BID_PRICE IS NOT NULL
AND BID_PRICE <> 0
group by INSTRUMENT_ID) y
WHERE st.QUOTE_DATE = '2005-11-14'
AND BID_PRICE IS NOT NULL
AND BID_PRICE <> 0
                AND st.QUOTE_TIME = maxtime and idx = st.INSTRUMENT_ID
    group by INSTRUMENT_ID,TRADING_SYMBOL, BID_PRICE) a,
(SELECT INSTRUMENT_ID,TRADING_SYMBOL, ASK_PRICE AS ap
FROM STOCK_QUOTE st, (Select INSTRUMENT_ID AS idx, max(QUOTE_TIME)
AS maxtime
from STOCK_QUOTE where QUOTE_DATE = '2005-11-14'
AND ASK_PRICE IS NOT NULL
AND ASK_PRICE <> 0
group by INSTRUMENT_ID) x
WHERE  st.QUOTE_DATE = '2005-11-14'
AND ASK_PRICE IS NOT NULL
AND ASK_PRICE <> 0
AND st.QUOTE_TIME = maxtime and idx = st.INSTRUMENT_ID
   group by INSTRUMENT_ID, TRADING_SYMBOL, ASK_PRICE) b
WHERE  a.INSTRUMENT_ID=b.INSTRUMENT_ID
)c
ORDER BY PER DESC
;
```

RAPCache

```
-- Find the 10 stocks with the highest percentage spreads.
-- Spread is the difference between the last ask-price and
-- the last bid-price.
-- Percentage spread is calculated as a percentage of the
-- bid-point price (average of ask and bid price).

-- This query will run on either the ASE or IQ platform.

set  rowcount 10
go

SELECT a.TRADING_SYMBOL, (ap-bp)*2/(ap+bp) AS PER
FROM   (SELECT INSTRUMENT_ID,TRADING_SYMBOL, BID_PRICE as bp
FROM STOCK_QUOTE st, (Select INSTRUMENT_ID AS idx, max(QUOTE_TIME)
AS maxtime
from STOCK_QUOTE
         where
         QUOTE_TIME between '2005-11-14 00:00:00' and '2005-11-14 23:59:59'
AND BID_PRICE IS NOT NULL
AND BID_PRICE <> 0
```

```
group by INSTRUMENT_ID,
TRADING_SYMBOL) y
WHERE
QUOTE_TIME between '2005-11-14 00:00:00' and '2005-11-14 23:59:59'
AND BID_PRICE IS NOT NULL
AND BID_PRICE <> 0
          AND st.QUOTE_TIME = maxtime and idx = st.INSTRUMENT_ID
          group by INSTRUMENT_ID, TRADING_SYMBOL,BID_PRICE) a,
(SELECT INSTRUMENT_ID, TRADING_SYMBOL, ASK_PRICE AS ap
FROM   STOCK_QUOTE st, (Select INSTRUMENT_ID AS idx, max(QUOTE_TIME)
AS maxtime
from STOCK_QUOTE
         where
QUOTE_TIME between '2005-11-14 00:00:00' and '2005-11-14 23:59:59'
AND ASK_PRICE IS NOT NULL
AND ASK_PRICE <> 0
group by INSTRUMENT_ID,
TRADING_SYMBOL) x
WHERE
QUOTE_TIME between '2005-11-14 00:00:00' and '2005-11-14 23:59:59'
AND ASK_PRICE IS NOT NULL
AND ASK_PRICE <> 0
AND st.QUOTE_TIME = maxtime and idx = st.INSTRUMENT_ID
group by INSTRUMENT_ID, TRADING_SYMBOL,ASK_PRICE) b
WHERE  a.INSTRUMENT_ID=b.INSTRUMENT_ID
ORDER BY PER DESC

go
set  rowcount 0
go
```

Interday queries

Interday queries reflect price quotes and trade prices during multiple trading days. These queries examine the tick data of the current day plus historical data and can provide an alert for changes to statistical models during the course of the trading day.

The Interday queries are optimized to run with the Risk Analytics Platform sample data in VLDBServer.

Note The Interday queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede the query with a commit statement.

Script name	Description
interday_tick_qry1.sql	Determine the volume-weighted price of a security considering only the ticks in a specified three-day interval.
interday_tick_qry2.sql	Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.
interday_tick_qry3.sql	Find the most active stocks in the “COMPUTER” industry for last three days.

interday_tick_qry1.sql

Determine the volume-weighted price of a security considering only the ticks in a specified three-day interval.

Output

The following output display the results of this query for the specified security AAA:

```
TRADING_SYMBOL   VOLUME_WEIGHTED_PRICE
AAA                   49.6641081572369
```

SQL

The following script contains the SQL statements for this query.

```
-- Determine the volume weighted price of a security considering
-- only the ticks in a specified three day interval.

commit;

SELECT TRADING_SYMBOL,
SUM(TRADE_SIZE*TRADE_PRICE)/SUM(TRADE_SIZE) as VOLUME_WEIGHTED_PRICE
FROM STOCK_TRADE
WHERE TRADE_DATE BETWEEN '2005-11-10'
AND '2005-11-14'
AND TRADING_SYMBOL ='AAA'
GROUP BY TRADING_SYMBOL;
```

interday_tick_qry2.sql

Determine the top 10 percentage losers for the specified date on the specified exchanges, sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.

Output

The following output displays the rows returned by this query:

```
INSTRUMENT_ID   TRADING_SYMBOL     PER_LOSER   LOSER_RANK
          765   BDL              -0.20549224            1
          302   ALQ              -0.09486665            2
          227   AIT              -0.07940709            3
          789   BEJ              -0.06967022            4
          489   ASV               0.03770739            5
           70   ACS               0.05735041            6
          568   AVW               0.08119079            7
           48   ABW               0.26094176            8
          283   AKX               0.26346122            9
          569   AVX               0.34009873           10
```

SQL

The following script contains the SQL statements for this query:

```
commit;

BEGIN
--Determine the top 10 percentage losers for the specified date on the
--specified exchanges sorted by percentage loss. The loss is calculated
--as a percentage of the last trade price of the previous day.


Select INSTRUMENT_ID, TRADING_SYMBOL, TRADE_PRICE, TRADE_DATE into #temp_tick3a
FROM  DBA.STOCK_TRADE st, (Select INSTRUMENT_ID AS idx, max(TRADE_TIME) AS
maxtime
from DBA.STOCK_TRADE where TRADE_DATE = '2005-11-14'
group by INSTRUMENT_ID) y
WHERE  st.TRADE_DATE =  '2005-11-14'
AND st.TRADE_TIME = maxtime and idx = st.INSTRUMENT_ID
;

create variable prev_day date;
set prev_day = (Select MAX(TRADE_DATE) from STOCK_HISTORY
where TRADE_DATE < '2005-11-14');
```

```
SELECT TOP 10 INSTRUMENT_ID,TRADING_SYMBOL, PER_LOSER, LOSER_RANK
FROM   (SELECT INSTRUMENT_ID, TRADING_SYMBOL, PER_LOSER,
RANK() OVER (ORDER BY per_loser ASC) LOSER_RANK
FROM   (SELECT t.INSTRUMENT_ID,t.TRADING_SYMBOL,
(t.mtp-y.mtp)*100/y.mtp PER_LOSER
FROM  (SELECT INSTRUMENT_ID,TRADING_SYMBOL,TRADE_PRICE  mtp
FROM #temp_tick3a) t,
(SELECT INSTRUMENT_ID,TRADING_SYMBOL,CLOSE_PRICE  mtp
FROM STOCK_HISTORY
WHERE TRADE_DATE = prev_day) y
WHERE  t.INSTRUMENT_ID=y.INSTRUMENT_ID
) a
) b
where PER_LOSER < 0
ORDER BY PER_LOSER;

drop variable prev_day;
END
```

interday_tick_qry3.sql

Find the most active stocks in the “COMPUTER” industry for last three days.

Output

The following output displays the first 25 rows returned by this query:

INSTRUMENT_ID	TRADING_SYMBOL	TRADESIZE	RANKING
289	ALD	554400	1
493	ASZ	538700	2
850	BGS	529400	3
346	ANI	519300	4
866	BHI	516700	5
394	APE	504600	6
44	ABS	502800	7
400	APK	492000	8
360	ANW	490500	9
560	AVO	488800	10
507	ATN	485800	11
886	BIC	484600	12
752	BCY	484400	13
616	AXS	477500	14
356	ANS	477200	15

```
487             AST         477000        16
 80             ADC         475700        17
230             AIW         474200        18
980             BLS         466700        19
810             BFE         464400        20
588             AWQ         463400        21
771             BDR         463000        22
930             BJU         462800        23
460             ARS         462300        24
773             BDT         458400        25
...
```

SQL

The following script contains the SQL statements for this query:

```
-- Find the most active stocks in the "COMPUTER" industry
-- for last three days.

commit;

SELECT st.INSTRUMENT_ID, st.TRADING_SYMBOL, SUM(TRADE_SIZE) TRADESIZE,
DENSE_RANK() OVER (ORDER by SUM(TRADE_SIZE) DESC) as RANKING
FROM STOCK_TRADE st
inner join INSTRUMENT ii
on ii.INSTRUMENT_ID = st.INSTRUMENT_ID
inner join SCND_IDST_CLS sc
on ii.SCND_IDST_CLS_ID = sc.SCND_IDST_CLS_ID
and sc.SIC_NAME = 'COMPUTERS'
WHERE  st.TRADE_DATE BETWEEN '2005-11-10' AND '2005-11-14'
GROUP BY st.INSTRUMENT_ID
,st.TRADING_SYMBOL
;
```

Historical market data queries

Historical market data queries compare price histories for different instruments over time. Historical market data does not change frequently, and updates typically occur at the end of the trading day.

The end-of-day data is consolidated into the historical TAQ time series in VLDBServer as well. Stock History, Stock Quotes, and Stock Trade tables contain entries for end-of-day data records.

The Historical market data queries are optimized to run with the Risk Analytics Platform sample data in VLDBServer.

Note The Historical market data queries for VLDBServer all begin with a commit statement. This commit statement causes the data to refresh, so the query accesses the most recent data. When you write your own queries for VLDBServer, be sure to precede the query with a commit statement.

Script name	Description
hist_qry1.sql	Get the closing price of a set of 10 stocks for a 10-year period and group into weekly, monthly, and yearly aggregates. For each aggregate period, determine the low, high, and average closing price value. Output is sorted by TRADING_SYMBOL and trade date.
hist_qry2.sql	Adjust all prices and volumes (prices are multiplied by the split factor and volumes are divided by the split factor) for a set of 1000 stocks to reflect the split events during a specified 300-day period, assuming that events occur before the first trade of the split date. These are called split-adjusted prices and volumes.
hist_qry3.sql	For each stock in a specified list of 1000 stocks, find the differences between the daily high and daily low on the day of each split event during a specified period.
hist_qry4.sql	Calculate the value of the S&P 500 and Russell 2000 index for a specified day using unadjusted prices and the index composition of the two indexes on the specified day.
hist_qry5.sql	Find the 21-day and 5-day moving average price for a specified list of 1000 stocks during a 6-month period. (Use split-adjusted prices).
hist_qry6.sql	(Based on the previous query) Find the points (specific days) when the 5-day moving average intersects the 21-day moving average for these stocks. Output is sorted by TRADING_SYMBOL and trade date.

Script name	Description
hist_qry7.sql	Determine the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (That is, allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average, the complete allocation for that stock is invested, and when the 20-day moving average crosses below the 5-month moving average, the entire position is sold. The trades are made on the closing price of the trading day.
hist_qry8.sql	Find the pair-wise coefficients of correlation in a set of 10 securities for a two year period. Sort the securities by the coefficient of correlation, indicating the pair of securities corresponding to that row.
hist_qry9.sql	Determine the yearly dividends and annual yield (dividends/average closing price) for the past 3 years for all the stocks in the Russell 2000 index that did not split during that period. Use unadjusted prices since there were no splits to adjust for.

hist_qry1.sql

Get the closing price of a set of 10 stocks for a 10-year period and group into weekly, monthly, and yearly aggregates. For each aggregate period, determine the low, high, and average closing price value. Output is sorted by TRADING_SYMBOL and trade date.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL  YEAR   MON    WEEK  MAX_PRICE  MIN_PRICE        AVG_PRICE
AAA             2005     2       7      26.24      25.48           25.925
AAA             2005     2       8      26.49      25.46           25.972
AAA             2005     2       9      27.28      26.48           26.746
AAA             2005     2      10      26.47      26.47            26.47
AAA             2005     2  (NULL)      27.28      25.46      26.25066667
AAA             2005     3      10      26.46      25.93          26.2625
AAA             2005     3      11      26.19       24.9           25.566
AAA             2005     3      12      26.16      25.38           25.792
AAA             2005     3      13      25.38      24.37           24.924
AAA             2005     3      14      25.61      25.11            25.36
AAA             2005     3  (NULL)      26.46      24.37      25.56086957
AAA             2005     4      14      25.87      25.87            25.87
AAA             2005     4      15      26.39      25.34           25.866
```

```
AAA             2005     4      16       26.1      25.07           25.634
AAA             2005     4      17      25.81      24.81           25.314
AAA             2005     4      18      27.94      26.59           27.284
AAA             2005     4  (NULL)      27.94      24.81      26.01714286
AAA             2005     5      19       27.1      26.55           26.828
AAA             2005     5      20      27.08      26.29            26.71
AAA             2005     5      21      27.08      26.54           26.862
AAA             2005     5      22      28.45      27.07           27.836
AAA             2005     5      23      27.88       27.6            27.74
AAA             2005     5  (NULL)      28.45      26.29      27.12090909
AAA             2005     6      23      27.05      26.24            26.69
...
```

SQL

The following script contains the SQL statements for this query:

```
-- Get the closing price of a set of 10 stocks for a 10-year period and
-- group into weekly, monthly and yearly aggregates.
-- For each aggregate period determine the low, high and average closing
-- price value.
-- The output should be sorted by INSTRUMENT_ID and trade date.

commit
;

SELECT sh.TRADING_SYMBOL,
DATEPART(yy,sh.TRADE_DATE) AS YEAR,
DATEPART(mm,sh.TRADE_DATE) AS MON,
DATEPART(wk,sh.TRADE_DATE) AS WEEK,
MAX(sh.CLOSE_PRICE) AS MAX_PRICE,
MIN(sh.CLOSE_PRICE) AS MIN_PRICE,
AVG(sh.CLOSE_PRICE) AS AVG_PRICE
FROM  STOCK_HISTORY sh
WHERE
sh.TRADE_DATE BETWEEN '2005-02-08'
       and '2015-02-07'
and sh.TRADING_SYMBOL in
('AAA','AAB','AAC','AAD','AAE','AAF','AAG','AAH','AAI','AAJ' )
GROUP BY ROLLUP ( sh.TRADING_SYMBOL,
DATEPART(yy,sh.TRADE_DATE),
        DATEPART(mm,sh.TRADE_DATE),
DATEPART(wk,sh.TRADE_DATE))
;
```

hist_qry2.sql

Adjust all prices and volumes (prices are multiplied by the split factor and volumes are divided by the split factor) for a set of 1000 stocks to reflect the split events during a specified 300-day period, assuming that events occur before the first trade of the split date. These are called split-adjusted prices and volumes.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL  TRADE_DATE   H_PRC    L_PRC    C_PRC    O_PRC       VOL
AAA             2005-03-03  793.52   728      740.88   740.88   27.46428571
AAA             2005-03-04  749.84   651      726.04   740.88   29.10714286
AAA             2005-03-07  764.4    672      733.32   726.04   29.10714286
AAA             2005-03-08  786.24   637      726.04   733.32   26.17857143
AAA             2005-03-09  707      637      711.48   726.04   26.67857143
AAA             2005-03-10  700      624.96    697.2   711.48   24
AAA             2005-03-11  749      618.24    711.2    697.2   23.75
AAA             2005-03-14  742.56   658      732.48    711.2   25.85714286
AAA             2005-03-15  757.12   686       725.2   732.48   26.85714286
AAA             2005-03-16  728      644      710.64    725.2   28.46428571
AAA             2005-03-17  721      700      724.92   710.64   30.14285714
AAA             2005-03-18  735      644      717.64   724.92   28
AAA             2005-03-21  707      665      710.64   717.64   28
AAA             2005-03-22  763      637      710.64   710.64   27.71428571
AAA             2005-03-23  763      658.56   696.36   710.64   24.92857143
AAA             2005-03-24  692.16   638.4    682.36   696.36   25.14285714
AAA             2005-03-25  725.76   651.84   689.36   682.36   26.89285714
AAA             2005-03-28  728      624.96   703.08   689.36   26.89285714
AAA             2005-03-29  728      686      710.08   703.08   25
AAA             2005-03-30  700      672      710.08   710.08   23.25
AAA             2005-03-31  735      672      717.08   710.08   21.35714286
AAA             2005-04-01  735      651      724.36   717.08   21.75
AAA             2005-04-04  735      665      724.36   724.36   19.78571429
AAA             2005-04-05  756      644      724.36   724.36   19.35714286
AAA             2005-04-06  778.96   679      738.92   724.36   19.35714286
...
```

SQL

The following script contains the SQL statements for this query:

```
-- Adjust all prices and Volumes (prices are multiplied by the split factor
-- and Volumes are divided by the split factor) for a set of 1000 stocks to
-- reflect the split events during a specified 300 day period, assuming that
-- events occur before the first trade of the split date.
-- These are called split-adjusted prices and Volumes.

commit
;

SELECT B.TRADING_SYMBOL, TRADE_DATE,
B.HIGH_PRICE * IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR)) H_PRC,
B.LOW_PRICE * IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR)) L_PRC,
B.CLOSE_PRICE * IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR)) C_PRC,
B.OPEN_PRICE *IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR)) O_PRC,
B.Volume/IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR)) VOL
FROM STOCK_HISTORY AS B
       left outer join SPLIT_EVENT A
       on B.INSTRUMENT_ID = A.INSTRUMENT_ID
       AND B.TRADE_DATE < A.EFFECTIVE_DATE
WHERE  B.TRADING_SYMBOL BETWEEN 'AAA' AND 'BML'
       AND LENGTH(B.TRADING_SYMBOL) = 3
       and B.TRADE_DATE BETWEEN '2005-03-03'
       and '2005-12-03'
GROUP BY B.TRADING_SYMBOL,
TRADE_DATE ,
B.HIGH_PRICE,
B.LOW_PRICE,
B.CLOSE_PRICE,
B.OPEN_PRICE,
B.Volume
;
```

hist_qry3.sql

For each stock in a specified list of 1000 stocks, find the differences between the daily high and daily low on the day of each split event during a specified period.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL      D_PRICE      TRADE_DATE
AAD                    3.71      2005-09-14
ABV                   25.64      2005-09-06
ABW                    7.17      2005-10-03
ACK                       1      2005-09-29
ACQ                   12.09      2005-08-25
ADI                   11.82      2005-09-28
ADR                    12.8      2005-09-27
AED                    3.34      2005-09-02
AEE                   12.08      2005-08-25
AEX                   23.85      2005-08-25
AFD                    5.66      2005-09-12
AFD                    4.56      2005-08-31
AFD                    9.14      2005-09-09
AFY                    22.5      2005-08-22
AGU                   16.09      2005-08-15
AGV                    3.03      2005-09-19
AGV                     2.6      2005-09-20
AGV                    3.12      2005-09-21
AGV                    2.76      2005-09-02
AIE                    2.37      2005-09-12
AIE                    1.11      2005-09-05
AIE                    2.51      2005-09-08
AIW                    24.4      2005-09-27
AJL                   23.24      2005-08-24
AJL                   15.72      2005-08-25
...
```

SQL

The following script contains the SQL statements for this query.

```
-- For each stock in a specified list of 1000 stocks, find the differences
-- between the daily high and daily low on the day of each split event
-- during a specified period.
```

```
commit
;

SELECT sh.TRADING_SYMBOL, sh.HIGH_PRICE - sh.LOW_PRICE AS D_PRICE,
sh.TRADE_DATE
FROM   STOCK_HISTORY AS sh
inner join SPLIT_EVENT A
on sh.INSTRUMENT_ID = A.INSTRUMENT_ID
AND sh.TRADE_DATE = A.EFFECTIVE_DATE
WHERE  sh.TRADING_SYMBOL BETWEEN 'AAA' AND 'BML'
AND LENGTH(sh.TRADING_SYMBOL) = 3
and sh.TRADE_DATE BETWEEN '2005-08-04'
and '2005-10-04'
ORDER  BY sh.TRADING_SYMBOL
;
```

hist_qry4.sql

Calculate the value of the S&P 500 and Russell 2000 index for a specified day using unadjusted prices and the index composition of the two indexes on the specified day.

Output

The following output displays the rows returned by this query:

```
INDEX_NAME      AVERAGE_CLOSE_PRICE
Russell 2000       49.47026052
S&P500            54.44644
```

SQL

The following script contains the SQL statements for this query.

```
-- Calculate the value of the S&P500 and Russell 2000 index
-- for a specified day using unadjusted prices and the index composition
-- of the 2 indexes (see appendix for spec) on the specified day.

commit
;

Select ii.INDEX_NAME, AVG(sh.CLOSE_PRICE) as AVERAGE_CLOSE_PRICE
FROM MARKET_INDEX AS ii
inner join INDEX_CMPSTN AS ic
```

```
      on ii.MARKET_INDEX_ID = ic.MARKET_INDEX_ID
inner join STOCK_HISTORY AS sh
on ic.INSTRUMENT_ID = sh.INSTRUMENT_ID
and sh.TRADE_DATE = '2005-03-03'
WHERE ii.INDEX_NAME in ('S&P 500','Russell 2000')
GROUP BY
ii.INDEX_NAME
;
```

hist_qry5.sql

Find the 21-day and 5-day moving average price for a specified list of 1000 stocks during a 6-month period. (Use split-adjusted prices).

Output

The following output displays the first 24 rows returned by this query:

```
TRADING_SYMBOL  TRADE_DATE      AVG_5DAY     AVG_21DAY
AAA             2010-06-01         381.72   383.7142857
AAB             2010-06-01         397.28    402.472381
AAC             2010-06-01        1452.32   1393.725714
AAD             2010-06-01   223.0616667   231.3666667
AAE             2010-06-01         481.05   482.5571429
AAF             2010-06-01   105.8666667   101.4019048
AAG             2010-06-01   288.1816667   285.0571429
AAH             2010-06-01        1743.07       1816.68
AAI             2010-06-01         105.43   103.2509524
AAJ             2010-06-01   487.2633333   463.6709524
AAK             2010-06-01   1081.116667   1035.864762
AAL             2010-06-01   326.8266667    317.287619
AAM             2010-06-01   688.7283333   682.9061905
AAN             2010-06-01        1246.46   1162.657143
AAO             2010-06-01          911.6         923.7
AAP             2010-06-01         700.46   696.4857143
AAQ             2010-06-01         320.64   319.6285714
AAR             2010-06-01   125.5466667   127.5790476
AAS             2010-06-01   2397.198333   2447.154762
AAT             2010-06-01   70.21666667   66.85952381
AAU             2010-06-01        1447.55   1429.114286
AAV             2010-06-01  199.2166667   328.3333333
AAW             2010-06-01        1921.27   1846.160952
AAX             2010-06-01        1792.05   1764.205238
...
```

SQL

The following script contains the SQL statements for this query:

```
-- Find the 21-day and 5-day moving average price for a specified
-- list of 1000 stocks during a 6-month period. (Use split adjusted prices) */


truncate table hist_temp;
commit;


insert hist_temp
SELECT number(),B.INSTRUMENT_ID, B.TRADING_SYMBOL,B.TRADE_DATE, B.CLOSE_PRICE,
IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR))
FROM STOCK_HISTORY AS B
left outer join SPLIT_EVENT as A
on B.INSTRUMENT_ID = A.INSTRUMENT_ID
AND B.TRADE_DATE < A.EFFECTIVE_DATE
WHERE B.TRADING_SYMBOL BETWEEN 'AAA' AND 'BML'
AND LENGTH(B.TRADING_SYMBOL) = 3
and B.TRADE_DATE >= DATEADD(DAY,-28,'2010-06-01')
and B.TRADE_DATE <= '2010-12-01'
GROUP BY B.INSTRUMENT_ID, B.TRADING_SYMBOL,
B.TRADE_DATE, B.CLOSE_PRICE
ORDER BY B.INSTRUMENT_ID,
B.TRADE_DATE;


SELECT x.TRADING_SYMBOL, x.TRADE_DATE, AVG_5DAY , AVG_21DAY
FROM (SELECT B.INSTRUMENT_ID, B.TRADING_SYMBOL, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR ) avg_5day
FROM hist_temp as B
left outer join hist_temp as C
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 5 and b.row_nbr
Where B.TRADE_DATE >= '2010-06-01'
GROUP BY B.INSTRUMENT_ID, B.TRADING_SYMBOL,
B.TRADE_DATE) x,
(SELECT B.INSTRUMENT_ID, B.TRADING_SYMBOL, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR ) avg_21day
FROM hist_temp as B
left outer join hist_temp as C
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 21 and b.row_nbr

Where B.TRADE_DATE >= '2010-06-01'
GROUP BY B.INSTRUMENT_ID, B.TRADING_SYMBOL,
```

```
B.TRADE_DATE) y
where x.INSTRUMENT_ID = y.INSTRUMENT_ID
and x.TRADE_DATE = y.TRADE_DATE
;
```

hist_qry6.sql

(Based on the previous query.) Find the points (specific days) when the 5-day moving average intersects the 21-day moving average for these stocks. Output is sorted by TRADING_SYMBOL and trade date.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL  TRADE_DATE      DAY_5     DAY_21   PREV_DAY5   PREV_DAY21
AAG             2012-06-04    71.6416    71.7977     71.8916      71.8238
ABI             2012-06-04   303.3733   301.6436    300.4533     301.4476
ABX             2012-06-04     1264.2  1260.7418     1258.16    1259.9085
ACD             2012-06-04      171.7   170.2854      169.16     169.8628
ACP             2012-06-04   103.3083   103.0081    102.2816       102.79
AGT             2012-06-04     855.95    889.129      952.56     915.4057
AHS             2012-06-04     1156.5  1157.2772   1158.6666    1157.4333
AIC             2012-06-04    412.875   412.1018      410.85     411.6085
AJE             2012-06-04   140.5466   140.5545    141.2666     140.7352
AKA             2012-06-04      41.64    41.5381       41.44       41.478
AKU             2012-06-04    804.225   801.3395       798.9     800.4385
ALE             2012-06-04   138.6116   137.6359     136.535     137.2133
ALO             2012-06-04     484.44   483.0109    479.6133     481.9047
AML             2012-06-04     272.68   272.4145    271.3333     272.3161
AOC             2012-06-04   519.5166   543.8159     579.775     559.9976
APZ             2012-06-04   272.6916   272.2113    270.4333     271.6261
AQQ             2012-06-04  1034.8333  1031.1181   1026.4333    1030.2333
AQT             2012-06-04     126.89     125.79      125.21     125.4542
ARD             2012-06-04  1074.6266    1072.01   1067.5866    1070.8709
ARU             2012-06-04   1013.475  1063.6445     1136.64    1095.3428
ASA             2012-06-04      406.4   405.4436    405.1466     405.3485
ASH             2012-06-04     310.75    309.499       306.65     308.8228
ATF             2012-06-04   550.7866    550.269       549.92     550.3809
ATK             2012-06-04   136.2716     137.43    137.6833     137.5733
ATQ             2012-06-04    42.1333    42.3045       42.35       42.319
...
```

SQL

The following script contains the SQL statements for this query:

```
--(Based on the previous query) Find the points (specific days) when the
--5-day moving average intersects the 21-day moving average for these stocks.
--The output is to be sorted by INSTRUMENT_ID and date.

truncate table hist_temp;
truncate table hist6_temp;
commit;

insert hist_temp
SELECT number(),B.INSTRUMENT_ID, B.TRADING_SYMBOL, B.TRADE_DATE,B.CLOSE_PRICE,
IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR))
FROM STOCK_HISTORY AS B
left outer join SPLIT_EVENT as A
on B.INSTRUMENT_ID = A.INSTRUMENT_ID
AND B.TRADE_DATE < A.EFFECTIVE_DATE
WHERE B.TRADING_SYMBOL BETWEEN 'AAA' AND 'BML'
AND LENGTH(B.TRADING_SYMBOL) = 3
and B.TRADE_DATE >= DATEADD(DAY,-28,'2012-06-01')
and B.TRADE_DATE <= '2012-12-01'
GROUP BY B.INSTRUMENT_ID,TRADING_SYMBOL,
B.TRADE_DATE, B.CLOSE_PRICE
ORDER BY B.INSTRUMENT_ID,
B.TRADE_DATE;

Insert hist6_temp
SELECT number(),x.INSTRUMENT_ID, x.TRADING_SYMBOL, x.TRADE_DATE, avg_5day,
avg_21day
FROM (SELECT B.INSTRUMENT_ID, B.TRADING_SYMBOL, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR)  avg_5day
FROM hist_temp as B
left outer join hist_temp as C
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 5 and b.row_nbr
Where B.TRADE_DATE >= '2012-06-01'
GROUP BY B.INSTRUMENT_ID, B.TRADING_SYMBOL,
B.TRADE_DATE) x,
(SELECT B.INSTRUMENT_ID, B.TRADING_SYMBOL, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR) avg_21day
FROM hist_temp as B
left outer join hist_temp as C
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 21 and b.row_nbr
Where B.TRADE_DATE >= '2012-06-01'
```

```
GROUP BY B.INSTRUMENT_ID,B.TRADING_SYMBOL,
B.TRADE_DATE) y
where x.INSTRUMENT_ID = y.INSTRUMENT_ID
and x.TRADE_DATE = y.TRADE_DATE
order by x.INSTRUMENT_ID, x.TRADE_DATE;


select z.TRADING_SYMBOL, z.TRADE_DATE, DAY_5, DAY_21, PREV_DAY5, PREV_DAY21
from (SELECT a.INSTRUMENT_ID, a.TRADING_SYMBOL, a.TRADE_DATE,  avg(b.avg_21day)
as prev_day21
from hist6_temp a, hist6_temp b
where a.INSTRUMENT_ID = b.INSTRUMENT_ID
and b.row_nbr between a.row_nbr - 2 and a.row_nbr - 1
group by a.INSTRUMENT_ID, a.TRADING_SYMBOL, a.TRADE_DATE) x,
(SELECT a.INSTRUMENT_ID, a.TRADING_SYMBOL, a.TRADE_DATE,  avg(b.avg_5day) as
prev_day5
from hist6_temp a, hist6_temp b
where a.INSTRUMENT_ID = b.INSTRUMENT_ID
and b.row_nbr between a.row_nbr - 2 and a.row_nbr - 1
group by a.INSTRUMENT_ID, a.TRADING_SYMBOL, a.TRADE_DATE) y,
(SELECT INSTRUMENT_ID, TRADING_SYMBOL, TRADE_DATE, avg_5day as day_5, avg_21day
as day_21
from hist6_temp) z
where z.INSTRUMENT_ID = x.INSTRUMENT_ID
and z.TRADE_DATE = x.TRADE_DATE
and z.INSTRUMENT_ID = y.INSTRUMENT_ID
and z.TRADE_DATE = y.TRADE_DATE
and sign(day_21-day_5) * sign(prev_day21-prev_day5) < 0;
```

hist_qry7.sql

Determine the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (that is, allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average, the complete allocation for that stock is invested, and when the 20-day moving average crosses below the 5-month moving average, the entire position is sold. The trades are made on the closing price of the trading day.

Output

The following output displays the result of this query:

```
STOCK_VALUE
289690.0039
```

SQL

The following script contains the SQL statements for this query:

```
BEGIN

-- Determine the value of $100,000 now if 1 year ago it was invested
-- equally in 10 specified stocks (i.e. allocation for each stock is $10,000).
-- The trading strategy is: When the 20-day moving avg crosses over the
-- 5 month moving avg the complete allocation for that stock is invested
-- and when the 20-day moving avg crosses below the 5 month moving avg
-- the entire position is sold.  The trades happen on the closing price
-- of the trading day.

truncate table hist_temp;
truncate table hist7_temp;
commit;

insert hist_temp
SELECT number(),B.INSTRUMENT_ID,B.TRADING_SYMBOL, B.TRADE_DATE,b.CLOSE_PRICE,
IFNULL(sum(A.SPLIT_FACTOR),1,sum(A.SPLIT_FACTOR))
FROM STOCK_HISTORY AS B
left outer join SPLIT_EVENT as A
on B.INSTRUMENT_ID = A.INSTRUMENT_ID
AND B.TRADE_DATE < A.EFFECTIVE_DATE
WHERE B.INSTRUMENT_ID BETWEEN 11 and 20
and B.TRADE_DATE >= DATEADD(DAY,-160,'2012-06-01')
and B.TRADE_DATE <= '2012-12-01'
GROUP BY B.INSTRUMENT_ID,B.TRADING_SYMBOL,
B.TRADE_DATE, B.CLOSE_PRICE
ORDER BY B.INSTRUMENT_ID,
B.TRADE_DATE;


Insert hist7_temp
SELECT number(),x.INSTRUMENT_ID, x.TRADE_DATE, avg_5mth , avg_21day
FROM (SELECT B.INSTRUMENT_ID, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR) avg_5mth
FROM hist_temp as B
left outer join hist_temp as C
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 160 and b.row_nbr
GROUP BY B.INSTRUMENT_ID,
B.TRADE_DATE) x,
(SELECT B.INSTRUMENT_ID, B.TRADE_DATE,
AVG(C.CLOSE_PRICE * B.SPLIT_FACTOR) avg_21day
FROM hist_temp as B
left outer join hist_temp as C
```

```
on B.INSTRUMENT_ID = C.INSTRUMENT_ID
and c.row_nbr BETWEEN b.row_nbr - 21 and b.row_nbr
GROUP BY B.INSTRUMENT_ID,
B.TRADE_DATE) y
where x.INSTRUMENT_ID = y.INSTRUMENT_ID
and x.TRADE_DATE = y.TRADE_DATE
order by x.INSTRUMENT_ID, x.TRADE_DATE;


select z.INSTRUMENT_ID, z.TRADE_DATE, diff, td2, diff2, pre_diff into
#hist7_temp
from (SELECT a.INSTRUMENT_ID, a.TRADE_DATE,  b.avg_21day - b.avg_5mth as
pre_diff
from hist7_temp a, hist7_temp b
where a.INSTRUMENT_ID = b.INSTRUMENT_ID
and b.row_nbr = a.row_nbr - 1
) x,
 (SELECT a.INSTRUMENT_ID, a.TRADE_DATE, b.TRADE_DATE as td2,
b.avg_21day - b.avg_5mth as diff2
from hist7_temp a, hist7_temp b
where a.INSTRUMENT_ID = b.INSTRUMENT_ID
and b.row_nbr = a.row_nbr + 1
) y,
(SELECT INSTRUMENT_ID, TRADE_DATE, avg_21day - avg_5mth as diff
from hist7_temp) z
where z.INSTRUMENT_ID = x.INSTRUMENT_ID
and z.TRADE_DATE = x.TRADE_DATE
and z.INSTRUMENT_ID = y.INSTRUMENT_ID
and z.TRADE_DATE = y.TRADE_DATE
and pre_diff*diff <=0
and NOT (pre_diff=0 and diff=0);

select sum(mp2.CLOSE_PRICE * (10000/mp1.CLOSE_PRICE)) as STOCK_VALUE
from #hist7_temp t7, STOCK_HISTORY mp1, STOCK_HISTORY mp2
where t7.INSTRUMENT_ID = mp1.INSTRUMENT_ID
and t7.INSTRUMENT_ID = mp2.INSTRUMENT_ID
and t7.TRADE_DATE = mp1.TRADE_DATE
and t7.td2 = mp2.TRADE_DATE;
END
```

hist_qry8.sql

Find the pair-wise coefficients of correlation in a set of 10 securities for a two-year period. Sort the securities by the coefficient of correlation, indicating the pair of securities corresponding to that row.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL    TRADING_SYMBOL    CORRELATION
AAI               AAH                 -4.29E+09
AAI               AAC                 -3.65E+09
AAF               AAH                 -3.04E+09
AAI               AAB                 -2.92E+09
AAF               AAC                 -2.58E+09
AAD               AAH                 -2.57E+09
AAG               AAH                 -2.56E+09
AAD               AAC                 -2.24E+09
AAG               AAC                 -2.22E+09
AAF               AAB                 -2.07E+09
AAJ               AAH                 -1.80E+09
AAA               AAH                 -1.79E+09
AAD               AAB                 -1.72E+09
AAG               AAB                 -1.71E+09
AAE               AAH                 -1.68E+09
AAJ               AAC                 -1.67E+09
AAE               AAC                 -1.63E+09
AAA               AAC                 -1.62E+09
AAI               AAE                 -1.29E+09
AAA               AAB                 -1.21E+09
AAJ               AAB                 -1.15E+09
AAE               AAB                 -1.11E+09
AAF               AAE                 -9.21E+08
AAI               AAJ                 -8.34E+08
AAG               AAE                 -6.35E+08
...
```

SQL

The following script contains the SQL statements for this query:

```
-- Find the pair-wise coefficients of correlation in a set of 10 securities
-- for a 2 year period. Sort the securities by the coefficient of correlation,
-- indicating the pair of securities corresponding to that row.

commit
;

SELECT a.TRADING_SYMBOL,b.TRADING_SYMBOL,
(Count(*) * sum(a.CLOSE_PRICE * b.CLOSE_PRICE) - sum(a.CLOSE_PRICE)
* sum(b.CLOSE_PRICE)/sqrt(count(*) * sum(a.CLOSE_PRICE * a.CLOSE_PRICE )
- (sum(a.CLOSE_PRICE) * sum(a.CLOSE_PRICE)))
* sqrt(count(*) * sum(b.CLOSE_PRICE * b.CLOSE_PRICE )
- (sum(b.CLOSE_PRICE) * sum(b.CLOSE_PRICE)))) as CORRELATION
from (Select TRADING_SYMBOL, TRADE_DATE,CLOSE_PRICE
from
STOCK_HISTORY AS B
WHERE B.TRADING_SYMBOL BETWEEN 'AAA' AND 'AAJ'
AND LENGTH(B.TRADING_SYMBOL) = 3
and B.TRADE_DATE BETWEEN '2005-02-08'
and '2007-02-07'
) a,
(Select TRADING_SYMBOL, TRADE_DATE,CLOSE_PRICE
from
STOCK_HISTORY AS B
WHERE B.TRADING_SYMBOL BETWEEN 'AAA' AND 'AAJ'
AND LENGTH(B.TRADING_SYMBOL) = 3
and B.TRADE_DATE BETWEEN '2005-02-08'
and '2007-02-07'   ) b
where a.TRADE_DATE = b.TRADE_DATE
group by a.TRADING_SYMBOL, b.TRADING_SYMBOL
order by correlation
;
```

hist_qry9.sql

Determine the yearly dividends and annual yield (dividends/average closing price) for the past 3 years for all the stocks in the Russell 2000 index that did not split during that period. Use unadjusted prices since there were no splits to adjust for.

Output

The following output displays the first 25 rows returned by this query:

```
TRADING_SYMBOL     YEAR         DIVIDEND
AXZ                2005      1.581407226
AYB                2006      1.381549972
AYG                2006      0.367898983
AYI                2006      5.216678323
AYN                2006      4.028782552
AYQ                2006      3450.287281
AYS                2006      3.694855018
AYU                2006      1.271758802
AYW                2006      2.619291171
AZA                2006       2.28069406
AZG                2006      0.990853336
AZH                2006      27.14814009
AZI                2006      1.061584765
AZJ                2005      0.604295833
AZK                2005      22.93923892
AZL                2005      0.902766204
AZN                2006      1.088005743
AZO                2006      1.789226001
AZV                2006      2.247136542
AZZ                2005      0.568150577
BAK                2006      1.266999535
BAL                2005       0.68438775
BAM                2005      0.499661767
BAN                2005       5.72367411
BAR                2006      1.146197323
...
```

SQL

The following script contains the SQL statements for this query:

```
--Determine the yearly dividends and annual yield (dividends/average closing
--price) for the past 3 years for all the stocks in the Russell 2000 index that
--did not split during that period. Use unadjusted prices since there were no
--stock_split to adjust for.

commit
;

SELECT sh.TRADING_SYMBOL, DATEPART(yy,TRADE_DATE) AS YEAR,
SUM(dividend_value)/AVG(CLOSE_PRICE) as DIVIDEND
FROM  MARKET_INDEX mi
inner join INDEX_CMPSTN AS ic
on mi.MARKET_INDEX_ID = ic.MARKET_INDEX_ID
inner join STOCK_HISTORY AS sh
on ic.INSTRUMENT_ID = sh.INSTRUMENT_ID
AND sh.TRADE_DATE BETWEEN '2005-04-04' and '2008-04-03'
inner join DIVIDEND_EVENT de
on de.INSTRUMENT_ID= sh.INSTRUMENT_ID
AND DATEPART(yy,TRADE_DATE)=DATEPART(yy,ANNOUNCED_DATE)
AND de.INSTRUMENT_ID NOT IN (SELECT se.INSTRUMENT_ID
FROM   SPLIT_EVENT se
WHERE  sh.INSTRUMENT_ID=se.INSTRUMENT_ID
AND   DATEPART(yy,TRADE_DATE)=
DATEPART(yy,EFFECTIVE_DATE))
WHERE mi.INDEX_NAME ='Russell 2000'
GROUP  BY sh.TRADING_SYMBOL,
       DATEPART(yy,TRADE_DATE)
order by sh.TRADING_SYMBOL,
       DATEPART(yy,TRADE_DATE)
;
```

Index

M

P

Q

R

S

T

V