RED BRICK®
The Data Warehouse Company®

RED BRICK® WAREHOUSE
Version 5.1

RED BRICK DATA MINE™ USER'S GUIDE

5.1

Revision number: 1
January, 1998
Part number: 456051

Red Brick Systems, Inc.
485 Alberto Way
Los Gatos, California 95032
USA
Telephone: +1 408 399 3200
+1 800 777 2585

RED BRICK®
The Data Warehouse Company®

Contents

About This Document

Purpose

This guide provides information needed to use the Red Brick Data Mine™ option to perform the data-mining process (discovery, understanding, and prediction) on data contained in a Red Brick® Warehouse database. It includes information necessary for the effective use of Red Brick Data Mine, as well as syntax definitions and procedural descriptions.

Audience

The intended users for this guide are:

- Model builders, who will use Red Brick Data Mine to build and refine models.
- Analysts, who will use the models to study the relationships and correlations between the inputs and outputs, and who will use “validated” models to make predictions on data.

Organization

This guide is organized as follows:

- Chapter 1, "Introduction to Integrated Data Mining Applications," provides a high-level view of what data mining is and how it can be useful. The chapter includes a general description of data mining, a description of the Red Brick Data Mine architecture, and a simple example of the data mining process.
- Chapter 2, "Defining a Model," describes the how to define a model for data mining and discusses the CREATE MODEL, CREATE MODEL LIKE, and DROP MODEL statements—how to use SQL to create a model, how to choose inputs and outputs, and how to split data into appropriate "buckets."
- Chapter 3, "Performing Discovery on a Model," discusses the process by which the model is populated with sample data using the INSERT INTO *model* command, including information on how to choose and generate a sample.
- Chapter 4, "Understanding the Model," discusses the UNDERSTAND sub-table and the use of the SELECT statement to evaluate the model.
- Chapter 5, "Making Predictions," discusses how to use the PREDICT sub-table to make predictions using the model.
- Chapter 6, "Refining the Model," describes how to make changes to the model so it will more accurately reflect the needs of the user. This chapter also discusses the ALTER MODEL statement, including how to add to, delete from, and change the model.
- Chapter 7, "Data Mining: A Detailed Example," uses a single example to illustrate the process of building a model and writing case and batch prediction queries. The example is based on data from the sample Aroma database.
- Appendix A, "Red Brick Data Mine Reference," contains reference information for the Red Brick Data Mine system tables.

Related Documentation

The standard documentation set for Red Brick Warehouse includes the following documents:

Installation and Configuration Guide	Installation and configuration information, as well as platform-specific material, about Red Brick Warehouse and related products. Customized for either UNIX-based or Windows NT systems.
Warehouse Administrator's Guide	Description of warehouse architecture, supported schemas, and other concepts relevant to warehouse databases. Procedural information for designing and implementing a warehouse database, maintaining a database, and tuning a database for performance. Includes a description of the system tables and the configuration file (*rbw.config*). Customized for either UNIX-based or Windows NT systems.
Table Management Utility Reference Guide	Description of the Table Management Utility, including all activities related to loading and maintaining data. Also includes information about data replication and the *rb_cm* copy management utility.
SQL Reference Guide	The complete language reference for the Red Brick Systems SQL implementation and RISQL® extensions for warehouse databases.
SQL Self-Study Guide	An example-based review of SQL and introduction to the RISQL extensions, the macro facility, and Aroma, the sample database.
RISQL Entry Tool and RISQL Reporter User's Guide	The complete guide to the RISQL Entry Tool, a command-line tool used to enter SQL statements, and the RISQL Reporter, an enhanced version of the RISQL Entry Tool with report-formatting capabilities.
Messages and Codes Reference Guide	The complete listing of all informational, warning, and error messages generated by warehouse products, including probable causes and recommended responses. Also includes event log messages that are written to the log files.
Release Notes	Information pertinent to the current release that was unavailable when the documents were printed.

In addition to the standard documentation set, the following documents are included for specific sites:

Red Brick Vista User's Guide	Description of the Red Brick Vista™ aggregate navigation and advice system, including procedures for rewriting queries and getting advice on the best set of aggregate tables and views to create. Includes detailed examples of queries whose performance can be dramatically increased by using aggregate navigation.
SQL-BackTrack for Red Brick Warehouse User's Guide	The complete guide to SQL-BackTrack™ for Red Brick Warehouse, a command-line interface for backing up and recovering warehouse databases. Includes procedures for defining backup configuration files, performing online and checkpoint backups, and recovering the database to a consistent state.
Client Connector Pack Installation Guide	Procedures for installing and configuring the Red Brick ODBC Driver, the RISQL Entry Tool, and the RISQL Reporter on client systems. Included for those sites that purchase the Client Connector Pack.
ODBC Connectivity Guide	Includes information about ODBC conformance levels as well as instructions for compiling and linking an ODBC application using the Red Brick ODBClib SDK.
Red Brick Data Mine User's Guide	Description of the data mining process, and procedural information for using the Red Brick Data Mine™ SQL-based interface to find hidden or unpredictable relationships among the data in a data set. Included for those sites that purchase the Red Brick Data Mine option.
Red Brick Data Mine Builder™ User's Guide	Description of the data mining process, and procedural information for performing data mining using Red Brick's GUI-based product in a Microsoft Windows environment.

Additional references you might find helpful include:

- An introductory-level book on SQL
- An introductory-level book on relational databases
- Documentation for your hardware platform and operating system

Online Documentation

The English version of the Red Brick Warehouse documentation is also available in Adobe Acrobat format (PDF) on a separate CD-ROM.

Conventions

Throughout Red Brick Systems technical publications, the following notation and syntax conventions are used:

- Computer input and output, including commands, code, and examples, appear in `Courier`.
- Information that you enter or that is being emphasized in an example appears in **`Courier bold`** to help you distinguish it from other text.
- Filenames, system-level commands, variables, and document titles appear in *Palatino italic* or *`Courier italic`*, depending on the context.
- Names of database tables and columns are capitalized (Sales table, Dollars column). Names of system tables and columns are in all uppercase (RBW_INDEXES table, TNAME column).

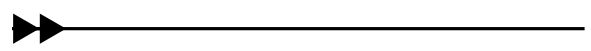
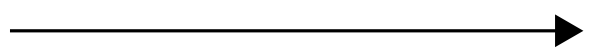
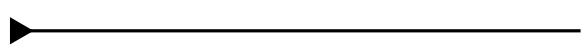
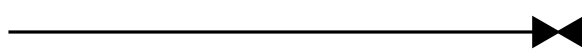
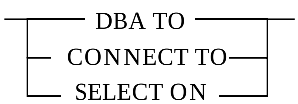
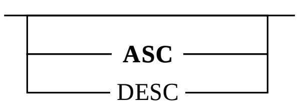
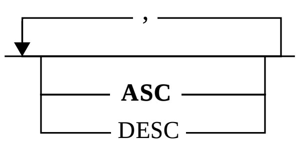
Syntax Notation

This guide uses the following conventions to describe the syntax of operating-system commands:

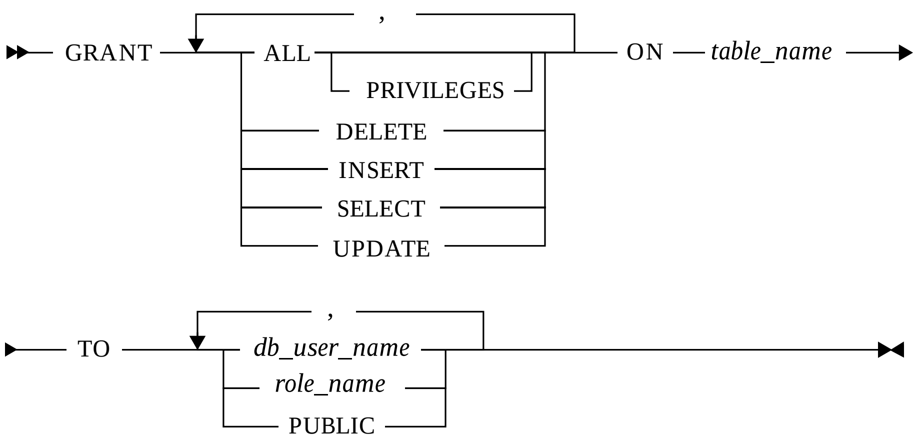
Command Element	**Example**	**Convention**
Values and parameters	*table_name*	Items that you replace with an appropriate name, value, or expression are in *italic* type style.
Optional items	[ ]	Optional items are enclosed by square brackets. Do not type the brackets.
Choices	ONE | TWO	Choices are separated by vertical lines; choose one if desired.
Required choices	{ONE | TWO}	Required choices are enclosed in braces; choose one. Do not type the braces.
Default values	<u>ONE</u> | TWO	Default values are underlined, except in graphics where they are in **bold** type style.
Repeating items	name, ...	Items that can be repeated are followed by a comma and an ellipsis. Separate the items with commas.
Language elements	( ) , ; .	Parentheses, commas, semicolons, and periods are language elements. Use them exactly as shown.

Syntax Diagrams

This guide uses diagrams built with the following components to describe the syntax for statements and all commands other than system-level commands:

Component	Meaning
	Statement begins.
	Statement syntax continues on next line. Syntax elements other than complete statements end with this symbol.
	Statement continues from previous line. Syntax elements other than complete statements begin with this symbol.
	Statement ends.
	Required item in statement.
	Optional item.
	Required item with choice. One and only one item must be present.
	Optional item with choice. If a default value exists, it is printed in **bold**.
	Optional items. Several items are allowed; a comma must precede each repetition.

The syntax elements shown above are combined to form a diagram as follows:



Complex syntax diagrams such as the one for the following statement are repeated as point-of-reference aids for the detailed diagrams of their components. Point-of-reference diagrams are indicated by their shadowed corners, gray lines, and reduced size:

Keywords and Punctuation

Keywords are words reserved for statements and all commands except system-level commands. When a keyword appears in a syntax diagram, it is shown in uppercase. You can write a keyword in upper- or lowercase, but you must spell the keyword exactly as it appears in the syntax diagram.

Any punctuation that occurs in a syntax diagram must also be included in your statements and commands exactly as shown in the diagram.

Identifiers and Names

Metavariables serve as placeholders for identifiers and names in the syntax diagrams and examples. A metavariable can be replaced by an arbitrary name, identifier, or literal, depending on the context. Metavariables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a metavariable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses metavariables to illustrate the general form of a simple SELECT statement:

▸▸— SELECT — *column_name* — FROM — *table_name* ——▸◂

When you write a SELECT statement of this form, you replace the metavariables *column_name* and *table_name* with the name of a specific column and table.

Customer Support

Please review the following information before contacting the Customer Support Center at Red Brick Systems.

Support Solutions Warehouse

The Support Solutions Warehouse is the Customer Support Center's external web site, an online resource that registered Red Brick customers can use to:

- Submit new cases.
- Read release notes.
- Find answers to frequently asked questions (FAQs).
- Search the Problems and Solutions database.

To use the Support Solutions Warehouse, point your web browser to the following URL and enter your registered username and password:

http://www.redbrick.com/RBCustomer/index.htm

If you do not have a registered username and password, contact the Customer Support Center by telephone, fax, or e-mail.

General and Technical Questions

If you have general sales-related questions or technical questions about Red Brick products or services, contact Red Brick Systems as follows:

Telephone	
General Questions	(408) 399-3200 or 1 (800) 777-2585
Technical Questions	(408) 399-7100 or 1 (800) 727-1866
FAX	
General Questions	(408) 399-3277
Technical Questions	(408) 399-3297
Internet e-mail	
General Questions	info@redbrick.com
Technical Questions	support@redbrick.com
World Wide Web	www.redbrick.com

Existing Cases

If you want to inquire about the status of an existing case, please have the case number ready. The case number will always be given to you by the support engineer who logs the case or first contacts you. This number is used to keep track of all the activities performed during the resolution of each problem.

New Cases

If you want to log a new case, please have the following information ready:

- Red Brick Warehouse version
- Platform and operating-system version
- Error messages returned by Red Brick Warehouse or the operating system
- Concise description of the problem, including any commands or operations performed prior to the occurrence of the error message
- List of Red Brick Warehouse and/or operating-system configuration changes made prior to the occurrence of the error message

If you think the problem concerns client-server connectivity, please have the following additional information ready:

- Name and version of the client tool in use
- Version of Red Brick ODBC Driver in use (if applicable)
- Name and version of client network and/or TCP/IP stack in use
- Error messages returned by the client application
- Warehouse and client locale specifications

Troubleshooting Tips

You can often reduce the time it takes to close your case by providing the smallest possible reproducible example of your problem. The more you can isolate the cause of the problem, the more quickly the support engineer can help you resolve it.

- For SQL query problems, try removing columns or functions, or restating WHERE, ORDER BY, or GROUP BY clauses until you can isolate the part of the statement causing the problem.
- For TMU load problems, verify the datatype mapping between the source file and the target table to ensure compatibility. Try loading a small test set of data to determine whether the problem concerns volume or data format.
- For connectivity problems, verify that the network is up and running by issuing the *rbping* command from the client to the host. If possible, try another client tool to see if the same problem arises.

Documentation Questions and Comments

If you have questions or comments about the Red Brick Warehouse documentation, please contact the Technical Publications Department at Red Brick Systems as follows:

Telephone	+1 408 399 3200 +1 800 727 1866 (USA only)
Internet e-mail	docs@redbrick.com

1

Introduction to Integrated Data Mining Applications

Data mining is the process of discovering unexpected patterns or trends in existing data so that you can use the discovered knowledge to manage the business more effectively. For example, a typical application might study the demographic information known about a list of customers to create a profile of people most likely to buy a given product, respond to a direct mail campaign, or default on a loan.

Because the Red Brick Data Mine™ option is fully integrated with the Red Brick® Warehouse database engine, the same business analysts who write assumption-driven queries have easy access to this kind of discovery-based application; moreover, the application is constructed with the same data definition language (SQL) as the database tables and uses that language to mine data from the same source.

This integration produces a powerful decision support system in which patterns revealed by data mining lead to further investigation with standard queries or, conversely, the results of standard queries provoke a decision to mine the data.

This chapter discusses data mining and the Red Brick Data Mine option in the following sections:

- Data Mining Concepts
- Integrated Data Mining Architecture
- The Data Mining Process—A Simple Example

Data Mining Concepts

Despite its name, *data mining* has little to do with "drill-down" queries; these types of queries are feasible in a data warehouse that has no data mining component. The difference between a data mining exercise and decision support queries lies in the analyst's approach to the data: Standard queries are assumption-driven, whereas data mining is discovery-based.

For example, an analyst who writes a query that compares sales last year to sales this year is looking for an accurate result to a routine business question, while an analyst who mines sales data is looking for patterns or trends that might be interesting and useful to understand. If the sales comparison query reveals that sales this year are up 200% from last year in some stores but only 100% in others, for example, the analyst might want to mine the data to discover any unexpected reasons for this discrepancy.

In other words, the data warehouse application helps analysts understand *what* happened, while data mining applications attempt to tell analysts *why* it happened. It is useful for a retailer to know, for example, what factor had the greatest impact on sales of a given product, and whether the retailer can control that factor or take advantage of that knowledge. Any number of factors can influence sales—price, style, packaging, promotions, placement in the store, season, weather, day of week, time of day, and so on. Without requiring analysts to ask explicit questions about each possible factor, a data mining application can read very large volumes of data and discover both obvious and hidden trends.

Data Mining Techniques

Given the general distinction between data mining exercises and data-warehousing queries, this section describes the specific goals and techniques behind data mining applications.

Although all data mining applications try to uncover hidden patterns, such patterns exist in different forms. For example, if two products tend to be bought in a single transaction, this is an example of an *associative* pattern. If one product tends to be bought subsequent to the purchase of another product, this is an example of a *sequential* pattern. Because different applications look for different kinds of patterns and trends, there are different approaches to data mining and different technologies to support those approaches.

Four common data mining techniques are known as *cluster analysis*, *linkage analysis*, *time-series analysis*, and *categorization analysis*.

Cluster Analysis

Cluster analysis identifies associations among data points. For example, an application might discover that meals charged on business-issued Gold Cards are typically purchased on weekdays and have an average value of $200. The same application might find that Platinum Cards are used more frequently for meals purchased on weekends and have an average value of $100.

Linkage Analysis

Linkage analysis typically identifies pairs of data points that have strong associations. For example, an application might find that beer and pretzels or soda and ice are routinely purchased together. The value of this kind of application is that far less obvious pairs can also be found—and this knowledge can be used in promotion or product-positioning strategies.

Time-Series Analysis

Time-series analysis looks for related sequences of events. For example, promotional toys might be bought by the same customers who recently rented a popular children's movie and gardening equipment might be purchased by recent buyers of a first home. Again, the value of this kind of application is that less predictable sequences of events can be discovered as well.

Categorization Analysis

Categorization analysis is a composite approach to data mining that embraces the previous three techniques and has the broadest appeal to business applications in different industries.

Categorization analysis attempts to explain the influence that a number of factors might have on one specific outcome. For example, given a financial profile of a loan candidate that includes data such as occupation, income, and monthly expenses, what is the likely disposition of a specific loan? Or given demographic information such as age, gender, educational background, and so on for a group of customers who have purchased a specific product, what is the profile of the perfect prospective customer?

The data mining exercise will reveal which factors, among all those used in the exercise, had the greatest influence. The model might return some non-surprising facts—more athletic shoes were sold during the 25%-off sale than at the normal list price; some moderate surprises—more athletic shoes are sold on Friday and Saturday than during the rest of the week combined; and some non-intuitive new insights—more athletic shoes are sold when white tube socks are prominently displayed on 2-for-1 sale than when they are not.

A data mining exercise consists of two main phases: *building a model* and *predicting future results*. A model defines the influencing factors (input data) and the potential outcomes (output values), and how they are selected and mapped from a larger pool of data that already exists in database tables. The set of inputs, which might be very large (100 or more), is a list of factors that *might* influence the output. The result of the data mining exercise is a much smaller list of inputs that individually or in combination with other inputs *do* influence the output.

In turn, the model gives you the ability to *predict* an output given a set of input characteristics. For example, if the model accurately reflects the tendency of customers with certain attributes (age, gender, income, and so on) to purchase a luxury automobile, the discovered results can be compared with a list of prospective buyers to determine those most likely to make a purchase in the near future. Because models can be used independently or in conjunction with query analysis, a warehouse query might be issued to generate a promotional mailing list from a Customer table. (With the Red Brick Data Mine option, data mining sub-tables can be joined to any base table or view in a Red Brick Warehouse database.)

Typical Data Mining Applications

Categorization analysis has useful applications in various industries—financial services, direct mail marketing, telecommunications, insurance, healthcare, retail sales—as a means of solving business problems by leveraging a deeper knowledge of the targeted consumers. Some of these applications are as follows:

- Promotion analysis—In retail sales, understanding which products customers often purchase in combination with other products can lead to changes and improvements to in-store displays, seasonal promotions, advertising campaigns, and general advertising strategies.
- Churn analysis—In telecommunications, discovering the profile of individuals likely to switch (or not to switch) long-distance carriers helps companies find ways to attract long-term customers and discourage short-term customers from switching carriers.
- Claims analysis—In insurance, isolating the characteristics of a good risk can save both time and money. In healthcare, cost-cutting measures can be discovered by analyzing the history of a claim and the profile of the claimant.

- Customer profiling—In almost any business, discovering the demographic profile of consumers most likely to buy or use a given product is critical to maintaining a competitive edge. The ability to compile effective target mailing lists is a common result of good profiling.
- Rate and usage analysis—In telecommunications, studying the details of customer calls helps analysts find ways to better serve customers and keep their business, as well as improve the day-to-day service available to those customers.
- Fraud detection—In insurance, healthcare, and credit-card businesses, discovering the typical characteristics of a fraudulent claim or application helps analysts predict and prevent future fraudulent activity.

Data Mining Technology

The Red Brick Data Mine option uses an advanced set of algorithms to perform categorization analysis of warehouse data quickly and accurately. The fundamental difference between this technology and traditional statistical analysis is that the results of a data mining exercise are not revealed in a linear form. In other words, data mining is a *transparent* form of analysis that allows you to see (in symbolic rather than numeric terms) all the subtleties of its findings, not just general trends. The analysis is also *restrained*, in the sense that it does not draw conclusions where none can be drawn.

This distinction is important because it serves as a reminder that data mining does not necessarily find interesting trends—in fact, the findings might be uninteresting and predictable. Data mining does not reveal insights that are not there, nor does it presume to be foolproof. For example, if a large customer list includes only five teenagers and all five bought a given product, this is not conclusive evidence that teenagers will generally buy that product. The data mining application will find a 100% associative pattern between teenagers and the product in question, but this rating is not meaningful unless it is proved to exist when the model is tested against a more representative sample.

The precise definition of the data mining model is also critical. You can mine thousands of rows of data yet return no novel insights about your business if the data necessary to arrive at such insights is not present in the model. For example, you cannot confirm that sales figures are higher on Fridays prior to three-day weekends if the requisite day and date attributes are not explicitly specified as inputs.

Finally, data mining is a process—a process that involves building and refining models in an attempt to discover the hidden relationships in data. The software can find the relationships—even the non-obvious, deeply buried ones—but users must specify what they are trying to understand, what the object of the model is, and what the list of factors might be. Furthermore, they must be willing to test several iterations of the same model with several data samples before trusting its universal accuracy and reliability as a predictive tool.

How Data Warehousing and Data Mining Work Together

The Red Brick Data Mine option is tightly integrated with the warehouse database for several reasons. First, the data to be mined must be suitable for mining—that is, it must come from a single, separate, and clean source. Data that is the subject of decision support queries shares the same set of requirements, so a data warehouse is a ready-made source of data for mining.

Second, it is very likely that not all the data at the source is *appropriate* for mining—for example, highly specific consumer attributes such as social security numbers, account numbers, and phone numbers are unlikely to be useful information in a fraud-detection exercise. Therefore, the data source must have an efficient and reliable means of sorting through and selecting the useful data. Again, these requirements match those of the data warehouse engine, which can select carefully constrained subsets of data from very large databases both quickly and reliably.

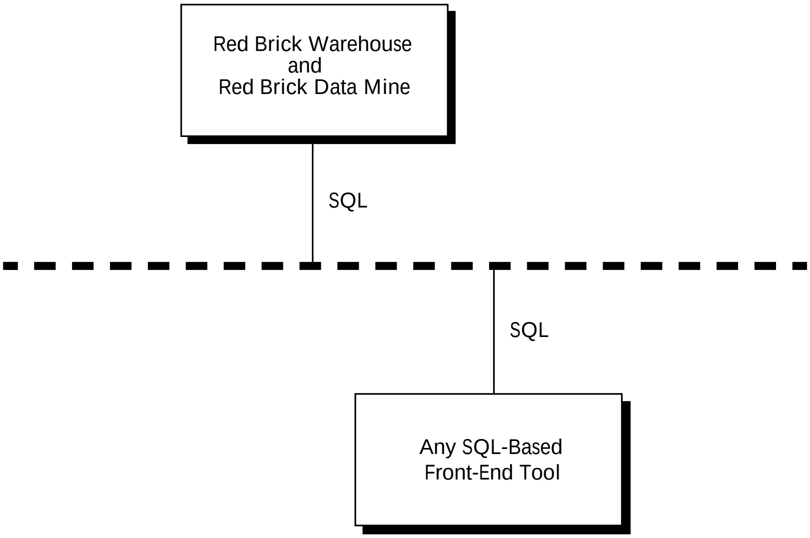
A third and critical role that the data source must play is to analyze the data *after* the data mining exercise has been completed. Although the end result of a successful exercise is the discovery of a new trend or pattern that went unnoticed through the use of standard query and reporting tools, those tools will now be very useful as the analyst uses the discovered knowledge to investigate the same data from a new angle or to follow a brand-new thought process.

For example, in the telecommunications industry, if the data mining application uncovers a relationship between customers who are likely to switch long-distance carriers and an increased number of service calls in the past six months, analysts might write a series of new queries that focus on service calls. Customers who have had increased service calls in recent months (a list generated by a simple warehouse query) could be the target of a campaign to persuade those customers not to switch.

In these ways, data mining and data warehousing are complementary almost to the extent that a data warehouse is a prerequisite to successful deployment of a data mining application. Moreover, when data is both mined and queried by the same server, using the same database, query tool, and query language, the whole process of data mining is made easier, more efficient, and more accessible to the end users.

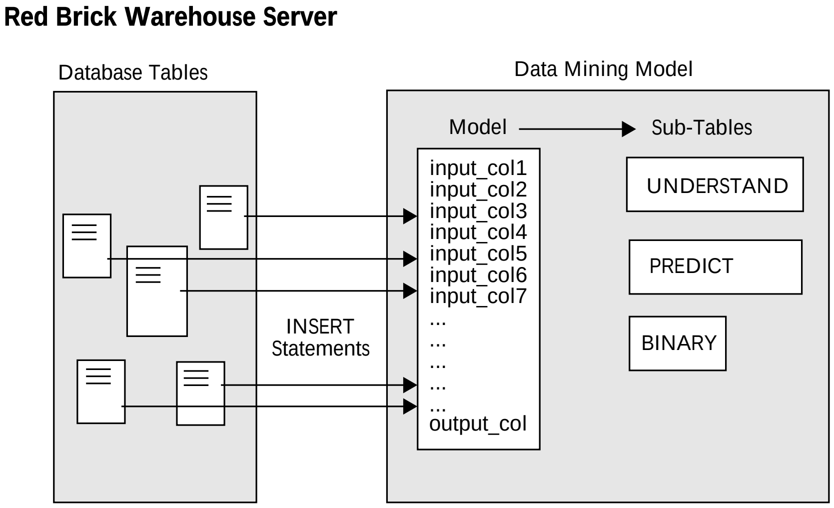
Integrated Data Mining Architecture

The Red Brick Data Mine option is tightly integrated with the Red Brick Warehouse server. The following diagram illustrates the simplicity of this integrated approach to data mining.



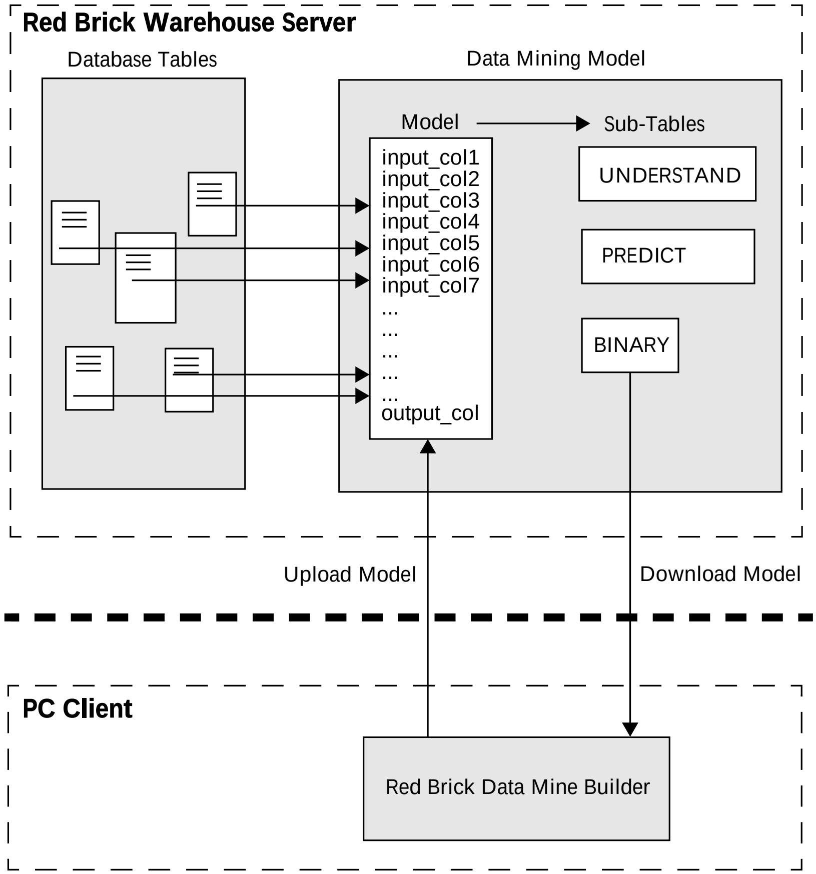
The result of this integration is *one product*—a data warehouse RDBMS with data mining capabilities as well as ad hoc query analysis functions, all accessible via SQL, using the RISQL® Entry Tool, the RISQL Reporter, or any other SQL-based front-end tool that is connected to the database.

Data mining in the relational database engine is accomplished by building models that behave like tables. The following diagram shows how the warehouse database serves as the data source for a data mining model, which consists of the parent "table" and three child "sub-tables."



The sub-tables are created automatically when the model is created. The data mining discovery process is performed on the server whenever data is inserted into the model; users can query the UNDERSTAND and PREDICT sub-tables to study the results of that process. The entire process of building a model and using it for prediction is done with a standard set of SQL commands.

Red Brick Data Mine Builder™ is a separately licensed client tool that can be used to create models in a graphical user interface (GUI) on a PC. These models can be "uploaded" to the warehouse server, while models created on the server in SQL can be "downloaded" to the PC client, as shown in the following diagram.



For detailed information about the Red Brick Data Mine Builder application, refer to the *Red Brick Data Mine Builder User's Guide.*

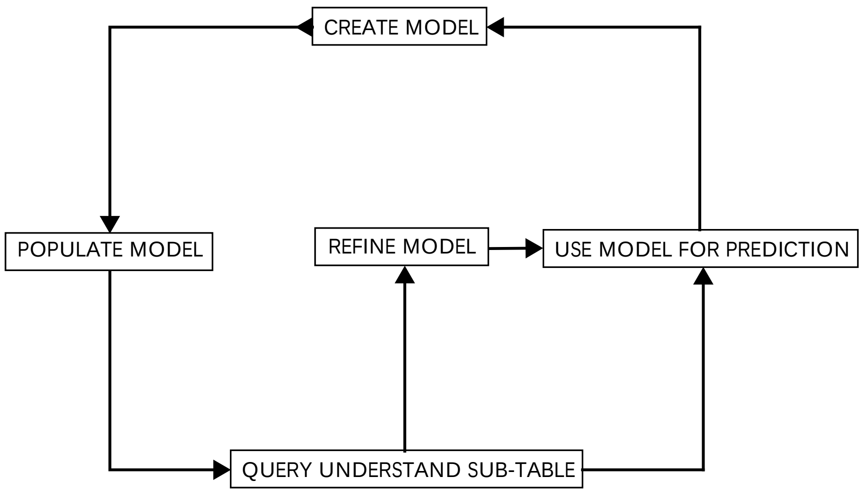
The Data Mining Process—A Simple Example

Data mining with the Red Brick Data Mine option is an iterative process:

1. Create a model.
2. Populate the model to perform discovery.
3. Query the UNDERSTAND sub-table to see the results of the discovery process.
4. Refine the model as required.
5. Use the model for prediction.

The first four steps represent a complete phase in the process that might be called the *model-building* phase. These steps are likely to be performed by an application developer—someone very familiar with the data warehouse and SQL. The fifth step represents a second phase that is more likely to be of interest to business analysts. Predictive models are typically shared and reused among various decision-making groups connected to the same data warehouse.

The following diagram illustrates the iterative nature of the data mining process:



For more details about each step in this process, refer to Chapters 2 through 6 of this guide. For a much more detailed example of the whole process, refer to Chapter 7, "Data Mining: A Detailed Example."

Creating a Model

Models are defined by issuing a CREATE MODEL statement, which must give the model a unique name, identify a single output column and its datatype, and list a number of input columns and their datatypes. For example:

```
create model popcorn
   (candy char(20) input,
   drinks char(20) input,
   popcorn char(3) output);
```

The purpose of this model is to discover what combinations of candy and drink purchases at a chain of movie theaters tend to be accompanied by a purchase of popcorn (an example of categorization analysis). The Popcorn column contains the values "Yes" and "No," while the other two columns contain generic names of types of candy and drinks (for example, "Chocolate" and "Soda").

Although only two inputs are used here, any number of input columns could be added to the model, and multiple versions of the model could be created to "rotate" the columns. For example, a second model could have the Popcorn and Drinks columns as inputs and the Candy column as the output. Such "product grouping" analyses are common in the retail industry. The goal of these analyses is to discover what products tend to be bought or consumed together, then find ways to promote or facilitate the sales of those products.

Another feature of the CREATE MODEL syntax not used in this example is the ability to "map" columns; for example, specific ranges of dollar values for an input column can be mapped to character strings in the model sub-tables. Examples of mapped columns are presented in Chapters 2 and 7.

A CREATE MODEL statement triggers the creation of a set of model sub-tables:

- *model_name*_UNDERSTAND—Contains the results of the data mining discovery process in a tabular form that can be studied via SQL queries. This sub-table is updated each time an INSERT statement is issued against the model.
- *model_name*_PREDICT—Contains the results of the data mining discovery process in a tabular form useful for predicting output values for new data sets. This sub-table is updated each time an INSERT statement is issued against the model.

- *model_name*_BINARY—Contains a binary file readable by the Red Brick Data Mine Builder, a separately licensed product that connects to the database from a PC client. This sub-table allows a model created with the Red Brick Data Mine option to be downloaded to a PC running the Red Brick Data Mine Builder application; this sub-table serves no other purpose.

In terms of their behavior, these sub-tables are similar to database tables; however, they differ in some important respects. For details, refer to Chapter 2, "Defining a Model."

Populating the Model

A model is populated and the data mining discovery process is triggered when an INSERT statement that selects rows from existing database tables is issued. In this case, assume that a predefined table or view exists in the database that contains the three columns used in the model:

```
CANDY        DRINKS       POPCORN
=====        ======       =======
Chocolate    Lemonade     No
NULL         Soda         Yes
Chocolate    Soda         Yes
Licorice     NULL         No
Chocolate    Soda         Yes
...
```

Each row reflects a sales transaction at a movie theater. The INSERT statement would look like this:

```
insert into popcorn
select * from popcorn_view;
```

This statement selects all the columns from the view and inserts them into the model. Although this is the simplest possible query, in most cases the INSERT statement will contain a much more complex query that "transforms" the warehouse data into a result set that matches a larger set of column definitions in the model. For example, the query might join several tables and perform calculations on certain columns.

When the INSERT statement is issued (via any SQL interface connected to the database), the data mining discovery process takes place automatically, and the UNDERSTAND and PREDICT sub-tables are updated without further user intervention. These sub-tables are the source of the knowledge discovered by the data mining exercise.

Querying the UNDERSTAND Sub-Table

The results of the discovery process are selected from the UNDERSTAND sub-table, using any SELECT statement. The contents of this sub-table are quite detailed, but the essential information is embodied in a few specific columns.

The most useful discovery ratings are the Importance values; these specify, numerically, the strength of the relationship (or correlation) between a given input value or set of input values and a given output value. A set of input values that, *taken together*, are significant is known as a *conjunction*; each conjunction is given an ID number.

A good way to constrain on the UNDERSTAND sub-table is to select all the columns but use the RANK function in the WHEN clause to return only those rows with the highest Importance ratings. For example:

```
select * from popcorn_understand
when rank(importance) <=3;
```

TYPE	ConjI	Input	Value	Imp	Disc	OccCt	OccFr	POPCORN
N	1	CANDY	Chocolate	178.08	65.17	10	35.71	No
N	1	DRINKS	Lemonade	178.08	65.17	10	35.71	No
N	NULL	DRINKS	NULL	100.00	100.00	11	39.29	No
N	NULL	CANDY	NULL	100.00	100.00	45	49.45	Yes
N	NULL	CANDY	Licorice	100.00	100.00	5	17.86	No

Note: Some of these columns have been renamed and reformatted using RISQL Reporter commands.

The strongest correlation in this model in terms of Importance value alone is a conjunction of Chocolate and Lemonade; when products in these two categories are purchased in a single transaction, popcorn tends *not* to be purchased. Note, however, that the Occurrence_Count (OccCt) value is low; only 10 transactions were recorded in which this correlation was found.

Taking all columns into account, the strongest correlation in this result set is represented by the fourth row, which suggests that when no candy is purchased (NULL in the Value column), popcorn *is* purchased (Yes). This row has the highest possible Importance value for a single input not involved in a conjunction (100), as well as relatively high Discrimination and Occurrence values.

The third and fifth rows are also potentially interesting correlations, but like the conjunction of the first two rows, they have very low occurrence counts.

Refining the Model

Models frequently require modifications in order to fine tune the discovery results. For example, the data set might not be representative or large enough or additional input and columns might be required.

A model can be refined by adding or dropping columns with the ALTER MODEL command. If necessary, you can remove the model and its sub-tables from the database with the DROP MODEL command or delete all the rows from a model with the DELETE command. Also, the CREATE MODEL...LIKE command, described in Chapter 2, "Defining a Model, " allows you to make a copy of a model and give it a new name; however, when you use this command the rows in the model are not copied, just the column definitions.

In this case, it might be interesting to know if the conjunction of Chocolate and Lemonade still amounts to a significant correlation when the model is tested against a larger data set; therefore, another INSERT statement could be issued to add some new rows to the model and automatically update the sub-tables.

Using the Model for Prediction

The PREDICT sub-table, like the UNDERSTAND sub-table, can be queried as soon as the model is populated with data. Typically, however, it is necessary to refine the model to some degree before using it for prediction.

Prediction is the process of querying the PREDICT sub-table to discover the most likely output for a given set of input values. You can query the PREDICT sub-table in two ways:

- On a case-by-case (row-by-row) basis with a query that contains WHERE clause constraints on some or all of the input columns.
- In "batch" mode by joining the PREDICT sub-table to a database table or view.

Note: Do not issue an unconstrained SELECT * query against the PREDICT table; this kind of query will return a very large number of rows. For more details, refer to Chapter 5, "Making Predictions."

This section presents an example of a case prediction query; an example of batch prediction is presented in Chapter 7, "Data Mining: A Detailed Example."

Assuming that the original popcorn model in this example is not altered or refined in any way, a SELECT statement like this could be issued to predict the output for a specified pair of inputs:

```
select *
from popcorn_predict
where candy = 'Licorice' and drinks = 'Soda';

CANDY        DRINKS       POPC SCORE Rank Valid? Disc
=====        ======       ==== ===== ==== ====== ====
Licorice     Soda         No     156    1 Y       38.00
Licorice     Soda         Yes     98    2 Y        0.00
```

It is not necessary—though it is recommended—to specify WHERE clause conditions on all the input columns. Nonetheless, this kind of query will return more than one row—one for each distinct value in the domain of the output column. In this case, two rows are returned, one for an output value of No and one for Yes.

The results of this query are difficult to judge without a sense of the relative weight of the Score values. Therefore, it is useful to know what the highest score in the entire PREDICT sub-table is; the following query quickly answers this question:

```
select max(score)
from popcorn_predict;

200
```

Given this knowledge, it is reasonable to say that a score of 156 is high; therefore, the prediction that Licorice and Soda purchases *do not* also involve Popcorn purchases looks reliable. However, note that the opposite case also returns a fairly high score of 98.

In conclusion, this model returns some potentially interesting correlations; however, their reliability is difficult to judge until more data has been inserted into the model and the discovery results have been revisited. For a more complex example of a model-building exercise that involves mapped columns and a larger data set, refer to Chapter 7, “Data Mining: A Detailed Example.”

2

Defining a Model

This chapter explains how to define a Red Brick Data Mine model. It is divided into the following main sections:

- Preliminary Planning
- Creating a Model, which includes a description of the syntax for the CREATE MODEL command
- Creating a Model from Another Model, which includes a description of the syntax for the CREATE MODEL LIKE command
- Dropping a Model, which includes a description of the syntax for the DROP MODEL command

Preliminary Planning

Before you write the CREATE MODEL statement that will define your model, you should complete the following preliminary planning steps:

- Define a goal.
- Examine and prepare your data.
- Identify which data columns you want to use as input columns and which one to use as output.
- Determine how you want to handle ranges of data, both discrete and continuous.

Defining a Goal

Before you create a Red Brick Data Mine model, you should have a clear goal in mind. This goal will help you make decisions about what information to include in your model, which columns to specify as input and output columns, and the data you want to use.

Examples of goals include:

- What are the profiles of customers with balanced, late payment, and overdrawn credit cards?
- What other items do people most commonly buy when purchasing beer or wine?
- What kind of patients respond best to a particular medical procedure?

Examining and Preparing Your Data

Creating an effective data mining model requires that you know and understand the data you will be examining. Familiarize yourself with the data columns and the types of data they contain. The more familiar you are with your data, the easier it will be to identify the components that a well-designed model must contain.

It is also very important that your data be “clean.” If data is not entered consistently (for example, if the value “Yes” is entered in various places as “Y”, “YES”, “y”, and “yes”), Red Brick Data Mine will look at these as four separate values, rather than as the single value they represent. If you have not enforced consistency on your data, you should go through it using a report tool or query tool and “clean it up” as much as possible before performing discovery on it. The more consistent your data is, the more reliable and accurate your results will be.

Identifying Input and Output Columns

To create a useful model, you need to understand your data and what you are trying to accomplish with it. For example, if your goal is to determine what other items people buy when buying beer or wine, it would be logical to define beer or wine purchases as your output column, and define other purchase items as input columns. Similarly, if your goal is to identify people who are good credit risks, an obvious choice for an output column would be account status, using other columns (such as income, payment history, and expenses) as inputs.

Other examples of goals and logical output choices include:

Goal	Output
What type of customer is likely to buy a given product?	A customer_type column should be designated as output, with product item columns as input. Creating multiple models using different product data for each model will help you identify customer type data for different products.
What is the best customer profile for my company?	A total_purchases column is one logical choice for output. If you designate other customer characteristics as input columns, then you can map various characteristics to total product purchases.

If, when examining your data, you determine that you cannot state your goal in terms of the relationship between a single output column and a series of input columns, it might help to create multiple models, using a different output column for each. This approach might eliminate non-productive directions and help you to refine your questions. If your goal seems to require multiple output columns, the results might be too broad for easy comprehension or meaningful use. Try rephrasing your question so it will work with a single output column.

When choosing input columns, do not limit your choices to what you think is correct—it is better to be broad, at least on the first attempt. If you are not certain which information you want, start by looking at the columns available in your data, then look at the values for these columns. Ask yourself these questions:

- Which columns do I know (or suspect) might influence the output column?
- Does the data set include more than one column describing the same data (age and birthday, for example)?
- How are the output column and the input columns related? For example, should you examine stores by city or by region? Are purchase levels of a product better described by season or by month?

Again, make logical choices where they exist, but do not limit yourself. Red Brick Data Mine will help you filter out columns that do not provide useful information.

Determining How to Handle Ranges of Data

If your data contains continuous data (such as monthly income, age, and so on) or large groups of discrete values (such as job classifications or zip codes), then you will probably want to consolidate this data in some meaningful way. For instance, if you are attempting to predict creditworthiness with monthly income as an input, you are most likely interested in ranges of monthly income values as opposed to individual income figures. Similarly, if your data contains zip codes, sorting them by geographic region is often more useful than looking at each individual zip code value.

Before you begin creating your model, you should determine how you want to organize this continuous and discrete data; that is, define the ranges into which you want to divide your continuous data, and decide how you want to group large sets of discrete values.

Choosing "Buckets" for Column Mapping of Continuous Values

When a column contains a large number of continuous values, it is often useful to group these values into smaller ranges, or "buckets." A continuous column is numeric and contains a quantitative value (such as age, size, or income). Dividing these values into buckets will make model results easier to understand; the buckets you choose depend on the type of data you are examining in your model.

For example, two columns in the Credit model are Monthly_Income and Monthly_Expenses. Each column is likely to contain many (hundreds, thousands, or more) unique values. Dividing the income and expense values into several buckets allows you to deal with a much smaller number of possible input values. For instance, suppose the Monthly_Income column had possible values ranging from 1000 to 10000, allowing potentially 9,001 individual values. Grouping these values into logical ranges (such as 1000 to 2000, 3000 to 5000, and so on) can make your data analysis more meaningful.

How you divide your data depends on its content and the goal of the application. A telecommunications company looking for those customers most likely to switch long-distance services, for example, might put age data into different buckets than would a pediatric hospital.

Mapping Bucket Datatypes

When you divide column values into buckets and assign each bucket a descriptive name, you might find that the datatype of the original values is often different from that of the final list of ranges. For example, suppose you have a series of age values, and you want to map ages 0 to 12 as "Child," ages 13 to 19 as "Teen," 20 to 64 as "Adult," and 65 and up as "Senior." In this example, the original datatype of the age values is INTEGER, but the datatype of the final designation is CHAR. To indicate this in the CREATE MODEL statement, you use the *mapping_clause* specification, which specifies the original datatype of the values and datatype to which the value of the final range list is mapped.

You do not need to map the datatype if the datatype of the original values is the same as that of the final mapped values. For instance, if you map a group of income values (type INTEGER) to a smaller group of numbered buckets (also type INTEGER), it is not necessary to map the datatypes.

For more information about mapping bucket datatypes, refer to the CREATE MODEL syntax beginning on page 2-15.

Example

The following example shows how an integer input column mapped to a character column is used to indicate the original datatype of the Age data described above:

```
create model purchases (
   age integer input
      mapping to char(6)
      mapping ranges (0 to 18 as 'Young', 19 to 65 as 'Adult',
      66 to max as 'Senior'),
   purchase char(20) output;
```

Mapping Ranges into Buckets

You use the MAPPING RANGES specification of the CREATE MODEL statement to divide your data into buckets and map each range to an identifier. You can map values in the following ways:

- A basic range of values, for example: 0 to 5 as Low, 6 to 10 as Middle, 11 to 20 as High.
- A range of values specifying inclusive or exclusive elements, for example: 2750, inclusive, to 3550, exclusive, as Medium.
- A range of values where the minimum and / or maximum values are unknown, for example: min to 0 as Low, 1 to max as High.
- A range of overlapping values, for example: 0 to 12 as Child, 11 to 19 as Teen, 18 to 99 as Adult.
- A range of values with gaps included, for example: 0 to 12 as Child, 13 to 19 as Student, 65 to 99 as Senior.

You can combine these mapping methods as needed in your model. For input columns, the specification can have overlapping ranges, which can result in multiple data discovery values for a single input value.

If your data mapping has gaps (values that are not covered within the specified ranges), then NULL (unknown) will be supplied during data discovery for any such values found.

The MAPPING RANGES specification is described in the section “CREATE MODEL Command: mapping_clause Specification” on page 2-19.

Mapping a Basic Range of Values

To map a basic range of values, specify the initial value in the range, the final value in the range, and the name of the "bucket" into which you want to map them.

You can map more than one range of input values to the same bucket, but the names of the buckets to which you map output values must be unique. For example, if you were mapping a series of input values representing ages, and you were only interested in people between the ages of 18 and 65, then you could map all the ages younger than 18 or older than 65 into the category "Other." However, if you are using age as an output value, then each of these two ranges (age 17 or under and age 66 or over) must have its own unique bucket name.

Example

The following example shows the mapping of a series of monthly income values (input) and a series of purchase levels (output) mapped into three buckets each:

```
create model purchases (
   input integer input
      mapping ranges (1000 to 2000 as 1, 2001 to 4000 as 2, 4001
      to 6000 as 3),
   purchases integer output
      mapping to char(4)
      mapping ranges (0 to 500 as 'Low', 501 to 1000 as 'Med',
      1001 to 2000 as 'High');
```

Mapping a Range of Values with Inclusive and Exclusive Elements

To map a range of values with one or more inclusive or exclusive elements, use the keywords INCLUSIVE and EXCLUSIVE (or INCL and EXCL) to designate that a value is inclusive (included in the range) or exclusive (excluded from the range).

You can use one or both attributes (INCLUSIVE and EXCLUSIVE) in a column mapping clause.

Example

The following examples show mappings of a series of monthly expense values using INCLUSIVE and EXCLUSIVE:

```
create model creditcard (
   income integer input,
   c_card_limit integer input
      mapping to char(4)
      mapping ranges (500 inclusive to 1000 exclusive as 'Low',
      1000 to 2999 as 'Medium', 3000 inclusive to 5000 as
      'High'),
   expenses integer output);
```

Note: If the INCLUSIVE and EXCLUSIVE attributes are omitted from your mapping, then the value on the left side of *to* defaults to INCLUSIVE, and the value on the right side of *to* defaults to EXCLUSIVE.

Mapping a Range of Values with Unknown Minimum and Maximum Values

To map a range of values where the minimum and/or maximum possible values are not known, use the keyword MIN to designate the minimum value in the range, and the keyword MAX to designate the maximum value in the range.

Example

The following examples show mappings of monthly expense values where the minimum and maximum values are not known when the model was created:

```
create model creditcard (
   income integer input,
   c_card_limit integer input
      mapping to char(4)
      mapping ranges (min to 1000 as 'Low', 1001 to 3000 as
      'Med', 3001 to max as 'High'),
   savings integer output);
```

Mapping a Series of Values with Overlapping Ranges

In some instances, you might want to map a particular value into more than one bucket. For example, if you are designing a marketing promotion aimed at young people, you might want to classify children of ages 11 and 12 in both the “Child” and “Teen” categories, to reflect the fact that older children might have interests in both these areas.

Note: Overlapping ranges are not allowed for mapped output columns.

Example

The following example shows a set of age data with 11- and 12-year-olds mapped into both the “Child” and “Teen” buckets:

```
create model y_purchase (
   age integer input
      mapping to char(5)
      mapping ranges (0 to 12 as 'Child', 11 to 19 as 'Teen',
      20 to 99 as 'Adult')
   no_purchased integer output);
```


Mapping a Series of Values with Gaps

You can allow gaps in the data that you are mapping into buckets. For example, if you are analyzing a special ticket promotion and want to determine how many Child and Senior tickets were purchased, you can map the ages of the ticket purchasers into buckets while leaving the other ages (people who are not children or seniors) out of the mapping.

Example

The following example maps a series of ages into “Child” and “Senior” buckets, while leaving a gap for those ages that do not fall within these categories:

```
create model tickets
   age integer input
      mapping to char(6)
      mapping ranges (0 to 13 as 'Child', 55 to max as 'Senior'),
   no_tickets integer output);
```

Consolidating Discrete Data

A discrete column contains a qualitative dimension, such as gender, profession, or city. In a discrete column, each value has a meaning of its own (such as zip codes; each individual value indicates a different region despite the fact that the data is numeric). If you have a large number of discrete values in a column, you might need to consolidate them to make them more useful. Examples of ways you can consolidate discrete data include:

- Using a query or report tool to consolidate telephone number data into regions (North, South, East, and West, for instance), then using these derived columns in your study.
- Assigning job description columns to logical numeric categories (for example, you might assign all managerial occupations the numbers 9000 – 9999 and all agricultural occupations 5000 – 5999), then mapping these values to buckets for use with your model.

Modifying Date Data

When preparing your data, keep in mind that absolute dates are not very useful for data mining purposes. A good strategy is to convert absolute date columns (such as Year_Started) to numeric values (such as Years_In_Business). It is also useful to use date functions in your query or report tool (or the MAPPING RANGES specification) to represent dates in terms of quarter, month, or week. This approach is more likely to reveal underlying patterns in your data.

Preparing Empty Columns

A NULL indicator in a column indicates that the value is missing or unknown. It is not the same as a 0 (zero) value or a blank or space. For example, in the Credit model, the column Nbr_Children allows both NULL (no data known) and 0 (the respondent has no children). If your data contains zero values or spaces to indicate an unknown value, these values should be changed to NULLs.

Creating a Model

When you have defined your goal and identified the input and output columns and determined how to handle discrete and continuous data, you are ready to create the model.

You create a new model using the CREATE MODEL statement. Each model is defined with the following elements:

- Name
- Description of the input columns
- Description of the output column
- Series of model sub-table names
- Threshold values that specify the requirements for data retrieval during model understanding and prediction as described on page 2-13.

Models can be modified as needed with an ALTER MODEL statement. For information about modifying models, refer to Chapter 6, "Refining the Model."

The complete syntax for the CREATE MODEL statement begins on page 2-15.

Model Sub-Tables

When the model is created, the system creates three corresponding model sub-tables, with the following default names:

- The UNDERSTAND sub-table, which stores the information Red Brick Data Mine finds during the data mining discovery process. The default name of this sub-table is *model_name*_UNDERSTAND. For more information about the UNDERSTAND sub-table, refer to Chapter 4, "Understanding the Model."
- The PREDICT sub-table, used to predict new output values given a set of input values. The default name of this sub-table is *model_name*_PREDICT. For more information about the PREDICT sub-table, refer to Chapter 5, "Making Predictions."
- The BINARY sub-table, which contains a binary file readable by the Red Brick Data Mine Builder, a separately licensed product that runs on PCs and connects to the database via an ODBC interface. The default name of this sub-table is *model_name*_BINARY. This sub-table allows a model created with the Red Brick Data Mine option to be downloaded to a PC running the Red Brick Data Mine Builder application. For more information about downloading Red Brick Data Mine models to Red Brick Data Mine Builder, refer to the *Red Brick Data Mine Builder User's Guide*.

In terms of their behavior, these sub-tables are similar to database tables; however, they differ in some important respects. The following table summarizes the rules and restrictions imposed on each sub-table:

Table	CREATE	DROP	ALTER	SELECT	INSERT	UPDATE	DELETE
MODEL	Yes	Yes	Yes	No	Yes	No	Yes
UNDERSTAND	No	No	No	Yes	No	No	No
PREDICT	No	No	No	Yes	No	No	No
BINARY	No	No	No	Yes	No	No	No

Note: A DROP or DELETE statement on a model applies to all the sub-tables as well. The SELECT operation is allowed on the BINARY sub-table because it is required to exchange data with Red Brick Data Mine Builder; the BINARY sub-table contains no information of interest to users.

You can use the default name or you can specify different names for each of these tables when you create the model, but these names must be unique among table names and model names in the database.

Example

The following simple CREATE MODEL statement illustrates the creation of a model where the names of the UNDERSTAND and PREDICT sub-tables are explicitly defined:

```
create model purchases (
   purchase1 char(20) input,
   purchase2 char(20) input,
   purchase3 char(20) output,
understand table purchaseresults,
predict table purchasefuture);
```

Note that because it is not specified, the BINARY sub-table keeps its default name, Purchases_BINARY, when this model is created.

Occurrence Threshold

The *occurrence threshold* defines the percentage in which a given correlation or conjunction of correlations is required before the relation is retrieved in the UNDERSTAND sub-table. Only those relations with an Occurrence Frequency score in the UNDERSTAND sub-table that is higher than the Occurrence Threshold will be retrieved.

For example, selecting from the UNDERSTAND sub-table with the occurrence threshold set to 0 will return all associations, no matter how small, while the same SELECT operation with the occurrence threshold set to 5 would result in the retrieval of only those associations that have a number greater than 5.0 in the Occurrence Freq column of the UNDERSTAND sub-table (in other words, those associations that occur more than 5% of the time).

If you do not specify an occurrence threshold value, the default value of 3.0 (3%) is used.

For more information about the UNDERSTAND sub-table and the Occurrence Threshold, refer to “Setting the Occurrence Threshold” on page 4-11.

Discrimination Threshold

The *Discrimination Threshold* is similar to the Occurrence Threshold, except that it applies during prediction instead of during understanding. The Discrimination Threshold affects the number of rows that are generated for a prediction on an input row.

If the Discrimination Threshold is set to 0.0, a prediction is generated for each possible output value. The Discrimination, or the difference in strength between the best prediction and the second best prediction, appears in the PREDICT sub-table. For all prediction values other than the best prediction, the Discrimination value is NULL.

If the Discrimination Threshold is set to a value greater than 0.0, then only the best prediction appears in the PREDICT sub-table. However, if the Discrimination value for the best prediction is equal to or less than the Discrimination Threshold value, no prediction appears.

If you do not specify a Discrimination Threshold value, the default of 0.0 (0%) is used.

Model Backup and Restore

You cannot perform an incremental backup on Red Brick Data Mine models using SQL-BackTrack for Red Brick Warehouse or the Table Management Utility Incremental Backup and Restore feature. If you perform an incremental backup on a Red Brick Warehouse database that contains Red Brick Data Mine models, the incremental backup will be performed only on the tables in the database; a full backup will be performed on all model files.

For more information about the Table Management Utility, refer to the *Table Management Utility Reference Guide*.

For more information about SQL-BackTrack for Red Brick Warehouse, refer to the *SQL-BackTrack for Red Brick Warehouse User's Guide*.

Model Security

The GRANT and REVOKE commands allow the granting and revoking of privileges for models and their associated UNDERSTAND, PREDICT, and BINARY sub-tables.

You can only grant those privileges that are allowed on each type of table; the GRANT ALL PRIVILEGES command implies the granting of the following privileges depending upon table type:

Table Type	Privileges Allowed	Privileges Not Allowed
model	INSERT, DELETE	SELECT, UPDATE
UNDERSTAND, PREDICT, and BINARY sub-tables	SELECT	INSERT, DELETE, UPDATE

For detailed information about the GRANT, REVOKE, and GRANT ALL PRIVILEGES commands, refer to the *SQL Reference Guide*.

CREATE MODEL Command

A CREATE MODEL statement defines the data mining model for subsequent discovery and analysis.

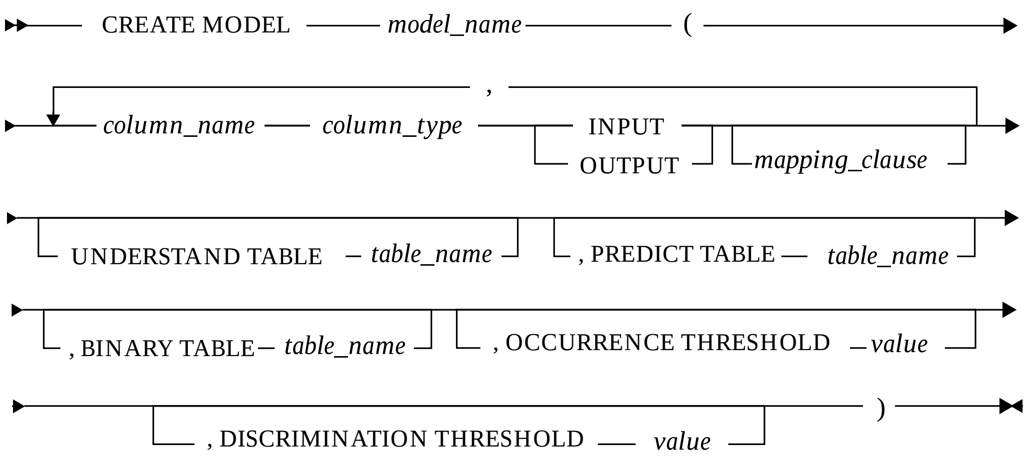
Authorization

To create a model, a user must meet one of the following requirements:

- Be a member of the DBA or RESOURCE system role.
- Have CREATE_ANY or CREATE_OWN authorization, either explicitly or through membership in a user-created role. (Enterprise Control and Coordination option only).

Syntax

The following syntax diagram shows how to construct a CREATE MODEL statement:



model_name

Specifies the name of the model being created, which must be:

- Unique in the database
- A valid database identifier

Note: Although the normal table identifier length is 128 characters, the maximum length of a model name is 117 characters. This restriction is necessary to allow enough space to append "_UNDERSTAND" (the longest of the three suffixes appended to the model name) when the additional tables are created.

column_name

Specifies the name of a column in the model. The name must be unique within the model, must be a valid database identifier, and must not be qualified. A model can have a maximum of 7,280 columns, of which exactly one must be defined as an output column.

column_type

The column type must be one of the following datatypes:

- CHARACTER or CHAR for fixed-length character string values
- DATE, TIME, or TIMESTAMP for datetime values
- DECIMAL or DEC for signed decimal values (same as NUMERIC)
- FLOAT for signed floating-point values (same as DOUBLE PRECISION)
- REAL for signed floating-point values
- INTEGER or INT for signed integer values (between $–2^{31}$ and $2^{31}–1$)
- SMALLINT for small signed integer values (between $–2^{15}$ and $2^{15}–1$)
- TINYINT for signed integer values (between $–2^{7}$ and $2^{7}–1$)

Note: The length of a column name plus the length of the column value for a column of type CHAR(128) or greater must not exceed 250 bytes.

For additional information about datatypes, refer to the *SQL Reference Guide*.

INPUT, OUTPUT

At least one column in the model must be marked as INPUT. Only one column can be marked as OUTPUT.

Examples

The following column-definition clauses define the input columns Gender and Pmt_History, and the output column Account_Status:

```
gender char(1) input
pmt_history int input
account_status char(20) output
```

mapping_clause

Specifies the datatypes and ranges into which the data will be mapped. For a description of the mapping clause, refer to page 2-19.

UNDERSTAND TABLE *table_name*

Specifies the name of the UNDERSTAND sub-table. If a name is not specified, a default value of *model_name*_UNDERSTAND is used.

PREDICT TABLE *table_name*

Specifies the name of the PREDICT sub-table. If a name is not specified, a default value of *model_name*_PREDICT is used.

BINARY TABLE *table_name*

Specifies the name of the BINARY sub-table. If a name is not specified, a default value of *model_name*_BINARY is used.

OCCURRENCE THRESHOLD *value*

Specifies the occurrence percentage of a given criterion or conjunction of criteria required before the correlation is retrieved in the UNDERSTAND sub-table.

If *value* is not specified, a default value of 3 (3%) is used.

This parameter does not affect the model contents, but rather the results retrieved from the model. For example, selecting from the UNDERSTAND sub-table with *value* set to 0 will return all associations no matter how small; if *value* is set to 5 for the same model, fewer values are returned.

The Occurrence Threshold value must be within the range of 0 to 100.

DISCRIMINATION THRESHOLD ***value***

Specifies the percentage difference that must exist between the Score value of the best prediction and the Score value of the next best prediction before the prediction appears in the PREDICT sub-table.

For example, if two output values are possible and the discrimination is set at 10%, no prediction result will be generated if the second best prediction's Score value is within 10% of that of the best prediction. Instead, an undetermined result will be returned.

If the Discrimination Threshold value is set to 0.0, ranked predictions will appear in the PREDICT sub-table for each of the possible output values, though only the best prediction (ranked 1) will display a discrimination value.

If the Discrimination Threshold is set to a number greater than 0.0, only the best prediction appears in the PREDICT sub-table. However, if the discrimination value of the best prediction is less than or equal to the Discrimination Threshold value, no prediction is generated.

The Discrimination Threshold is similar to the Occurrence Threshold, except that it applies during the prediction process rather than during the understanding process.

If the value is not specified, a default value of 0.0 is used.

The Discrimination Threshold value must be within the range of 0 to 100.

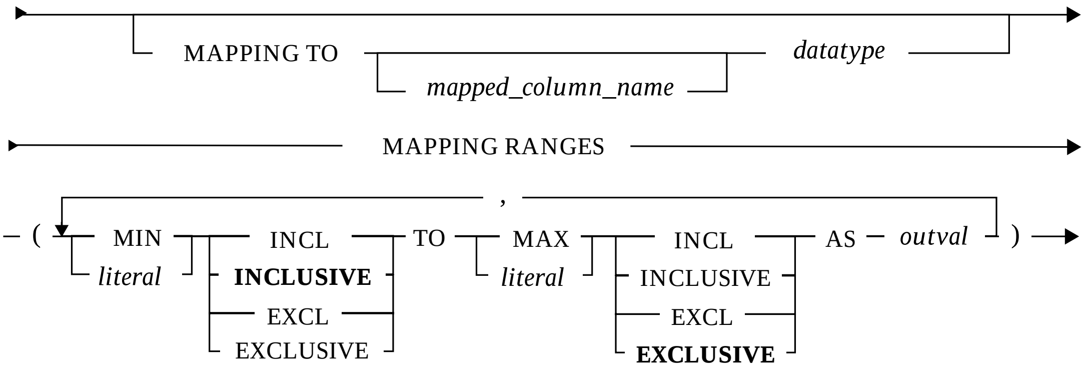
CREATE MODEL Command: mapping_clause Specification

The *mapping_clause* specification specifies the “buckets” into which the data is mapped, and also the datatypes to which each column will be mapped if the bucket’s datatype is different from that of the original data.

Note: If you define the MAPPING TO specification, you must define MAPPING RANGES as well. If the original source of the data and the ranges to which it will be mapped have the same datatype, you do not need to define a MAPPING TO specification.

Syntax

The following syntax diagram shows how to construct a *mapping_clause*:



MAPPING TO

Indicates that the specified datatype is to be mapped to the datatype defined in the *mapping_clause*. If the MAPPING TO specification is not present, the input data type is assumed to be the same as the model column type.

mapped_column_name

Specifies the name of the mapped input or output column. If no name is specified, a default name of the form *column_name_MAPPED* is automatically used. If specified for an output column, the *mapped_column_name* is used in the UNDERSTAND and PREDICT sub-tables. If specified for an input column, the *mapped_column_name* is used only in the PREDICT sub-table.

For example, if an output column originally named Dollars is given Sales_Pct as its *mapped_column_name*, the name Sales_Pct will appear in both sub-tables. However, if the Dollars column is not given a *mapped_column_name*, the name Dollars_MAPPED will appear in both sub-tables. In both cases, any query of the sub-tables that constrains on the Dollars column will receive a syntax error.

datatype

Specifies the column type must be one of the following datatypes:

- CHARACTER or CHAR for fixed-length character string values
- DATE, TIME, or TIMESTAMP for datetime values
- DECIMAL or DEC for signed decimal values (same as NUMERIC)
- FLOAT for signed floating-point values (same as DOUBLE PRECISION)
- REAL for signed floating-point values
- INTEGER or INT for signed integer values (between -2^{31} and $2^{31}-1$)
- SMALLINT for small signed integer values (between -2^{15} and $2^{15}-1$)
- TINYINT for signed integer values (between -2^{7} and $2^{7}-1$)

Note: The length of a column name plus the length of the column value for a column of type CHAR(128) or greater must not exceed 250 bytes.

For additional information about datatypes, refer to the *SQL Reference Guide*.

MAPPING RANGES

Indicates the ranges, or "buckets," into which the values will be mapped.

MIN

Indicates the minimum value included in the data.

literal

Specifies a fixed sequence of characters or a numeric constant.

INCL, INCLUSIVE

Indicates that the specified value is included within the range of values. If INCLUSIVE and EXCLUSIVE are not present, the default is INCLUSIVE on the left side of to and EXCLUSIVE on the right side.

EXCL, EXCLUSIVE

Indicates that the specified value is not included within the range of values. If INCLUSIVE and EXCLUSIVE are not present, the default is INCLUSIVE on the left side of to and EXCLUSIVE on the right side.

MAX

Indicates the maximum value included in the data.

outval

Specifies the name of the range or "bucket" to which a set of values is mapped. The *outval* value must be unique.

Sample CREATE MODEL Statements

The following CREATE MODEL statements illustrate the process described in the previous sections.

Example 1: The Credit Model

The Credit model's purpose is to analyze personal data about individuals (such as income, occupation, and monthly expense) and aid in predicting whether a given individual is a good credit risk. The following CREATE MODEL statement is used to create this model:

```
create model credit (
   gender char(8) input,
   marital_status char(20) input,
   nbr_children integer input,
   occupation char(15) input,
   home char(4) input,
   mo_expenses integer input
      mapping ranges (min to 505 as 0, 506 to 1113 as 1,
      1113 to 1720 as 2, 1720 to 2327 as 3, 2327 to 2934 as 4,
      2934 to 3541 as 5, 3542 to max as 6),
   mo_income integer input
      mapping ranges (min to 2003 as 0, 2004 to 3398 as 1,
      3398 to 4795 as 2, 4795 to 6187 as 3, 6187 to 7581 as 4,
      7581 to 8975 as 5, 8976 to max as 6),
   checking_acct char(3) input,
   savings_acct char(3) input,
   mc_limit char(20) input,
   vi_limit char(20) input,
   ax_limit char(20) input,
   pmt_history integer input,
   account_status char(20) output
);
```

Note that the names of the model sub-tables have not been specified in this CREATE MODEL statement, so they will use the default names: Credit_UNDERSTAND, Credit_PREDICT, and Credit_BINARY.

The OCCURRENCE_THRESHOLD and DISCRIMINATION_THRESHOLD values have also not been specified, so the default values of 3.0 and 0.0, respectively, will be used.

2

Example 2: the Oilprices Model

The purpose of the Oilprices model is to predict, based on various input values, the trends for prices of oil.

```
create model oilprices
   pricetrend float output
      mapping to char(4)
      map ranges (MIN to 0 as ‘Down’, 0 to 9.5 as ‘Avg’, 9.5 to
      MAX as ‘Up’),
   inflation integer input
      mapping to char(8)
      mapping ranges (0 to 0 as ‘NONE’, 0 to 7 as ‘LOW’, 5 to
      12 as ‘MODERATE’),
   averagempg decimal(3,0) input,
understand table oilresults,
predict table oilfutures,
occurrence threshold 2,
discrimination threshold 4
);
```

Note that in this model, the UNDERSTAND sub-table has been named Oilresults, and the PREDICT sub-table has been named Oilfutures. The BINARY sub-table uses its default name, Oilprices_BINARY.

Note also that the occurrence threshold and discrimination threshold have been set to 2% and 4%, respectively.

Creating a Model from Another Model

When you have created a model and populated it with data using the INSERT INTO MODEL statement (refer to Chapter 3, "Performing Discovery on a Model," for more information about inserting data into a model), the model contains both the model definition and the results obtained from performing discovery on your data. You can use the CREATE MODEL LIKE statement to create other models using the definition of an existing model, so that you can use the same model to analyze different data sources using identical model definitions. You can also use the CREATE MODEL LIKE statement to duplicate a model similar to one you want to create, then use ALTER MODEL to make specific changes. For more information about altering an existing model, refer to Chapter 6, "Refining the Model."

To create a model using the definition of an existing model, you specify the name of the new model, the name of the model from which you are taking the definition, and, optionally, provide names for the three associated model sub-tables (UNDERSTAND, PREDICT, and BINARY).

Threshold Values for the New Model

When creating a model using another model, the OCCURRENCE_THRESHOLD and DISCRIMINATION_THRESHOLD values are copied from the source model. You cannot alter these values when creating the new model; when it has been created, you can alter them using the ALTER MODEL statement.

CREATE MODEL LIKE Command

A CREATE MODEL LIKE statement allows you to create other models using the definition of an existing model.

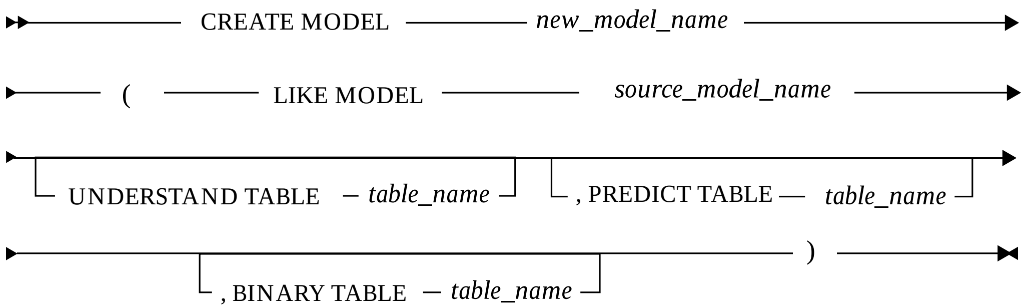
Authorization

To create a model, a user must meet one of the following requirements:

- Be a member of the DBA or RESOURCE system role.
- Have CREATE_ANY or CREATE_OWN authorization, either explicitly or through membership in a user-created role (Enterprise Control and Coordination option only)

Syntax

The following syntax diagram shows how to construct a CREATE MODEL LIKE statement:



new_model_name

Specifies the name of the model to be created. The name must:

- Be unique in the database
- Contain no more than 117 characters.

Note that the normal table identifier length is 128 characters; the 117-character restriction for model names is necessary to allow the system enough space to append the default "_UNDERSTAND" suffix to the model name when creating the model's corresponding UNDERSTAND sub-table.

source_model_name

Specifies the name of an existing model from which the new model's definition will be taken. After the new model is created, no special associations are maintained between the source model and the new model.

UNDERSTAND TABLE *table_name*

Specifies the name of the new model's UNDERSTAND sub-table, which must be different from any other model or table names in the database.

PREDICT TABLE *table_name*

Specifies the name of the new model's PREDICT sub-table, which must be different from any other model or table names in the database.

BINARY TABLE *table_name*

Specifies the name of the new model's BINARY sub-table, which must be different from any other model or table names in the database.

Note: The new model uses the same OCCURRENCE THRESHOLD and DISCRIMINATION THRESHOLD values as the source model. For more information about occurrence threshold and discrimination threshold values, refer to page 2-17.

Example

The following example shows how to create a second model, Credit2, from the source model Credit, renaming the UNDERSTAND and PREDICT sub-tables:

```
create model credit2 (
   like model credit,
   understand table credit2results,
   predict table credit2future
);
```

Dropping a Model

Dropping a model deletes the model definition, the discovery results, and all the associated model sub-tables.

To drop a model, you use a DROP MODEL statement, specifying the name of the model you want to drop.

DROP MODEL Command

The DROP MODEL command allows you to delete the model definition, all discovery results, and all associated model sub-tables.

Authorization

To drop a model, a user must either

- Be a member of the DBA system role.
- Have DROP_ANY authorization, either explicitly or through membership in a user-created role. (Enterprise Control and Coordination option only).
- Be a member of the RESOURCE system role and be the creator of the model.
- Have DROP_OWN authorization and be the creator of the model. (Enterprise Control and Coordination option only.)

Syntax

The following syntax diagram shows how to construct a DROP MODEL statement:

DROP MODEL *model_name*

model_name
Specifies the name of an existing model to be dropped.

Note: Associated model sub-tables are also dropped when a model is dropped.

Example

The following example shows how to drop the Credit model, including all its associated model sub-tables:

```
drop model credit;
```

3

Performing Discovery on a Model

The process of inserting data into a model and triggering Red Brick Data Mine to perform analysis on the data is called *discovery*. During the discovery process, Red Brick Data Mine identifies the relationships among the data and inserts the relationships into the UNDERSTAND sub-table.

This chapter describes how to insert data into the model to begin the process of discovery. It is divided into the following main sections:

- Sampling Your Data
- Inserting Data Into the Model
- INSERT INTO MODEL Command
- Deleting Data From a Model
- DELETE FROM MODEL Command

Sampling Your Data

Although you can use full data sets with Red Brick Data Mine, in some cases you might want to limit your domain size by dividing your data source into several smaller domains. This approach allows you to specify different domains for the discovery and prediction processes.

You can often learn more about your model if you use different subsets of your data in each domain, but your model is more accurate if you use the full set for the discovery process. If you have a complex data set, it is probably advantageous to try both approaches to find which one works best for your data.

Why Use Subsets?

If your data set is large and complex, you might want to limit the time spent processing it, at least for the initial discovery, by using a subset of the data instead of the entire data set. However, it is not usually necessary to do this, because Red Brick Data Mine can process most large data sets easily.

Subset Strategies

A standard strategy for defining subsets involves dividing the data and processing each subset separately, using a varying selection of subsets for discovery. After these subsets are processed, you compare the results.

For example, from a one-million-row data set, choose a 5% or 10% subset for your first analysis. After you have created a model and performed discovery on this data subset, run the model over some other subsets of data (or try the full set) to see how accurate your model is. Does it remain consistent over all the data? When you are satisfied with the model, you can build it against the full data set.

It is also important to partition your data correctly. For example, if you are not sure whether to use the full international data or only data from certain countries for your model, ask yourself how you would describe the data to a colleague. Which data should be looked at first? What level of detail is important? Is unfamiliar data more meaningful than familiar data? Is there another area of interest or a new way to examine the data? When you have answered these questions, you might decide to use the entire data set, or decide to select subsets of it (with a WHERE clause in your query tool, for instance).

By crafting your queries carefully (using WHERE clauses, OR clauses, and other methods of selecting parts of your data), you can create subsets of specific data. By using these clauses on many columns, you can create a pseudo-random subset of data on which to perform discovery. Note, however, that this is not a truly random sample.

For more information about writing SQL statements, refer to the *SQL Reference Guide*.

Inserting Data into the Model

The data mining discovery process is triggered by inserting data into a model. You can insert as many or as few rows of data as you wish, using one or more INSERT operations.

When you insert data into the model, the number and types of columns inserted into the model must correspond to the model's definition. For example, if the Gender column of the Credit model is defined as type CHAR(1), then the data inserted into the model for Gender must be of type CHAR(1) as well.

However, recall that if the datatype is mapped to another datatype, the datatype of the inserted data must match the original datatype, not the type to which it is mapped. For example, if the datatype for a column named Age was originally INTEGER, but was mapped to CHAR(4), then the data inserted must be in the form INTEGER.

Inserting a Single Row of Data

To insert a single full row of data into a model, you do not need to specify the column names; the command assumes that all columns have been specified and are ordered as they occur in the named model.

To insert a partial row of data into a model, you must specify the column names in the order in which the data is to be inserted, followed by the values to be inserted.

Example

The following INSERT INTO *model* statement inserts a single row of data into the Credit model, where all the values are known:

```
insert into credit values ('Female', 'Married', 0,
'Professional', 'Rent', 4425, 7450, 'Yes', 'Yes',
'middle', 'low', 'low', 12, 'Balanced');
```

The following INSERT INTO *model* statement inserts a single partial row of data (in this case, the Gender, Marital_Status, and Occupation values) into the Credit model:

```
insert into credit (Gender, Marital_Status, Occupation)
   values ('Female', 'Married', 'Professional');
```

The values for the nonspecified columns are assumed to be NULL (unknown).

Inserting Data Using a SELECT Statement

You can also insert data into a model using a SELECT statement. The result table of the query is inserted into the named model.

The number of columns returned by the query must be the same as the number of columns that occur in the column list. The value of the first column of the query's result table is inserted into the first column specified in the column list, the value of the second column is inserted into the second, and so on.

Example

The following INSERT INTO *model* statement inserts the complete contents of the Creditstats table into the Credit model. In this example, the Creditstats table has the same number of columns, and the columns are of the same datatypes, as in the Credit table.

```
insert into credit
   select * from creditstats;
```

Example

The following example shows the syntax for inserting a subset of the contents of the Creditstats table (in this case, the Mo_Expenses, Mo_Income, and Account_Status columns) into the Credit model:

```
insert into credit(Mo_Expenses, Mo_Income, Account_Status)
   select Mo_Expenses, Mo_Income, Account_Status
   from creditstats;
```

NULL Values and Discovery

Column values with the value NULL are considered to be unknown during the discovery process.

INSERT INTO model Command

An INSERT INTO *model* statement inserts one or more rows into a specified model.

Note: The INSERT INTO *model* statement is similar to the INSERT statement used for inserting data into a table. Refer to the *SQL Reference Guide* for more information about the INSERT statement.

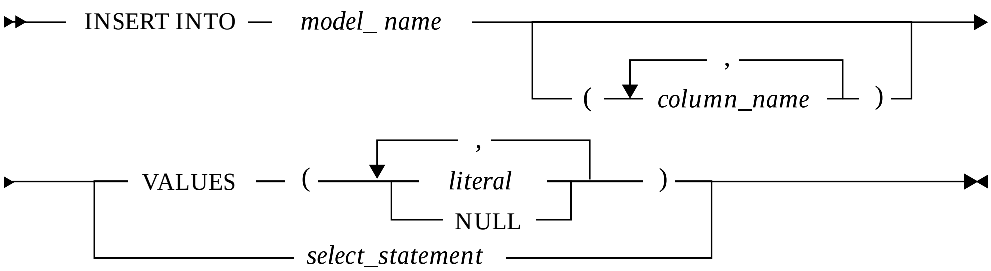
Authorization

To insert rows into a model, a user must either:

- Be a member of the DBA system role.
- Be the creator of the model.
- Have INSERT privileges on the model.

Syntax

The following syntax diagram shows how to construct an INSERT INTO *model* statement:



model_name
Specifies the name of the model where rows are to be inserted.

column_name
Specifies a column in the named model.

- The list of names can be in any order.
- A name can occur in the list only once.

If no columns are specified, the command assumes that all columns have been specified and are ordered as they occur in the named model.

VALUES

If a VALUES clause is specified, it must contain the same number of entries as the column list. If a column list is not specified, the entries in the VALUES clause must correspond with the number of columns and order of columns in the named model.

The INSERT command inserts the first entry into the first column specified, the second value into the second value specified, and so on.

literal

A character or numeric constant.

NULL

Specifies the NULL value to be inserted into the column.

select_statement

The result table of the query is inserted into the named model. The query can return one or more rows; if no rows are returned, the following message is returned:

```
** INFORMATION ** (209) Rows inserted: 0.
```

The number of columns returned by the query must be the same as the number of columns that occur in the column list. The value of the first column of the query's result table is inserted into the first column specified by the column list, the value of the second column is inserted into the second, and so on.

Deleting Data from a Model

When you have created a model and revised it until it fits your needs, you can re-use the same model to mine different data. To do this, you delete all the discovery data from the model while leaving the model's definition intact.

The DELETE FROM *model* statement allows you to remove the discovery results from a model while preserving the model's definition. It removes all the rows from the model, as well as removing all the rows from the model sub-tables.

Example

The following DELETE FROM *model* statement removes the rows from the Credit model and all its associated model sub-tables:

```
delete from credit;
```

DELETE FROM model Command

A DELETE statement deletes all rows of data from a specified model and its associated model sub-tables.

Authorization

To delete rows from a model, a user must either:

- Be a member of the DBA system role.
- Be the creator of the model.
- Have DELETE privilege on the model.

Syntax

The following syntax diagram shows how to construct a DELETE statement:

DELETE FROM *model_name*

Unlike the standard SQL DELETE statement, the DELETE FROM *model* statement does not allow a WHERE clause.

model_name
Specifies the name of the model from which rows are to be deleted.

Example

The following statement removes the discovery results from the Credit model and all its associated model sub-tables, while preserving the definition of the model:

```
delete from credit;
```

4

Understanding the Model

After Red Brick Data Mine has discovered the relationships among your data, you must be able to make sense of these relationships in order to determine how to fine-tune your model to make it more accurate, or just to understand the “business rules” that the data mining process has discovered. This chapter explains the UNDERSTAND sub-table and how you can use it to analyze your discovery results.

This chapter is divided into the following main sections:

- The UNDERSTAND Sub-Table
- Understanding Discovery Results
- Using Discovery Results

The UNDERSTAND Sub-Table

The UNDERSTAND sub-table is automatically created when the model is created. After data has been inserted into the model, this table stores the associations found by Red Brick Data Mine during the discovery process. Each row in the UNDERSTAND sub-table represents an association between an input value (or conjunction of values) and an output value.

Only SELECT operations are allowed on the UNDERSTAND sub-table. For example:

```
select type, conjunctionid, importance, discrimination,
  account_balance from credit_understand
where type = 'I';
```

This section begins with a brief description of each column in the UNDERSTAND sub-table. A more complete discussion of how this table is used for the understanding process follows the descriptions.

The UNDERSTAND sub-table contains the following information.

Column Name	Column Type	Description
TYPE	CHAR(1)	The type of correlation discovered: **N** indicates a normal correlation; **M** indicates a minimal correlation **I** indicates an irrelevant correlation. Refer to "Type" on page 4-4.
CONJUNCTIONID	INTEGER	If the correlation exists as part of a conjunction, this column displays a number common to this and the other parts of the conjunction. If the correlation is not part of a conjunction, this column contains NULL.
INPUT_COLUMN_NAME	CHAR(128)	Name of input column.
INPUT_COLUMN_VALUE	CHAR(1024)	Value of the input column specified in INPUT_COLUMN_NAME.
IMPORTANCE	FLOAT	The measurement of the correlation between an input or conjunction of inputs on the given output. A higher value indicates greater importance.

Column Name	Column Type	Description
DISCRIMINATION	FLOAT	The distance between the importance of this input value on this output value and the nearest other output value. A higher Discrimination value indicates greater discrimination.
OCCURRENCE_COUNT	DECIMAL(12,0)	Count of the number of rows inserted into the model which had this input value or conjunction of input values for the given output value. This value is NULL when the correlation's Type is "I".
OCCURRENCE_FREQ	FLOAT	Percentage of rows inserted into the model which had this input value or conjunction of input values for the given output value. This value is NULL when the correlation's Type is "I".
OUTPUT_COLUMN	Same as the datatype of the output value	This column contains the name of the output column as specified in the CREATE MODEL statement.

Understanding Discovery Results

The UNDERSTAND sub-table provides the results of Red Brick Data Mine's discovery process, including:

- The strength and frequency of relationships between inputs and outputs
- Inputs or sets of inputs resulting in a given output value
- The differences between the significance of one set of correlations versus the significance of the next most common set.

This section explains each column of the UNDERSTAND sub-table in detail.

Type

The Type value shows the relevance of a relationship based on how often a certain input or set of inputs occurs together.

A *Normal* correlation occurs when the input value or group of input values has some impact on the given output, but is not exclusive to that output. For example, if some homeowners and some renters in the data have balanced accounts, “Home: Own” and “Home: Rent” are both Normal correlations for the output “Account_Status: Balanced.”

A *Minimal* correlation occurs when a particular output is produced *only* by one particular input or conjunction of inputs, and no other input or conjunction produces that output. For example, if the input “Home: Rent” produces only the output “Account_Status: Balanced,” and no other value in the “Home” column produces a balanced account status, then “Home: Rent” is a minimal condition for Account_Status: Balanced.

An *Irrelevant* correlation indicates an input value or group of input values that have no impact on the given output value; in other words, the combination of the specified input value and the specified output value does not occur in the data. These values might or might not have impact on other output values. For example, you might discover that Nbr_Children: 3 is an irrelevant condition for Account_Status: Balanced, meaning that no instances of Nbr_Children: 3 occurred in combination with a balanced account status in the data.

You might be able to drop values that indicate irrelevant correlations from your model.

ConjunctionID

The ConjunctionID value identifies the inputs included in a conjunction. A conjunction is a set of input values that appear together in connection with a given output value. For example, the input values for Marital_Status, Pmt_History, and Mo_Income, taken individually, might have little bearing on a customer’s account status, but when taken together, might indicate something more useful. For example, the discovery might find that married women who own their own businesses and who have one child tend to be better credit risks than people with other characteristics.

The ConjunctionID value is a number that occurs for each input value that is part of a conjunction. Each conjunction has its own number. For example, if "Marital_Status: Married," "Pmt_History: 12," and "Mo_Income: 5" are all part of a conjunction linking to a single output value, then each of these values will have the same ConjunctionID value.

You can identify conjunctions easily by ordering the UNDERSTAND sub-table by the ConjunctionID value. This value is NULL if the correlation is not part of a conjunction.

Note: If the same conjunction affects more than one output value, then the same ConjunctionID will occur in combination with multiple output values.

Input_Column_Name, Input_Column_Value

These columns show the column name and column value of the individual input, such as "Marital_Status: Married," or "Home: Rent".

Importance

This value shows the strength of the correlation between an input or conjunction and the given output. A higher number for this value indicates a higher relative number of times the given input or conjunction resulted in the given output. You should not look at this value alone; it can be deceptive if the number of occurrences is low. For example, if your discovery shows a high correlation between women who rent their homes and good credit, but there is only one instance of a woman who rents her home in your data set, then this high Importance score is misleading.

The Importance value is usually between 1 and 100, although it can be larger when it occurs with a conjunction.

Discrimination

Discrimination shows the distance between the importance of the specified input value on the specified output and the importance of the specified input value and the nearest other output. A higher number indicates a greater level of discrimination. For example, suppose the Importance of the conjunction "Home: Own" and "Pmt_History: 12" to the output "Account_Status: Balanced" was 100 and the Discrimination value was 5. This indicates that this conjunction has similar importance to another output (such as "Account_Status: Payment Late.") This is likely not as interesting as discovering that the Discrimination value was 90, which would indicate that the conjunction "Home: Own" and "Pmt_History: 12" occurs much less frequently with the next most likely output value than it does with the output value "Account_Status: Balanced."

Occurrence_Count

The Occurrence_Count value shows the number of times the particular input value or conjunction resulted in the given output value. For example, consider the following table:

Input	Output
Blue	Red
Red	Blue
Blue	Red

In this simple set of input/output, the Occurrence Count is as follows:

Blue/Red: 2
Blue/Blue: 0
Red/Blue: 1
Red/Red: 0

The Occurrence_Count value is NULL for correlations of type "I" (irrelevant).

Occurrence_Freq

This value shows how often an input value or conjunction is connected with a particular output value, expressed as a percentage. You should look at this value in connection with the Importance value to verify that the occurrence is common enough to consider important: A high Occurrence_Freq value does not necessarily mean a high Importance value. For example, the input value "Gender: Male" might have a similar frequency for multiple output values, but this can mean that there are more men than women in the Discovery data set.

The Occurrence_Freq value is NULL for correlations of type "I" (irrelevant).

Output_Column

This column has the same name as the model's output column and contains the values for that column. For example, for the output column "Account_Status," the Output_Column column would look similar to this:

```
Account_Status
---------------------
Balanced
Overdraft
Payment Late
```

Using Discovery Results

When Red Brick Data Mine performs discovery on the data you have inserted into your model, it stores the results of this discovery process in the UNDERSTAND sub-table. To make use of this data for your business, you must be able to view the data in a form that is meaningful to you. This will allow you to see which results are valuable and which components of your model (or your data) you might want to alter or discard in order to obtain useful results. You view the discovery results by submitting queries to the UNDERSTAND sub-table.

This section describes how you can use the UNDERSTAND sub-table to make sense of your data mining results.

UNDERSTAND Sub-Table Data

The UNDERSTAND sub-table, when populated with the results of Red Brick Data Mine's discovery process, contains rows of data representing correlations between various input values and the selected output value. Each row represents a relationship between a particular column containing a particular input value and the output column containing a particular output value. In the case of a conjunction, the relationship is between a group of input values that have the same ConjunctionID and a particular output value. For example, consider this small subset of the discovery results for the Credit model:

```
select type, conjunctionid, input_column_name,
   input_column_value, importance, account_status
from creditresults;
```

```
TYPE  ConjID Input            Value        Imp    Acct_Status
====  =====  =====            =====        ===    ===========
N     NULL   NBR_CHILDREN     2           86.22  Balanced
I     NULL   MARITAL_STATUS   Widowed      0.00  Balanced
N     NULL   PMT_HISTORY      0          100.00  Overdraft
N        3   MARITAL_STATUS   Single     300.00  Payment Late
N        3   HOME             Rent       300.00  Payment Late
N        3   MO_INCOME        4          300.00  Payment Late
```

In this example, the results show that:

- The input column Nbr_Children with the value 2 is a Normal correlation for the account status value Balanced, meaning that this input value occurred more often for the account status Balanced than it did for any other account status. The Importance value is relatively high as well, indicating that the input value Nbr_Children:2 might be a good predictor of account status Balanced.
- The input column Marital Status with the value Widowed is an Irrelevant correlation for the account status value Balanced. This means that the input column value Widowed did not occur in the data in combination with Account Status: Balanced, and is therefore not a good predictor of this account status.
- The input column Pmt_History with the value of 0 (meaning zero months of payment history) shows a high Importance in combination with Account Status Overdraft. It is also a Normal correlation; this input value occurred for Account Status: Overdraft more than it did for any other output value. Again, this correlation is a fair indicator that people with no credit history might have overdrawn accounts.
- The conjunction of the input values Marital Status: Single, Home: Rent, and Mo_Income: 4 (indicated by the value 3 in the ConjunctionID column, labeled “ConjID” in the example) shows that these three values, taken together, have a high Importance score in relation to Account Status: Payment Late. Note, however, that each of these input values, taken alone, might not indicate the same information as the three taken together. For example, a Mo_Income value of 4 (falling in the fourth of five “buckets”) by itself might suggest a better credit risk than it does when in conjunction with the other two input values. This consideration can be explored by looking at other columns in the UNDERSTAND sub-table.

Note that several of the UNDERSTAND sub-table columns (such as Discrimination and Occurrence_Freq) are missing from this example; these are also valuable in deciding which relationships are important and which are not as important. The complete UNDERSTAND sub-table is discussed in greater detail in the following sections.

Querying the UNDERSTAND Sub-Table

The UNDERSTAND sub-table, taken in its entirety, probably contains far too much data to make sense to you. By submitting queries to Red Brick Data Mine, you can distill the data into meaningful subsets that allow you to view the relationships found during the discovery process.

The SELECT operation is the only operation allowed on the UNDERSTAND sub-table. You cannot add to, delete from, or alter this table in any way.

The following SELECT statement queries the Credit_UNDERSTAND table. The results of this query displays information about those input values that occurred with the account status Balanced and that had high Importance scores, indicating that they might be good predictors of a Balanced account status.

```
select type, conjunctionid, input_column_name,
   input_column_value, importance
from credit_understand
where importance > 80 and account_status = 'Balanced'
order by importance desc, conjunctionid;


TYPE  ConjID Input              Value           Importance
====  ====== =====              =====           ==========
N          2 NBR_CHILDREN       1                   200.00
N          2 PMT_HISTORY        8                   200.00
N       NULL PMT_HISTORY        7                   100.00
N       NULL NBR_CHILDREN       2                    86.22
```

Note: The column headings for these results have been altered to allow them to fit on the page.

From the results of this query you can see that, in this set of data, three relationships satisfied the criteria (Importance of greater than 80 and Account_Status: Balanced). They have been ordered by the Importance value with the highest values first and then by ConjunctionID. It is usually useful to order your results by ConjunctionID; because each component of a conjunction has the same ConjunctionID, ordering in this way displays all the components of each conjunction together, which aids you in identifying them in large amounts of data.

Filtering Discovery Data Using the Occurrence Threshold

Several of the examples shown previously have not included any Occurrence information, although this information is important in making the determination about which relationships are meaningful and useful. You can produce smaller, easier-to-understand sets of query results by setting the Occurrence Threshold in the model creation statement, causing the discovery process to ignore any relationships that do not fall above your specified thresholds. With a large amount of data, setting these thresholds can filter out many non-interesting relationships during the discovery process, leaving you with more meaningful information to analyze.

Setting the Occurrence Threshold

The Occurrence Threshold in the CREATE MODEL and ALTER MODEL statements specifies the number of times, expressed as a percentage, that a particular relationship must occur before the relationship appears in the UNDERSTAND sub-table. The default Occurrence Threshold value is 3, meaning that a relationship must occur at least 3% of the time before the relationship is retrieved.

For example, suppose the discovery process identified a relationship between Nbr_Children: 3 and Account_Status: Balanced. However, upon examining the discovery results more closely, you find that there are only four instances in your data (which consists of thousands of rows) of customers with three children. If you set the Occurrence Threshold to 0, this relationship shows up in your discovery results, but if you set it higher (such as at the default level of 3%), this relationship is not added to the UNDERSTAND sub-table.

For more information about setting the Occurrence Threshold value in a CREATE MODEL statement, refer to “Occurrence Threshold” on page 2-13. For more information about changing the Occurrence Threshold in an ALTER MODEL statement, refer to “Altering Columns in a Model” on page 6-9.

RED BRICK®

5

Making Predictions

This chapter explains the process for using Red Brick Data Mine to perform prediction on your data. It is divided into the following main sections:

- The Prediction Process
- The PREDICT Sub-Table
- Using the PREDICT Sub-Table to Make Predictions

The Prediction Process

Prediction is the process of finding out, given the information found in the model during the discovery process, the expected output values for one or more given input rows. When you have mined your data and discovered trends and relationships, prediction helps you make informed decisions about new input data.

For example, the Credit model contains information about customers and their creditworthiness, based on a number of factors including income, home ownership, occupation, and others. Running discovery on this model and then examining the results in the UNDERSTAND sub-table tells you that people who have checking accounts, long credit histories, high credit card limits, and who own their homes tend to be better credit risks than people who have the opposite characteristics.

Suppose you have a new group of customers who are applying for credit, and you have the job of determining whether your bank will extend credit to them. Prediction allows you to run data for these new customers against your existing model and determine whether each customer is likely to be a good or a bad credit risk, based on the trends you found during discovery.

Of course, this method is not infallible. The prediction you generate is only as good as the discovery model you have defined; if the model has incomplete, inaccurate, or minimal data, then your predictions will reflect this. You might find that you will need to go back and redefine your model. Refer to Chapter 6, “Refining the Model,” for more information about refining your model.

The PREDICT Sub-Table

The PREDICT sub-table is created automatically when the model is created. This table is used after data has been inserted into the model, to predict new output values given a set of input values.

The PREDICT sub-table consists of the Cartesian product of all the unique values in each of the input and output columns. In other words, there is a prediction for every combination of values in the input and output columns.

Only SELECT operations are allowed against the PREDICT sub-table. For example:

```
select score, valid_rank, discrimination, account_status
   from credit_predict
where home = 'Rent'
and occupation = 'Professional';
```

PREDICT Sub-Table Columns

The PREDICT sub-table contains the following information:

Column Name	Column Type	Description
OUTPUTCOLUMN	Same as the output column.	The output column predicted value.
INPUTCOLUMN(1...N)	Same as the specified input column.	One column for each of the input columns.
PRE_COLUMNS(1...N)	Same as the type of the mapped column.	One column for each MAPPING RANGES column.
SCORE	INTEGER	The strength of the prediction. Higher values indicate greater strength.
VALID_RANK	INTEGER	For valid predictions, the rank of this prediction relative to other valid predictions. A rank of 1 means that this is the best prediction. This field is NULL if the prediction is not valid.

Column Name	Column Type	Description
VALID	CHAR(1)	“Y” if the prediction is valid; “N” if the prediction is not valid.
DISCRIMINATION	FLOAT	The difference in score between the best prediction and the next best prediction for the given inputs. (This value is NULL for all but the best prediction.)

Understanding Prediction Results

The PREDICT sub-table provides the results of Red Brick Data Mine’s prediction process, including:

- The strength of the prediction.
- Ranking of predictions for each output value.
- The difference in strength between the best prediction and the second best prediction.

This section explains each column in the PREDICT sub-table in detail.

Outputcolumn

This field indicates the output column for the specified model. The name of the output column corresponds to the output column name as defined in the model. The type of the output column corresponds to the model-defined output type. For example, in the Credit model, this column is Account_Status, and its datatype is CHAR(20).

In the case of mapped columns, there are two columns for each mapped column: the *pre-mapped* and the *post-mapped* column. The *pre-mapped* column is used for prediction constraints, but it always contains NULL when displayed. The *post-mapped* column contains the bucket labels defined when the model was created.

Inputcolumn

One column is defined in the PREDICT sub-table for each input column defined in the model. The name and type of each of these input columns will be the same as those identified in the associated model. If an input column is mapped to a datatype (for more information, refer to “Mapping Bucket Datatypes” on page 2-5), then the input column type corresponds to the definition specified in the *mapping_clause*.

In the case of mapped columns, there are two columns for each mapped column: the *pre-mapped* and the *post-mapped* column. The *pre-mapped* column is used for prediction constraints, but it always contains NULL when displayed. The *post-mapped* column contains the bucket labels defined when the model was created.

Pre_Column

If a column is mapped, Pre_Column contains the pre-mapped value of that column. For example, if you have an INTEGER column that has been mapped to a CHAR(4), then Pre_Column contains the column's original INTEGER value.

When displayed, this column contains NULL; you generally do not need to display this column.

Score

The Score value is an integer value that indicates the strength of the prediction. A higher number in this field indicates a stronger prediction. This value is similar to Importance in the UNDERSTAND sub-table. For more information about the Importance value, refer to "Importance" on page 4-5.

Valid_Rank

If a prediction is valid, this field indicates its rank relative to all the other valid predictions. If the prediction is not valid, this field has the value 0 (zero).

Valid

This field indicates whether or not the prediction is valid. If it is valid, the value in the Valid field is "Y"; if it is not valid, the value is "N". A valid prediction is one that satisfies the minimal requirements for the predicted output value, as defined in the model's Discrimination Threshold and Occurrence Threshold parameters.

Discrimination

This field indicates the difference in Score between the best prediction and the nearest other prediction for the given inputs. If this value is small, this might indicate that this output value and the nearest other are similarly likely to occur for the specified input value, or it might indicate that Red Brick Data Mine does not have enough information to make a strong distinction between the two predictions. If this value is large, it indicates that the likelihood of the most likely output value occurring is significantly stronger than the next one in rank.

For example, in the Credit model, the three possible output values (using the Account_Status field) are Balanced, Payment_Late, and Overdraft. If, for a given set of inputs, the Discrimination between an output of Balanced and an output of Overdraft is 5, then this might indicate that the inputs used are not very valuable in determining account status (it cannot be accurately predicted that a person with the given characteristics will have a balanced or an overdrawn account); if the Discrimination is 90, then this might indicate that the higher-ranked prediction is useful in predicting the desired outcome (a person with the given characteristics is significantly more likely to have a balanced account than an overdrawn one).

If the model's Discrimination Threshold is set to 0.0, ranked predictions appear in the PREDICT sub-table for all the possible output values. A Discrimination value is calculated only for the difference between the Score of the best prediction and that of the second best prediction in this case; Discrimination values for predictions on the other output values are NULL.

If the Discrimination Threshold is set to a number greater than 0.0, only the best prediction appears in the PREDICT sub-table. However, note that if the discrimination value of the best prediction is less than or equal to the Discrimination Threshold value, no prediction is generated.

Using the PREDICT Sub-Table to Make Predictions

You make predictions by querying the PREDICT sub-table and constraining on specific input column values. For example, suppose you have created a Credit model that analyzes the effect of inputs such as income, home ownership, and other traits on creditworthiness. You have performed discovery on your model, and refined it until you feel that it is an accurate predictor of creditworthiness. At this point, you select rows from the PREDICT sub-table or join rows from a different table with the PREDICT sub-table to get predictions about which customers should or should not receive credit.

You can do two types of predictions with Red Brick Data Mine:

- Case (or single-row) predictions, where you specify a single row of values and obtain a prediction for only this row.
- Batch prediction, where you insert data into a table, join this table with the PREDICT sub-table, and then analyze the predictions by selecting desired data from the PREDICT sub-table.

Case Predictions

A case (or single-row) prediction is one where you select a single row of data from the PREDICT sub-table. The results return a prediction for each of the possible output values in the model, ranked according to their validity.

For example, suppose you are a bank employee whose job it is to determine if a customer should be extended credit. Using the Credit model, you can enter the customer's information and then query the Credit_PREDICT table to determine the best prediction about the customer's potential creditworthiness.

The SELECT statement might look something like this:

```
select account_status, score as Sc, valid_rank as Rank,
   valid as Valid?, discrimination as Disc
from creditfuture
where gender = 'Female'
   and marital_status = 'Married'
   and nbr_children = 2
   and occupation = 'Principal'
   and home = 'Own'
   and mo_expenses = 2500
   and mo_income = 5000
   and checking_acct = 'yes'
   and savings_acct = 'yes'
   and mastercard_limit = 'High'
   and visa_limit = 'Very High'
   and ax_limit = 'Low'
   and pmt_history = 11;
```

Red Brick Data Mine assumes that any values not supplied are unconstrained; that is, all the possible values in that column’s domain. For a “true” case prediction, you should supply a values for each available input.

The following results represent the prediction for the query:

```
ACCOUNT_STATUS       Sc    Rank Valid? Disc
==============       ===   ==== ====== ====
Balanced               221    1 Y      51.00
Overdraft              110    2 Y       NULL
Payment Late            73    3 Y       NULL
```

The query returns one prediction for each possible output values in the model. In this case, there are three predictions (one for account_status Balanced, one for Overdraft, and one for Payment Late).

Analyzing these results, you can see that the best prediction (indicated by the rank of 1) is Account_Status: Balanced. In other words, according to this model, a person with the characteristics described in the query is most likely to have a Balanced account. The second most likely output, Overdraft, is 51% less likely to occur, as indicated by the 51.00 value in the Discrimination column for the Balanced prediction.

Note: To obtain the prediction shown in the example, the model’s Discrimination Threshold must be set to 0.0. If the Discrimination Threshold is set to a value greater than 0.0, only the prediction for a Balanced account status (the best prediction) appears.

Batch Predictions

A batch prediction is one where you join the PREDICT sub-table with one or more database tables containing the rows for which you want predictions.

For example, suppose that you are responsible for determining whether each person in a large group (such as the daily group of applicants for credit in your district) should be extended credit. You have the applicants' data, obtained from their credit applications, in a table named Applicants. Your model, Credit, has as its inputs the same inputs required of the credit applicants. A batch query against this model would look similar to the following:

```
select name, score, valid, valid_rank, discrimination,
     account status
from credit_predict natural join applicants;
```

Red Brick Data Mine will predict, using the information it deduced during the discovery process, the most likely account status of each person in the Applicants table. The results will look similar to these partial results:

NAME	SCORE	VALID	VALID_RANK	DISCRIMINATION	ACCOUNT_STATUS
Smith, B	334	Y	1	8.00	Overdraft
Smith, B	310	Y	2	NULL	Balanced
Smith, B	303	Y	3	NULL	Payment Late
Lee, G	297	Y	1	11.00	Overdraft
Lee, G	266	Y	2	NULL	Payment Late
Lee, G	257	Y	3	NULL	Balanced
Jackson, C	282	Y	1	9.00	Payment Late
Jackson, C	259	Y	2	NULL	Overdraft
Jackson, C	225	Y	3	NULL	Balanced

Notice that three predictions—ranked 1, 2, and 3—are returned by this query, because there are three possible output values. Note that if the model's Discrimination Threshold was set to a value greater than 0.0, only those predictions with a Valid_Rank value of 1 appear.

Unconstrained SELECT* Queries

Do not issue an unconstrained SELECT * query against the PREDICT sub-table; this kind of query will return an extremely large number of rows.

The number of rows returned by an unconstrained SELECT * query is determined by the following calculation:

```
(CIV1 + 1) * (CIV2 +1) ... (CIVn +1) * (COV)
```

where:

CIV1 is the count of unique input values in column 1 inserted into the model;

CIV2 is the count of unique input values in column 2 inserted into the model;

CIV*n* is the count of unique input values in column *n* inserted into the model;

n is the number of input columns in the model definition; and

COV is the number of unique output values inserted into the model.

So, for example, if you have a model with 5 input columns with the following number of unique values in each input column and in the output column (I1 through I5 and O, respectively):

Column	Unique Values
I1	10
I2	6
I3	9
I4	5
I5	4
O	7

then the calculation is:

(10 + 1) * (6 + 1) * (9 + 1) * (5 + 1) * (4 + 1) * 7 = 161,700 rows

The input columns each have 1 added for the NULL value; the output column does not. For mapped input columns, the unique count is the number of ranges defined for that column (‘Low’, ‘Med’, ‘High’ is 3, for example) plus one for the NULL value.

Improving Performance by Constraining Values

A SELECT operation from the PREDICT sub-table is more efficient if inputs are constrained to some set of values; this is because Red Brick Data Mine generates a set of output values based on the given set of input values. If any input value is not restricted, the Red Brick Warehouse server needs to generate every single combination of that column's input value. This process can be very time consuming, especially for large numbers of inputs.

Any input column that is NULL is assumed to be unknown. You can use the IS NULL predicate to make predictions with input values that are unknown. For example, if you want to predict the creditworthiness of a customer but do not know the customer's payment history, you can use the following query:

```
select score, valid_rank, valid, discrimination,
   account_status
from credit_predict
where gender = 'Male'
   and marital_status = 'Married'
   ...
   and pmt_history is null;
```

This query will return predictions using the specified column constraints, and assume that the Pmt_History value is unknown. If you do not specify IS NULL for Pmt_History in this query, Red Brick Data Mine will generate output values based on every combination of the specified input values and all possible values for Pmt_History.

Constraining on Mapped Columns

If the output column in a model is mapped, you cannot constrain on the pre-mapped column name when querying the PREDICT sub-table, even though both the pre-mapped and the post-mapped column exist in the table.

However, if an input column is mapped, you can specify either the pre-mapped column name or the post-mapped column name. For performance reasons, it is preferable to constrain on pre-mapped input columns rather than post-mapped input columns.

If the mapped value for a given column is unknown, both the pre-mapped column and the post-mapped column will contain NULLs; to constrain on rows where the value of the mapped column is unknown, you must use the IS NULL predicate for the post-mapped column. Otherwise, the prediction query cannot cover the case where the column value is unknown. Note that you cannot use the IS NULL predicate for the pre-mapped column; if you do so, an error will result.

For example, if the Mo_Income column is mapped but you want to constrain on rows where the Mo_Income value is unknown, a case prediction query might look like this:

```
select score, valid_rank, valid, discrimination,
   account_status
from credit_predict
where home = 'Rent'
   and gender = 'Male'
   and occupation = 'Principal'
   and mo_income_mapped is null;
```

Setting the Discrimination Threshold

The Discrimination Threshold specifies how far apart, expressed as a percentage, the Score values of two relationships with the same output must be before they appear in the PREDICT sub-table. The default value for the Discrimination Threshold is 0.0, meaning that predictions for each of the possible output values appear in the PREDICT sub-table.

If you set the Discrimination Threshold to 0, every relationship, regardless of discrimination, appears in the PREDICT sub-table. A ranked prediction is generated for each possible output value, with the best prediction showing a Discrimination percentage and all other predictions showing NULL. Setting the Discrimination Threshold to a higher value allows you to filter out those relationships that might not be interesting to you because they are too close together to predict, but this will result in only the best prediction appearing in the PREDICT sub-table.

For instance, if your results showed that Married: Yes and Account_Status: Balanced has a high Score value but a low Discrimination, while Married: No also has a high Score value, this might indicate that marital status is not a good predictor of creditworthiness. Setting the Discrimination Threshold to filter out such relationships helps you to identify more easily the relationships that might prove better indicators for your output values.

For more information about setting the Discrimination Threshold value in a CREATE MODEL statement, refer to "Discrimination Threshold" on page 2-13. For more information about changing the Discrimination Threshold in the ALTER MODEL statement, refer to "Altering Columns in a Model" on page 6-9.

Testing Your Predictions Using Known Data

If you are unsure whether your predictions are accurate, you can test them by generating predictions on known data and examining the results for accuracy. For example, you can determine whether your model is competent at predicting good credit risks by performing prediction using data about people whom you know are good credit risks. If your model is accurate, it should make the correct prediction about the known data. If your model does not make the appropriate predictions, you should redefine your model or use different data (or a larger size domain) for your discovery process.

RED BRICK®

6

Refining the Model

Data mining is an iterative process. You will probably need to refine your model a number of times as you go through the process of discovery, understanding, and prediction.

For instance, you might find that certain columns are more (or less) relevant to your study or that you want to change the way in which information is grouped. You might discover that you are filtering out too much, or not enough, information with your occurrence or discrimination thresholds. It is quite likely that you will find that you need to tune your model so that it will provide you with the data you need in a form that is useful to you.

This chapter describes how to refine your model by adding, removing, or modifying its columns. It contains the following sections:

- ALTER MODEL Command
- Adding Columns to a Model
- Dropping Columns from a Model
- Altering Columns in a Model

ALTER MODEL Command

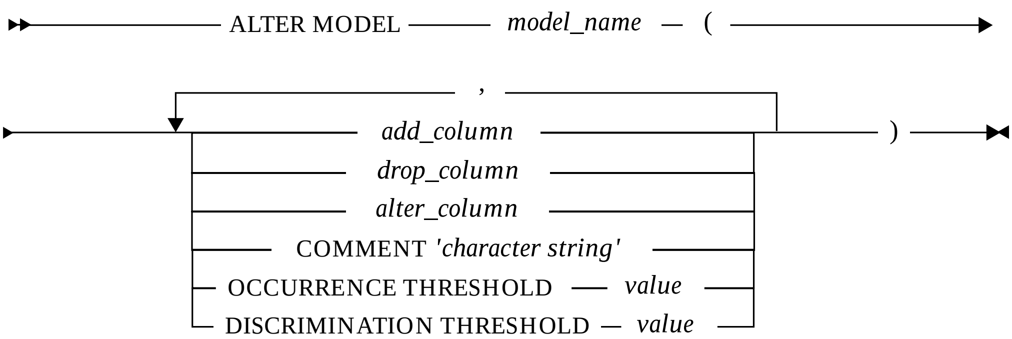
In the process of refining your model, you will need to modify it—add or remove columns, change the discrimination or occurrence thresholds, or make other changes so that your model accurately reflects your needs. You make changes to the model by using the ALTER MODEL statement.

You can alter a model by adding columns, dropping columns, making changes to column definitions, or adding comments to columns. You can also change the values for the occurrence threshold and the discrimination threshold.

Note: If a model contains no rows, you can alter a model freely. If the model contains any rows, there are some restrictions on ALTER MODEL operations. Refer to the descriptions of the individual operations for more information.

Syntax

The following syntax diagram shows how to construct an ALTER MODEL statement:



model_name

Specifies the name of the existing model you are altering.

add_column

Describes the column to be added to the specified model. For more information about adding columns to a model, refer to "Adding Columns to a Model" on page 6-4.

RED BRICK®

USA SALES OFFICES

1040 Crowne Point Parkway, Suite 250, Atlanta, GA 30338 +1 770 804 2440
2215 York Road, Suite 409, Oak Brook, IL 60521 +1 630 472 8500
1120 Avenue of the Americas, 4th Floor, New York, NY 10036 +1 212 626 6815
5314 Arapaho Road, Dallas, TX 75248 +1 972 702 1750
2010 Corporate Ridge, 7th Floor, McLean, VA 22102 +1 703 883 9310

UK SALES OFFICE

Red Brick Systems UK Ltd., 45 Berkeley Square, Mayfair, London W1A 1EB
United Kingdom +44 171 290 8373

AUSTRALASIA HEADQUARTERS

Red Brick Systems Australasia Pty. Ltd., Level 20, 99 Walker Street,
North Sydney, NSW 2060 Australia +61 02 9911 7744

JAPAN HEADQUARTERS

Red Brick Japan Co. Ltd., Level 16 Shiroyama Hills, 4-3-1 Toranomon,
Minato-ku, Tokyo 105 Japan +81 3 5403 4638