About This Document
   Purpose ix  
   Audience ix  
   Organization x  
   Related Documentation xi  
   Conventions xiii  
      Syntax Notation xiii  
      Syntax Diagrams xiv  
      Keywords and Punctuation xvi  
      Identifiers and Names xvi  
   Customer Support xvii  
      Support Solutions Warehouse xvii  
      General and Technical Questions xvii  
      Troubleshooting Tips xix  
      Documentation Questions and Comments xix  

1 What's New in this Release
   Query-Priority Concurrency  1-2  
   Parallel REORG and Deferred Parallel Index Building  1-3  
   Parallel Query and Parallel Load on Windows NT  1-3  
   Where to Find Information  1-4
## Contents

2 Working with a Versioned Database

- Determining Whether You Need Versioning 2-2
  - Cost/Benefit Analysis 2-2
  - Costs of the Version Log 2-2
  - Load Window 2-3
  - Updating Records in a Referenced (Dimension) Table 2-4
  - Incremental Loads During Production Hours 2-4
  - New Table and Index Data into New Segments 2-4
  - Increased Recoverability 2-5
- Understanding the Version Log 2-6
  - Database Blocks 2-6
  - Multiple Read Operations 2-7
  - Changed Blocks 2-9
  - Versioned DELETE Operations 2-9
- Creating and Sizing the Version Log 2-10
  - Allocating the Disk Space for the Version Log 2-10
  - Adding Space to the Version Log 2-11
  - Enabling Versioning 2-11
  - Dropping the Version Log 2-12
- Locking Mechanism 2-13
- Single Statement Transactions 2-15
- Isolation Level 2-15
- Strategies for Maintaining a Versioned Database 2-18
- Vacuum Cleaner Daemon/Thread 2-19
- Trickle Feed Applications 2-20
- Example—Creating a Versioned Aroma Database 2-20
3 Warehouse Administrator’s Guide

| Warehouse Processes on UNIX Platforms | 3-2 |
| Interprocess Communication | 3-3 |
| Warehouse Daemon Process | 3-4 |
| Server Processes | 3-4 |
| Administration Daemon Process | 3-4 |
| Log Daemon Process | 3-4 |
| Process Checker Daemon | 3-5 |
| Vacuum Cleaner Daemon | 3-5 |
| Shared Memory | 3-5 |

| Warehouse Service on Windows NT | 3-6 |
| Designed for Microsoft BackOffice | 3-6 |
| Communication Between Threads | 3-7 |
| Warehouse API Thread | 3-8 |
| Red Brick Warehouse Server Threads | 3-8 |
| Administration Thread | 3-8 |
| Log Thread | 3-9 |
| Red Brick Warehouse Listener Thread | 3-9 |
| Control-C Coordination Thread | 3-9 |
| Process Checker Thread | 3-9 |
| Vacuum Cleaner Thread | 3-10 |
| Shared Memory | 3-10 |

Versioned Databases in a DSS Environment | 3-11 |
Modifying the Configuration File on UNIX Platforms | 3-12 |
Modifying the Configuration File on Windows NT Platforms | 3-15 |

Appendix B—Configuration File | 3-18 |
| RBWAPI MAX_ACTIVE_DATABASES | 3-18 |
| RBWAPI PROCESS_CHECKING_INTERVAL | 3-18 |
| OPTION TMU_VERSIONING | 3-18 |
| OPTION TMU_COMMIT_RECORD_INTERVAL | 3-18 |
| OPTION TMU_COMMIT_TIME_INTERVAL | 3-18 |
| OPTION VERSIONING | 3-18 |
| OPTION TRANSACTION_ISOLATION_LEVEL | 3-19 |
| TUNE INDEX_TEMPSPACE_DUPLICATESPILLPERCENT | 3-19 |

Appendix C—System Tables and Dynamic Statistic Tables | 3-20 |
| RBW_LOADINFO Table | 3-20 |
| RBW_SEGMENTS Table | 3-22 |
| RBW_STORAGE Table | 3-25 |
| RBW_USERAUTH Table | 3-26 |
| DST_COMMANDS Table | 3-29 |
| DST_DATABASES Table | 3-32 |
| DST_LOCKS Table | 3-35 |
| DST_SESSIONS Table | 3-36 |

Fully Qualified Database Pathnames (Windows NT only) | 3-37 |
4 SQL Reference Guide

ALTER DATABASE  4-2
CREATE INDEX  4-8
SET VERSIONING  4-10
SET TRANSACTION ISOLATION LEVEL  4-11
New Scalar Functions  4-12
   EXP  4-12
   LN  4-13
   SQRT  4-15
   Macros for Statistical Functions  4-16
   LENGTH  4-19
SET FIRST DAYOFWEEK Command  4-21
   Syntax  4-21
   Usage Notes  4-21
Supported Date Formats  4-22
   Input Formats  4-22
   U.S.-English Locales  4-22
   All Other Locales  4-22
   Entering the Year Datepart in the Japanese Locale  4-23
   Output Formats  4-23
   U.S. English  4-23
   All Other Locales  4-24
   ANSI_DATETIME_DISPLAY_FORMAT option  4-24
Miscellaneous Changes and Additions  4-26
   Maximum Row Size  4-26
   Chapter 3 of the SQL Reference Guide  4-26
5 Table Management Utility Reference Guide
Reorganizing Tables and Indexes 5-2
The REORG Operation 5-2
Data Processing During the REORG Operation 5-5
REORG Syntax 5-9
Usage Notes 5-14
SET Statements and Parameters to Control Behavior 5-17
Understanding Versioned LOAD Operations 5-22
Benefits 5-22
Costs 5-23
Enabling Versioning for the TMU 5-26
Setting Versioned Commit Intervals 5-27
SET TMU COMMIT RECORD INTERVAL 5-28
SET TMU COMMIT TIME INTERVAL 5-29
Miscellaneous Changes and Additions 5-31
SET TMU BUFFERS Command 5-31
EBCDIC to ASCII Conversion 5-31
TMU Aggregate Operators 5-33
Datetype Fieldtypes 5-34
Fieldtype Conversions 5-35
Locales 5-35

6 Installation and Configuration Guide
Kernel Parameters 6-2
Digital AlphaServer 6-2
HP 9000 Computer 6-4
Sun SPARC-Based Systems 6-6
Silicon Graphics Servers 6-8
New Files 6-9
UNIX Systems 6-9
Windows NT Systems 6-9
Locales 6-10
Required Disk Space (UNIX Platforms) 6-14
CD-ROM Mount Command (NCR Platform Only) 6-14

7 Red Brick Vista User’s Guide
From Columns in Hierarchies Must Be NOT NULL 7-2
Online Copies of Query Rewrite Case Studies 7-2
SELECT Privileges on Precomputed Views 7-2

8 ODBC Connectivity Guide
Sample ODBC Program 8-2
ODBClib Dynamically Linked Shared Library (AIX platform only) 8-2
Contents

9 Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms
   Parallel Query Tuning Parameters  9-2
   Enabling Parallel Query Processing  9-3
   Limiting I/O Contention with the FILE_GROUP Parameter  9-4
   Increasing Disk Group Tasks with the GROUP Parameter  9-6
   Limiting Available Tasks  9-8
      TOTALQUERYPROCS  9-8
      QUERYPROCS  9-9
   Usage Notes for TOTALQUERYPROCS and QUERYPROCS  9-9
   Setting Minimum Row Requirements with ROWS_PER_TASK Parameters  9-10
      ROWS_PER_SCAN_TASK  9-11
      ROWS_PER_FETCH_TASK and ROWS_PER_JOIN_TASK  9-13
   Forcing the Number of Parallel Tasks with the FORCE_TASKS Parameters  9-16
      FORCE_TASKS Syntax  9-17
      FORCE_SCAN_TASKS  9-19
      FORCE_FETCH_TASKS and FORCE_JOIN_TASKS  9-20
      FORCE_HASHJOIN_TASKS  9-22
   Enabling Partitioned Parallelism for Aggregation  9-23
   Other Limitations to Parallelism  9-25
   Analysis of System Resources and Workload  9-27
      Disk Usage  9-27
      Memory Usage  9-28
      CPU Allocation  9-30
   Tuning for Specific Query Types  9-31
      Parallel STARjoin Queries  9-31
      Parallel Table Scans  9-33
      SuperScan Technology  9-34
      About Reasonable Values  9-34
   Summary: Basic Guidelines  9-35

10 Messages and Codes Reference Guide
   Chapter 1, “Event Log Messages”  10-2
   Chapter 2, “Informational, Warning, and Error Messages”  10-5

Index
About This Document

**Purpose**

This document supplements the Red Brick® Warehouse documentation set. It contains new and changed information.

**Audience**

The intended users of this guide are system and database administrators and managers who are responsible for system administration, the administration of a Red Brick Warehouse on a Windows NT or UNIX system, and the design and implementation of warehouse databases. In this guide, these users are collectively designated as the warehouse administrator.

Knowledge of the operating system, basic system administration procedures, and relational databases is assumed.
About This Document

This document contains information that supplements the Red Brick Warehouse 5.1 documentation set and is organized as follows:

Chapter 1, “What’s New in this Release,” gives a brief overview of the new features in this release.

Chapter 2, “Working with a Versioned Database,” provides conceptual and practical information about whether you need concurrency and how concurrency works in Red Brick Warehouse.


Chapter 4, “SQL Reference Guide,” includes new and changed syntax information that supplements the SQL Reference Guide.

Chapter 5, “Table Management Utility Reference Guide,” includes new and changed syntax information that supplements the Table Management Utility Reference Guide.


Chapter 8, “ODBC Connectivity Guide,” includes additions and changes to the ODBC Connectivity Guide.

Chapter 9, “Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms,” Provides information on parallel query administration on systems running the Windows NT operating system.

Chapter 10, “Messages and Codes Reference Guide,” lists new and changed messages, along with their causes and responses, for this release.
### Related Documentation

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

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation and Configuration</td>
<td>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.</td>
</tr>
<tr>
<td>Guide</td>
<td></td>
</tr>
<tr>
<td>Warehouse Administrator’s</td>
<td>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.</td>
</tr>
<tr>
<td>Guide</td>
<td></td>
</tr>
<tr>
<td>Table Management Utility</td>
<td>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.</td>
</tr>
<tr>
<td>Reference Guide</td>
<td></td>
</tr>
<tr>
<td>SQL Self-Study Guide</td>
<td>Example-based review of SQL and introduction to the RISQL extensions, the macro facility, and Aroma, the sample database.</td>
</tr>
<tr>
<td>RISQL Entry Tool and</td>
<td>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.</td>
</tr>
<tr>
<td>RISQL Reporter User’s Guide</td>
<td></td>
</tr>
<tr>
<td>Messages and Codes Reference</td>
<td>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.</td>
</tr>
<tr>
<td>Guide</td>
<td></td>
</tr>
<tr>
<td>Release Notes</td>
<td>Information pertinent to the current release that was unavailable when the documents were printed.</td>
</tr>
</tbody>
</table>
In addition to the standard documentation set, the following documents are included for specific sites:

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Red Brick Vista User’s Guide</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>SQL-BackTrack for Red Brick Warehouse User’s Guide</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Client Connector Pack Installation Guide</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ODBC Connectivity Guide</strong></td>
<td>Information about ODBC conformance levels and instructions for compiling and linking an ODBC application using the Red Brick ODBClib SDK.</td>
</tr>
<tr>
<td><strong>Red Brick Data Mine User’s Guide</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

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 Red Brick Warehouse documentation set 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, and variables appear in Palatino italic or Courier italic, depending on the context.
- Document titles always appear in Palatino italic.
- 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:

<table>
<thead>
<tr>
<th>Command Element</th>
<th>Example</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values and parameters</td>
<td>table_name</td>
<td>Items that you replace with an appropriate name, value, or expression are in italic type style.</td>
</tr>
<tr>
<td>Optional items</td>
<td>[ ]</td>
<td>Optional items are enclosed by square brackets. Do not type the brackets.</td>
</tr>
<tr>
<td>Choices</td>
<td>ONE</td>
<td>TWO</td>
</tr>
<tr>
<td>Required choices</td>
<td>{ONE</td>
<td>TWO}</td>
</tr>
<tr>
<td>Default values</td>
<td>ONE</td>
<td>TWO</td>
</tr>
<tr>
<td>Repeating items</td>
<td>name, ...</td>
<td>Items that can be repeated are followed by a comma and an ellipsis. Separate the items with commas.</td>
</tr>
<tr>
<td>Language elements</td>
<td>() , ; .</td>
<td>Parentheses, commas, semicolons, and periods are language elements. Use them exactly as shown.</td>
</tr>
</tbody>
</table>
**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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶️</td>
<td>Statement begins.</td>
</tr>
<tr>
<td></td>
<td>Statement syntax continues on next line. Syntax elements other than complete statements end with this symbol.</td>
</tr>
<tr>
<td>▶️</td>
<td>Statement continues from previous line. Syntax elements other than complete statements begin with this symbol.</td>
</tr>
<tr>
<td>▶️</td>
<td>Statement ends.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Required item in statement.</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Optional item.</td>
</tr>
<tr>
<td>DBA TO</td>
<td>Required item with choice. One and only one item must be present.</td>
</tr>
<tr>
<td>CONNECT TO</td>
<td>Optional item with choice. If a default value exists, it is printed in <strong>bold</strong>.</td>
</tr>
<tr>
<td>SELECT ON</td>
<td>Optional items. Several items are allowed; a comma must precede each repetition.</td>
</tr>
<tr>
<td>ASC</td>
<td></td>
</tr>
<tr>
<td>DESC</td>
<td></td>
</tr>
</tbody>
</table>
The syntax elements shown above are combined to form a diagram as follows:

```
GRANT ALL PRIVILEGES ON ALL TABLE_NAME TO db_user_name, role_name, PUBLIC
```

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:

```
SELECT select_list FROM from_clause WHERE where_clause
```

The point-of-reference diagram is then followed by an expanded diagram of the shaded portion—in this case, the `select_list`:
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
What's New in this Release

This chapter provides an overview of the new features in this release and describes where to find the information in this manual. The following topics are included:

- Query-Priority Concurrency
- Parallel REORG and Deferred Parallel Index Building
- Parallel Query and Parallel Load on Windows NT
- Where to Find Information
What's New in this Release
Query-Priority Concurrency

Query-Priority Concurrency

Decision support applications typically involve analysis of historical data. With Red Brick® Warehouse, this analysis translates to querying the database. In contrast to online transaction processing (OLTP) databases, everything in Red Brick Warehouse is optimized for query speed. OLTP systems typically are optimized to update small numbers of records very quickly, and updates to the database take precedence over analytic queries. Red Brick Warehouse is optimized for the opposite priorities—query performance takes precedence over update performance.

With previous versions of Red Brick Warehouse, updates to the database (including SQL INSERT, UPDATE, and DELETE operations and TMU LOAD DATA operations) require either that the whole database is not available for querying or that the table(s) being modified is not available for querying during these operations. Beginning with Version 5.1.4, you can use query-priority concurrency to allow updates and queries to occur at the same time on a Red Brick Warehouse database.

Query-priority concurrency is a concurrency scheme designed for data warehouse applications. It allows data modification to take place simultaneously, or concurrently, with query operations. This is achieved without compromising query performance.

OLTP concurrency models are typically optimized for a large number of small operations, both modification and read (query), that access only a small number of records. Data warehouse queries typically access a very large number of records from the database. Query-priority concurrency allows modification of the database to occur concurrently with data warehouse queries.

Red Brick Warehouse uses versioning to implement query-priority concurrency. With versioning, database administrators can configure versioned databases to allow concurrent modification and querying of the database. For details about versioning, refer to Chapter 2, “Working with a Versioned Database,” in this document.
**What's New in this Release**

*Parallel REORG and Deferred Parallel Index Building*

The TMU REORG operation is now capable of reorganizing tables and indexes in parallel. On multi-processor systems, parallel processing can dramatically speed up the time to perform a REORG operation on large tables and on the indexes of large tables.

Additionally, the REORG operation can be used to build indexes using parallel processing. There is a new way to create a deferred index, which creates an index without populating it with data and then uses the parallel REORG features to populate the index using parallel processing.

For detailed information about the REORG operation, refer to “Reorganizing Tables and Indexes” on page 5-2. For the syntax and information about building deferred indexes, refer to “CREATE INDEX” on page 4-8.

**Parallel Query and Parallel Load on Windows NT**

Parallel query and load are now available on Windows NT platforms as well as on UNIX platforms. Parallel query takes advantage of computers with multiple processors to process large queries more efficiently. Similarly, the Parallel Table Management Utility takes advantage of computers with multiple processors to load data into a database more efficiently.

For detailed information on parallel query on Windows NT platforms, refer to Chapter 9, “Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms.”
Where to Find Information

This document contains both new and changed information in the following chapters:

- Chapter 1, “What’s New in this Release”
- Chapter 2, “Working with a Versioned Database”
- Chapter 3, “Warehouse Administrator’s Guide”
- Chapter 4, “SQL Reference Guide”
- Chapter 5, “Table Management Utility Reference Guide”
- Chapter 6, “Installation and Configuration Guide”
- Chapter 7, “Red Brick Vista User’s Guide”
- Chapter 8, “ODBC Connectivity Guide”
- Chapter 9, “Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms”
- Chapter 10, “Messages and Codes Reference Guide”

The following table lists where you can find specific information in this document.

<table>
<thead>
<tr>
<th>To Find...</th>
<th>Refer To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual information about versioned databases</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>Steps to enable versioning for a database</td>
<td>Page 2-11</td>
</tr>
<tr>
<td>Syntax for the ALTER DATABASE command</td>
<td>Page 4-2</td>
</tr>
<tr>
<td>A description of the two isolation levels</td>
<td>Page 2-15</td>
</tr>
<tr>
<td>Changed UNIX kernel parameters</td>
<td>Chapter 6</td>
</tr>
<tr>
<td>New syntax for TMU REORG operations</td>
<td>Page 5-9</td>
</tr>
<tr>
<td>How to use the TMU with a versioned database</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>Changes to the system tables and DSTs</td>
<td>Page 3-20</td>
</tr>
<tr>
<td>New rbw.config file parameters</td>
<td>Page 3-18</td>
</tr>
<tr>
<td>A description of the Warehouse process for UNIX platforms</td>
<td>Page 3-2</td>
</tr>
<tr>
<td>A description of the Warehouse threads for Windows NT platforms</td>
<td>Page 3-6</td>
</tr>
<tr>
<td>Additions and changes to the Red Brick Vista User’s Guide</td>
<td>Chapter 7</td>
</tr>
<tr>
<td>Information on parallel query on Windows NT</td>
<td>Chapter 9</td>
</tr>
<tr>
<td>New error messages for this release</td>
<td>Chapter 10</td>
</tr>
</tbody>
</table>
This chapter covers issues and procedures involved with versioned databases. The following topics are included:

- Determining Whether You Need Versioning
- Understanding the Version Log
- Creating and Sizing the Version Log
- Locking Mechanism
- Single Statement Transactions
- Isolation Level
- Strategies for Maintaining a Versioned Database
- Vacuum Cleaner Daemon/Thread
- Trickle Feed Applications
- Example—Creating a Versioned Aroma Database
Determining Whether You Need Versioning

Versioning allows you to query the database while it is being changed. Versioning also provides an increased level of recoverability in case of a catastrophic failure. If neither of these things are important to the operation of your database, then you probably do not need to use versioning.

This section provides information on the cost/benefit analysis you must do to determine whether versioning is right for your application, as well as describing some scenarios that are appropriate for versioned databases.

Cost/Benefit Analysis

As with most database administration issues, deciding whether to use concurrency involves some cost/benefit analysis. There are no right or wrong answers. Costs include the disk space for the version log, increased complexity of your system, and slightly more administrative duties. Benefits include increased database availability and better recovery from catastrophic failure.

Your user community will tell you how important availability is. Perhaps you have users in San Francisco, California, accessing the same database as users in Tokyo, Japan. The time difference means that the database needs to be available nearly all the time for both sets of users to have access during business hours.

Or perhaps you only have a small number of users who need access only between the hours of 9:00 AM and 4:00 PM. If this is the case, it might be fine to continue performing database maintenance at night with the database unavailable to users. Whatever your situation, you must perform the analysis to determine what is right in your environment.

Costs of the Version Log

The version log has some costs associated with it. The costs are relatively small, but are not zero. When performing your cost/benefit analysis, the most obvious cost is the disk space that the version log requires. To ensure good query performance, the version log should reside on its own storage subsystem, preferably with several dedicated disk drives. These are extra budget and system resources that you must allocate.
There is potentially some cost in the version log in the form of additional I/O. When you modify a versioned database, all of the changed blocks are first written to the version log. They are eventually re-written back to the database files, after the transaction is complete and when no other users are reading the modified parts of the database. If there are no modified blocks in a transaction (for example, all new table and index data), there is nothing written to the version log and therefore no additional I/O cost.

Additionally, consider the cost to administer the version log. Although it is not very complex, it does add to the overall complexity of your system. It also uses some resources on your system.

For more information about the version log, refer to “Understanding the Version Log” on page 2-6.

**Load Window**

The “load window” is the period of time available for database maintenance when the database does not need to be available for query operations. If you find that the load window is getting shorter, then concurrency might be appropriate for your system; it will allow you to load data while the database is available for query operations.

You can use concurrency to decrease or eliminate the period of time when the database is unavailable. Whether this makes sense in your environment depends on the answers to the following questions:

- How much of your load operation changes existing blocks in the database?
- Is the cost of creating the version log justified by the benefit?
- How important is the extra availability for your users?

**Note:** Versioning is not supported for the PTMU. If your LOAD operations use the PTMU, you will not be able to run them as versioned transactions. If you want to continue to perform parallel loads with the PTMU but find your load window shrinking, consider splitting your load operations into two or more phases. The first phase can be run in blocking mode (database not available for query operations) using the PTMU. You can do this for as long as your load window allows. The next phase or phases can be done in versioned mode (database is available for query operations). This scheme allows satisfactory maintenance and loading with a shorter load window, with the remaining operations completing while the database is available to your users.
Updating Records in a Referenced (Dimension) Table

An example where versioning can be extremely useful is when you modify a referenced, or dimension, table. This is especially useful when the dimension table is fairly large (for example, 1,000,000 rows), although it is useful for tables of all sizes. This type of operation is sometimes referred to as “dimension table cleansing.”

When you modify a dimension table in blocking mode (VERSIONING OFF), a lock is placed on the dimension table. This blocks read access to the table for the duration of the operation.

When you modify the same table with versioning enabled (VERSIONING ON), however, a consistent version of the tables involved is still available for query operations. When deciding whether to run these transactions in versioned mode, consider the amount of data they are modifying. If it is modifying a relatively small amount of data (for example, changing some customer addresses in a large customer table), then versioning will have a very small cost.

If you modify every row in the table, however, many blocks are written to the version log, which will have a higher cost. You must determine whether the benefit is worth the cost.

Incremental Loads During Production Hours

Versioning allows you to perform loads during production hours. If these loads are small enough to fit into the version log, this is a good application for versioning. You can also control the size of the version log to fit the demands of your database modifications.

New Table and Index Data into New Segments

For load operations that add new table and index data in new segments, the cost of versioning is very low. Because very few existing blocks are changed in the database, very few blocks are written to the version log. New data goes directly to the table and index PSUs in the database.

For an example of a database where new segments are used for new data and index entries, refer to Appendix D of the Warehouse Administrator’s Guide.
Increased Recoverability

When you run a transaction with versioning enabled, no existing blocks in the database are directly modified. Those blocks that need modification are modified in the version log, not in the database files. This has the benefit of providing complete recoverability to the point when the transaction began because none of the database files have been changed.

Consider, for example, a situation where you are loading data with versioning enabled. In the middle of the operation, the power fails. When the system comes back up, the database is still operational and in the same state as before the operation began. Any transactions that were in process during the power failure are aborted and the database is automatically rolled back to a consistent state. You can then start the load operation again. If this happens without versioning, it is possible for the database to be left in an inconsistent state, requiring some recovery action to make the database operational.
Understanding the Version Log

The version log stores data that has changed due to INSERT, UPDATE, DELETE, and certain TMU operations. The version log resides in a single segment that can contain up to 250 separate physical storage units (PSUs). Each PSU maps to an operating-system file.

Database Blocks

The database is divided into blocks. Each database block in a Red Brick Warehouse database uses 8K of disk space. The following figure shows a database with 5 blocks of data:

The first block in the database is named DB0, and the last block is named DB4. If a query reads the whole database, it reads the following blocks:

{(DB0), (DB1), (DB2), (DB3), (DB4)}

Suppose this database is a versioned database and therefore contains a version log. When a transaction changes block DB0 of the database and commits the change, the new version of that block is stored as revision number 1 of that block in the version log:
Now if a query reads the whole database, it reads the following blocks:

\{ (VL0,1), (DB1), (DB2), (DB3), (DB4) \}

If another transaction changes block 0 and block 2 of the database, the result is as follows:

New transactions read the latest version of the database. Because block 0 contains two revisions, the latest revision is revision number 2 in this example. Therefore if a query reads the whole database, it reads the following blocks:

\{ (VL0,2), (DB1), (VL2, 2), (DB3), (DB4) \}

**Note:** This explanation assumes the vacuum cleaner has not cleaned any blocks throughout the duration of this example. If it had, those blocks would have been moved from the version log to the database files. For more information on the vacuum cleaner, refer to “Vacuum Cleaner Daemon/Thread” on page 2-19.

**Multiple Read Operations**

It is possible for multiple read (typically query) operations to read different versions of the database. When a query begins, it reads the latest revision of the database at the time the query begins. If another query begins at a later time, that query reads the latest version of the database as of that later time. Note that the database might have changed from the time the first query began to the time the second query began.
**Example**

Consider three queries that take place one minute apart. Assume the queries are long-running and take 5 minutes to complete. After the first query begins, an UPDATE operation commits its changes to the database. After the second query begins, another UPDATE operation commits its changes. Then the third query begins. Each of these queries will access a different version of the database, as illustrated in the following figure:

In this example, query #1 reads the data only from the database files, query #2 and query #3 read some of the data from the database files and some from the version log. Each query reads the latest committed version of the database, regardless of whether it currently resides in the database files, the version log, or a combination of the two.
**Changed Blocks**

When an operation is performed that modifies the database, all modifications that change existing blocks are written to the version log; all new blocks are written directly to the database files. After the blocks are written to disk, the transaction is complete, whether they are written directly to the database files or to the version log. Blocks that are written to the version log, however, will eventually be re-written back to the database files when previous versions are no longer needed (this operation is performed by the vacuum cleaner).

Therefore, anything that is written to the version log must eventually be re-written back to the database files. This extra overhead on the system is mainly increased I/O. The version log is most efficient when the number of blocks written to the version log is relatively small. The more blocks that are written to the version log, the more blocks that eventually will be re-written to the database files.

When you are deciding whether to run a transaction with versioning enabled, consider the volume of changed blocks in your transaction. This volume is directly proportional to the extra cost of writing to the version log.

**Versioned DELETE Operations**

When performing DELETE operations on a versioned database, note that any deleted blocks are treated as changed blocks. As long as you are deleting a small number of blocks relative to the number of blocks in the a segment or a table, this is fine. This type of versioned DELETE operation allows users to query a table for the duration of the operation.

A versioned delete of a whole segment or table, however, might perform slowly because it creates a new version of each block that is deleted. Furthermore, because the blocks in the table exist as empty blocks, a subsequent load operation into a that table will create another version of each block in the version log. Deleting everything in a table is better done as a non-versioned transaction or with an ALTER SEGMENT CLEAR operation, which is a blocking-only operation and which resets the segments to have no allocated blocks.
Creating and Sizing the Version Log

In order to have a versioned database, you must first create a version log. This section includes issues with allocating space for the version log and provides an overview of the procedure for creating a version log.

Allocating the Disk Space for the Version Log

You must allocate all of the version log disk space when you create the version log in order to ensure that the space is actually available on disk. The amount of data that is written to the version log is directly related to the amount of blocks in the existing database that are changing. New blocks are not written to the version log; they are written directly to the database files. As the amount of data in the version log increases and as the query demand on the modified data increases, the more important it is to configure the version log on high performance devices in order to minimize the performance cost of the version log.

For maximum performance from the version log, Red Brick Systems recommends the version log have a dedicated storage subsystem that is separate from the rest of the database. Ideally, the storage subsystem should have several separate disk drives. Disk striping and/or RAID level 0 or level 1 devices are recommended for the version log.

The main performance cost of the version log is in extra disk I/O due to the data being first written to the version log, then written back to the database by the vacuum cleaner. The reason for placing the version log on several disks and/or on RAID devices is to allow I/O to occur from several disks at one time, thus having little or no performance impact on query operations.

Before you create the version log, you must decide how large you want it to be. The size needed is equal to the number of changed blocks in your database, plus about 20% for overhead.

When the version log runs out of space during a transaction, the transaction aborts and the system rolls back to its state before that transaction began. You can monitor how much the space the version log is using by checking the VERSION_LOG_MAXIMUM_USED column of the DST_DATABASES table.
Working with a Versioned Database
Creating and Sizing the Version Log

Adding Space to the Version Log

If you need to add space to the version log, you must perform the following steps:

1. Disable versioning by entering the following command:
   ```
   alter database stop versioning;
   ```
2. Drop the version log by entering the following command:
   ```
   alter database drop version log;
   ```

   **Note:** You must stop versioning and wait for the vacuum cleaner to complete its cleaning of the version log before you can drop the version log.
3. Add more space to the segment with ALTER SEGMENT operations.
4. Create the version log again, as in the following example:
   ```
   alter database create version log in version_log_segment;
   ```

Enabling Versioning

To enable versioning in a database, you must perform the following steps.

1. Create the segment in which the version log will reside.
2. Ensure that no users are connected to the database. If necessary, you can perform an ALTER SYSTEM QUIESCE operation and/or ALTER SYSTEM CANCEL USER SESSION operations.
3. Use an ALTER DATABASE command to specify a segment as the version log segment. For example, to specify the segment named `versionlog` as the version log, enter the following:
   ```
   alter database create version log in versionlog;
   ```

   This command allocates all of the physical space for the version log, so it might take some time to complete. All other users must be off the system before you can execute this command, and all other connections to the database are refused while the version log is initializing.
4. Issue an ALTER DATABASE START VERSIONING command to start versioning.
This procedure sets up a versioned database. If you want transactions to run as versioned transactions, you must set versioning on with the OPTION VERSIONING ON rbw.config file parameter or the SET VERSIONING ON command. Similarly, if you want TMU operations to run as versioned transactions, you must set versioning on with the OPTION TMU_VERSIONING ON rbw.config file parameter or a SET TMU_VERSIONING ON statement in your TMU control file.

Note: If these parameters are set to ON and no version log exists for the database or if the database is not versioning enabled (by the ALTER DATABASE START VERSIONING command), then any transaction that modifies the database will fail with an error.

**Dropping the Version Log**

To drop the version log for a database, perform the following steps.

1. Stop versioning on the database by entering the following command:
   
   ```
   alter database stop versioning;
   ```

   This disallows any new versioning transactions.

2. Wait until the vacuum cleaner finishes emptying the version log. The version log is empty when the value of the CURRENT_REVISION column in the DST_DATABASES table is equal to the value of the LATEST_MERGED_REVISION column, as in the following example:

   ```
   RISQL> select current_revision, latest_merged_revision
       from dst_databases;
   CURRENT_REV LATEST_MERG
   1           1
   ```

3. Ensure that no users are connected to the database. If necessary, you can perform an ALTER SYSTEM QUIESCE operation and/or ALTER SYSTEM CANCEL USER SESSION operations.

4. Drop the version log by entering the following command:

   ```
   alter database drop version log;
   ```
Locking Mechanism

Red Brick Warehouse has six types of table locks. There are three types of read locks and three types of write locks. Read locks ensure that a transaction reads a consistent view of a table. Write locks are more restrictive and allow other users various levels of read access to the table for the duration of the lock.

The different types of read and write locks allow several levels of concurrency in the database, ranging from disallowing any concurrent transactions (blocking) to allowing a write transaction on a table while read transactions are taking place on versions of the same table or a table that with a primary key/foreign key relationship. The following table provides definitions and descriptions for the six types of locks.

<table>
<thead>
<tr>
<th>Lock Type</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO</td>
<td>Read-Only</td>
<td>Read transaction that allows any other read or write transactions except a blocking transaction on the table.</td>
</tr>
<tr>
<td>RK</td>
<td>Read-Key</td>
<td>Read transaction that does not allow other transactions to change primary key values in the table.</td>
</tr>
<tr>
<td>RD</td>
<td>Read-Data</td>
<td>Read transaction that does not allow any write transactions to the table.</td>
</tr>
<tr>
<td>WD</td>
<td>Write-Data</td>
<td>Write transaction that does not modify existing primary key values in the database (for example, INSERT or UPDATE of non-key columns).</td>
</tr>
<tr>
<td>WK</td>
<td>Write-Key</td>
<td>Write transaction that modifies existing primary key values in the database (for example, DELETE or UPDATE of key columns).</td>
</tr>
<tr>
<td>WB</td>
<td>Write-Blocking</td>
<td>Write transaction that does not allow any other read or write transactions on the table.</td>
</tr>
</tbody>
</table>
The following table shows how the various types of locks interact with each other. The shaded area indicates blocking transactions for the lock combinations; the unshaded area indicates concurrent transactions.

<table>
<thead>
<tr>
<th></th>
<th>RO</th>
<th>RK</th>
<th>RD</th>
<th>WD</th>
<th>WK</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>RK</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>RD</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>WD</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>WK</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>WB</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

You can see what types of locks are on tables by querying the TYPE column of the DST_LOCKS table.
**Single Statement Transactions**

A *transaction* in a Red Brick Warehouse database is defined as a single executable statement. There is no syntax for BEGIN TRANSACTION and COMMIT, but they are implicit at the beginning and end, respectively, of every SQL statement and of every TMU operation executed. The duration of the transaction is the time it takes for the statement to fully execute.

A *versioned transaction* is a transaction on a versioned database that changes the database (for example, INSERT, UPDATE, DELETE, and LOAD operations). Versioned transactions create new versions of the blocks they modify. These new versions reside in the version log until the vacuum cleaner writes them back to the database files.

**Isolation Level**

For versioned SQL transactions, there are two user-controlled *isolation levels* available: SERIALIZABLE and REPEATABLE READ. The isolation level is the level of concurrency allowed for the duration of the transaction. Depending on the level being used, different read locks are used on the tables to allow different levels of access to those tables.

The SERIALIZABLE mode is the most restrictive; other versioned transactions cannot use (read) a table locked by another transaction for use in modifying another table. This ensures that the new versioned transaction reads the latest version of the table before using it to modify another table.

The REPEATABLE READ mode is less restrictive; versioned transactions can read an older version of a table that is locked by another transaction for use in modifying another table. This allows a new versioned transaction to proceed with an older version of the table, thus allowing the new transaction to execute sooner, but without guaranteeing that it is using the latest version of one table to modify another table.

The isolation level is controlled through the SET TRANSACTION ISOLATION LEVEL command and the OPTION TRANSACTION_ISOLATION_LEVEL *rbw.config* file parameter. For the syntax of this command, refer to page 4-11 in Chapter 4, “SQL Reference Guide.”
Example 1

To illustrate the difference between the isolation levels, consider a database with two tables A and B with no foreign key relationships, containing rows with values x and y respectively. The following figure shows the different results that occur with the two different isolation levels:

Note the results differ in SERIALIZABLE and REPEATABLE READ modes. This is because they are each accessing different versions of the database.
Example 2

Consider an application where you have a 100,000,000-row customer table. Assume that table has a versioned transaction (a TMU LOAD operation) that just started that is adding new customers whose last names begin with the letter “A.” You know that this transaction takes about 15 minutes to complete.

Your manager suddenly calls and says she needs you to add all the customers whose last name starts with the letter “S” to the “Preferred Customer” table. The “Preferred Customer” table is in the same database but is not related to the customer table with any primary key/foreign key relationships. She needs the operation completed in 10 minutes. The SQL INSERT statement for this operation is as follows:

```sql
insert into preferred_customer
    select last_name, first_name, id, address
from customer
    where last_name like 'S%'
    group by last_name, first_name, id, address
    order by last_name, first_name;
```

If you run this transaction in SERIALIZABLE mode, it must wait for the LOAD operation to complete. The reason it must wait is because it is using data from one table, which is currently being modified, to modify another table. But you know that you are not going to use any of the data that is currently being modified. You are interested in customers with “S” last names, and only customers with “A” last names are being changed.

So you run the transaction in REPEATABLE READ mode by issuing the following command before executing the INSERT statement:

```
set transaction isolation level repeatable read;
```

You then run the query and it executes without waiting.

**Note:** In this example, if the LOAD operation added new customers whose last names begin with the letter “S” instead of the letter “A,” and if you ran the INSERT operation in REPEATABLE READ mode, you might not get the results you expect. You will not see the latest changes made with this INSERT operation.
Strategies for Maintaining a Versioned Database

After you have a version log set up with all of the space allocated, very little maintenance is required. It is a good idea, however, to monitor the activity of the version log to get an idea whether you have over- or under-allocated disk space. Statistics for the version log are stored in the system tables and dynamic statistic tables (DSTs). The VERSION_LOG_USED, VERSION_LOG_AVAILABLE, and VERSION_LOG_MAXIMUM_USED columns of the DST DATABASES table are particularly useful in monitoring the activity of the version log. If you find you are filling up the version log, consider allocating more space.

If you using SQL-BackTrack for Red Brick Warehouse to back up a versioned database, SQL-BackTrack waits for the vacuum cleaner to finish cleaning the version log before performing a checkpoint backup. For information on SQL-BackTrack, refer to the SQL-BackTrack for Red Brick Warehouse User’s Guide.
 Vacuum Cleaner Daemon/Thread

The vacuum cleaner daemon (UNIX) or thread (Windows NT) moves the data committed in the version log to the database files. There is one vacuum cleaner for each versioned database. This operation is done in the background. After data is committed to the version log, the vacuum cleaner waits for any queries that are accessing previous version(s) of the data to complete, then writes the newly modified data back to the database files. Then it frees the space in the version log for use by new transactions.

In most situations, the vacuum cleaner can perform its work in the background, cleaning out the version log without adversely affecting anything else on the server. If you find it is having an impact, however, then there are manual controls you can use to start and stop the vacuum cleaner. Under normal conditions, you should not stop the vacuum cleaner because it might cause the version log to run out of space.

If you perform a blocking transaction that modifies a versioned database (for example, a LOAD operation with SET TMU VERSIONING OFF), the vacuum cleaner daemon/thread cleans the version log before the transaction executes. The blocking transaction then performs its work solely on the database files, not on the version log.

Because the vacuum cleaner waits for queries to finish reading previous versions before writing the new versions back to the database files, you might notice that the vacuum cleaner is active on your system even when no one is connected to the database.

To manually disable the vacuum cleaner, enter the following command:

```
alter database stop cleaning;
```

This allows you to control when the cleaning occurs. Because it is an I/O intensive process, it can potentially cause a performance bottleneck if there are queries that also need to perform extensive of I/O on the same devices.

**Caution:** If you manually stop the vacuum cleaner, it is possible for the version log to run out of disk space. If this happens, the transaction is aborted and the database is rolled back to its previous state; all the changes from that transaction are lost.

If you have stopped the vacuum cleaner manually and want to start it back up when the system is quiet, you can start it with the following command:

```
alter database start cleaning;
```
Trickle Feed Applications

A “trickle feed” application is one in which a relatively small volume of incremental data is added to the database during production hours. This allows periodic commits to occur during database modification. A typical trickle feed application has the requirement of having up-to-date or nearly up-to-date information. For example, if a database stores information on stock ticker activity, you can load in new data in small batches at set time intervals (every 15 minutes, for example). If you had to disallow queries during that time, the database would be unavailable during much of the time the stock market is open. Versioning allows you to build such an application.

The TMU has features that allow you to set up intervals at which a load operation commits. These intervals can be set up to periodically commit after a specified number of rows have been loaded, after a specified time period has elapsed, or both. Using these features in a versioned database allows trickle feed applications to be easily implemented. For information on using these TMU features, refer to “Setting Versioned Commit Intervals” on page 5-27.

Example—Creating a Versioned Aroma Database

The Aroma sample database is installed when you install Red Brick Warehouse. Aroma is a retail database with approximately 69,000 rows of data. The following procedure enables versioning in the Aroma database.

1. Create the Aroma database. If you have deleted Aroma, re-create it as described in Appendix A of the Warehouse Administrator’s Guide.

2. Create the segment in which the version log will reside. From the RISQL prompt, enter the following:

   RISQL> create segment versionlog
   storage 'version.log' maxsize 10000;

3. Create the version log in the new segment by entering the following command:

   RISQL> alter database create version log in versionlog;

4. Start versioning by entering the following command:

   RISQL> alter database start versioning;

5. Enable versioned transactions globally with the OPTION VERSIONING ON rbw.config file parameter or for a session with the following command:

   RISQL> set versioning on;

You can now run concurrent transactions on the Aroma database.
This chapter contains new and changed material from the Warehouse Administrator’s Guide for both UNIX and Windows NT platforms. The following topics are included:

- Warehouse Processes on UNIX Platforms
- Warehouse Service on Windows NT
- Versioned Databases in a DSS Environment
- Modifying the Configuration File on UNIX Platforms
- Modifying the Configuration File on Windows NT Platforms
- Appendix B—Configuration File
- Appendix C—System Tables and Dynamic Statistic Tables
- Fully Qualified Database Pathnames (Windows NT only)
This section replaces pages 1-6 to 1-7 in Chapter 1 of the V5.1 Warehouse Administrator’s Guide for UNIX Platforms.

The Red Brick Warehouse server is implemented by a set of cooperating processes that communicate using the UNIX System V Interprocess Communication (IPC) mechanism, shared memory, and semaphores. A single process, the warehouse daemon, starts and monitors separate server processes for each user session. This implementation is well-suited for multi-processor systems because each server process can be run on a different processor.
Interprocess Communication

The following figure illustrates the interprocess communication that occurs among the warehouse daemon, other daemons, warehouse server processes, and the Red Brick ODBC Driver. When a user uses a client tool to access a warehouse database, the client tool communicates through the Red Brick ODBC Driver with the warehouse daemon and server processes. When a user accesses the database with the RISQL Entry Tool or the RISQL Reporter, these tools communicate directly with the warehouse daemon and server processes.
Warehouse Daemon Process

The warehouse daemon process (rbwapid) manages the separate server processes and the communication between the separate server (rbwsvr) and client processes. The daemon creates and controls the number of server processes, manages exceptional conditions, and handles the termination and cleanup of the server processes. In a standard configuration, the rbwapid process is started automatically at system startup and runs continuously.

Server Processes

The Red Brick Warehouse server uses a process-per-user architecture in which a new independent process named rbwsvr is created for each user session accessing the warehouse. An rbwsvr process accepts SQL statements from clients (for example, the RISQL Entry Tool or other client tools via the Red Brick ODBC Driver), checks the statement syntax, executes the command, and returns any output to the client. Each client session is serviced by its own warehouse server process; each server process exists until the client terminates the session. Additional server processes are created as needed to perform parallel query processing.

Administration Daemon Process

The administration daemon process (rbwadmd) collects statistics for the dynamic statistic tables (DSTs) and performs the actions specified by ALTER SYSTEM commands. The rbwadmd process is started when the rbwapid process is started.

For more information about the rbwadmd process, refer to Chapter 7, “Managing Database Activity with the Enterprise Control and Coordination Option,” in the Warehouse Administrator’s Guide.

Log Daemon Process

The log daemon process (rbwlogd) writes records to the log file when various events occur in Red Brick Warehouse; this daemon is started automatically when the warehouse daemon (rbwapid) starts. The warehouse administrator can specify which events to log; if nothing is specified, the log daemon logs only a restricted set of events intended to help the Red Brick Customer Support Center diagnose problems.

For more information about the rbwlogd process, refer to Chapter 7, “Managing Database Activity with the Enterprise Control and Coordination Option,” in the Warehouse Administrator’s Guide.
Process Checker Daemon

The process checker daemon (rbwpchk) is a process that looks for abnormally-terminated connections and cleans up any shared resources that might be left around by the abnormal termination. This ensures proper cleanup of the system, even when processes are terminated outside the control of Red Brick Warehouse (for example, `kill -9`). The process checker daemon is started by the `rbwapid` daemon.

Vacuum Cleaner Daemon

The vacuum cleaner daemon (rbwvcd) is present in versioned databases. There is one rbwvcd process per database. The purpose of the vacuum cleaner daemon is to merge committed data from the version log into the database files, then to free the space in the version log. The vacuum cleaner daemon is started when you first create the version log or when the first connection is made to a versioned database.

When changes occur in a versioned database, the changed blocks are initially written to the version log. While these changes are taking place, the existing blocks are available for query (read) operations. After the new blocks are committed to the version log but before the new blocks are moved back into the database files, query operations access the new blocks in the version log.

The vacuum cleaner daemon waits until no users are reading the old version of the database, then proceeds to “vacuum” the changed blocks from the version log back into the database files. After the data have been moved from the version log, the vacuum cleaner then removes those blocks from the version log, freeing space in the version log for more versioning transactions.

For more information on the version log, refer to Chapter 2, “Working with a Versioned Database.”

Shared Memory

All databases access global shared memory. For versioned databases, a portion of shared memory is also allocated for each database.
Warehouse Service on Windows NT

This section replaces pages 1-6 to 1-7 in Chapter 1 of the V5.1 Warehouse Administrator’s Guide for Windows NT Platforms.

Red Brick Warehouse is implemented on Windows NT by a single multithreaded process which runs as a Windows NT service. The individual threads communicate using shared memory. A single thread, the Red Brick API (rbwapid) thread, starts and monitors separate server threads for each user session. The Table Management Utility (TMU) runs as a separate process that communicates with the Red Brick Warehouse service.

Note: On Windows NT, a thread is a portion of a process that performs a specific function. A process can consist of many threads. A Windows NT service is a multithreaded process containing one or more threads that are always running and can start and stop other threads.

Designed for Microsoft BackOffice

Red Brick Warehouse for Windows NT is designed with Systems Management Server support for Microsoft BackOffice. To conform to Microsoft BackOffice requirements, Red Brick Warehouse for Windows NT includes the following features:

• Runs as a Windows NT service
• Supports the WinSock 2.0 network protocol stack
• Supports an unattended installation using Systems Management Server

Communication Between Threads

The following figure illustrates the communication that occurs among threads in the Red Brick Warehouse Service. When a user accesses a warehouse database with RISQL Entry Tool, RISQL Reporter, or another client tool, the tool communicates through Red Brick ODBC Driver with the API and server threads.
**Warehouse API Thread**

The Red Brick Warehouse API thread (rbwapid) manages the separate server (rbwsvr) threads and the communication between the separate server threads and their associated clients. The rbwapid thread creates and controls the number of server threads, manages exceptional conditions, and handles the termination and cleanup of the server threads.

In a standard configuration, the Red Brick Warehouse Service is started at system startup and the rbwapid thread runs continuously.

**Red Brick Warehouse Server Threads**

On the Windows NT operating system, the Red Brick Warehouse server functions are performed by threads, one per user session, that are part of the Red Brick Warehouse Service. This multithreaded architecture includes a separate thread named rbwsvr that is created for each user session accessing the warehouse. An rbwsvr thread accepts SQL or RISQL statements from a client tool via the Red Brick ODBC Driver, checks the statement syntax, executes the command, and returns any output to the client application via the Red Brick ODBC Driver. Each client session, RISQL Entry Tool session, and RISQL Reporter session is serviced by its own warehouse server thread; each server thread exists until the client terminates the session.

**Note:** The names of the threads (rbwsvr, rbwapid, etc.) are for reference and they correspond to the respective UNIX processes that are visible on UNIX versions of Red Brick Warehouse. To view the active threads on Red Brick Warehouse for Windows NT, use the rbshow.exe program shipped with Red Brick Warehouse.

**Administration Thread**

The administration thread (rbwadmd) collects statistics for the dynamic statistic tables (DSTs) and performs the actions specified by ALTER SYSTEM commands. This thread is part of the Enterprise Control and Coordination feature. The rbwadmd thread is started when the rbwapid thread is started.

For more information about the rbwadmd thread, refer to Chapter 7, “Managing Database Activity with the Enterprise Control and Coordination Option,” in the Warehouse Administrator’s Guide.
**Log Thread**

The log thread (rbwlogd) writes records to the log file when various events occur in Red Brick Warehouse; this thread is started automatically when the rbwapid thread starts. The warehouse administrator can specify which events to log; if nothing is specified, the log thread logs only a restricted set of events intended to help the Red Brick Customer Support Center diagnose problems.

For more information about the rbwlogd thread, refer to Chapter 7, “Managing Database Activity with the Enterprise Control and Coordination Option,” in the *Warehouse Administrator’s Guide*.

**Red Brick Warehouse Listener Thread**

The Red Brick Warehouse listener thread (rbwlsnr) is a thread in the Red Brick Warehouse Service that listens for connections coming in to Red Brick Warehouse. The listener thread reads the rbw.config file and determines the port on which to listen and how many simultaneous connections to allow (based on the value of the MAX_SERVERS parameter).

Starting the warehouse service automatically starts the Red Brick Warehouse listener thread.

**Control-C Coordination Thread**

The control-c coordination thread (rbwconc) listens for control-c interrupts from the various clients connected to Red Brick Warehouse. This thread is started by the listener thread, and there is one control-c coordination thread per 60 concurrent connections.

When a control-c or cancel operation is detected from a client application, the rbwconc thread ensures that the rbwsvr thread that requested the cancel operation terminates the operation in a clean and consistent manner.

**Process Checker Thread**

The process checker thread (rbwpchk) is a thread in the Red Brick Warehouse service that looks for abnormally terminated connections and cleans up any shared resources that might be left around by the abnormal termination. This ensures proper cleanup of the system, even when processes are terminated outside of the control of Red Brick Warehouse (for example, a thread exit or an End Task request from the Task Manager). The process checker thread is started by the rbwapid thread.
**Vacuum Cleaner Thread**

The vacuum cleaner thread (rbwvcd) is present in versioned databases. There is one rbwvcd thread per database. The purpose of the vacuum cleaner thread is to merge committed data from the version log into the database files, then to free the space in the version log. The vacuum cleaner thread is started when you first create the version log or when the first connection is made to a versioned database.

When changes to the database occur in a versioned database, the changed blocks are initially written to the version log. While these changes are taking place, the existing committed blocks are available for query (read) operations. After the new blocks are committed to the version log but before the new blocks are moved back into the database files, query operations access the new blocks in the version log.

The vacuum cleaner thread waits until no users are reading the old version of the database, then proceeds to “vacuum” the changed blocks from the version log back into the database files. After the data have been moved from the version log, the vacuum cleaner then removes those blocks from the version log, freeing space in the version log for more versioning transactions.

For more information on the version log, refer to Chapter 2, “Working with a Versioned Database.”

**Shared Memory**

All databases access global shared memory. For versioned databases, a portion of shared memory is also allocated for each database.
Versioned Databases in a DSS Environment

This section is supplemental to Chapter 2, “Key Concepts,” of the Warehouse Administrator’s Guide.

Decision support systems (DSS) are primarily used to perform analysis on large amounts of data. Business questions are answered by performing complex queries against a large, comprehensive database. DSS queries often access very large numbers of rows in many tables. These complex queries are where the real value of a DSS database is realized.

Online transaction processing (OLTP) databases, in contrast to DSS databases, are primarily used to keep track of records, not to analyze the records. OLTP transactions typically select a small number of specific records for query and modification. Consequently, update performance is the critical factor in OLTP systems. To support this environment, very complex concurrency models are typical features in OLTP systems.

Because querying is where the value of a DSS database is realized, the concurrency model in a DSS environment must favor query performance over update performance. That is the basis of query-priority concurrency in Red Brick Warehouse.

Typically, a DSS database is unavailable for query operations while changes to the database occur. Historically, this has not been a problem because the DSS databases typically had small user communities and the downtime could be easily planned. As these systems have evolved, however, user communities of DSS databases have grown, thus raising the demand for more and more availability. Additionally, the value added from the information gained through querying these databases has made these systems much more business-critical.

Versioning is a mechanism where readers, typically users querying the database, can access the database while writers (for example, INSERT, UPDATE, DELETE, REORG, and LOAD operations) create a new version of the database with little or no impact on the readers on the system.

Red Brick Warehouse uses a version log to temporarily store the new versions of the database. The version log contains only changes to existing parts of the database; new data is added directly to the database files.

For more information on setting up and using versioned databases, refer to Chapter 2, “Working with a Versioned Database.”
Modifying the Configuration File on UNIX Platforms

This section replaces pages 8-29 to 8-30 in Chapter 8 of the V5.1 Warehouse Administrator’s Guide for UNIX Platforms.

The configuration file, rbw.config, is created when the warehouse software is installed, using information provided during the installation procedure. This information is used by the warehouse server and by the TMU.

You, as the redbrick user, can edit the rbw.config file (using any standard text editor) as conditions at your site change. Because changes you make to this file have various effects on the processes that use it, refer to the following table to determine what other actions you must take.

<table>
<thead>
<tr>
<th>If you change:</th>
<th>Action required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHMEM or MAPFILE SERVER</td>
<td>Stop and restart warehouse daemon.</td>
</tr>
<tr>
<td>MAX_SERVERS, MAX_ACTIVE_DATABASES</td>
<td>1. Stop warehouse daemon.</td>
</tr>
<tr>
<td>PROCESS_CHECKING_INTERVAL</td>
<td>2. Use ipcrm to remove prior memory segment or reboot (this step is only necessary after severe errors occur).</td>
</tr>
<tr>
<td></td>
<td>3. Restart warehouse daemon.</td>
</tr>
<tr>
<td>CLEANUP_SCRIPT</td>
<td>Stop and restart warehouse daemon.</td>
</tr>
<tr>
<td>LOGFILE_SIZE</td>
<td>Do not change.</td>
</tr>
<tr>
<td>QUERYPROCS</td>
<td></td>
</tr>
<tr>
<td>TOTALQUERYPROCS</td>
<td></td>
</tr>
<tr>
<td>ADMINADVISOR_LOGGING</td>
<td></td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td></td>
</tr>
<tr>
<td>MESSAGE_DIR</td>
<td></td>
</tr>
<tr>
<td>LOCALE</td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td>Stop and restart server-monitoring daemon (rbw.servermon).</td>
</tr>
<tr>
<td>License keys</td>
<td>No action needed. Current TMU and server processes do not recognize, but new processes will.</td>
</tr>
<tr>
<td>If you change:</td>
<td>Action required:</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FILE_GROUP</td>
<td>No action needed. Current server processes do not recognize, but new processes will.</td>
</tr>
<tr>
<td>ROWS_PER_SCAN_TASK</td>
<td></td>
</tr>
<tr>
<td>ROWS_PER_FETCH_TASK</td>
<td></td>
</tr>
<tr>
<td>ROWS_PER_JOIN_TASK</td>
<td></td>
</tr>
<tr>
<td>FORCE_SCAN_TASK</td>
<td></td>
</tr>
<tr>
<td>FORCE_FETCH_TASK</td>
<td></td>
</tr>
<tr>
<td>FORCE_JOIN_TASK</td>
<td></td>
</tr>
<tr>
<td>FORCE_HASHJOIN_TASK</td>
<td></td>
</tr>
<tr>
<td>FORCE_AGGREGATION_TASK</td>
<td></td>
</tr>
<tr>
<td>PARTITIONED_PARALLEL_AGGREGATION</td>
<td></td>
</tr>
<tr>
<td>CROSS_JOIN</td>
<td></td>
</tr>
<tr>
<td>ARITHABORT</td>
<td></td>
</tr>
<tr>
<td>ALLOW_POSSIBLE_DEADLOCKS</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_DATA_SEGMENT</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_INDEX_SEGMENT</td>
<td></td>
</tr>
<tr>
<td>TEMPORARY_DATA_SEGMENT</td>
<td></td>
</tr>
<tr>
<td>TEMPORARY_INDEX_SEGMENT</td>
<td></td>
</tr>
<tr>
<td>OPTION ADVISOR_LOGGING</td>
<td></td>
</tr>
<tr>
<td>PRECOMPUTED_VIEW parameters segments</td>
<td></td>
</tr>
<tr>
<td>IGNORE_PARTIAL_INDEXES</td>
<td></td>
</tr>
<tr>
<td>PARTIAL_AVAILABILITY</td>
<td></td>
</tr>
<tr>
<td>QUERY_TEMPSPACE parameters</td>
<td></td>
</tr>
<tr>
<td>RESULT_BUFFER</td>
<td></td>
</tr>
<tr>
<td>RESULT_BUFFER_FULL_ACTION</td>
<td></td>
</tr>
<tr>
<td>VERSIONING</td>
<td></td>
</tr>
<tr>
<td>TRANSACTION_ISOLATION_LEVEL</td>
<td></td>
</tr>
</tbody>
</table>
### If you change:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMU_BUFFERS</td>
<td>No action needed. Current TMU processes do not recognize, but new processes will.</td>
</tr>
<tr>
<td>TMU_OPTIMIZE</td>
<td></td>
</tr>
<tr>
<td>TMU_CONVERSION_TASKS</td>
<td></td>
</tr>
<tr>
<td>TMU_INDEX_TASKS</td>
<td></td>
</tr>
<tr>
<td>TMU_SERIAL_MODE</td>
<td></td>
</tr>
<tr>
<td>AUTOROWGEN</td>
<td></td>
</tr>
<tr>
<td>TMU_VERSIONING</td>
<td></td>
</tr>
<tr>
<td>TMU_COMMIT_RECORD_INTERVAL</td>
<td></td>
</tr>
<tr>
<td>TMU_COMMIT_TIME_INTERVAL</td>
<td></td>
</tr>
<tr>
<td>FILLFACTOR parameters</td>
<td>No action needed. Current server and TMU processes do not recognize, but new processes will.</td>
</tr>
<tr>
<td>INDEX_TEMPSPACE parameters</td>
<td></td>
</tr>
<tr>
<td>PASSWORD parameters</td>
<td></td>
</tr>
</tbody>
</table>

### Note:

In the `rbw.config` file, those parameters preceded by “TUNE” affect performance; the parameters preceded by “OPTION” affect behavior.

For a description and example of the `rbw.config` file and a table that lists which parameters can also be set with SQL or TMU SET commands, refer to Appendix B of the *Warehouse Administrator’s Guide*.

The RBW_OPTIONS system table lists current values for all parameters that you can change; it is updated whenever a SET command changes a value during a session. The values displayed are the values that apply to the current session.
Modifying the Configuration File on Windows NT Platforms

This section replaces pages 8-34 to 8-35 in Chapter 8 of the V5.1 Warehouse Administrator's Guide for Windows NT Platforms.

The configuration file, rbw.config, is created when the warehouse software is installed, using information provided during the installation procedure. This information is used by the warehouse server and by the TMU.

You, as the redbrick user, can edit the rbw.config file (using any standard text editor) as conditions at your site change. Because changes you make to this file have various effects on the processes that use it, refer to the following table to determine what other actions you must take.

<table>
<thead>
<tr>
<th>If you change:</th>
<th>Action required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER</td>
<td>From the Control Panel, stop the warehouse service and then restart it.</td>
</tr>
<tr>
<td>MAX_SERVERS</td>
<td></td>
</tr>
<tr>
<td>CLEANUP_SCRIPT</td>
<td></td>
</tr>
<tr>
<td>LOGFILE_SIZEADMIN</td>
<td></td>
</tr>
<tr>
<td>ADVISOR_LOGGING</td>
<td></td>
</tr>
<tr>
<td>MAX_ACTIVE_DATABASES</td>
<td></td>
</tr>
<tr>
<td>PROCESS_CHECKING_INTERVAL</td>
<td></td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td></td>
</tr>
<tr>
<td>MESSAGE_DIR</td>
<td>Set during installation; do not change.</td>
</tr>
<tr>
<td>LOCALE</td>
<td></td>
</tr>
<tr>
<td>License keys</td>
<td>No action needed. Current warehouse threads and TMU processes do not recognize, but new ones will.</td>
</tr>
</tbody>
</table>

License keys
### If you change:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARITHABORT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>ALLOW_POSSIBLE_DEADLOCKS</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>DEFAULT_DATA_SEGMENT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>DEFAULT_INDEX_SEGMENT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TEMPORARY_DATA_SEGMENT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TEMPORARY_INDEX_SEGMENT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>OPTION ADVISOR_LOGGING</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>PRECOMPUTED_VIEW</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>SEGMENTS</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>IGNORE_PARTIAL_INDEXES</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>PARTIAL_AVAILABILITY</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>QUERY_MEMORY_LIMIT</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>QUERY_TEMPSPACE</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>RESULT_BUFFER</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>RESULT_BUFFER_FULL_ACTION</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>VERSIONING</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TRANSACTION_ISOLATION_LEVEL</td>
<td>No action needed. Current server threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TMU_BUFFERS</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TMU_OPTIMIZE</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>AUTOROWGEN</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TMU_VERSIONING</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TMU_COMMIT_RECORD_INTERVAL</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>TMU_COMMIT_TIME_INTERVAL</td>
<td>No action needed. Current TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>FILLFACTOR parameters</td>
<td>No action needed. Current server threads and TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>INDEX_TEMPSPACE parameters</td>
<td>No action needed. Current server threads and TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>PASSWORD parameters</td>
<td>No action needed. Current server threads and TMU processes do not recognize, but new ones will.</td>
</tr>
<tr>
<td>ADMIN parameters</td>
<td>No action needed; current threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>REPORT_INTERVAL</td>
<td>No action needed; current threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td>Other ADMIN parameters</td>
<td>No action needed; current threads do not recognize, but new ones will.</td>
</tr>
<tr>
<td></td>
<td>From the Control Panel, stop the warehouse service and then restart it or use the ALTER SYSTEM command to restart logging and admin threads.</td>
</tr>
</tbody>
</table>

### Note:
In the `rbw.config` file, those parameters preceded by “TUNE” affect performance; the parameters preceded by “OPTION” affect behavior.
For a description and example of the rbw.config file and a table that lists which parameters can also be set with SQL or TMU SET commands, refer to Appendix B, “Configuration File.”

The RBW_OPTIONS system table lists current values for all parameters that you can change; it is updated whenever a SET command changes a value during a session. The values displayed are the values that apply to the current session.
Appendix B—Configuration File

The following new parameters supplement the parameters documented in Appendix B of the Warehouse Administrator’s Guide.

**RBWAPI MAX_ACTIVE_DATABASES**
Specifies the maximum number of databases allowed per rbwapid daemon.
Default: 30

**RBWAPI PROCESS_CHECKING_INTERVAL**
Specifies the interval, in seconds, after which the process checker daemon (rbwpck) checks for abnormally terminated processes.
Default: 5

**OPTION TMU_VERSIONING**
Specifies whether TMU operations are executed as versioned or blocking. When set to ON, all TMU operations are run as versioned transactions. Does not apply to PTMU operations.
Default: OFF

**OPTION TMU_COMMIT_RECORD_INTERVAL**
Specifies the number (a positive integer) of records to load into a table before each commit operation. Only effective on versioned transactions.
Default: OFF

**OPTION TMU_COMMIT_TIME_INTERVAL**
Specifies the amount of time (in minutes) for a load operation to proceed before each commit operation. Only effective on versioned transactions.
Default: OFF

**OPTION VERSIONING**
Specifies whether INSERT, UPDATE, and DELETE SQL transactions are executed as versioned or blocking. When set to ON, transactions are run as versioned transactions.
Default: OFF
**OPTION TRANSACTION_ISOLATION_LEVEL**

Specifies how restrictive a DML transaction is with respect to the type of lock put on the table(s) being changed. A value of SERIALIZABLE is the most restrictive and a value of REPEATABLE_READ is the least restrictive. Has an effect only when versioning is enabled.

Default: SERIALIZABLE

**TUNE INDEX_TEMPSPACE_DUPLICATESPILLPERCENT**

Specifies the percentage of the INDEX TEMPSPACE MAXSPILLSIZE used for the temporary storage of discarded duplicate rows during a REORG operation. The percent is an integer between 0 and 100 (inclusive).

Default: 2
Appendix C—System Tables and Dynamic Statistic Tables

This section describes the system tables and the dynamic statistic tables (DSTs) that have changed for this release of Red Brick Warehouse. The following tables have changed:

- RBW_LOADINFO Table
- RBW_SEGMENTS Table
- RBW_STORAGE Table
- RBW_USERAUTH Table
- DST_COMMANDS Table
- DST_DATABASES Table
- DST_LOCKS Table
- DST_SESSIONS Table

RBW_LOADINFO Table

The RBW_LOADINFO table describes data loads into tables and offline segments. This table is updated by LOAD DATA statements and contains one row for each load operation. Only the most recent 256 rows are retained; older rows are deleted automatically. This table contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNAME</td>
<td>CHAR(128)</td>
<td>Name of table.</td>
</tr>
<tr>
<td>SEGNAME</td>
<td>CHAR(128)</td>
<td>Name of segment if load performed into an offline segment; NULL if not an offline load.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>CHAR(128)</td>
<td>Name of user that performed load.</td>
</tr>
<tr>
<td>STARTED</td>
<td>TIMESTAMP</td>
<td>Date and time load started.</td>
</tr>
<tr>
<td>FINISHED</td>
<td>TIMESTAMP</td>
<td>Date and time load completed.</td>
</tr>
<tr>
<td>MODE</td>
<td>CHAR(20)</td>
<td>Mode used to insert or modify rows (INSERT, REPLACE, APPEND, MODIFY, MODIFY AGGREGATE, UPDATE, UPDATE AGGREGATE).</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(128)</td>
<td>NULL or ABORTED.</td>
</tr>
<tr>
<td>INSERTED</td>
<td>INTEGER</td>
<td>Number of rows inserted into table.</td>
</tr>
</tbody>
</table>
### Column Name | Column Type | Column Description
--- | --- | ---
UPDATED | INTEGER | Number of existing rows in the table that were updated.
SKIPPED | INTEGER | Number of rows in the input file that were skipped, as specified by a START RECORD clause.
DISCARDED | INTEGER | Number of rows in the input file that were rejected and discarded.
AUTOROWGEN | CHAR(1) | Whether any rows were automatically generated (Y or N).
COMMENT | CHAR(256) | User-specified comment or descriptive data; NULL if not specified in the LOAD DATA statement.
TRANSACTION_TYPE | CHAR (32) | Transaction type for the command. Possible values: READ_ONLY, VERSIONING, BLOCKING.
READ_REVISION | DECIMAL (10,0) | The revision number of the database being accessed by the transaction.
NEW_REVISION | DECIMAL (10,0) | The revision number of the database the transaction creates. NULL for aborted or uncommitted transactions.
**RBW_SEGMENTS Table**

The RBW_SEGMENTS table describes all segments in the system. When a member of the DBA or RESOURCE system role or a user with ACCESS_ANY authorization displays information in the RBW_SEGMENTS table, the user sees all segments. When a member of the CONNECT system role displays this table, the user sees no segments. This table is updated by ALTER SEGMENT, CREATE SEGMENT, DROP SEGMENT, CREATE TABLE, DROP TABLE, CREATE INDEX, DROP INDEX, BACKUP, RESTORE, and LOAD DATA commands. It contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CHAR(128)</td>
<td>Name of segment.</td>
</tr>
<tr>
<td>TNAME</td>
<td>CHAR(128)</td>
<td>Name of table that uses segment for row data or indexes; set to NULL for unattached segments.</td>
</tr>
<tr>
<td>CREATOR</td>
<td>CHAR(128)</td>
<td>Creator of segment.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td>Date and time of segment creation.</td>
</tr>
<tr>
<td>NPSUS</td>
<td>INTEGER</td>
<td>Number of physical storage units (PSUs) used for segment.</td>
</tr>
<tr>
<td>NCOLS</td>
<td>INTEGER</td>
<td>Number of columns used to segment the data or index.</td>
</tr>
<tr>
<td>MINKEY</td>
<td>CHAR(256)</td>
<td>Minimum key value in the segment; displays first 256 characters.</td>
</tr>
<tr>
<td>MAXKEY</td>
<td>CHAR(256)</td>
<td>Maximum key value in the segment; displays first 256 characters.</td>
</tr>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>Segment ID.</td>
</tr>
<tr>
<td>TOTALFREE</td>
<td>INTEGER</td>
<td>Kilobytes of unused space in segment.</td>
</tr>
<tr>
<td>INAME</td>
<td>CHAR(128)</td>
<td>Name of index that uses segment; set to NULL for unattached and row data segments.</td>
</tr>
<tr>
<td>ONLINE</td>
<td>CHAR(1)</td>
<td>Whether a segment is online (Y or N). For System segment (NULL).</td>
</tr>
<tr>
<td>OPTICAL</td>
<td>CHAR(1)</td>
<td>Whether a segment contains one or more optical PSUs (Y or N). (NULL for system segment.)</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Column Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INTACT</td>
<td>CHAR(1)</td>
<td>Flag indicating whether segment is intact (Y), or there is detected, unrepaired damage (N). For System segment (NULL).</td>
</tr>
<tr>
<td>INSYNCH</td>
<td>CHAR(1)</td>
<td>Whether the row contents are synchronized with the indexes of table: for offline segments: Y or N; for online segments: Y; for index segments, unattached segments, and System segment: NULL.</td>
</tr>
<tr>
<td>BACKUPELEVEL</td>
<td>INTEGER</td>
<td>Current segment backup level; all blocks within all files of this segment are backed up to at least this level (TMU Backup option only).</td>
</tr>
<tr>
<td>LAST_OFFLINE</td>
<td>TIMESTAMP</td>
<td>Date and time the segment last set offline; initially contains segment creation time.</td>
</tr>
<tr>
<td>LAST_ONLINE</td>
<td>TIMESTAMP</td>
<td>Date and time segment last set online; initially contains segment creation time.</td>
</tr>
<tr>
<td>LAST_LOAD</td>
<td>TIMESTAMP</td>
<td>Completion time of the last offline load into segment; initially set to NULL.</td>
</tr>
<tr>
<td>LAST_BACKUP</td>
<td>TIMESTAMP</td>
<td>Date and time segment was last backed up; initially set to NULL (TMU Backup option only).</td>
</tr>
<tr>
<td>LAST_RESTORE</td>
<td>TIMESTAMP</td>
<td>Date and time of most recent restore of segment from backup; initially set to NULL (TMU Backup option only).</td>
</tr>
<tr>
<td>COMMENT</td>
<td>CHAR(256)</td>
<td>User-specified comment or descriptive data; NULL if not set with the ALTER SEGMENT command.</td>
</tr>
</tbody>
</table>
### Column Name | Column Type | Column Description
--- | --- | ---
LOCAL_ID | SMALLINT | The segment ID that is returned in the RBW_SEGID pseudo-column.
USAGE | CHAR (32) | The current function that a segment is performing. Possible values: UNUSED, TABLE, INDEX, LOAD_WORK, LOAD_INDEX, BACKUP_DATA, or VERSION_LOG.
**RBW_STORAGE Table**

The RBW_STORAGE table describes the physical storage units (PSUs) in the system. When a member of the DBA or RESOURCE system role or a user with ACCESS_ANY authorization displays information in the RBW_STORAGE table, the user sees all PSUs. When a member of the CONNECT system role displays this table, the user sees no PSUs. This table is updated by CREATE SEGMENT, DROP SEGMENT, CREATE TABLE, DROP TABLE, CREATE INDEX, DROP INDEX, and BACKUP commands and contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGNAME</td>
<td>CHAR(128)</td>
<td>Segment name containing PSU.</td>
</tr>
<tr>
<td>SEGID</td>
<td>SMALLINT</td>
<td>Segment ID containing PSU.</td>
</tr>
<tr>
<td>PSEQ</td>
<td>INTEGER</td>
<td>Sequence number of PSU in segment.</td>
</tr>
<tr>
<td>LOCATION</td>
<td>CHAR(1024)</td>
<td>The location of PSU.</td>
</tr>
<tr>
<td>MAXSIZE</td>
<td>INTEGER</td>
<td>Kilobytes of maximum allowed size of PSU.</td>
</tr>
<tr>
<td>INITSIZE</td>
<td>INTEGER</td>
<td>Kilobytes of initial allocated size of PSU.</td>
</tr>
<tr>
<td>EXTENDSIZE</td>
<td>INTEGER</td>
<td>Kilobytes of increment size to use when extending PSU.</td>
</tr>
<tr>
<td>USED</td>
<td>INTEGER</td>
<td>Kilobytes of current area in use in PSU.</td>
</tr>
<tr>
<td>INTACT</td>
<td>CHAR(1)</td>
<td>Flag indicating whether PSU is intact (Y), or there is detected, unrepaired damage (N). For System segment (NULL).</td>
</tr>
<tr>
<td>BACKUPLEVEL</td>
<td>SMALLINT</td>
<td>Backup level of PSU; all blocks within this PSU backed up to at least this level (TMU Backup option only).</td>
</tr>
<tr>
<td>PHYSICAL_LOCATION</td>
<td>CHAR (1024)</td>
<td>Actual location of the PSU</td>
</tr>
</tbody>
</table>
**RBW_USERAUTH Table**

The RBW_USERAUTH table describes access rights granted to users. When a member of the DBA system role or a user with ACCESS_ANY authorization displays information in the RBW_USERAUTH table, the user sees all users’ authorizations. When a member of the RESOURCE or CONNECT system role displays this table, the user sees only the user’s own authorization. This table is updated by GRANT and REVOKE commands and contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANTEE</td>
<td>CHAR(128)</td>
<td>User or role(^1) granted authorization.</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>CHAR(128)</td>
<td>User granting authorization to GRANTEE.</td>
</tr>
<tr>
<td>DBAAUTH</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE is a member of DBA system role (Y, N, R, or I)(^2).</td>
</tr>
<tr>
<td>RESAUTH</td>
<td>CHAR(1)</td>
<td>Whether grantee is a member of RESOURCE system role (Y, N, R, or I)(^2).</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td>Date and time authorization granted.</td>
</tr>
<tr>
<td>LOCKED(^1)</td>
<td>CHAR(1)</td>
<td>Whether or not GRANTEE’s account is locked due to failed connection attempts (Y or N); NULL for roles.</td>
</tr>
<tr>
<td>EXPIRED(^1)</td>
<td>CHAR(1)</td>
<td>Whether or not GRANTEE’s account is expired because of failure to change password before the defined expiration date (Y or N); NULL for roles.</td>
</tr>
<tr>
<td>PASSWORD_TS</td>
<td>TIMESTAMP</td>
<td>Date and time GRANTEE’s database password was added or last changed.</td>
</tr>
<tr>
<td>USER_MANAGEMENT</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can alter, create and drop database users and change passwords (Y, N, R, or I)(^2).</td>
</tr>
</tbody>
</table>
### System Tables and Dynamic Statistic Tables

#### Column Name | Column Type | Column Description
--- | --- | ---
GRANT_TABLE | CHAR(1) | Whether GRANTEE can grant object privileges to database users and to roles (Y, N, R, or I).
ROLE_MANAGEMENT | CHAR(1) | Whether GRANTEE can alter, create, drop, grant, and revoke roles (Y, N, R, or I).
ALTER_ANY | CHAR(1) | Whether GRANTEE can alter indexes, segments, tables, macros, views, and synonyms (Y, N, R, or I).
PUBLIC_MACROS | CHAR(1) | Whether GRANTEE can create and drop PUBLIC macros (Y, N, R, or I).
ACCESS_ANY | CHAR(1) | Whether GRANTEE can select data from all database objects, including private user information in the system tables (Y, N, R, or I).
MODIFY_ANY | CHAR(1) | Whether GRANTEE can insert, update, delete, and load any data (Y, N, R, or I).
DROP_ANY | CHAR(1) | Whether GRANTEE can drop objects created by any user (Y, N, R, or I).
CREATE_ANY | CHAR(1) | Whether GRANTEE can create any object, including those that use another’s resources (Y, N, R, or I).
LOCK_DATABASE | CHAR(1) | Whether GRANTEE can lock the database (Y, N, R, or I).
BACKUP_DATABASE | CHAR(1) | Whether GRANTEE can backup the database (Y, N, R, or I).
RESTORE_DATABASE | CHAR(1) | Whether GRANTEE can restore the database (Y, N, R, or I).
UPGRADE_DATABASE | CHAR(1) | Whether GRANTEE can upgrade the database (Y, N, R, or I).
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REORG_ANY</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can re-organize any table or index (Y, N, R, or I)².</td>
</tr>
<tr>
<td>OFFLINE_LOAD</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can use any segment as a working segment for offline loads or synchronize segments after offline loads (Y, N, R, or I)².</td>
</tr>
<tr>
<td>ALTER_SYSTEM</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can issue the ALTER SYSTEM command to perform database administration tasks (Y, N, R, or I)².</td>
</tr>
<tr>
<td>ACCESS_SYSINFO</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can query the dynamic statistic tables for statistics about database activity (Y, N, R, or I)².</td>
</tr>
<tr>
<td>ALTER_TABLE_INTO_ANY</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can alter own tables into other users' segments (Y, N, R, or I)².</td>
</tr>
<tr>
<td>CREATE_OWN</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can create own objects: indexes, private macros, segments, synonyms, tables, and views (Y, N, R, or I)².</td>
</tr>
<tr>
<td>DROP_OWN</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can drop own objects (Y, N, R, or I)².</td>
</tr>
<tr>
<td>ALTER_OWN</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can alter own indexes, macros, segments, synonyms, tables and views (Y, N, R, or I)².</td>
</tr>
<tr>
<td>GRANT_OWN</td>
<td>CHAR(1)</td>
<td>Whether GRANTEE can grant object privileges on own objects to other users (Y, N, R, or I)².</td>
</tr>
<tr>
<td>IGNORE_QUIESCE</td>
<td>CHAR(1)</td>
<td>Flag indicating whether a user can access a quiesced database (Y) or is locked out (N).</td>
</tr>
</tbody>
</table>
Task authorizations can be granted to user-created roles.  

Y—User has the task authorization.  
N—User does not have the task authorization.  
R—User has the task authorization directly through a role.  
I—User has the task authorization indirectly through a role.

**DST_COMMANDS Table**

The DST_COMMANDS table contains information about each command issued against the database by a currently connected session. This information consists of cumulative statistics for each command. Any subprocesses that the command spawns are included in the statistics calculations.

The following table lists and describes the DST_COMMANDS columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNAME</td>
<td>CHAR(128)</td>
<td>Logical database name.</td>
</tr>
<tr>
<td>UNAME</td>
<td>CHAR(128)</td>
<td>User name for the user issuing the command.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>CHAR(128)</td>
<td>Name of node on which command is running (for MPP servers).</td>
</tr>
</tbody>
</table>
### System Tables and Dynamic Statistic Tables

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>INTEGER</td>
<td>Process ID for session running the command.</td>
</tr>
<tr>
<td>STARTED</td>
<td>TIMESTAMP</td>
<td>Start time of the command.</td>
</tr>
<tr>
<td>STATE</td>
<td>CHAR(64)</td>
<td>Connecting, Idle, Executing, Returned x rows; computed y rows (for query),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inserted x rows (for Insert), Deleted x rows (for Delete), Updated x rows (for Update)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>CHAR(1024)</td>
<td>Starting text of current command prior to macro expansion.</td>
</tr>
<tr>
<td>CACHE_READS</td>
<td>INTEGER</td>
<td>Number of times that a block was found in local buffer cache (avoiding a logical read request).</td>
</tr>
<tr>
<td>CACHE_WRITES</td>
<td>INTEGER</td>
<td>Number of times that a block was found in local buffer cache (avoiding a logical write request).</td>
</tr>
<tr>
<td>LOGICAL_READS</td>
<td>INTEGER</td>
<td>Number of logical reads performed by the command.</td>
</tr>
<tr>
<td>LOGICAL_WRITES</td>
<td>INTEGER</td>
<td>Number of logical writes performed by the command.</td>
</tr>
<tr>
<td>PHYSICAL_READS</td>
<td>INTEGER</td>
<td>Number of physical reads performed by the command (NULL for platforms that do not support this statistic).</td>
</tr>
<tr>
<td>PHYSICAL_WRITES</td>
<td>INTEGER</td>
<td>Number of physical writes performed by the command (NULL for platforms that do not support this statistic).</td>
</tr>
<tr>
<td>SYSTEM_CPUTIME</td>
<td>DEC(9,2)</td>
<td>System CPU time used by the command (in seconds).</td>
</tr>
<tr>
<td>USER_CPUTIME</td>
<td>DEC(9,2)</td>
<td>User CPU time used by the command (in seconds).</td>
</tr>
</tbody>
</table>
### System Tables and Dynamic Statistic Tables

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPILL_COUNT</td>
<td>INTEGER</td>
<td>The number of spill files used in an operation</td>
</tr>
<tr>
<td>MEMORY_USED</td>
<td>INTEGER</td>
<td>Amount of memory being used by the command (in kilobytes).</td>
</tr>
<tr>
<td>TEMPSPACE_USED</td>
<td>INTEGER</td>
<td>Amount of spill space used by the command (in kilobytes).</td>
</tr>
<tr>
<td>PARALLELISM</td>
<td>INTEGER</td>
<td>Number of parallel tasks being performed by the command.</td>
</tr>
<tr>
<td>TRANSACTION_TYPE</td>
<td>CHAR (32)</td>
<td>Transaction type for the command. Possible values: READ_ONLY, VERSIONING,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BLOCKING.</td>
</tr>
<tr>
<td>READ_REVISION</td>
<td>DECIMAL (10,0)</td>
<td>The revision number of the database being accessed by the transaction.</td>
</tr>
<tr>
<td>NEW_REVISION</td>
<td>DECIMAL (10,0)</td>
<td>The revision number of the database the transaction creates. NULL for aborted or uncommitted transactions.</td>
</tr>
<tr>
<td>TRANSACTION_ISOLATION_LEVEL</td>
<td>CHAR (32)</td>
<td>The isolation level of the transaction. Possible values: REPEATABLE_READ, SERIALizable.</td>
</tr>
<tr>
<td>LAST_UPDATED</td>
<td>TIMESTAMP</td>
<td>Timestamp of when this row was last updated.</td>
</tr>
</tbody>
</table>
DST_DATABASES Table

The DST_DATABASES table contains statistics that indicate the overall level of activity against a database. It also contains the database location and state (quiescent or active).

The following table lists and describes all the DST_DATABASES columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNAME</td>
<td>CHAR(128)</td>
<td>Logical database name.</td>
</tr>
<tr>
<td>DBLOCATION</td>
<td>CHAR(1024)</td>
<td>Database directory path.</td>
</tr>
<tr>
<td>CURRENT_CONNECTS</td>
<td>INTEGER</td>
<td>Current number of connected sessions.</td>
</tr>
<tr>
<td>PEAK_CONNECTS</td>
<td>INTEGER</td>
<td>Maximum number of concurrent sessions.</td>
</tr>
<tr>
<td>TOTAL_CONNECTS</td>
<td>INTEGER</td>
<td>Cumulative count of connected sessions.</td>
</tr>
<tr>
<td>TOTAL_FATAL_EXITS</td>
<td>INTEGER</td>
<td>Number of times a session has terminated abnormally. This includes any invalid login attempts and failed attempts against quiesced databases.</td>
</tr>
<tr>
<td>TOTAL_COMMANDS</td>
<td>INTEGER</td>
<td>Number of commands executed against this database.</td>
</tr>
<tr>
<td>QUIESCED</td>
<td>CHAR(1)</td>
<td>Flag indicating whether the database is quiesced (Y) or active (N).</td>
</tr>
<tr>
<td>ADMINDB</td>
<td>CHAR(1)</td>
<td>Flag indicating whether the database is the administration database (Y) or a user-created database (N).</td>
</tr>
<tr>
<td>BACKUP_SEGMENT</td>
<td>CHAR(128)</td>
<td>Name of the backup segment (populated only if a backup segment exists for use with SQL-BackTrack for Red Brick Warehouse).</td>
</tr>
</tbody>
</table>
### VERSION_LOG_SEGMENT

**Column Name**: VERSION_LOG_SEGMENT  
**Column Type**: CHAR (128)  
**Description**: The name of the segment containing the version log. NULL if version log does not exist.

### VERSIONING.Started

**Column Name**: VERSIONING.STARTED  
**Column Type**: CHAR (1)  
**Description**: Whether versioning is currently enabled on the database (Y, N, or NULL if version log does not exist).

### ACTIVE.VACUUM.CLEANERS

**Column Name**: ACTIVE.VACUUM.CLEANERS  
**Column Type**: INTEGER  
**Description**: The number of vacuum cleaner daemon processes active for the database (0 or 1). NULL if version log does not exist.

### CURRENT.REVISION

**Column Name**: CURRENT.REVISION  
**Column Type**: DECIMAL (10, 0)  
**Description**: The database revision number of the most recently committed version of the database. NULL if version log does not exist.

### OLDEST_ACTIVE.REVISION

**Column Name**: OLDEST_ACTIVE.REVISION  
**Column Type**: DECIMAL (10, 0)  
**Description**: The database revision number of the oldest committed version of the database having at least one active read process. NULL if version log does not exist.

### LATEST.MERGED.REVISION

**Column Name**: LATEST.MERGED.REVISION  
**Column Type**: DECIMAL (10, 0)  
**Description**: The database revision number of the latest version of the database that has been merged from the version log to the main database PSUs. NULL if version log does not exist.

### VERSION_LOG.USED

**Column Name**: VERSION_LOG.USED  
**Column Type**: INTEGER  
**Description**: The amount of disk space (in kilobytes) used in the version log for new versions of database blocks. NULL if version log does not exist.

### VERSION_LOG.AVAILABLE

**Column Name**: VERSION_LOG.AVAILABLE  
**Column Type**: INTEGER  
**Description**: The amount of free disk space (in kilobytes) available in the version log. NULL if version log does not exist.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION_LOG_MAXIMUM_USED</td>
<td>INTEGER</td>
<td>The maximum amount of disk space (in kilobytes) used in the version log. Also known as the “high water mark.” Can be reset with the ALTER SYSTEM RESET STATISTICS command. NULL if version log does not exist.</td>
</tr>
<tr>
<td>MAXREVISIONS</td>
<td>INTEGER</td>
<td>The maximum number of active revisions allowed in the database. NULL if version log does not exist. Can be changed with an ALTER DATABASE CREATE VERSION LOG IN statement.</td>
</tr>
<tr>
<td>LAST_UPDATED</td>
<td>TIMESTAMP</td>
<td>Timestamp of when this row was last updated.</td>
</tr>
</tbody>
</table>
**DST_LOCKS Table**

The DST_LOCKS table contains information about the locks that each session is holding or waiting for. If the session is waiting for a lock, the DST_LOCKS table gives some information on the process that is holding that lock (blocking).

The following table lists and describes the DST_LOCKS columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNAME</td>
<td>CHAR(128)</td>
<td>Database logical name.</td>
</tr>
<tr>
<td>UNAME</td>
<td>CHAR(128)</td>
<td>User name.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>CHAR(128)</td>
<td>Node name where the process is running (for MPP systems).</td>
</tr>
<tr>
<td>PID</td>
<td>INTEGER</td>
<td>PID of this process.</td>
</tr>
<tr>
<td>TNAME</td>
<td>CHAR(128)</td>
<td>Table being locked (NULL for segment lock).</td>
</tr>
<tr>
<td>SEGNAME</td>
<td>CHAR(128)</td>
<td>Name of the segment for a segment lock (NULL for table only lock).</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td>Time of the lock request.</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR(2)</td>
<td>Lock type used for the current transaction. Possible values: read-only (RO), read-key (RK), read-data (RD), write-blocking (WB), write-key (WK), and write-data (WD).</td>
</tr>
<tr>
<td>BLOCKER_UNAME</td>
<td>CHAR(128)</td>
<td>Name of user holding lock or NULL if current process is holding the lock.</td>
</tr>
<tr>
<td>BLOCKER_PID</td>
<td>INTEGER</td>
<td>PID of process holding lock that is blocking current attempt or NULL if the current process holds the lock.</td>
</tr>
<tr>
<td>BLOCKER_NODE</td>
<td>CHAR(128)</td>
<td>Node name where the blocking process is running (for MPP systems) or NULL if the current process is holding the lock.</td>
</tr>
<tr>
<td>LAST_UPDATED</td>
<td>TIMESTAMP</td>
<td>Timestamp of when this row was last updated.</td>
</tr>
</tbody>
</table>
DST_SESSIONS Table

The DST_SESSIONS table contains information on each session currently connected to the database. This information includes both cumulative statistics over all of the commands issued by the session and peak statistics (the maximum single command values).

The following column has been added to the DST_SESSIONS table. For a description of the other columns, refer to Appendix C of the Warehouse Administrator’s Guide.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_TEMPSPACE_DUPICATESPILLPERCENT</td>
<td>INTEGER</td>
<td>Percentage of the index building temporary space allocated for duplicates. Only valid for REORG operations.</td>
</tr>
</tbody>
</table>
Fully Qualified Database Pathnames (Windows NT only)

Add the following notes to the “Initializing the Database” section on page 5-3 of the Warehouse Administrator’s Guide for Windows NT Platforms.

When you use the dbcreate utility to initialize a database, you must fully qualify the pathname for the database directory by including the drive letter. For example:

```
c:\> dbcreate -create -l AROMA -d c:\disk1\new_db
```

The database pathname is added to the rbw.config file automatically. If you edit the database section of this file, be sure that the pathname for each database is fully qualified.

If you have two logical database names in the rbw.config file that refer to the same physical database, the pathnames must be identical for the two entries.

When you delete a database with the dbcreate utility, you must specify the pathname as it is listed in the rbw.config file. For example, if you have the following entry in your rbw.config file:

```
DB AROMA f:\redbrick\aroma_db
```

When you delete this database, you must specify the exact pathname with the dbcreate utility as follows:

```
c:\> dbcreate -delete -d f:\redbrick\aroma_db
```
Warehouse Administrator's Guide
Fully Qualified Database Pathnames (Windows NT only)
This chapter contains additions and changes to the SQL Reference Guide. When there are additions to an existing command, the description of the entire command is included. The following commands are included:

- ALTER DATABASE
- CREATE INDEX
- SET VERSIONING
- SET TRANSACTION ISOLATION LEVEL
- New Scalar Functions
- SET FIRST DAYOFWEEK Command
- Supported Date Formats
- Miscellaneous Changes and Additions
**ALTER DATABASE**

The ALTER DATABASE command has the following purposes:

- To specify a segment as the backup segment.
- To drop the backup data stored in the backup segment.
- To create a version log in a segment.
- To drop the version log.
- To start and stop the processing of versioning transactions.
- To start and stop the vacuum cleaner daemon/thread for the database.

Only one segment per database can be defined as the backup segment. If no backup segment is defined, SQL-BackTrack backup operations cannot be performed.

Only one segment per database can be defined as the version log. If no version log is defined, versioning transactions cannot be performed.

The backup-related components of the ALTER DATABASE command are available only for Red Brick Warehouse installations that have SQL-BackTrack for Red Brick Warehouse enabled with a license key.

**Authorization**

To issue the ALTER DATABASE command, a user must meet at least one of the following requirements:

- Be a member of the DBA system role.
- Have BACKUP_DATABASE and RESTORE_DATABASE authorization, either explicitly or through membership in a user-created role (to perform backup operations).
- Have ALTER_SYSTEM authorization, either explicitly or through membership in a user-created role (to perform version log operations).
**Syntax**

The following diagram shows how to construct an ALTER DATABASE statement:

```
ALTER DATABASE  
  CREATE BACKUP DATA IN segment_name  
  DROP BACKUP DATA  
    version_log_specification  
    DROP VERSION LOG  
    START VERSIONING  
    STOP VERSIONING  
    START CLEANING  
    STOP CLEANING  
    TERMINATE  
  DATABASE logical_database_name
```

**CREATE BACKUP DATA IN segment_name**

Names an existing but unused segment as the backup segment for the database. The segment is created in the usual way with a CREATE SEGMENT statement and can consist of multiple PSUs. When the command is issued, the named segment is marked as the backup segment in the DST_DATABASES table. For information about the CREATE SEGMENT command, refer to “CREATE SEGMENT” on page 8-96 of the *SQL Reference Guide*.

**DROP BACKUP DATA**

Removes the backup data (bitmap information) from the database and changes the backup segment to a regular segment; the segment itself is not dropped. When this command has been issued, backup and recovery operations can no longer be performed.

**Caution:** Do not drop the backup data unless you no longer intend to perform backup and recovery operations on the database. Databases cannot be recovered to a checkpoint backup that used a backup segment that has since been dropped and re-created. Database recoveries do not work across multiple instances of the backup segment.
version_log_specification
Creates a version log for the database using the disk space in the named, unused segment. The following syntax diagram shows how to construct a version_log_specification.

```
CREATE VERSION LOG IN segment_name
```

**MAXREVISIONS number**
Specifies the maximum number of versions of the database allowed in the version log at a given time. The default number of revisions is 500.

A database can have only one version log. The version log must be created before versioning transactions can be enabled for the database. All of the disk space for the version log is allocated by this statement, therefore this operation might take some time (all of the PSUs in the segment are extended to their MAXSIZE values before the operation completes).

No other users can be connected to the database during the CREATE VERSION LOG operation; all other connections are refused after the operation begins.

If an error occurs during the CREATE VERSION LOG operation, drop the version log and run the operation again.

After creating the version log, you must start versioning (ALTER DATABASE START VERSIONING) before the database accepts versioning transactions.

Creating the version log starts the vacuum cleaner daemon/thread for the database.
DROP VERSION LOG
Terminates the vacuum cleaner, releases the versioning shared resources, and detaches the segment containing the version log. The disk space used by the segment is not released. This behavior differs from the DROP TABLE statement, which shrinks the PSUs back to their initsize values. If versioning is still active or the version log still contains active data that has not yet been processed by the vacuum cleaner daemon/thread, no action is taken and an error occurs.

You must stop versioning with an ALTER DATABASE STOP VERSIONING command and wait for the vacuum cleaner to finish cleaning the version log before you can drop the version log. The version log is empty when the value of the CURRENT_REVISION column in the DST_DATABASES table is equal to the value of the LATEST_MERGED_REVISION column. It is necessary to drop the version log before performing an UPGRADE operation.

No other user can be connected to the database during a DROP VERSION LOG operation.

START VERSIONING
Allows new versioning transactions to be processed by the database. It is only necessary to start versioning if versioning has been explicitly stopped or if the version log has just been created for the database. The version log must exist in order to execute this command.

STOP VERSIONING
Stops the database from accepting new versioning transactions. After performing an ALTER DATABASE STOP VERSIONING operation, new transactions that modify the database will fail with an error (as opposed to automatically becoming blocking transactions).

The versioning state of a database remains persistent after a restart of Red Brick Warehouse. Therefore, if the rbwapid daemon (UNIX systems) or the Red Brick Warehouse service (Windows NT systems) is shut down while versioning is started for a database, then versioning is restarted when the daemon/service is restarted. Similarly, if versioning is stopped for a database when the rbwapid daemon/Red Brick Warehouse service is shut down, then versioning for that database remains stopped when the daemon/service is restarted.

START CLEANING
Starts the vacuum cleaner daemon/thread for the database if it is not already running.
STOP CLEANING

Stops the vacuum cleaner daemon/thread for the database if it is running. If the rbwapid daemon (UNIX systems) or the Red Brick Warehouse service (Windows NT systems) is shut down while the vacuum cleaner is stopped, then the vacuum cleaner is automatically restarted when the daemon/service is restarted.

TERMINATE

Ends the database session and releases all the shared resources for that database. If there are other users on the connected to the database, then the command fails with an error. If run from the ADMIN database, you must specify the DATABASE logical_database_name clause indicating which database’s shared resources are released.

Usage Notes

The backup segment does not need to be “offline” for the ALTER DATABASE command to work.

You cannot use a DROP SEGMENT statement to remove the backup segment from the database. However, after an ALTER DATABASE DROP BACKUP DATA statement has been issued, the segment is a regular segment and can be dropped.

You cannot use a DROP SEGMENT statement to remove the version log from the database. However, after an ALTER DATABASE DROP VERSION LOG statement has been issued, the segment is a regular segment and can be dropped.

Extending the PSUs in the segment specified for the version log can take some time because all of the disk space must be allocated. The system administrator is advised to use care when setting the MAXSIZE parameters on the PSUs of this segment as they define the amount of disk storage that is immediately allocated and assigned to the version log.

The location of the version log can affect performance. If the version log is heavily used (for example, if your LOAD operations change a large number of blocks in the database), access to the version log can become a bottleneck to system performance; therefore, it is best to place the version log on high performance storage such as is provided by RAID level 0 or level 1. To ensure the best performance, the device(s) in which the version log resides should be independent of the devices in which your database files reside. Because good write performance is needed, higher RAID levels are not recommended.
Because the version log might contain data from any number of segments in the database, loss of the version log can potentially require the entire database to be recovered from backup. To safeguard against this, the version log should be placed on reliable storage such as is provided by RAID level 0.

**Example—Defining a Backup Segment**

The following SQL statements show how to create a segment, then define it as the backup segment.

1. Create the segment:
   ```sql
   create segment sqlbacktrack_seg
   storage '/test/bt1' maxsize 1024,
   storage '/test/bt2' maxsize 1024,
   storage '/test/bt3' maxsize 1024,
   storage '/test/bt4' maxsize 1024;
   ```

2. Define the segment as the backup segment:
   ```sql
   alter database create backup data in sqlbacktrack_seg;
   ```

**Example—Creating a Version Log**

The following SQL statements show how to create a segment, then define it as the version log.

1. Create the segment:
   ```sql
   create segment versionlog_seg
   storage '/test/vl1' maxsize 1024,
   storage '/test/vl2' maxsize 1024,
   storage '/test/vl3' maxsize 1024,
   storage '/test/vl4' maxsize 1024;
   ```

2. Define the segment as the version log:
   ```sql
   alter database create version log in versionlog_seg;
   ```
   This command allocates all of the physical space for the version log, so it might take some time to complete.

3. Start versioning for the database:
   ```sql
   alter database start versioning;
   ```
CREATE INDEX

This section supplements the description of the CREATE INDEX command beginning on page 8-71 of the SQL Reference Guide. This section only describes the new option to create a deferred index. A deferred index is an index whose structure is created with a CREATE INDEX command, but the index is not populated until you perform a REORG operation. The REORG operation can take advantage of parallelism, which can populate indexes on large tables faster than with the CREATE INDEX command.

The CREATE INDEX command creates one or more indexes in addition to the primary key index that is automatically created for each table. Multiple indexes can be created with a single CREATE INDEX statement, but they must all index the same table.

Syntax

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

```
CREATE INDEX —index_specifier
   ,
   DEREFERRED
   IMMEDIATE
   ABORT
   CONTINUE
   ERROR
   TARGET
   STAR
```

**IMMEDIATE**

Creates and builds the index at the time the command executes.

**DEFERRED**

Creates the index structures and creates an entry in the RBW_INDEXES system table with DEFERRED as the value in the STATE column. The index is not populated with data at this time. The index must be populated with a TMU REORG operation before it can be used.
A CREATE INDEX...DEFERRED operation skips the potentially time-consuming phase of building the index from the data (or deriving it from another index). Everything that can be done without building the index is done; this includes associating the specific segment(s) with the table and various semantic validation checks. This operation is normally followed by a parallel TMU REORG operation, which builds the index more efficiently than the server can.

An index in the DEFERRED state is empty and thus invisible to all functions that read or write the contents of an index. These functions include:

- Queries
- INSERT/UPDATE/DELETE operations
- Load/Unload operations

Additionally, deferred indexes are not visible to the compiler; therefore, query plans cannot be generated for them.

An index in the DEFERRED state is visible to those functions that operate only on the structure of an index. In general, these functions make no distinction between DEFERRED indexes and fully built indexes. DEFERRED indexes are visible to the following functions:

- DROP INDEX: The index can be dropped.
- ALTER INDEX: All operations are allowed.
- DROP TABLE: Deletes all indexes in the table including the DEFERRED indexes.
- ALTER TABLE—DROP COLUMN: This operation is prohibited if the specified column is a component of the DEFERRED index.
- CREATE INDEX: Another index with the same name or the same type cannot be created on the same columns.
- ALTER SEGMENT: All operations allowed on non-DEFERRED indexes are allowed on DEFERRED indexes.

For information on using the TMU REORG operation to populate a deferred index, refer to “Reorganizing Tables and Indexes” on page 5-2.
SET VERSIONING

The SET VERSIONING command determines whether database transactions are run as blocking transactions or versioning transactions. This command applies to versioned databases only and affects DML (data manipulation language) operations that change data in the database (INSERT, UPDATE, and DELETE) only.

Syntax

The following syntax diagram shows how to construct a SET VERSIONING statement.

```
 SET VERSIONING [OFF | ON]
```

OFF
Specifies that statements are to be run as blocking transactions.

ON
Specifies that statements are to be run as versioning transactions.

Usage Notes

If the connected database does not have a version log, then a SET VERSIONING ON command causes transactions that modify the database (INSERT, UPDATE, and DELETE statements) to fail with an error.

If the connected database has a version log but versioning is currently stopped (ALTER DATABASE STOP VERSIONING), then transactions that modify the database (INSERT, UPDATE, and DELETE statements) that would be run as versioning transactions fail with an error.

The SET VERSIONING command does not have an effect on DML operations for models and temporary tables; those operations are always performed as non-versioned (blocking) transactions.

This command also exists as the OPTION VERSIONING parameter in the rbw.config file. For more information about this command, refer to the Warehouse Administrator’s Guide.
**SET TRANSACTION ISOLATION LEVEL**

The SET TRANSACTION ISOLATION LEVEL command controls the type of read locks used by an SQL statement that reads one table and changes another table, where there are no key relationships between the two tables. This command applies to versioned databases only.

**Syntax**

The following syntax diagram shows how to construct a SET TRANSACTION ISOLATION LEVEL statement.

```
➤ SET TRANSACTION ISOLATION LEVEL  SERIALIZABLE
          REPEATABLE READ ➢
```

**SERIALIZABLE**

Specifies that an RD (Read Data) lock is used for a transaction that reads one table to modify another table. The RD lock does not permit simultaneous versioning modification on a table being read by another versioning transaction; the later transaction waits for the earlier one to complete before it can read the table. This is the most restrictive mode.

**REPEATABLE READ**

Specifies that an RO (Read Only) lock is used. The RO lock allows a versioning modification of the table concurrent with a read operation. This allows a transaction to read an older version of a table which is currently being modified in order to modify another table (for example, an INSERT INTO...SELECT statement). This mode is less restricted than SERIALIZABLE.

This command also exists as the OPTION TRANSACTION_ISOLATION_LEVEL parameter in the rbw.config file. For more information about this parameter and examples of the differences between the two settings, refer to page 2-15 in Chapter 2, “Working with a Versioned Database.”
New Scalar Functions

This section describes scalar functions and supplements the function descriptions in Chapter 5 of the SQL Reference Guide. This section is divided into the following subsections:

- EXP
- LN
- SQRT
- Macros for Statistical Functions
- LENGTH

EXP

The EXP function returns the exponential value of the function argument; that is, EXP(x) returns the value \( y \) such that \( e^x = y \). (\( e = 2.71828183 \).)

Syntax

The following syntax diagram shows how to construct an expression with the EXP function:

```
<table>
<thead>
<tr>
<th>expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP</td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td>expression</td>
</tr>
<tr>
<td>)</td>
</tr>
</tbody>
</table>
```

expression

The expression can be any datatype but must represent a numeric value. For example:

- 25 — valid
- 8.634E2 — valid
- 14.695 — valid
- '25' — valid
- 'RAJ' — not valid

Result

This function returns a DOUBLE PRECISION value.

If expression is of the datatype TINYINT, SMALLINT, INTEGER, REAL, FLOAT, DECIMAL, or NUMERIC, it is converted to a DOUBLE PRECISION value.
If $expression$ is a CHARACTER expression representing a numeric value, the EXP function converts this value to a double precision floating point, calculates the exponential value, and returns the exponential value as a double precision. If the character string does not represent a numeric value, the server returns an error message. If the argument is NULL, the function returns NULL.

If the result overflows the DOUBLE PRECISION datatype, the server returns a platform-specific overflow result.

**Example**

The expression

```
EXP (2.0)
```

returns 7.39. ($e^2=7.39; e=2.71828183.$)

**LN**

The LN function returns the natural logarithm of the function argument; that is, $\text{LN}(y)$ returns the value $x$ such that $e^x=y$. ($e=2.71828183.$)

**Syntax**

The following syntax diagram shows how to construct an expression with the LN function:

```
LN ( expression )
```

**expression**

The expression can be any datatype but must represent a numeric value. For example:

- 25—valid
- 8.634E2—valid
- 14.695—valid
- 'RAJ'—not valid
Result

The function returns a DOUBLE PRECISION value. If the argument is NULL, the function returns NULL. If the length of the result exceeds the length of the DOUBLE PRECISION datatype, the server returns an “out of range” error message.

If the argument is a TINYINT, SMALLINT, INTEGER, REAL, FLOAT, DECIMAL, or NUMERIC value, it is converted to a DOUBLE PRECISION value.

If the argument is negative or zero, the behavior is dictated by the ARITHIGNORE/ARITHABORT setting, similar to divide-by-zero processing. If ARITHIGNORE is ON (or ARITHABORT is OFF), the server returns NULL for computing natural logarithm on negative arguments. If ARITHABORT is ON (or ARITHIGNORE is OFF), the server produces an error message. For more information about the ARITHIGNORE/ARITHABORT settings, refer to Chapter 9 of the SQL Reference Guide.

If the argument is a character expression that represents a positive number, the argument is converted to a double precision floating-point value; otherwise, the server returns an error message.

Example

The expression

\[ \ln(25.0) \]

returns 3.22. \( (e^{3.22}=25; e=2.71828183.) \)
**SQRT**

The SQRT function returns the square root of the function argument.

**Syntax**

The following syntax diagram shows how to construct an expression with the SQRT function:

```
SQRT ( expression )
```

**expression**

The expression can be any datatype but must represent a numeric value. For example:

- 25 — valid
- 8.634E2 — valid
- 14.695 — valid
- 'RAJ' — not valid

**Result**

The function returns a DOUBLE PRECISION value. If the argument is NULL, the function returns NULL. If the length of the result exceeds the length of the DOUBLE PRECISION datatype, the server returns an “out of range” error message.

If the argument is a TINYINT, SMALLINT, INTEGER, REAL, FLOAT, DECIMAL, or NUMERIC value, it is converted to a DOUBLE PRECISION value.

If the argument is negative, the behavior is dictated by the ARITHIGNORE/ARITHABORT setting, similar to divide-by-zero processing. If ARITHIGNORE is ON (or ARITHABORT is OFF), the server returns NULL for computing square root on negative arguments. If ARITHABORT is ON (or ARITHIGNORE is OFF), the server produces an error message. For more information about the ARITHIGNORE/ARITHABORT settings, refer to Chapter 9 of the SQL Reference Guide.
If the argument is a character expression that represents a numeric value, the SQRT function converts the value to a double precision floating-point value, calculates the square root value, and returns the square root value as a double precision value. If the character string does not represent a numeric value, the server returns an error message.

**Example**

The expression

```
sqrt(25.0)
```

returns 5.000.

**Macros for Statistical Functions**

The following macros can be built using the EXP, LN, and SQRT scalar functions:

- Power
- Log
- Log10
- Variance
- Standard Deviation
- Population Variance
- Population Standard Deviation

**Power**

The following macro calculates \( x \) raised to the \( y \)th power:

```
create macro power(x, y) as
  (exp( (y) * ln((x))) )
```

**Log**

The following macro calculates the logarithm of a number \( x \) to base \( y \):

```
create macro log(x, y) as
  ( ln ((x)) / ln((y)) )
```
Log10

The following macro calculates the logarithm of a number \( x \) to base 10:

\[
\text{create macro log10}(x) \text{ as } (\frac{\ln(x)}{\ln(10)})
\]

Variance

The following macro calculates the variance for a column \( \text{col}_\text{name} \):

\[
\text{create macro variance}(\text{col}_\text{name}) \text{ as } \frac{(\text{count}(\text{col}_\text{name}) \times \text{sum}((\text{col}_\text{name}) \times (\text{col}_\text{name}))) - (\text{sum}(\text{col}_\text{name})\times \text{sum}(\text{col}_\text{name})))}{(\text{count}(\text{col}_\text{name}) \times (\text{count}(\text{col}_\text{name}) - 1))}
\]

Note: The Variance macro does not use the EXP, LN, or SQRT functions, but it is included here because variance is used as a component of the Standard Deviation macro.

Standard Deviation

The following macro calculates the standard deviation for a column \( \text{col}_\text{name} \):

\[
\text{create macro stddev}(\text{col}_\text{name}) \text{ as } \sqrt{\text{abs}((\text{count}(\text{col}_\text{name}) \times \text{sum}((\text{col}_\text{name}) \times (\text{col}_\text{name}))) - (\text{sum}(\text{col}_\text{name})\times \text{sum}(\text{col}_\text{name}))) / (\text{count}(\text{col}_\text{name}) \times (\text{count}(\text{col}_\text{name}) - 1)))}
\]

Population Variance

The following macro calculates the population variance for a column \( \text{col}_\text{name} \):

\[
\text{create macro population_variance}(\text{col}_\text{name}) \text{ as } \frac{(\text{sum}(\text{col}_\text{name}) \times (\text{col}_\text{name})) - (\text{sum}(\text{col}_\text{name})\times \text{sum}(\text{col}_\text{name})))}{(\text{count}(\text{col}_\text{name}) \times \text{count}(\text{col}_\text{name}))}
\]

Note: The Population Variance macro does not use the EXP, LN, or SQRT functions, but it is included here because population variance is used as a component of the Population Standard Deviation macro.
Population Standard Deviation

The following macro calculates the population standard deviation for a column `col_name`:

```sql
create macro population_stddev(col_name) as
  sqrt( abs(( sum((col_name) * (col_name)) -
             (sum((col_name)) * sum((col_name))) / count((col_name))) ) / count((col_name))) ;
```

Example

The following query uses the `stddev` macro to calculate the standard deviation on the average sales of Earl Grey tea in the Western region during the first quarter of 1995.

```sql
select store_name, sum(dollars) as Sales,
       avg(dollars) as AvgSales,
       stddev(dollars) as S_Dev
from period natural join sales
     natural join store
     natural join market
     natural join product
where region = 'West'
and prod_name = 'Earl Grey'
and qtr='Q1_95'
and year = 1995
group by qtr, store_name
order by qtr, store_name;
```

<table>
<thead>
<tr>
<th>STORE_NAME</th>
<th>SALES</th>
<th>AVGSALES</th>
<th>S_DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beaches Brew</td>
<td>1159.50</td>
<td>39.98275862</td>
<td>15.35</td>
</tr>
<tr>
<td>Cupertino Coffee Supply</td>
<td>628.50</td>
<td>28.56818181</td>
<td>15.93</td>
</tr>
<tr>
<td>Instant Coffee</td>
<td>717.50</td>
<td>42.20588235</td>
<td>14.92</td>
</tr>
<tr>
<td>Java Judy’s</td>
<td>595.50</td>
<td>35.02941176</td>
<td>18.34</td>
</tr>
<tr>
<td>Roasters, Los Gatos</td>
<td>911.00</td>
<td>41.40909090</td>
<td>18.13</td>
</tr>
<tr>
<td>San Jose Roasting Company</td>
<td>395.00</td>
<td>28.21428571</td>
<td>15.64</td>
</tr>
</tbody>
</table>
**LENGTH**

The **LENGTH** function computes the number of characters in a string.

**Syntax**

The following syntax diagram shows how to construct an expression with the **LENGTH** function:

```
LENGTH ( c_expression )
```

- **c_expression**
  - The **c_expression** must be a character datatype.

**Result**

- If the argument is not NULL, the function returns an integer result specifying the number of characters in the string; otherwise, the result is NULL.

- The actual length of the returned string is equal to the maximum length of the character string as defined in the CREATE TABLE statement. The **SUBSTR**, **TRIM**, **LTRIM**, and **RTRIM** functions can reduce the maximum length of a character string.

**Note:** The **LENGTH** function returns the number of characters, not the number of bytes, in a string.
Example 1

The following query returns the length in characters of the district names in the Market table. Note that because no function such as TRIM or SUBSTR is used to reduce the maximum length of the character string, the query returns the maximum value of the character string.

```
select distinct district, length(district) as char_len
> from market;
```

<table>
<thead>
<tr>
<th>DISTRICT</th>
<th>CHAR_LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta</td>
<td>20</td>
</tr>
<tr>
<td>Boston</td>
<td>20</td>
</tr>
<tr>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>20</td>
</tr>
<tr>
<td>Minneapolis</td>
<td>20</td>
</tr>
<tr>
<td>New Orleans</td>
<td>20</td>
</tr>
<tr>
<td>New York</td>
<td>20</td>
</tr>
<tr>
<td>San Francisco</td>
<td>20</td>
</tr>
</tbody>
</table>

Example 2

The following query returns the longest district name in the Market table. The TRIM function is used so the actual number of characters in each district name is considered, rather than the maximum length of the character string:

```
select distinct district from market
    where length(trim(district)) =
    (select max(length(trim(district))) from market)
```

<table>
<thead>
<tr>
<th>DISTRICT</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Francisco</td>
</tr>
</tbody>
</table>
**SET FIRST DAYOFWEEK Command**

Add the following SET command description to Chapter 9 of the *SQL Reference Guide*.

The SET FIRST DAYOFWEEK command overrides the default numbering system defined by the warehouse locale for the days of the week. This command affects the results returned by the EXTRACT and DATENAME functions when the `weekday` argument is used. (For detailed information about these functions, refer to Chapter 5, “Scalar Functions.”)

To specify the first day of the week for all server sessions, use the NLS_LOCALE FIRST_DAYOFWEEK parameter in the *rbw.config* file.

**Syntax**

The following syntax diagram shows how to construct a SET FIRST DAYOFWEEK statement:

```
SET FIRST DAYOFWEEK [DEFAULT] daynum
```

**DEFAULT**
Resets the first day of the week to its default value in the current warehouse locale. For example, in a German database, the default value is 2 (Monday).

**daynum**
Specifies an integer value from 1 to 7 that maps to the days of the week in the default U.S.-English locale: 1 = Sunday, 2 = Monday, and so on.

**Usage Notes**

This command has no effect on *day names*. Setting the first day of the week to a different number does not change how the server assigns day names to dates. For example, January 1, 1998, is *Thursday* in an English database and *Donnerstag* in a German database regardless of which day of the week is treated as the first.

The RBW_OPTIONS system table stores the current value of this parameter under the option name FIRST_DAYOFWEEK.

**Example**

The following statement sets the first day of the week to 7 (Saturday):

```
set first dayofweek 7
```
Supported Date Formats

This information supplements the date input and display information in Chapter 5 and Appendix C of the SQL Reference Guide.

Input Formats

Red Brick Systems supports the following input formats for the entry of date data. The formats supported depend upon the locale (language and territory) of the database.

U.S.-English Locales

In U.S.-English locales (locales with the language defined as English and the territory as UnitedStates), Red Brick Systems strongly recommends that dates be entered using the ANSI SQL-92 date format. For example:

```sql
date '1997-12-31'
```

Red Brick Systems also provides support for alternative date formats as a transition from SQL Server date formats to ANSI SQL-92; new database development and tool support should use the ANSI SQL-92 definitions.

Alternative date format support in the U.S. English locale is described fully in Appendix C of the SQL Reference Guide.

All Other Locales

Dates using ANSI SQL-92 format are supported for all non-U.S.-English locales.

One alternative date format is also supported for each non-U.S.-English locale. The format varies by locale; for example, for the Japan.Japanese locale, the supported format has the following structure:

```sql
yyyy#mm#dd#
```

where # is a double-byte character representing the appropriate datepart (year, month, or day).
For example, the following date:

1997年01月06日

shows the alternative long date format with the Japanese double-byte year, month, and day characters.

Note: No other alternative date formats are supported for non-U.S.-English locales. Attempting to enter dates in these non-supported formats will result in an error.

**Entering the Year Datepart in the Japanese Locale**

To avoid ambiguity when entering dates in the Japanese alternative date format, always enter the year using four digits, not two. Although you can enter the year using either two or four digits, the system interprets each entry as a four-digit year. Thus, the following:

1997年01月06日

is interpreted as:

1997-01-06

while

97年01月06日

is interpreted as

0097-01-06

**Output Formats**

The way in which dates are displayed on the client side varies widely by locale. The following section describes the differences in default output formats for U.S.-English and non-U.S.-English locales.

**U.S. English**

The output format for dates in the U.S.-English locale is as follows:

```
yyyy-mm-dd
```

For example:

1998-04-23

This output format cannot be changed, except by processing with scalar functions.
All Other Locales

The default output format for non-U.S.-English dates varies by locale. For the Japan.Japanese locale, the format is as follows:

\text{yy/mm/dd}

For example:

\text{97/12/31}

In order to display non-U.S.-English locale dates using a four-digit year, use the \text{ANSI_DATETIME_DISPLAY_FORMAT} setting.

\text{ANSI_DATETIME_DISPLAY_FORMAT} option

The \text{ANSI_DATETIME_DISPLAY_FORMAT} option defines the method by which dates will be displayed on the client side in non-U.S.-English locales.

The option is set in the following locations, depending on the platform:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 3.1</td>
<td>\text{rbodbc16.ini}</td>
</tr>
<tr>
<td>Windows NT</td>
<td>\text{rbodbc32.ini}</td>
</tr>
<tr>
<td>Windows 95</td>
<td>\text{rbodbc32.ini}</td>
</tr>
<tr>
<td>UNIX</td>
<td>environment variable</td>
</tr>
</tbody>
</table>

Setting \text{ANSI_DATETIME_DISPLAY_FORMAT} to ON uses YYYY-MM-DD as the display format for non-U. S.-English locales.

Setting \text{ANSI_DATETIME_DISPLAY_FORMAT} to OFF displays the year using the default display format for the locale.

The default setting is OFF.
Example

In the Japan Japanese locale, dates entered in the following way:

2001 #10 #31#
1901 #10 #31#

are displayed as follows with ANSI_DATETIME_DISPLAY_FORMAT set to OFF:

01/10/31

With ANSI_DATETIME_DISPLAY_FORMAT set to ON, the same dates are displayed as follows:

2001-10-31
1901-10-31

Conversely, a date entered in the following way:

97 #10 #31#

is displayed as follows with ANSI_DATETIME_DISPLAY_FORMAT set to OFF:

97/10/31

but as follows with ANSI_DATETIME_DISPLAY_FORMAT set to ON:

0097-10-31
Miscellaneous Changes and Additions

Note the following changes to the SQL Reference Guide.

Maximum Row Size

Add the following note to step 2 of “How SELECT Statements Are Processed” on page 7-51 of the SQL Reference Guide.

Note: If the row size of the final result or any intermediate result is greater than the maximum size of a row, the SELECT statement will fail. For more information about the maximum size of a row, refer to the “Warehouse Limits” section of the Warehouse Administrator’s Guide.

Chapter 3 of the SQL Reference Guide

Make the following correction to Chapter 3 of the SQL Reference Guide.

The syntax diagram on page 3-8 incorrectly shows the ESC keyword in front of the second string_expression for the LIKE predicate. The correct keyword is ESCAPE.
This chapter contains new and changed material for the *Table Management Utility Reference Guide*. The following topics are included:

- Reorganizing Tables and Indexes
- Understanding Versioned LOAD Operations
- Enabling Versioning for the TMU
- Setting Versioned Commit Intervals
- Miscellaneous Changes and Additions
Reorganizing Tables and Indexes

This section includes new descriptions and syntax for the REORG operation and replaces Chapter 6 of the Table Management Utility Reference Guide.

The REORG operation populates indexes, maintains referential integrity, and improves internal storage of a table’s indexes. The parallel processing capability of the REORG operation can result in significant time savings.

This section contains the following subsections:

• The REORG Operation
• Data Processing During the REORG Operation
• REORG Syntax
• Usage Notes
• SET Statements and Parameters to Control Behavior

The REORG Operation

The REORG operation performs the following functions:

• It checks referential integrity, if applicable for the target table, and either deletes rows that violate it or invalidates any affected indexes. (Referential integrity is the relational property that each foreign key value in a table exists as a primary key value in the referenced table.)

• It performs an internal reorganization of one or more of the table’s indexes (all types) to improve the internal storage of this information and thereby the performance when the index is used to access data. It can rebuild all indexes, selectively rebuild one or more named indexes, or selectively rebuild one or more segments of one or more named indexes.

• It populates a DEFERRED index that is created with a CREATE INDEX statement. (A DEFERRED index is an empty index structure that can be populated at a later date. For more information about DEFERRED indexes, refer to “CREATE INDEX” on page 4-8 of this document.)
If a table changes over time, you might need to reorganize it and any other tables and indexes affected by changes to that table. A REORG operation is necessary in the following cases:

- To rebuild the affected indexes if you use a database restore operation to restore individual segments of a table or index.
- Whenever modifications to a database affect more than about 30% of the data, you should run the TMU with a REORG statement for any tables directly modified. Periodically rebuilding such tables and indexes with a REORG statement ensures referential integrity and optimal performance.
- To reorganize invalid STAR indexes. Certain operations can invalidate STAR indexes. For example, increasing the MAXROWS PER SEGMENT or the MAXSEGMENTS parameter on a table can invalidate STAR indexes on tables that reference the altered table. These operations always generate a warning message that says STAR indexes based on the altered table might be invalid, in which case the affected STAR indexes need to be reorganized. You can either reorganize affected indexes when the message is issued or schedule the REORG operation for a more convenient time. However, any non-query (INSERT, UPDATE, or DELETE) operation against a table that has an invalid index will result in an error message that says the index must be reorganized. You must perform a REORG operation before the table can be accessed for an INSERT, UPDATE, DELETE, or LOAD operation.

REORG is unnecessary in the following cases:

- If no changes are made to the database except by complete loads of data (and you do not use a RESTORE…FORCE operation to restore individual segments).
- If the table and indexes are segmented alike and new index data is loaded into new index segments corresponding to new table segments. For an example of this type of setup, refer to Appendix D of the Warehouse Administrator's Guide.

Options

The REORG operation offers the following options:

- The choice of reorganizing a part of an index or the whole index. In the case of a partial-index REORG operation, you can restrict the operation to a specified list of one or more segments for each index being reorganized. (Multiple segments for multiple indexes defined on a single table can be rebuilt in a single REORG operation.)
The choice, during a partial REORG operation, of keeping segments to be reorganized either online or offline. If the segments are offline, the online portion of the index can still be used in queries while the REORG is in progress.

The choice, during a partial REORG operation, of scanning only a portion of the table to rebuild an index. You can scan one or more data segment(s) instead of scanning the entire table. This option can be used only if the same column is used to segment the table and the indexes being rebuilt and if all the keys for the index segment being rebuilt can be found in the specified data segment(s).

The choice of disabling referential integrity checking. This feature offers substantial timesaving when referential integrity checking is not necessary to build the required index.

A choice of behavior options for rows that fail referential integrity checking or result in a duplicate index key:
– The affected rows can be deleted from the table and from all indexes defined on the table.
– Any index affected by the error can be marked invalid without affecting the reorganization of other indexes.
– The REORG operation can be aborted immediately in case of an error in any of these indexes. All indexes being rebuilt by the REORG will be marked invalid.

The choice of recording rows that fail referential integrity checking in a separate file.

A choice of how the discarded rows are reported. They can be reported along with all other REORG messages or, optionally, recorded in a user-specified file, or separated into multiple files depending on the type of failure (duplicate rows or referential integrity failure). If the user has not specified a discard file, a message notifies the user when a row is discarded.

Note: To use the REORG statement, you must be a member of the DBA system role or be the owner of the table.
Data Processing During the REORG Operation

Both serial and parallel REORG operations perform identical functions. However, a parallel REORG operation uses separate tasks concurrently whereas a serial REORG operation uses only one task that proceeds serially from one stage to the next. The REORG operation consists of the following stages:

- Coordinator Stage
  - Validates the REORG command.
  - Acquires all necessary locks and sets the state of each index being rebuilt to prevent other users from accessing the index.
  - Clears the indexes or segments of the index being reorganized.

Also during this stage, a parallel REORG operation:
  - Determines how many additional tasks to use for each stage.
  - Assigns work to each task.
  - Determines the order of the tasks in the REORG pipeline.
  - Creates the processes (UNIX) or threads (Windows NT) for each stage in the REORG pipeline and starts their execution.

- Input Stage
  - Reads each row from the target table.
  - Passes the data on to a conversion task.

- Conversion Stage
  - Checks referential integrity on all foreign keys if reference checking is enabled. If reference checking is disabled, checks referential integrity only on foreign keys used in any STAR index being rebuilt.
  - Constructs a key for each index being rebuilt.
  - Identifies which index builder task has work to do for the current row. The row is skipped if the key value does not belong to any index segment being rebuilt.
  - Passes the data on to the first index builder task in the pipeline.

- Index Builder Stage
  - Inserts key values into their assigned segments of an index.
  - Passes the data on to the next index builder task or to the cleanup task.
• Cleanup Stage
  – Performs the function required for the selected ON DISCARD option.
  – Marks the successfully rebuilt indexes valid.
  – Reports status of the REORG operation.

The REORG operation uses its parallel processing capability to improve performance in two ways:
• It uses separate processes or threads for each stage, creating a pipeline in which rows are passed from one stage to the next, with multiple rows being processed simultaneously. Even on systems with a single CPU, multiple processes or threads can take advantage of I/O and CPU overlap, which might reduce elapsed time.
• On systems with multiple CPUs, additional input, conversion, and index builder tasks are created to further improve the pipeline.

The parallel processing capability of the REORG operation improves with an increase in the number of PSUs in a table or with an increase in the number of segments in an index.
The following figure illustrates the sequence of tasks in a REORG operation.

**Coordinator Stage**

During the coordinator stage, the coordinator task receives the REORG command and checks the validity of the REORG parameters. It determines how many tasks to use, assigns work for each stage and determines the order of the tasks in the REORG pipeline. After the completion of all the stages, the coordinator terminates the REORG operation.

**Input Stage**

During the input stage, each row from the target table is read and the information is passed on to the conversion stage. (In some cases, to perform the REORG operation faster, a STAR index might be scanned instead of the table. This choice is invisible to the user.) The number of input tasks cannot exceed the number of physical storage units (PSUs) in the target table.
Conversion Stage

During the conversion stage, referential integrity is checked (if you have enabled this option) and a key is constructed for each index being rebuilt. The index-building work to be done on each row is identified, and the row is then directed to the next stage in the pipeline.

Index-Building Stage

During the index-building stage, a key value is inserted into a specified segment of an index by an index-builder task. An index-builder task can insert keys into multiple segments of one index, all the segments of one index, or all the segments of a set of indexes. Multiple index-building tasks handle different subsets of segments of the same index and each index is built at a different stage of the pipeline. If the OPTIMIZE option is ON, sorting and merging strategies are used to add rows to the index. There can be multiple index-builder tasks operating simultaneously. The number of index-building tasks cannot exceed the total number of index segments being built. No index builder tasks are allocated for segments not being rebuilt.

Cleanup Stage

During the cleanup stage, the discarded rows are removed and recorded in the discard files as specified by the user. The cleanup task, depending on the selected ON DISCARD option, takes action as follows:

- If the DELETE ROW option is selected, the discarded rows are recorded in the discard file as specified by the user. If the maximum discard count is exceeded, the cleanup task terminates the REORG operation and marks all indexes as invalid. If versioning is enabled, a table and its indexes are restored to their previous state.

- If the INVALIDATE INDEX option is selected, any index that encounters an error while building is marked invalid.

- If the ABORT option is selected, the REORG operation will terminate and leave all indexes in an invalid state.

The successfully rebuilt indexes are marked valid and the status of the REORG operation is reported to the coordinator task.
**REORG Syntax**

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

```
REORG table_name

  SEGMENT ( segment_name )

INDEX ( index_name )

  SEGMENT ( segment_name )

EXCLUDE DEFERRED INDEXES

INCLUDE

ON DISCARD

  INVALIDATE INDEX

ABORT

DELETE ROW ( discardfile_clause )

DISCARDS n
```

*OPTIMIZE OFF*

*REFERENCE CHECKING ON OFF*

*RECALCULATE RANGES*

*ON*
REORG `table_name`
Specifies the table to be reorganized.

SEGMENT `segment_name`
Specifies the table segment(s) to be scanned for a partial REORG of an index, or the index segment(s) to be rebuilt by the REORG operation.

Table segments can be specified only if the table and all indexes specified in the index clause are segmented on the same column and if all the rows whose keys belong in the specified index segments are present in the specified data segments. When no segments are specified, the entire table will be scanned. Rows scanned that do not belong in any index segment being rebuilt will be ignored.

Index segments must be attached to the named index. They can be either online or offline, but all segments must be in the same online or offline state. When no segments are specified, all segments attached to that index will be rebuilt.

INDEX `index_name`
Specifies the index that is to be rebuilt. If this clause is not present, all non-deferred indexes defined on the table are rebuilt. The index name for a user-created index is specified in the CREATE INDEX statement. The name for a system-generated primary key index is the string "_PK_IDX" appended to the table name. For example, the primary index for the Market table is MARKET_PK_IDX.

[EXCLUDE, INCLUDE] DEFERRED INDEXES
Specifies whether deferred indexes (indexes that are not populated at the time of creation) should be included or excluded from the REORG operation. The default is EXCLUDE DEFERRED INDEXES. A deferred index cannot be subjected to a partial REORG operation.

RECALCULATE RANGES
Specifies that the ranges for any STAR index rebuilt with this REORG statement are to be recalculated to split index entries evenly among the segments for the index. Use this option when you have changed a MAXROWS PER SEGMENT or a MAXSEGMENTS value for a table that participates in the STAR index. If this option is included, at least one index in the index list must be a STAR index.

RECALCULATE RANGES cannot be specified in the REORG statement for a partial-index REORG. For information about segment ranges, refer to the SQL Reference Guide.

RECALCULATE RANGES cannot be specified when versioning is enabled.
OPTIMIZE ON, OFF
Specifies that the index or indexes are to be rebuilt in the OPTIMIZE mode. This option overrides the OPTIMIZE mode set in the rbw.config file. If this clause is not present in the REORG statement, the rbw.config file determines the default behavior. For more information on OPTIMIZE, refer to Chapter 3 of the Table Management Utility Reference Guide.

REFERENCE CHECKING ON, OFF
Specifies whether a referential integrity check should be conducted. If the option is set to ON, all foreign keys are checked. If the option is set to OFF, only those foreign keys that are part of a STAR index that is being rebuilt are checked. The default is ON.

If you have already ensured that the data has no referential integrity violations, referential integrity checking can be turned off while building a deferred index.

During a partial REORG operation, referential integrity will be checked only for those rows actually read from the table.

ON DISCARD
Indicates the action to be taken when data rows fail the referential integrity checks or contain duplicate index key values. The three options are DELETE ROW, INVALIDATE INDEX, and ABORT. The default is DELETE ROW.

INVALIDATE INDEX
Invalidates any unique index with a duplicate key or any STAR index that could not be created or rebuilt because of a referential integrity violation. Also invalidates any index that could not be created or rebuilt due to the operation running out of disk space. (The error can be corrected later by deleting a row, adding a missing primary key value to the referenced table, or dropping a foreign key constraint.) The rebuilding of other indexes will continue. If versioning is enabled, queries can access the previous version of the table and its indexes. If versioning is disabled, queries can access the table but cannot access the indexes being rebuilt.

ABORT
Aborts the REORG operation if an error occurs in any of the indexes. If versioning is enabled, the indexes are rolled back to their initial state. If versioning is not active, all indexes being rebuilt are marked invalid. Indexes not being rebuilt are not affected. If ABORT is selected and an index runs out of disk space, the REORG operation will terminate. If versioning is enabled, queries can access the table and its indexes. If versioning is disabled, queries can access the table but cannot access the indexes being rebuilt.
DELETE ROW
Deletes duplicate rows and rows with referential integrity failures from the table and all indexes. Rows with referential integrity violations are deleted from the table immediately. Rows with duplicate key values are not deleted until the end of the REORG after all table rows have been processed. If versioning is enabled, queries can access the table and its indexes. If versioning is disabled, queries cannot access the table or its indexes.

The DELETE ROW option cannot be used during a partial-index REORG operation. Hence, either the INVALIDATE INDEX option or the ABORT option must be selected.

discardfile clause
The discardfile clause specifies where to store duplicate rows or rows that fail the referential integrity check.

The following syntax diagram shows how to construct a discardfile clause:

```
DISCARDFILE 'filename'
|   |
|RI_DISCARDFILE 'filename'
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
|   |
RI_DISCARDFILE 'filename'
Name of the file to which to discard the records that violate referential integrity. This clause cannot be used on a table that does not reference other tables. This option can be used only when the ON DISCARD DELETE ROW option is specified.

The filename must satisfy operating-system file-specification conventions and must be enclosed in single quotes.

Note: The redbrick user must have write permission for the discard files.

RI_DISCARDFILE table_name 'filename'
A table name-filename pair that names a table referenced by a foreign key in the table being reorganized and a file in which to record the discarded rows that violate referential integrity with respect to the named table.

These name pairs provide a separate discard file for each named table. If a single record violates referential integrity with respect to multiple referenced tables, that record will be written to the file associated with each of those tables.

Multiple pairs can be specified. If some but not all referenced tables are listed here, records that violate referential integrity with respect to tables missing from the list are written either to the file following the OTHER keyword, or if that keyword is missing, to the standard discard file (following the DISCARDFILE keyword).

The filename(s) must satisfy operating-system file-specification conventions and must be enclosed in single quotes; pairs must be separated by commas.

Note: The redbrick user must have write permission for the discard files.

OTHER 'filename'
Specifies a file in which to discard any rows that violate referential integrity with respect to referenced tables not named in the table name-filename pairs. If a table name-filename pair list is present and this clause is omitted, then any records that violate referential integrity with respect to tables missing from the list are written to the standard discard file (following the DISCARDFILE keyword).

The filename must satisfy operating-system file-specification conventions and must be enclosed in single quotes.

Note: The redbrick user must have write permission for the discard files.
DISCARDS \( n \)

Specifies the maximum number of discarded rows allowed. When this number is exceeded, the REORG operation aborts. If the value specified for \( n \) count is 0, there is no limit on the number of discarded rows.

**Usage Notes**

**Referential Integrity**

Referential integrity is always preserved for databases, except in the following cases:

- When you use the OFFLINE OVERRIDE REFCHECK option in ALTER SEGMENT statements.
- When you use the CLEAR OVERRIDE REFCHECK option in ALTER SEGMENT statements.
- When you use the OVERRIDE REFCHECK option in SQL DELETE statements.
- When you use the ON DISCARD DELETE ROW option in REORG statements.
- When you use the REPLACE mode in a LOAD DATA statements on a table when rows in that table are referenced by another table.

In any of these cases, rows can be deleted from a referenced table in violation of referential integrity. Because delete operations performed by REORG do not cascade to referencing tables, if you reorganize a referenced table, you must also reorganize each table that references the reorganized table. To restore referential integrity, you must perform a REORG operation with the ON DISCARD DELETE ROW and the REFERENCE CHECKING ON options enabled for all tables that reference the table from which rows were deleted. The REORG operation will delete any rows that reference a deleted row, thus restoring referential integrity.

**Note:** Be sure that you perform the REORG operation on the referencing table, not the table from which the rows were deleted (the referenced table).
Example

This example illustrates how rows are deleted from referencing tables in a REORG operation. Assume the Fact1 table references the Dim1 table, which in turn references the Out1 table:

![Diagram showing the relationship between Out1, Dim1, and Fact1 tables]

After some rows are deleted from Out1, a REORG operation is performed on Dim1. Any rows in Dim1 that reference rows that were deleted from Out1 (that violate referential integrity) are deleted by the REORG operation to preserve referential integrity. However, rows in Fact1 that reference deleted rows in Dim1 are not deleted by the REORG operation on Dim1. To delete these rows and preserve referential integrity, you must also perform a REORG operation on Fact1.

Locking Behavior

The REORG operation initially places a read-lock on the database (if the RECALCULATE RANGES option is specified, the REORG operation write-locks the database). The lock on the database is released after the REORG operation locks the table that it is modifying.

If versioning is enabled, queries can access the previous version of the table and its indexes while the table is being reorganized. Versioning operates only in serial mode. For more information on versioning, refer to Chapter 2, “Working with a Versioned Database,” in this document.

Partial-Index REOG

The limitations of a partial-index REORG operation are as follows:

- Referential integrity is checked only for specified data segments of the table. Hence, referential integrity cannot be ensured for the entire table.
- The listed segments must all be either online or offline. If all the segments are online, an index that is being rebuilt by the REORG operation remains invisible to users until the rebuilding process has been completed. If those segments being reorganized are offline, the other segments that are online remain visible throughout the REORG operation.
• Only an index that was valid at the outset will be marked valid upon completion of the REORG operation. An invalid index will not be marked valid; to mark the index valid, you must rebuild the entire index.

• Rows will not be deleted from the table or from any index.

• A partial-index REORG operation cannot be performed on a DEFERRED index.

Online/Offline Operation

When the segments being reorganized are offline, the indexes will not be marked invalid. Instead, an internal flag will be set to indicate that the segment is being rebuilt. When this flag is set, the segment cannot be brought online. When the REORG operation completes successfully, the flag is reset. If the REORG is unsuccessful, the segment will remain inaccessible until it is successfully reorganized.

During a full-index REORG operation, all index segments must be online and will remain invisible to users until the rebuilding process has been completed.

Note: A segment marked “damaged” cannot be rebuilt by a REORG operation until the problem has been corrected.

Disk Space

An index that runs out of disk space will be marked invalid, but the excess rows will not be deleted from the table. If the DELETE ROW option or INVALIDATE INDEX option is specified, only the index that runs out of space is marked invalid and the process of building other indexes will continue. If the ABORT option is specified, the REORG operation will terminate immediately. If versioning is enabled, all REORG changes are rolled back to their initial state. If versioning is not active, all indexes being rebuilt are marked invalid.

Discardfile Format

Discarded rows are written in external format. They can be loaded into a table using a load script generated by an UNLOAD command. For more information, refer to “Unloading or Loading External-Format Data” on page 4-12 of the Table Management Utility Reference Guide.
SET Statements and Parameters to Control Behavior

The SET commands and the TUNE parameters control parallelism in the REORG operation. To specify parameters for a specific session, enter the SET statement in the TMU control file; for all sessions, edit the TUNE parameters in the rbw.config file.

SET TMU TASKS

The following commands control the number of input and index builder tasks during a REORG operation:

- SET TMU MAX TASKS which specifies an upper bound on the total number of tasks allocated for input tasks and index builder tasks. The total number of tasks specified by the MAX TASKS command must be at least two.
- SET TMU INPUT TASKS which specifies the number of tasks allocated to scan the target table. Because the number of conversion tasks is always identical to the number of INPUT tasks, this option also controls the number of conversion tasks.
- SET TMU INDEX TASKS which specifies the number of tasks allocated for indexes.

The SET TMU INPUT TASKS command and the SET TMU INDEX TASKS command should not be used in conjunction with the SET TMU MAX TASKS command.

Syntax

The following syntax diagram shows how to construct a SET TMU TASK statement or a TUNE TMU_TASKS rbw.config file parameter:

```plaintext
SET       TMU MAX TASKS  num_tasks ; 
           TMU INPUT TASKS 
           TMU INDEX TASKS 

TUNE       TMU_MAX_TASKS  num_tasks 
           TMU_INPUT_TASKS 
           TMU_INDEX_TASKS
```
**num_tasks**

Integer that indicates the number of tasks. If the `num_tasks` is set to 0, the REORG operation resets to its default behavior.

**Note:** The REORG operation might not allocate all the specified INPUT or INDEX tasks if the tasks are deemed excessive.

**Examples**

The following examples illustrate SET commands that can be used to change parameters for a specific session:

```plaintext
SET TMU MAX TASKS 5;
SET TMU INPUT TASKS 3;
SET TMU INDEX TASKS 2;
```

The following examples illustrate entries in the `rbw.config` file that apply to all sessions:

```plaintext
TUNE TMU_MAX_TASKS 5
TUNE TMU_INPUT_TASKS 3
TUNE TMU_INDEX TASKS 2
```

**Temporary Space Management**

If the OPTIMIZE option is ON, as data rows are read and indexed, intermediate index results are stored in memory until they reach a threshold value, at which point they are written (spilled) to disk. The following parameters control how much temporary space—both memory and disk—is used when the OPTIMIZE option is ON:

- **INDEX TEMPSPACE DIRECTORIES**, which specifies temporary space directories to be used by the TMU for index-building operations.
- **INDEX TEMPSPACE THRESHOLD**, which specifies the size at which index-building operations spill to disk.
- **INDEX TEMPSPACE MAXSPILLSIZE**, which specifies the maximum amount of temporary space on disk that can be consumed by index-building operations.
- **TEMPSPACE DUPLICATESPILLPERCENT**, which specifies the percentage of the INDEX TEMPSPACE MAXSPILLSIZE that can be used for recording duplicate rows.

**Note:** When the OPTIMIZE option is OFF, the entire INDEX TEMPSPACE MAXSPILLSIZE may be used for recording duplicate rows.
This section describes both the SET commands and the TUNE parameters in the \texttt{rbw.config} file that control temporary space allocation and management. For more information about temporary space management, which affects not only TMU and PTMU operations but also SQL DDL statements, refer to the \textit{Warehouse Administrator's Guide}.

\textbf{Syntax}

The following syntax diagram shows how to construct a SET INDEX TEMPSPACE statement or a TUNE INDEX\_TEMPSPACE parameter:

\begin{itemize}
  \item \textbf{DIRECTORY 'dir\_path', DIRECTORIES 'dir\_path', \ldots}
    Specifies a directory or a set of directories that are to be used for temporary files; \textit{dir\_path} must be a full pathname. To define a set of directories using entries in the \texttt{rbw.config} file, enter multiple lines. The order in which the directories are specified has no effect because the order in which they are used is random (determined internally) and no user control is possible.

  \begin{itemize}
    \item \textbf{UNIX}
      The default directory is \texttt{/tmp}.
    \item \textbf{Windows NT}
      The default directory is \texttt{c:\tmp}.
  \end{itemize}

  \item \textbf{THRESHOLD value}
    Specifies the amount of memory used before writing intermediate results from an index-building operation to disk. For operations involving multiple indexes, this threshold value is divided equally among the indexes being built.

  The size must be specified as kilobytes (K) or megabytes (M) by appending K or M to the number. Note that no space is allowed between the number and the unit identifier (K, M). For example: 1024K, 500M.
\end{itemize}
The threshold value must be specified before the corresponding MAXSPILLSIZE value is specified; it must precede the MAXSPILLSIZE entry in the rbw.config file.

A value of 0 causes files to be written to disk after the first 200 rows or index entries.

The default threshold for index-building operations is 10 megabytes (10M).

**MAXSPILLSIZE size**

Specifies the total maximum amount of temporary space per operation. For an operation involving multiple indexes, this space is divided equally among the indexes being built.

The size must be specified as kilobytes (K), megabytes (M), or gigabytes (G) by appending K, M, or G to the number. Note that no space is allowed between the number and the unit identifier (K, M, G). For example: 1024K, 500M, 8G.

The default MAXSPILLSIZE value is 1 gigabyte (1G). The maximum MAXSPILLSIZE value is 2047 gigabytes.

**DUPLICATESPILLPERCENT percent**

Specifies the percentage of the INDEX TEMPSPACE MAXSPILLSIZE used for the temporary storage of discarded duplicate rows. The percent is an integer between 0 and 100 (inclusive). The default is two percent.

**RESET**

Resets the index-building TEMPSPACE parameters to the values specified in the rbw.config file.

**Usage Notes**

In addition, use the following guidelines when setting index-building temporary space parameters:

- Always set the THRESHOLD value before setting the MAXSPILLSIZE value.
- Remember that the INDEX TEMPSPACE parameter settings in the rbw.config file affect not only TMU index-building operations but also SQL index-building operations.
Examples

The following examples illustrate SET commands that can be used to change parameters for a specific session:

```
SET INDEX TEMPSPACE THRESHOLD 2M;
SET INDEX TEMPSPACE MAXSPILLSIZE 3G;
SET INDEX TEMPSPACE DUPLICATESPILLPERCENT 5;
```

UNIX
```
SET INDEX TEMPSPACE DIRECTORIES '/disk1/itemp',
    '/disk2/itemp';
```

Windows NT
```
SET INDEX TEMPSPACE DIRECTORIES 'd:\itemp', 'e:\itemp';
```

The following example illustrates how to reset the INDEX_TEMPSPACE parameters to the values specified in the rbw.config file:

```
SET INDEX TEMPSPACE RESET;
```

The following examples illustrate entries in the rbw.config file that apply to all sessions:

```
TUNE INDEX_TEMPSPACE THRESHOLD 20M
TUNE INDEX_TEMPSPACE_MAXSPILLSIZE 8G
TUNE INDEX_TEMPSPACE_DUPLICATESPILLPERCENT 5
```

UNIX
```
TUNE INDEX_TEMPSPACE_DIRECTORY /disk1/itemp
TUNE INDEX_TEMPSPACE_DIRECTORY /disk2/itemp
TUNE INDEX_TEMPSPACE_DIRECTORY /disk3/itemp
```

Windows NT
```
TUNE INDEX_TEMPSPACE_DIRECTORY d:\itemp
TUNE INDEX_TEMPSPACE_DIRECTORY e:\itemp
TUNE INDEX_TEMPSPACE_DIRECTORY f:\itemp0
```
**Understanding Versioned LOAD Operations**

When deciding whether to run a LOAD operation in versioned mode, you must decide if the cost of the versioned operation is worth the benefit gained from it. This section discusses these costs and benefits. Every situation is unique, and you must perform your own analysis to determine the cost/benefit trade-offs in your data warehouse environment.

**Benefits**

The main benefits of versioned LOAD operations compared with blocking LOAD operations are:

- Increased availability of the database.
- Increased recoverability of the database.
- Increased opportunity for loading data.

Increased availability results because the database is available to query while the LOAD operation is taking place. This allows users to query the database as usual while the load is taking place.

Increased recoverability results with versioned operations because the existing database files do not change until the operation is complete. All changes to existing database blocks are done in the version log. This has the benefit of keeping the old version of the database completely consistent during the transaction. If a catastrophic failure occurs (for example, UNIX `kill -9`, Windows NT `End Task`, or a power failure), the database will not be damaged after recovery from the failure. The system automatically notices that the transaction never completed and rolls the database back to its original state.

Increased opportunity for loading data results because the effective “load window,” or the time available to modify the database, is increased because the database can be available for query operations while you are modifying it. Increasing the time available for LOAD operations does not compromise query performance.

If availability and recoverability are not important to your system during your LOAD operations, then you probably do not need to run the TMU with versioning enabled. If these things are important, then you must analyze the costs.
Costs

When analyzing the costs of running versioned LOAD operations, consider the following:

- The version log
- The effect on load performance

Version Log

The version log itself has some costs associated with it. The most obvious is the disk space it requires. To ensure good query performance, the version log should reside on its own storage subsystem, preferably with several dedicated disk drives. These are extra budget items and system resources that you must allocate.

Additionally, consider the cost to administer the version log. Although it is not very complex, it does add to the overall complexity of your system. Also, it uses resources on your system.

For more information about setting up the version log, refer to “Creating and Sizing the Version Log” on page 2-10.

Effect on Load Performance

The main factor affecting load performance of a versioned TMU operation is the number of existing database blocks that change. If there are primarily new blocks and few changed blocks, then the performance impact is negligible.

Because the TMU updates everything in the database necessary to keep it consistent, when you analyze the number of changed blocks from your TMU operation, consider all of the following:

- Existing rows in tables that change.
- Any changes to referenced tables.
- Any changes to indexes affected by the operation.
Example 1

Consider a simple star schema with a single fact table that references three dimension tables as in the following figure:

Assume that the fact table (Fact1) is segmented by month (these values come from the the Period table). Assume also that there is a single STARindex created on the three foreign keys of the Fact1 table, and that the index is also segmented by month. The only other indexes in this database are the primary key B-TREE indexes of the three dimension tables.

Because of the segmentation of the fact table and the STARindex, when a new month of data is loaded into this database, all the table data goes into a new Fact1 table segment and all of the corresponding index data goes into a new STARindex segment. Therefore, there is no changed data when a new month of data is loaded.

This scenario has a very small cost for running in versioned mode because there is no table or index data written to the version log; the data is all written directly to the database files because it did not previously exist in the database.
Example 2

Consider the same scenario as in the previous example, but with three TARGET indexes on the fact table: one for each foreign key column. Unlike the STAR index, these TARGET indexes are not segmented like the data and therefore contain data for all time periods in the database. When you load the new month of data, many blocks in the TARGET indexes change. Each of those changed blocks are written to the version log. Because of the increased I/O of reading from the database files and writing to the version log files, this operation might not perform as well as a non-versioned operation. There is also some cost after the transaction is committed; the vacuum cleaner must write the new blocks back to the database files and then clean out the version log.

You must weigh this cost with the benefits of doing a versioned load. If your users demand 24 hour availability, 7 days a week, it might be worth the extra overhead. However, if no one uses the system at night and the LOAD operations easily complete during that downtime, you might want to lower your overhead and perform blocking LOAD operations.
Enabling Versioning for the TMU

A TMU operation can be run as a versioned transaction on databases where versioning is enabled. The SET TMU VERSIONING statement and OPTION TMU_VERSIONING rbw.config file parameter specify whether TMU operations are executed as versioned or blocking transactions. A versioned transaction allows query (read) operations to occur on the previously committed version of the database while a new version is being written. A blocking transaction locks all of the tables involved and does not allow query (read) operations to begin until the transaction has completed.

The following TMU operations can run as versioned transactions:
- Online LOAD operations
- REORG
- SYNCH

When TMU VERSIONING is set to ON, all TMU operations are run as versioned transactions. When TMU VERSIONING is set to OFF, all TMU operations are run as blocking transactions. The default value is OFF.

Note: Versioning is not supported for the PTMU, therefore this option does not affect PTMU operations.

To control TMU versioning for a single session, enter a SET statement in the TMU control file; for all sessions, edit the OPTION parameter in the rbw.config file. The syntax is as follows:

```
SET TMU VERSIONING OFF ;
OPTION TMU_VERSIONING OFF
```

```
Setting Versioned Commit Intervals

Versioned databases allow you to build applications that modify a database while it is still available to other users for query operations. That way, as new transactions commit new data to the database, the new data is then available for querying. In the default mode, a versioned transaction completes an entire load operation before the commit operation occurs.

If you want commit operations to occur more frequently than once per load operation, the TMU supports two commit intervals—SET TMU COMMIT RECORD INTERVAL and SET TMU COMMIT TIME INTERVAL. These commit intervals allow you to specify how many records to load between each commit and to specify how much time elapses between each commit.

Commit intervals are useful if you have a steady stream of data to load into a database and want that data to become visible to users every so often, without having to stop and restart the load. For example, consider stock ticker data that comes in constantly. You might choose to update your database every 15 minutes with the latest data. Or consider retail transactions that come in from all the grocery stores corporate-wide, and you want to update your database every 5,000 transactions. These types of trickle feed applications allow users to analyze nearly real time data.

Additionally, you can use the commit interval as a checkpoint mechanism for normal versioned loads. Each time a load commits after a given commit interval, the changes become visible to new users. Also, if the load operation fails during the next interval, it rolls back to the latest committed version. Therefore, if you are loading 1,000,000 rows and set the TMU COMMIT RECORD INTERVAL at 100,000 records, and if the load then fails after loading 999,000 records, then the load rolls back to the last commit, which would be after 900,000 records. If you were performing a normal versioned load (no commit intervals set) that failed in such a manner, the transaction would roll back to the state before any records were loaded.

When setting versioned commit intervals, consider the total number of commit operations that a load operation will create and make sure it is reasonable with respect to the maximum number of revisions configured (ALTER DATABASE CREATE VERSION LOG IN...MAXREVISIONS) and the size of the version log. If the operation creates too many versions or if it fills up the version log, the operation will fail. This only becomes a problem if there are many commit intervals very close together and/or there are read transactions reading the version log, thus not allowing the vacuum cleaner to move the new blocks from the version log to the database files.
**SET TMU COMMIT RECORD INTERVAL**

The TMU COMMIT RECORD INTERVAL command specifies the number of records to load into a table between each commit operation.

To control TMU COMMIT RECORD INTERVAL for a single session, enter a SET statement in the TMU control file; for all sessions, edit the OPTION parameter in the `rbw.config` file. The syntax is as follows:

```
 SET TMU COMMIT RECORD INTERVAL record_count [ OFF ];
```

```
 OPTION TMU_COMMIT_RECORD_INTERVAL record_count [ OFF ];
```

*record_count*

Number of records, indicated by a positive integer, after which the load commits. After each commit, the load proceeds until the `record_count` is reached again and then another commit occurs. Setting the value to 0 disables the record interval, and is the same as specifying OFF (the default).

**Usage Notes**

After each record interval commit operation, an information message is issued indicating the number of rows that have been loaded.

When performing a versioned load with OPTIMIZE ON, each commit operation requires an index merge to occur. Depending on the values of the INDEX TEMPSPACE parameters, this might result in more merges during the index building phases and therefore less efficient indexes than if the load operation completed as a single transaction.

The TMU COMMIT RECORD INTERVAL command is only valid with versioned load operations; blocking transactions ignore this command.

If used in conjunction with the TMU COMMIT TIME INTERVAL command, a commit is performed when either condition is met. After the commit occurs, both counters are reset and loading continues until the next interval (either TIME or RECORD) occurs.

If the current transaction aborts, the database rolls back to the state of the last completed commit interval.
**SET TMU COMMIT TIME INTERVAL**

The TMU COMMIT TIME INTERVAL command specifies the amount of time, in minutes, to load data into a table before each commit operation.

To control TMU COMMIT TIME INTERVAL for a single session, enter a SET statement in the TMU control file; for all sessions, edit the OPTION parameter in the rbw.config file. The syntax is as follows:

```
| SET TMU COMMIT TIME INTERVAL minutes	 OFF |
```

```
| OPTION TMU_COMMIT_TIME_INTERVAL minutes	 OFF |
```

*minutes*

A positive integer specifying the number of minutes after which a load operation commits. After each commit, the load proceeds until the number of minutes is reached again and then another commit occurs. Setting the value to 0 disables the time interval, and is the same as specifying OFF (the default).

The time specified in the TMU COMMIT TIME INTERVAL command is the time between when the transaction begins and when the commit operation begins. The commit operation might take some time to complete.

**Usage Notes**

After each time interval commit operation, an information message is issued indicating the number of rows that have been loaded.

When performing a versioned load with OPTIMIZE ON, each commit operation requires an index merge to occur. Depending on the values of the INDEX TEMPSPACE parameters, this might result in more merges during the index building phases and therefore less efficient indexes than if the load operation completed as a single transaction.

The TMU COMMIT TIME INTERVAL command is only valid with versioned load operations; blocking transactions ignore this command.

If used in conjunction with the TMU COMMIT RECORD INTERVAL command, a commit is performed when either condition is met. After the commit occurs, both counters are reset and loading continues until the next interval (either TIME or RECORD) occurs.

If the current transaction aborts, the database rolls back to the state of the last completed commit interval.
### Example

This example shows a control file with a LOAD DATA statement for the Sales table in a versioned Aroma database where the commit interval is set for every 20,000 records. The following is the text of the TMU control file:

```plaintext
set tmu versioning on;
set tmu commit record interval 20000;
load data inputfile '$RB_CONFIG/sample_input/aroma_sales.txt'
recordlen 86
insert
into table sales (perkey position(2) integer external(11) nullif(1)='%',
classkey position(14) integer external(11) nullif(13)='%',
pronkkey position(26) integer external(11) nullif(25)='%',
srnkkey position(38) integer external(11) nullif(37)='%',
pronkkey position(50) integer external(11) nullif(49)='%',
quantity position(62) integer external(11) nullif(61)='%',
dollars position(74) decimal external(12)
radix point '.' nullif(73)='%');
```

The following is the output of this TMU operation:

```
% $RB_CONFIG/bin/rb_tmu -d aroma sales.tmu system manager
(C) Copyright 1991-1998 Red Brick Systems, Inc., Los Gatos, California, USA
All rights reserved.
Version 5.1.5(92836)TST
** STATISTICS ** (500) Time = 00:00:00.01 cp time, 00:00:00.00 time, Logical IO count=0
** INFORMATION ** (366) Loading table SALES.
** INFORMATION ** (8555) Data-loading mode is INSERT.
** INFORMATION ** (8707) Versioning is active.
** INFORMATION ** (8710) Interval commit set to 20000 records.
** INFORMATION ** (352) Row 3 of index SALES_STAR_IDX is out of sequence. Switching
to standard optimized index building. Loading continues...
** INFORMATION ** (8708) Performing interval commit...
** INFORMATION ** (513) Starting merge phase of index building SALES_STAR_IDX.
** INFORMATION ** (367) Rows: 20000 inserted. 0 updated. 0 discarded. 0 skipped.
** INFORMATION ** (8709) Commit complete. Loading continues...
** INFORMATION ** (8708) Performing interval commit...
** INFORMATION ** (513) Starting merge phase of index building SALES_STAR_IDX.
** INFORMATION ** (367) Rows: 40000 inserted. 0 updated. 0 discarded. 0 skipped.
** INFORMATION ** (8709) Commit complete. Loading continues...
** INFORMATION ** (8708) Performing interval commit...
** INFORMATION ** (513) Starting merge phase of index building SALES_STAR_IDX.
** INFORMATION ** (367) Rows: 60000 inserted. 0 updated. 0 discarded. 0 skipped.
** INFORMATION ** (8709) Commit complete. Loading continues...
** INFORMATION ** (315) Finished file /redbrick/sample_input/aroma_sales.txt. 69941 rows read from this file.
** INFORMATION ** (513) Starting merge phase of index building SALES_STAR_IDX.
** INFORMATION ** (367) Rows: 69941 inserted. 0 updated. 0 discarded. 0 skipped.
** STATISTICS ** (500) Time = 00:00:29.21 cp time, 00:00:33.73 time, Logical IO count=1044
```

Notice that this operation performs four commit operations—one after inserting 20,000, 40,000, 60,000, and 69,941 records.
**Miscellaneous Changes and Additions**

This section lists some miscellaneous changes and additions to the *Table Management Utility Reference Guide*. The following topics are included:

- SET TMU BUFFERS Command
- EBCDIC to ASCII Conversion
- TMU Aggregate Operators
- Datetime Fieldtypes
- Fieldtype Conversions
- Locales

**SET TMU BUFFERS Command**

Note the following change to the syntax description on page 2-12 of the *Table Management Utility Reference Guide*.

The maximum value for the SET TMU BUFFERS command (TUNE TMU_BUFFERS parameter) has been increased from 8,208 blocks to 65,534 blocks. If you increase the value significantly, monitor performance carefully because interactions between the operating system and the database implementation sometimes cause a very large buffer cache to decrease rather than increase performance.

**EBCDIC to ASCII Conversion**

Add the following usage notes to the section on the Format clause of the LOAD DATA statement in Chapter 3 of the *Table Management Utility Reference Guide*.

If you are using the FORMAT IBM option to load data, note the following restrictions:

- Only single-byte EBCDIC code pages are supported.
- Conversion to ASCII is limited to EBCDIC characters within the syntactic character set (CS 640). These characters are identified in the following table.
- In addition to CS 640, the TMU converts the EBCDIC exclamation point (!) based on its definition in IBMUSCanada code page 037 (Hex 5A, Dec 90).
- All other code points not addressed here have unsupported TMU mappings to ASCII; attempting to map such characters yields unpredictable results.
IBM Syntactic Character Set (CS 640)

<table>
<thead>
<tr>
<th>Character</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Z</td>
<td>Left parenthesis</td>
</tr>
<tr>
<td>a-z</td>
<td>Right parenthesis</td>
</tr>
<tr>
<td>0-9</td>
<td>Equal</td>
</tr>
<tr>
<td>(</td>
<td>Greater than</td>
</tr>
<tr>
<td>.</td>
<td>Asterisk</td>
</tr>
<tr>
<td>&quot;</td>
<td>Question mark</td>
</tr>
<tr>
<td>/</td>
<td>Plus</td>
</tr>
<tr>
<td>%</td>
<td>Underscore</td>
</tr>
<tr>
<td>:</td>
<td>Comma</td>
</tr>
<tr>
<td>&amp;</td>
<td>Hyphen</td>
</tr>
<tr>
<td>;</td>
<td>Space</td>
</tr>
<tr>
<td>'</td>
<td>(Left parenthesis</td>
</tr>
<tr>
<td>&lt;</td>
<td>(Right parenthesis</td>
</tr>
</tbody>
</table>

Two Approaches to Loading EBCDIC Data

There are two ways to load EBCDIC data with the TMU:

- Use the FORMAT IBM keywords in the Format clause.
- Specify an IBM character set in the NLS_LOCALE clause.

The FORMAT IBM and NLS_LOCALE specifications are mutually exclusive. You can use only one of these specifications in each TMU control file.

Users who are certain that they are loading only characters that comply with CS 640 can use either the FORMAT IBM or the NLS_LOCALE specification, but the FORMAT IBM approach yields higher performance. However, users who are unsure whether their input data complies with CS 640 should use the NLS_LOCALE clause to select an EBCDIC character set that is fully compatible with the specified language. Although load performance might not be optimal, this approach ensures the integrity of both the loaded data and database objects (such as indexes) that are built based on that data.

For a complete list of supported character sets for each language, refer to page 6-10.
**TMU Aggregate Operators**

In addition to the ADD, SUBTRACT, MIN, and MAX aggregate operators described on page 3-59 of the *Table Management Utility Reference Guide*, the following new aggregate operators have been added to optimize the AGGREGATE function in cases where there is a NULL value in an input record or a table:

**MIN_NONULL**
Keeps the smaller of the value in the input record and the value in the corresponding table column. If the value in the input record is NULL, the value in the table column will be retained. If the value in the table column is NULL, it will be replaced by the value in the input record.

**MAX_NONULL**
Keeps the larger of the value in the input record and the value in the corresponding table column. If the value in the input record is NULL, the value in the table column will be retained. If the value in the table column is NULL, it will be replaced by the value in the input record.

**ADD_NONULL**
Adds the value in the input record to the corresponding table column. When the value in the input record is NULL, the value in the table column remains unchanged. A NULL value in the table column will be treated as 0.

**SUBTRACT_NONULL**
Subtracts the value in the input record from the value in the corresponding table column. When the value in the input record is NULL, the value in the table column remains unchanged. A NULL value in the table column will be treated as 0.
Datetime Fieldtypes

Replace the table on page 3-92 of the Table Management Utility Reference Guide with the following table.

The TMU converts this string into the Metaphor DIS date format and stores it as an integer when loading a table. The format must be one of the following:

<table>
<thead>
<tr>
<th>Format</th>
<th>Example: April 10, 1996</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYJJJ or</td>
<td>96/100</td>
</tr>
<tr>
<td>YYYYJJJ</td>
<td>1996/100</td>
</tr>
<tr>
<td>YYMD or</td>
<td>960410</td>
</tr>
<tr>
<td>YYYYMD</td>
<td>1996/4/10</td>
</tr>
<tr>
<td>MDYY or</td>
<td>4/10/96</td>
</tr>
<tr>
<td>MDYYYY</td>
<td>04101996</td>
</tr>
<tr>
<td>DMYY or</td>
<td>10/4/96</td>
</tr>
<tr>
<td>DMYYYYY</td>
<td>10041996</td>
</tr>
</tbody>
</table>

where:

D One or two digits specifying the day of the month.

JII Three digits specifying day of the year, in Julian format.

M One or two digits specifying the month of the year.

YY and YYYY Two and four digits respectively specifying a year. A two-digit year \( nn \) is interpreted as 19\( nn \).
Fieldtype Conversions

The table on page 3-111 of the Table Management Utility Reference Guide, which defines how blanks and empty fields in CHARACTER or NUMERIC EXTERNAL fields are mapped to column values, should be corrected as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Column Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC EXTERNAL field with blanks</td>
<td>Default column value</td>
</tr>
<tr>
<td>Empty NUMERIC EXTERNAL field</td>
<td>Default column value</td>
</tr>
<tr>
<td>CHARACTER field with blanks</td>
<td>Blank</td>
</tr>
<tr>
<td>Empty CHARACTER field</td>
<td>Default column value</td>
</tr>
</tbody>
</table>

Note the following behavior regarding empty (NULL) fields.

- If an empty field is loaded into a column defined as NOT NULL, the default column value (not a NULL indicator) is loaded into that column.
- If a column is defined as NOT NULL DEFAULT NULL, and an empty field for that column is encountered, the load operation terminates.

Locales

Replace Appendix C of the Table Management Utility Reference Guide with the expanded list of locale components listed in “Locales” on page 6-10 of this document.
This chapter contains additions and changes to the *Installation and Configuration Guide*. The following topics are included:

- Kernel Parameters
- New Files
- Locales
- Required Disk Space (UNIX Platforms)
- CD-ROM Mount Command (NCR Platform Only)
Kernel Parameters

Version 5.1.5 has increased shared memory and semaphore requirements compared with earlier releases of Red Brick Warehouse. Consequently, on some UNIX platforms, some kernel parameter values must be changed. After changing kernel parameters, a system reboot is typically required for the changes to take effect. For information about modifying kernel parameters, refer to your operating system documentation. This section lists changes to the “Kernel Parameters” sections of the Installation and Configuration Guide for UNIX Platforms.

Digital AlphaServer

The following table lists new kernel parameter recommendations for the Digital AlphaServer platform, and replaces the values for the same parameters on page D-3 of the V5.1 Installation and Configuration Guide for UNIX Platforms.

The recommendations for all parameters that do not appear in the following table remain unchanged.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_max</td>
<td>20 + 8 * MAXUSERS + MAX (load_processes, 2 * max_rbw_users + max_parallel_tasks) + 2 * max_active_databases</td>
</tr>
<tr>
<td>sem-mni</td>
<td>MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 30</td>
</tr>
<tr>
<td>sem-mns</td>
<td>3 * MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 60</td>
</tr>
<tr>
<td>shm-max</td>
<td>If you are not using versioned databases, use 16777216 Otherwise, use: MAX (16777216, [858636 + max_active_revision * 272 + sem-mni * 48 + size_of_version_log * 4])</td>
</tr>
<tr>
<td>shm-mni</td>
<td>If max_parallel_tasks = 0, use: (100 + max_active_databases) Otherwise, use: (100 + max_rbw_users + max_parallel_tasks * 2 + max_active_databases)</td>
</tr>
</tbody>
</table>

Note: The sem-mni, shm-max, shm-mni, msg-mnb, msg-tql, and msg-mni parameters need to be set in the /etc/sysconfigtab file.
where:

- **max_active_databases**
  Maximum number of active databases as specified by the MAX_ACTIVE_DATABASES `rbw.config` file parameter (the default is 30).

- **max_rbw_users**
  Maximum number of users licensed for your warehouse installation.

- **max_parallel_tasks**
  TOTALQUERYPROCS from the `rbw.config` file. If not specified, use 0.

- **size_of_version_log**
  The size, in kilobytes, of version log for your largest database.

- **max_active_revision**
  Maximum number of active revisions for your largest database (the default is 500, and it can be specified with the MAXREVISIONS specification of an ALTER DATABASE CREATE VERSION LOG statement).

- **max_query_tasks**
  QUERYPROCS from the `rbw.config` file (or the largest value used in any SET QUERYPROCS command if larger). If not specified, use 0.

- **load_processes**
  If no parallel load or parallel REORG operations, use 0. Otherwise, use:
  
  \[ 3 + (3 \times \text{num_cpus}) + \text{max_nuniq_idx} \]

- **max_nuniq_idx**
  Maximum number of non-unique indexes on any single table to be loaded by a parallel load.

- **num_cpus**
  The number of CPUs on the machine.
HP 9000 Computer

The following table lists new kernel parameter recommendations for the HP 9000 platform, and replaces the values for the same parameters on page E-3 of the V5.1 Installation and Configuration Guide for UNIX Platforms.

The recommendations for all parameters that do not appear in the following table remain unchanged.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>semmni</td>
<td>( \text{MAX} (\text{max}<em>\text{rbw}</em>\text{users}, \text{max}<em>\text{parallel}</em>\text{tasks}) + (20 \times \text{num}_\text{cpus}) + 30 )</td>
</tr>
<tr>
<td>semms</td>
<td>( 3 \times \text{MAX} (\text{max}<em>\text{rbw}</em>\text{users}, \text{max}<em>\text{parallel}</em>\text{tasks}) + (20 \times \text{num}_\text{cpus}) + 60 )</td>
</tr>
<tr>
<td>nproc</td>
<td>( 30 + 8 \times \text{MAXUSERS} + \text{MAX} (\text{load}<em>\text{processes}, 2 \times \text{max}</em>\text{rbw}<em>\text{users} + \text{max}</em>\text{parallel}<em>\text{tasks}) + 2 \times \text{max}</em>\text{active}_\text{databases} )</td>
</tr>
<tr>
<td>shmmax</td>
<td>If you are not using versioned databases, use 67108864. Otherwise, use: ( \text{MAX} (67108864, [1120904 + \text{max}<em>\text{active}</em>\text{revision} \times 272 + \text{semmni} \times 48 + \text{size}<em>\text{of}</em>\text{version}_\text{log} \times 4]) )</td>
</tr>
<tr>
<td>shmmni</td>
<td>If ( \text{max}<em>\text{parallel}</em>\text{tasks} = 0 ), use ( 100 + \text{max}<em>\text{active}</em>\text{databases} ). Otherwise, use ( \text{MIN} (1024, \text{max}<em>\text{rbw}</em>\text{users} + 100 + \text{max}<em>\text{parallel}</em>\text{tasks} \times 2) + \text{max}<em>\text{active}</em>\text{databases} )</td>
</tr>
</tbody>
</table>

where:

- \( \text{max}_\text{active}_\text{databases} \)
  Maximum number of active databases as specified by the \( \text{MAX}_\text{ACTIVE}_\text{DATABASES} \) \text{rbw.config} file parameter (the default is 30).

- \( \text{max}_\text{rbw}_\text{users} \)
  Maximum number of users licensed for your warehouse installation.

- \( \text{max}_\text{PSUs} \)
  Maximum number of PSUs (files) that you expect to assign to a database. For help in determining this number, refer to the Warehouse Administrator’s Guide.
**max_parallel_tasks**
TOTALQUERYPROCS from the rbw.config file. If not specified, use 0.

**size_of_version_log**
The size, in kilobytes, of version log for your largest database.

**max_active_revision**
Maximum number of active revisions for your largest database (the default is 500, and it can be specified with the MAXREVISIONS specification of an ALTER DATABASE CREATE VERSION LOG statement).

**max_query_tasks**
QUERYPROCS from the rbw.config file (or the largest value used in any SET QUERYPROCS command if larger). If not specified, use 0.

**load_processes**
If no parallel load or parallel REORG operations, use 0.
Otherwise, use:
3 + (3 * num_cpus) + max_nuniq_idx

**max_nuniq_idx**
Maximum number of non-unique indexes on any single table to be loaded by a parallel load.

**num_cpus**
The number of CPUs on the machine.
**Sun SPARC-Based Systems**

The following table lists new kernel parameter recommendations for the Sun SPARC platform, and replaces the values for the same parameters on page F-3 of the V5.1 *Installation and Configuration Guide for UNIX Platforms*.

The recommendations for all parameters that do not appear in the following table remain unchanged.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_nprocs</td>
<td>100 + MAX (load_processes, 2 * max_rbw_users + max_parallel_tasks) + 2 * max_active_databases</td>
</tr>
<tr>
<td>semsys:seminfo_semmni</td>
<td>MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 30</td>
</tr>
<tr>
<td>semsys:seminfo_semmns</td>
<td>3 * MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 60</td>
</tr>
<tr>
<td>shmsys:shminfo_shmmax</td>
<td>If you are not using versioned databases, use 16777216</td>
</tr>
<tr>
<td></td>
<td>Otherwise, use: MAX (16777216, [793072 + max_active_revision * 272 + seminfo_semmni * 48 + size_of_version_log * 4])</td>
</tr>
<tr>
<td>shmsys:shminfo_shmmni</td>
<td>If max_parallel_tasks = 0, use (100 + max_active_databases).</td>
</tr>
<tr>
<td></td>
<td>Otherwise, use: (100 + max_rbw_users + max_parallel_tasks * 2 + max_active_databases)</td>
</tr>
</tbody>
</table>

where:

- **max_active_databases**
  
  Maximum number of active databases as specified by the MAX_ACTIVE_DATABASES rbw.config file parameter (the default is 30).

- **max_rbw_users**
  
  Maximum number of users licensed for your warehouse installation.

- **max_parallel_tasks**
  
  TOTALQUERYPROCS from the rbw.config file. If not specified, use 0.
size_of_version_log
   The size, in kilobytes, of version log for your largest database.

max_active_revision
   Maximum number of active revisions for your largest database
   (the default is 500, and it can be specified with the MAXREVISIONS
   specification of an ALTER DATABASE CREATE VERSION LOG
   statement).

max_query_tasks
   QUERYPROCS from the rbw.config file (or the largest value used in
   any SET QUERYPROCS command if larger). If not specified, use 0.

load_processes
   If no parallel load or parallel REORG operations, use 0.
   Otherwise, use:
   \[ 3 + (3 \times \text{num_cpus}) + \text{max_nuniq_idx} \]

max_nuniq_idx
   Maximum number of non-unique indexes on any single table to
   be loaded by a parallel load.

num_cpus
   The number of CPUs on the machine.
Silicon Graphics Servers

The following table lists new kernel parameter recommendations for the Silicon Graphics Server platform, and replaces the values for the same parameters on page G-2 of the V5.1 Installation and Configuration Guide for UNIX Platforms.

The recommendations for all parameters that do not appear in the following table remain unchanged.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>semmni</td>
<td>MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 30</td>
</tr>
<tr>
<td>semmns</td>
<td>3 * MAX (max_rbw_users, max_parallel_tasks) + (20 * num_cpus) + 60</td>
</tr>
<tr>
<td>shmmni</td>
<td>If max_parallel_tasks = 0, use (400 + max_active_databases) Otherwise, use MIN (1000, max_rbw_users + 400 + max_parallel_tasks * 2) + max_active_databases</td>
</tr>
</tbody>
</table>

where:

- **max_active_databases**
  Maximum number of active databases as specified by the MAX_ACTIVE_DATABASES rbw.config file parameter (the default is 30).

- **max_rbw_users**
  Maximum number of users licensed for your warehouse installation.

- **max_parallel_tasks**
  TOTALQUERYPROCS from the rbw.config file. If not specified, use 0.

- **num_cpus**
  The number of CPUs on the machine.
New Files

This section lists new files for this release.

UNIX Systems

This section supplements Appendix A of the Installation and Configuration Guide for UNIX Platforms and lists new files that exist in Red Brick Warehouse beginning with Version 5.1.5.

The following table describes the new files in Version 5.1.5 in the redbrick directory.

New Files Under redbrick Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.RB_HOST.semaphores</td>
<td>Contains a list of semaphores used by a Red Brick Warehouse installation, where RB_HOST is the value of your RB_HOST environment variable.</td>
</tr>
<tr>
<td>.RB_HOST.sharedmemory</td>
<td>Contains a list of shared memory keys used by a Red Brick Warehouse installation, where RB_HOST is the value of your RB_HOST environment variable.</td>
</tr>
</tbody>
</table>

Windows NT Systems

This section supplements Appendix A of the Installation and Configuration Guide for Windows NT Platforms and lists the new rbshow.exe file that exists in Red Brick Warehouse for Windows NT systems.

Directories and Files Under redbrick Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN\</td>
<td>Directory for warehouse server binary files and scripts.</td>
</tr>
<tr>
<td>rbshow.exe</td>
<td>Command-line program that shows activity on a Red Brick Warehouse system. Type rbshow at the Windows NT prompt for the syntax.</td>
</tr>
</tbody>
</table>
Replace the table in Appendix B, “Recommended Locale Specifications,” with the following expanded list of locale components. (The same changes apply to the lists of locales in Appendix C of the Table Management Utility Reference Guide and Appendix C of the RISQL Entry Tool and RISQL Reporter User’s Guide.)

<table>
<thead>
<tr>
<th>Language</th>
<th>Territory</th>
<th>Code Page (Character Set)</th>
<th>Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>UnitedStates</td>
<td>US-ASCII Latin1, MS1252, UTF-8, IBM037, IBM285</td>
<td>Binary, Default</td>
</tr>
<tr>
<td></td>
<td>Australia</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Canada</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>South Africa</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UnitedKingdom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German</td>
<td>Germany</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8, IBM273</td>
<td>Default, Binary</td>
</tr>
<tr>
<td></td>
<td>Austria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>German-Switzerland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>France</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8, IBM297</td>
<td>Default, Binary</td>
</tr>
<tr>
<td></td>
<td>French-Belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>French-Canada</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>French-Switzerland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>Spain</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8</td>
<td>Spanish, TraditionalSpanish, Binary</td>
</tr>
<tr>
<td></td>
<td>Argentina</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chile</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mexico</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italian</td>
<td>Italy</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8, IBM280</td>
<td>Default, Binary</td>
</tr>
<tr>
<td></td>
<td>Italian-Switzerland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portuguese</td>
<td>Portugal</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8, IBM037</td>
<td>Default, Binary</td>
</tr>
<tr>
<td></td>
<td>Brazil</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Territory</td>
<td>Code Page (Character Set)</td>
<td>Sort</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>---------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Norwegian</td>
<td>Norway</td>
<td>Latin1</td>
<td>Danish</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS1252</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO-8859-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td>Swedish</td>
<td>Sweden</td>
<td>Latin1</td>
<td>Swedish</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS1252</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO-8859-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td>Danish</td>
<td>Denmark</td>
<td>Latin1</td>
<td>Danish</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS1252</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO-8859-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td>Finnish</td>
<td>Finland</td>
<td>Latin1</td>
<td>Finnish</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS1252</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO-8859-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td>Japanese</td>
<td>Japan</td>
<td>JapanEUC</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS932</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM930</td>
<td></td>
</tr>
<tr>
<td>CanadianFrench</td>
<td>French-Canada</td>
<td>Latin1</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS1252</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISO-8859-9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM297</td>
<td></td>
</tr>
<tr>
<td>TraditionalChinese</td>
<td>Taiwan</td>
<td>EUC-TW</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS950</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM937</td>
<td></td>
</tr>
<tr>
<td>SimplifiedChinese</td>
<td>China</td>
<td>MS936</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM935</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Territory</td>
<td>Code Page (Character Set)</td>
<td>Sort</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>---------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Albanian</td>
<td>Albania</td>
<td>ISO-8859-2, MS1250, Latin1, MS1252, ISO-8859-9, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Arabic</td>
<td>Saudi Arabia</td>
<td>ISO-8859-6, MS1256, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Bulgaria</td>
<td>ISO-8859-5, MS1251, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Byelorussian</td>
<td>Belarus</td>
<td>ISO-8859-5, MS1251, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Catalan</td>
<td>Catalonia</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Croatian</td>
<td>Croatia</td>
<td>ISO-8859-2, MS1250, UTF-8</td>
<td>Croatian Binary</td>
</tr>
<tr>
<td>Czech</td>
<td>Czech Republic</td>
<td>ISO-8859-2, MS1250, UTF-8</td>
<td>Czech Binary</td>
</tr>
<tr>
<td>Dutch</td>
<td>Netherlands</td>
<td>Latin1, MS1252, ISO-8859-9, UTF-8, IBM037</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Estonian</td>
<td>Estonia</td>
<td>ISO-8859-4, ISO-8859-10, MS1257, UTF-8</td>
<td>Estonian Binary</td>
</tr>
<tr>
<td>Farsi</td>
<td>Iran</td>
<td>ISO-8859-6, UTF-8</td>
<td>Default Binary</td>
</tr>
<tr>
<td>Language</td>
<td>Territory</td>
<td>Code Page (Character Set)</td>
<td>Sort</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>---------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Greek</td>
<td>Greece</td>
<td>ISO-8859-7 MS1253 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Hebrew</td>
<td>Israel</td>
<td>ISO-8859-8 MS1255 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Hungarian</td>
<td>Hungary</td>
<td>ISO-8859-2 MS1250 UTF-8</td>
<td>Hungarian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Korean</td>
<td>Korea</td>
<td>MS949 MS1361 UTF-8</td>
<td>Binary</td>
</tr>
<tr>
<td>Latvian</td>
<td>Latvia</td>
<td>ISO-8859-4 ISO-8859-10 MS1257 UTF-8</td>
<td>Latvian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>Lithuania</td>
<td>ISO-8859-4 ISO-8859-10 MS1257 UTF-8</td>
<td>Lithuanian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Macedonian</td>
<td>Macedonia</td>
<td>ISO-8859-5 MS1251 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Romanian</td>
<td>Romania</td>
<td>ISO-8859-2 MS1250 UTF-8</td>
<td>Romanian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Russian</td>
<td>Russia</td>
<td>ISO-8859-5 MS1251 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>Serbian</td>
<td>Yugoslavia</td>
<td>ISO-8859-2 MS1250 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>CyrillicSerbian</td>
<td>Yugoslavia</td>
<td>ISO-8859-5 MS1251 UTF-8</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Binary</td>
</tr>
</tbody>
</table>
Required Disk Space (UNIX Platforms)

Change the minimum free disk space requirement on page 1-4 of the Installation and Configuration Guide for UNIX Platforms from 100 megabytes to 145 megabytes.

CD-ROM Mount Command (NCR Platform Only)

The command for mounting the CD-ROM on NCR UNIX SVR4 MP-RAS on page 2-4 of the Installation and Configuration Guide for UNIX Platforms is incorrect. Replace it with the following:

```
/etc/mount -F cdfs device_name /cdrom
```
This chapter includes changes and additions to the Red Brick Vista User’s Guide:
The following topics are included:

• From Columns in Hierarchies Must Be NOT NULL
• Online Copies of Query Rewrite Case Studies
• SELECT Privileges on Precomputed Views
**From Columns in Hierarchies Must Be NOT NULL**

Add the following restriction to the description of explicit hierarchies in Chapter 3 of the *Red Brick Vista User’s Guide*.

When you create a hierarchy from one column to another, the first column (the *from* column) must have been declared NOT NULL when the table was created. Otherwise, an error message is displayed.

**Online Copies of Query Rewrite Case Studies**

Scripts that match the query-rewriting examples presented in Chapter 4 of the *Red Brick Vista User’s Guide* are installed with the Aroma database. A separate script for each example is located in the following directory:

```
/redbrick_dir/sample_input/vista_examples
```

Each script contains the SQL statements required to create the tables and views, insert data into the tables, set up the query-rewriting environment, and execute the rewritten query. The scripts are intended to be run in order (1 through 6) and contain the appropriate DROP statements to remove tables and views created in each case.

**SELECT Privileges on Precomputed Views**

The DBA must grant SELECT privileges on a precomputed view to any database users whose queries might be automatically rewritten to use the precomputed view. If a user does not have SELECT privileges on a precomputed view but does have SELECT privileges on the detail table, the query will complete but no rewrite will occur.

**Example**

Consider Case 1 which begins on page 4-3 of the *Red Brick Vista User’s Guide*. This example shows the privileges that must be granted in order for the new user to run the query and get a rewrite.

1. Connect to the database as the SYSTEM user and create a new user named *elaine*:
   
   ```
   RISQL> connect to aroma as system manager;
   RISQL> grant connect to elaine with elaine;
   ```
2. If you try to run the query in Step 5 on pages 4-4 and 4-5 of the Red Brick Vista User’s Guide, the following error occurs:

```
** ERROR ** (35) Insufficient authority to SELECT from table SALES.
```

3. Grant the database user elaine SELECT access to the tables in the query:

```
RISQL> grant select on sales to elaine;
RISQL> grant select on period to elaine;
RISQL> grant select on store to elaine;
```

4. Connect to the database as elaine and run the query:

```
RISQL> connect to aroma as elaine elaine;
RISQL> set stats info;
RISQL> select store_name, qtr, sum(dollars) as total_sales 
> from sales, period, store 
> where sales.perkey = period.perkey 
> and sales.storekey = store.storekey 
> group by store_name, qtr;
```

```
** STATISTICS ** (500) Compilation = 00:00:00.41 cp time, 
00:00:00.55 time, Logical IO count=74
STORE_NAME                     QTR   TOTAL_SALES
Roasters, Los Gatos            Q1_94        43011.50
San Jose Roasting Company      Q1_94        55763.25
...                              
Olympic Coffee Company         Q1_96        54868.50
Coffee Connection              Q1_96        31697.00
```

```
** STATISTICS ** (1457) EXCHANGE (ID: 2) Parallelism over 1 times 
High: 4 Low: 4 Average: 4.
** STATISTICS ** (500) Time = 00:00:00:44.98 cp time, 00:00:00:25.22 time,
Logical IO count=0 
** INFORMATION ** (256) 156 rows returned.
```

Notice that the query is not rewritten because the database user elaine does not have the SELECT privilege on the precomputed view
quarterly_store_sales_view.

5. Connect to the database as the SYSTEM user and grant SELECT access to the precomputed view.

```
RISQL> connect to aroma as system manager;
RISQL> grant select on quarterly_store_sales_view to elaine;
```
6. Connect to the database as elaine and run the query again:

RISQL> connect to aroma as elaine elaine;
RISQL> set stats info;
RISQL> select store_name, qtr, sum(dollars) as total_sales
   > from sales, period, store
   > where sales.perkey = period.perkey
   >     and sales.storekey = store.storekey
   > group by store_name, qtr;
** STATISTICS ** (500) Compilation = 00:00:00.37 cp time,
00:00:00.36 time, Logical IO count=76
STORE_NAME                     QTR   TOTAL_SALES
Roasters, Los Gatos           Q1_94        43011.50
San Jose Roasting Company     Q1_94        55763.25
...                             
Olympic Coffee Company        Q1_96        54868.50
Coffee Connection             Q1_96        31697.00
** INFORMATION ** (1462) SQL statement was rewritten to use one or
more precomputed views.
** STATISTICS ** (500) Time=00:00:00.06 cputime, 00:00:00.07 time,
Logical IO count=6
** INFORMATION ** (256) 156 rows returned.
RISQL>

The information message indicates that this time, the query was rewritten
to use the precomputed view. The elapsed time also decreased from
25.22 seconds to 0.07 seconds.
This chapter includes changes and additions to the ODBC Connectivity Guide:
The following topics are included:
• Sample ODBC Program
• ODBClib Dynamically Linked Shared Library (AIX platform only)
Sample ODBC Program

The sample ODBC program shipped with the Red Brick ODBClib SDK is different from the one shipped with V5.1.2. The new sample program, including the source code and makefile, is located in the `redbrick_dir/lib/example` directory, where `redbrick_dir` is the directory in which the Red Brick ODBClib SDK is installed. The new program code replaces the code currently printed on pages 4-4 to 4-12 of the ODBC Connectivity Guide.

Note: There is no sample program for the 32-bit Silicon Graphics version of the ODBClib SDK.

ODBClib Dynamically Linked Shared Library
(AIX platform only)

The name of the ODBClib shared library on the IBM AIX platform has changed from `librbodbc.so` to `librbodbcshared.a`. If you have applications that link with this shared library, you must change the name in the link statements. The name of the static library remains `librbodbc.a`. The definitions for the compile and link lines on page 3-5 of the ODBC Connectivity Guide should now read as follows:

IBM RISC System/6000—AIX

The following definitions apply to the compile and link commands for the AIX operating system:

```
# Define libraries to link
LIBS=$(REDBRICK_DIR)/lib/librbodbc.a -lm
CFLAGS=

#If you want to use Red Brick ODBClib shared library
#define LIBS as follows
#define LIBS=$(REDBRICK_DIR)/lib/librbodbcshared.a -lm
```

Use commands in the following form to compile and link an application on the AIX operating system:

```
$ cc $(CFLAGS) -I$(REDBRICK_DIR)/include -c rb_client.c
$ cc $(CFLAGS) -o rb_client rb_client.o $(LIBS)
```
Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms

This chapter contains information on parallel query processing on Windows NT platforms and is supplemental to the Warehouse Administrator’s Guide for Windows NT Platforms. The information about parallel query processing on UNIX platforms is in Chapter 10 of the Warehouse Administrator’s Guide for UNIX Platforms.

Query performance can be improved by using multiple tasks to process queries against large tables. Because multiple tasks consume more system resources—CPU, disk, processes/threads, and memory—than a single task, several parameters in the rbw.config file allow you to control the amount of parallel processing, balancing query response time against the available system resources and demands of other users. Queries that involve a relation scan of a large table or a STAR index are candidates for parallel processing.

This chapter describes query performance improvements that can be achieved by parallel query processing and is organized as follows:

• Parallel Query Tuning Parameters
• Enabling Parallel Query Processing
• Limiting I/O Contention with the FILE_GROUP Parameter
• Increasing Disk Group Tasks with the GROUP Parameter
• Limiting Available Tasks
• Setting Minimum Row Requirements with ROWS_PER_TASK Parameters
• Forcing the Number of Parallel Tasks with the FORCE_TASKS Parameters
• Enabling Partitioned Parallelism for Aggregation
• Other Limitations to Parallelism
• Analysis of System Resources and Workload
• Tuning for Specific Query Types
• Summary: Basic Guidelines
The extent to which parallel query processing is used is based on the tuning parameters listed in the following table. After these parameters are set, parallelism on demand is enabled whenever the specified conditions are met; without any further tuning, the end user will benefit from parallel processing.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUNE FILE_GROUP</td>
<td>Reduces seek contention on disk devices.</td>
</tr>
<tr>
<td>TUNE GROUP</td>
<td>Sets number of parallel tasks per file group (disk group).</td>
</tr>
<tr>
<td>TUNE TOTALQUERYPROCS</td>
<td>Sets maximum number of tasks available for parallel query processing at one time by all warehouse server threads.</td>
</tr>
<tr>
<td>TUNE QUERYPROCS*</td>
<td>Sets maximum number of tasks available for parallel query processing at one time by a single warehouse server thread.</td>
</tr>
<tr>
<td>TUNE ROWS_PER_SCAN_TASK*</td>
<td>Limits number of parallel tasks applied to a query that performs a relation scan (that is, does not use an index).</td>
</tr>
<tr>
<td>TUNE ROWS_PER_JOIN_TASK*</td>
<td>Limits number of parallel tasks applied to the index-probing portion of a query.</td>
</tr>
<tr>
<td>TUNE ROWS_PER_FETCH_TASK*</td>
<td>Limits number of parallel tasks applied to the row-data-fetching phase of a query.</td>
</tr>
<tr>
<td>TUNE FORCE_FETCH_TASKS*</td>
<td>Sets the number of parallel tasks for fetching rows in queries that use a STAR index.</td>
</tr>
<tr>
<td>TUNE FORCE_JOIN_TASKS*</td>
<td>Sets the number of parallel tasks for joining tables in queries that use a STAR index.</td>
</tr>
<tr>
<td>TUNE FORCE_SCAN_TASKS*</td>
<td>Sets the number of parallel tasks for relation scans of tables.</td>
</tr>
<tr>
<td>TUNE FORCE_HASHJOIN_TASKS*</td>
<td>Sets the number of parallel tasks for hybrid hash joins.</td>
</tr>
<tr>
<td>TUNE FORCE_AGGREGATION_TASKS**</td>
<td>Sets the number of parallel tasks for partitioned parallel aggregation.</td>
</tr>
<tr>
<td>TUNE PARTITIONED_PARALLEL_AGGREGATION*</td>
<td>Enables or disables parallelism for aggregation operations.</td>
</tr>
</tbody>
</table>

*These parameters can also be set with a SET command entered anywhere SQL statements can be entered.
Enabling Parallel Query Processing

To enable parallel processing, you must set the QUERYPROCS and TOTALQUERYPROCS parameters in the configuration file (rbw.config) to a value greater than zero.

You can control the extent of parallelism for query processing in either of two ways:

- Specify the minimum number of rows you want a task to process with the ROWS_PER_TASK parameters, which prevent incurring the overhead of parallel processing for trivial queries. These parameters are intended to handle the general purpose day-to-day processing.

- Specify the number of parallel tasks you want to use to process a query, regardless of the number of rows per task, with the FORCE_TASKS parameters. These parameters, which allow you more control over CPU resources, are designed for use on specific queries on which you want to specify the number of parallel tasks to be used in order to get the query processed quickly, regardless of the resources used. These parameters override the ROWS_PER_TASK parameters.

All parameters can be entered as TUNE parameters in the rbw.config file so they affect all server sessions; the order of these parameters in the rbw.config file is not significant. Alternatively, you can enter them as SQL SET statements, in which case they affect the current session only.

The parallel tuning parameters are discussed in detail in the following sections.
Limiting I/O Contention with the FILE_GROUP Parameter

The FILE_GROUP parameter definitions tell a warehouse server what PSUs are on the same disk in order to reduce seek contention on individual disk devices; these groups of PSUs are referred to as disk groups. This parameter limits the amount of parallelism: In general, at most one task will be allocated per disk group for each operation (such as a scan), unless the TUNE GROUP parameter is used to specify that more than one task can be used for a specific group.

Note: Because of the SuperScan technology used for disk I/O, tasks from multiple servers performing relation scans on tables can access the same data with a single read operation, which can reduce seek contention across server threads.

These entries are read at server startup so changes will be in effect for all sessions started after a change to the rbw.config file.

Syntax

To specify disk groups, enter a line for each disk group in the rbw.config file using the following syntax:

```
TUNE --- FILE_GROUP --- disk_group_id --- pathname_prefix
```

`disk_group_id`
Identifier used to group files into the same disk group; this must be an integer in the range of 1 to 32767.

`pathname_prefix`
Any string of characters delimited by whitespace or end-of-line.

Usage Notes

To determine what group a file belongs to, the filename (as specified in CREATE SEGMENT statements) is converted to an absolute filename. Then the longest pathname prefix that is a left substring of the filename is located. The `disk_group_id` associated with this pathname prefix is the ID of the group to which the file belongs.

If no matching pathname prefix exists for a particular filename, the file is considered to be in its own private group with a limit of one task applied to that group.
Any number of pathname prefixes can be mapped to the same disk group; each mapping requires a separate TUNE FILE_GROUP statement.

If multiple TUNE FILE_GROUP statements contain identical pathname prefixes, the last such entry is used and all others are discarded. With the exception of duplicate elimination, the order of the entries has no impact.

If you want to allow more than one task per disk group, you can increase this limit with the TUNE GROUP parameter.

**Example**

A table has its data segmented as follows:

<table>
<thead>
<tr>
<th>seg1</th>
<th>seg2</th>
<th>seg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f:\psu11</td>
<td>h:\psu21</td>
<td>i:\psu31</td>
</tr>
<tr>
<td>g:\psu12</td>
<td>i:\psu22</td>
<td>j:\psu32</td>
</tr>
</tbody>
</table>

To reduce seek contention by placing PSUs on the same physical disk in the same file group, enter the following lines in the `rbw.config` file:

```
TUNE FILE_GROUP 1 f:\
TUNE FILE_GROUP 2 g:\
TUNE FILE_GROUP 3 h:\
TUNE FILE_GROUP 4 i:\
TUNE FILE_GROUP 5 j:\
```

Note that FILE_GROUP 4 contains both `psu22` and `psu31`, which are on the same disk (i). Putting these PSUs in the same disk group prevents a query process from assigning two tasks to work simultaneously on the two PSUs.
Increasing Disk Group Tasks with the GROUP Parameter

The GROUP parameter provides a way to allow some parallel processing to occur within a disk group. In those cases where PSUs are striped across multiple disks, such as disk arrays or multiple disks grouped together as logical volumes, you can use this parameter to allow parallel processing for a specific disk group.

Syntax

To specify parallelism for a disk group, enter a line for each disk group in the rbw.config file using the following syntax:

```
TUNE GROUP disk_group_id num_tasks
```

`disk_group_id`
Identifier used to define a disk group with the FILE GROUP parameter; this must be an integer value.

`num_tasks`
Maximum number of outstanding tasks against the specified disk group. If you do not specify a GROUP parameter for a disk group, the default is one task at a time for that disk group.

Usage Notes

Even though the FILE_GROUP parameter is designed specifically to limit I/O contention within a specific disk group, multiple disks that are striped to appear as one logical volume to the operating system can support more I/O activity. You can increase parallel I/O activity to disk groups of this type (RAID disks or striped logical volumes) with the GROUP parameter.

In cases where queries are CPU-intensive rather than I/O-bound and there is excess CPU capacity, you can use the GROUP parameter to allow additional parallelism to take advantage of all the CPU capacity.

You need to enter a separate GROUP parameter for each disk group you want to modify.
Example

Suppose in the example on page 9-5 that f: and g:\ are actually logical volumes that are striped across five physical disks each. You could adjust the parallelism per disk group to accommodate CPU-intensive queries that are heavily concentrated in seg1 by entering the following lines in the rbw.config file:

```
TUNE GROUP 1 5
TUNE GROUP 2 5
```

Any query that accessed seg1 now could have as many as five I/O requests outstanding against f: and g:\.
Limiting Available Tasks

You can control the number of tasks available for parallel query processing and the allocation of those tasks in a multi-user environment. The TOTALQUERYPROCS parameter specifies the maximum number of tasks available for parallel queries at one time on all the servers controlled by a single daemon, providing a mechanism to control the system load imposed by parallel queries. The QUERYPROCS parameter specifies the maximum number of concurrent parallel tasks to be used in processing a single query, providing a mechanism to control the resources allocated to a single server (user).

The algorithm that allocates tasks to queries employs a “graceful decrease” mechanism to ration remaining tasks when the demand is high. After 50% of the total tasks available for processing queries have been allocated, subsequent queries are allocated fewer tasks per query.

Example

Assume TOTALQUERYPROCS is 1,000, QUERYPROCS is 100, and five queries have each been allocated 100 tasks so that 500 tasks out of the 1,000 total tasks have been allocated. In this case, a “graceful decrease” sets in and subsequent queries each receive fewer than 100 tasks apiece (even if they request 100). Once the ratio of allocated tasks to total tasks drops below 50%, queries again receive the requested number of tasks, up to the limit imposed by the QUERYPROCS value.

TOTALQUERYPROCS

To specify a limit on the total number of tasks available for parallel queries to all servers controlled by of a single warehouse daemon, enter a line in the rbw.config file using the following syntax:

```
TUNE TOTALQUERYPROCS max_parallel_tasks
```

`max_parallel_tasks`

A non-negative integer in the range of 0 to 32767. (A value of 0 or 1 effectively disables parallel query execution.)

The count specified by `max_parallel_tasks` does not include those tasks allocated to the base servers (the number limited by the MAX_SERVERS parameter). Changing this parameter might also require operating-system parameters to be changed, as described in the Installation and Configuration Guide. Individual operating systems have limits or operating-system parameters that limit the allowable range of TOTALQUERYPROCS.
Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms

Limiting Available Tasks

**QUERYPROCS**

To specify a limit on the total number of parallel tasks available for a single query for all sessions, enter a line in the `rbw.config` file using the following syntax:

```
TUNE QUERYPROCS num_per_query
```

To specify a limit on the total number of parallel tasks available for a single query for specific sessions, enter a SET command using the following syntax:

```
SET QUERYPROCS num_per_query -
```

**num_per_query**

A non-negative integer in the range of 0 to 32767. Specifying a value of 0 effectively disables parallel query processing. This number is an upper bound; other factors such as the ROWS_PER_TASK and FORCE_TASKS parameters and the TOTALQUERYPROCS parameter can also limit the number of tasks available for a single query. Also note that the distribution of data might result in some tasks completing before others, which might reduce the amount of parallelism to less than expected.

**Usage Notes for TOTALQUERYPROCS and QUERYPROCS**

If either the TOTALQUERYPROCS parameter or the QUERYPROCS parameter is 0 or not present, no parallel processing is done.

If multiple TOTALQUERYPROCS or QUERYPROCS statements are present, the last such entry is used and all others are discarded. With the exception of duplicate elimination, the order of the entries has no impact.

The TOTALQUERYPROCS entry in the `rbw.config` file is read at daemon startup so changes are not effective until the `rbwapid` daemon is restarted.

The QUERYPROCS entry in the `rbw.config` file is read at server startup so changes are not effective until a new server is started. If the SET command attempts to set QUERYPROCS to a value greater than the value in the `rbw.config` file, the value in the `rbw.config` file is used.
You can provide some guidelines to be used by the server in determining how much parallelism to use. If the number of rows to be processed is small, the overhead of parallel tasks outweighs the benefits. To prevent this, you can provide a row-per-task limit, which in effect says, “Do not start a parallel task unless the rows-per-task is greater than x.” The following three parameters are used for different types of queries and at different points in query processing:

- **ROWS_PER_SCAN_TASK** is the minimum estimated number of rows to be scanned by a relation scan before a parallel relation scan is performed. If the expected number of rows to be scanned exceeds this number, parallel scan tasks are used. This number does not affect queries that use an index, only those that do a relation scan of the table.

- **ROWS_PER_FETCH_TASK** is the minimum estimated number of data rows returned during the fetch portion of a STARjoin before parallel fetch tasks are used.

- **ROWS_PER_JOIN_TASK** is the minimum estimated number of index entries returned during the join processing (index-probing) portion of a STARjoin before parallel join tasks are used.

These parameters can be set for all sessions with entries in the `rbw.config` file, which are read at server startup; therefore, changes are effective only for new server sessions started after the change is made. If multiple values for a given parameter are specified, the last entry of each type is used and all others are discarded. With the exception of duplicate elimination, the order of the entries has no impact. These parameters can also be set for a specific session with a SET command.

The following sections describe how to select values for these parameters. In general, the default values supplied with Red Brick Warehouse inhibit parallelism. If you are not satisfied with the resource consumption and/or query response time and feel that more or less parallelism would improve performance, adjust the values accordingly.
ROWS_PER_SCAN_TASK

The ROWS_PER_SCAN_TASK parameter sets a lower limit to the number of rows each scan task must return in order to justify its existence, thus limiting the number of parallel tasks initiated for a relation scan. This limit affects queries that use no index but scan an entire table.

The server uses this parameter value as follows to determine the maximum number of tasks to use for a query of this type:

1. Each disk group with at least the number of rows specified by ROWS_PER_SCAN_TASK is assigned tasks based on the following formula:

\[
\text{MIN}\left(\frac{\text{rows\_in\_group}}{\text{rows\_per\_task}}, \frac{\text{max\_tasks\_per\_group}}{\text{ROWS\_PER\_SCAN\_TASK}}\right)
\]

Specified by ROWS_PER_SCAN_TASK

Specified by GROUP or 1 if no corresponding GROUP entry

2. The number of rows in each disk group containing fewer than the specified number of rows are added together. This total row number is then divided by the number of rows specified by ROWS_PER_SCAN_TASK to determine how many additional tasks to allocate:

\[
\text{Number of additional tasks} = \frac{\text{total\_rows}}{\text{rows\_per\_task}}
\]

Specified by ROWS_PER_SCAN_TASK

Note: The number of rows (in a disk group) refers to the number of rows for which space has been allocated in a PSU, and this number might exceed the number of rows visible to a query; for example, space might be allocated for rows that have since been deleted.

Syntax

To specify the value used for rows_per_task in the preceding equation for all sessions, enter a line in the rbw.config file using the following syntax:

```
TUNE ROWS_PER_SCAN_TASK rows_per_task
```

To specify the value used for rows_per_task in the preceding equation for specific sessions, enter a SET command using the following syntax:

```
SET ROWS_PER_SCAN_TASK rows_per_task ;
```
**rows_per_task**
An integer in the range of 1 to $2^{31}$. A higher value provides less parallelism in returning rows from the queried table, and a lower value, more parallelism. (Red Brick Systems recommends this number be at least 5,000.)

**Example**

Assume a table has space for 18,000,000 rows allocated across three segments, and each segment contains two PSUs. Each PSU is in its own disk group. The first PSU in each segment has been allocated up to its maximum size, which is sufficient to hold 4,500,000 rows, and the second PSU in each segment has sufficient space allocated to hold 1,500,000 rows. The following figure illustrates this table:

Assuming a maximum of one task per disk group and a ROWS_PER_SCAN_TASK value of 2,000,000, the maximum number of tasks that could be allocated to a relation scan of the table is computed as follows:

1. There are three disk groups, one for each of the large PSUs in each segment, that are each large enough to hold over 2,000,000 rows. These disk groups contribute three tasks to the maximum task count, even though each PSU is more than twice as large as the value for the ROWS_PER_SCAN_TASK parameter.

2. There are three disk groups, one for each of the small PSUs in each segment, whose allocated space is insufficient to hold 2,000,000 rows. Thus the total number of rows that could be held in the allocated space in all these groups is added up to yield 4,500,000 rows. This number is divided by 2,000,000 and rounded down to yield two tasks.
3. The resulting maximum task count is $3 + 2 = 5$. So a relation scan of this table would be processed by at most 5 parallel tasks.

**Note:** The number of tasks actually used is also bounded by the values set for the TOTALQUERYPROCS and QUERYPROCS parameters.

**ROWS_PER_FETCH_TASK and ROWS_PER_JOIN_TASK**

The ROWS_PER_FETCH_TASK and ROWS_PER_JOIN_TASK parameters determine how many parallel tasks are used to process queries that use a STAR index. Because queries vary in the amount of work done during the index-probing phase and the row-data-processing phase, you can set different limits for each phase. For example, if your queries tend to require a lot of processing after each row is fetched (GROUP BY, SUM, MIN, and so on), you should assign fewer rows per task for the fetch phase than for the join phase so that more tasks are used for the fetch phase.

These parameters allow you to control parallel processing for queries that use a STAR index based on the following guidelines:

- The more tightly constrained a query is on the columns that participate in the STAR index, the smaller the number of parallel tasks needed to probe the index efficiently during the join phase. The ROWS_PER_JOIN_TASK parameter defines what “tight” is and how many tasks to use during the join phase.
- The more rows to be returned and/or the more processing of row data to be done, the larger the number of parallel tasks that can be used effectively during the fetch phase. The ROWS_PER_FETCH_TASK parameter determines how many tasks to use during the fetch phase.
Both parameters use the following equations:

Estimated rows

\[
\text{Estimated rows} = \frac{\langle \text{count}(*) \rangle \text{ from fact_table} \cdot \prod \langle \text{count}(*) \rangle \text{ from dimension tables}}{\prod \langle \text{number of rows from dimension table} \rangle}
\]

Number of tasks

\[
\text{Number of tasks} = \left\lfloor \frac{\text{estimated rows}}{\text{rows per task}} \right\rfloor
\]

Specified by \text{TUNE} \text{ROWS\_PER\_JOIN\_TASK} and \text{TUNE} \text{ROWS\_PER\_FETCH\_TASK}

**Estimated rows**

This equation is applied after the constraints on the dimension tables are processed and the number of rows in each dimension table that satisfy the constraints is known. These numbers are multiplied together and then multiplied by the ratio of rows in the referencing table to all possible values in the STAR index—a type of density or sparseness, to determine an estimated number of rows to return from the STAR index.

**Number of tasks**

This equation uses the estimated number of rows and the rows-per-task parameter to calculate how many tasks to use during each phase of query processing.

**Syntax**

To specify the minimum number of rows per task used in the preceding equation for each phase for all sessions, enter lines in the \textit{rbw.config} file using the following syntax:

```
  - TUNE -- ROWS\_PER\_JOIN\_TASK -- rows\_per\_task
  - TUNE -- ROWS\_PER\_FETCH\_TASK -- rows\_per\_task
```

To specify the minimum number of rows per task used in the preceding equation for each phase for specific sessions, enter a SET command using the following syntax:

```
  - SET -- ROWS\_PER\_JOIN\_TASK -- rows\_per\_task ;
  - SET -- ROWS\_PER\_FETCH\_TASK -- rows\_per\_task ;
```
Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms

Setting Minimum Row Requirements with ROWS_PER_TASK Parameters

**rows_per_task**

Integers in the range of 1 to $2^{31}$. A higher value provides less parallelism, and a lower value, more parallelism. In no case will the system run a query in parallel if the number of tasks given by the preceding equations is less than 2.

**Tip:**

As a general rule, the values of ROWS_PER_JOIN_TASK and ROWS_PER_FETCH_TASK should each be at least 5,000 to justify the use of parallel tasks.

**Example**

This example illustrates how the equations on page 9-14 are used. A referencing (fact) table Fact has 3,000,000 rows. Three referenced (dimension) tables are used in its STAR index: Product with 2,000 rows, Market with 50 rows, and Period with 156 rows. The ROWS_PER_JOIN_TASK parameter is specified to be 90,000 rows per task, and the ROWS_PER_FETCH_TASK parameter is specified to be 50,000.

After the constraints are processed, all the rows from the Product table, 10 rows from the Market table, and 52 rows from the Period table satisfy the constraints:

\[
\text{Estimated rows} = \frac{(3,000,000) \cdot (2,000 \times 10 \times 52)}{(2,000 \times 50 \times 156)}
\]

\[
= 200,000
\]

Based on ROWS_PER_JOIN_TASK, which is specified as 90,000 rows per task, apply the equation for the number of tasks that will be used to process the STAR index as follows:

\[
\text{Number of join tasks} = \text{floor}(200,000 / 90,000) = 2 \text{ tasks}
\]

Based on ROWS_PER_FETCH_TASK, which is specified as 50,000 rows per task, the number of tasks used to fetch the data and perform any result pre-aggregation is calculated as follows:

\[
\text{Number of fetch tasks} = \text{floor}(200,000 / 50,000) = 4
\]

These figures indicate that $2 + 4 = 6$ tasks could be used to process the query based on the values specified for ROWS_PER_JOIN_TASK and ROWS_PER_FETCH_TASK.
Forcing the Number of Parallel Tasks with the FORCETASKS Parameters

The FORCETASKS parameters allow you to explicitly specify the number of parallel tasks that are used to process a query. The ROWS_PER_TASK parameters require that you determine a minimum number of rows needed to justify starting a parallel task, an implicit limit. These parameters, except for the FORCE_HASHJOIN_TASKS parameter, are analogous to the ROWS_PER_TASK parameters in their target queries:

- FORCESCAN_TASKS, which specifies a maximum number of tasks that can be used for relation-scan operations. This parameter does not affect queries that use an index, only those that perform a relation scan.
- FORCEFETCH_TASKS, which specifies the maximum number of parallel tasks that can be used for the fetch portion of a query using a STAR index.
- FORCEJOIN_TASKS, which specifies the maximum number of parallel tasks that can be used for the join portion of a query using a STAR index.
- FORCE_HASHJOIN_TASKS, which specifies the maximum number of parallel tasks that can be used for each hybrid hash join in a query.
- FORCE_AGGREGATION_TASKS, which specifies the maximum number of parallel tasks that can be used for each group in a partitioned parallel query.

These parameters are designed to be used only in cases where you want to override the general allocation of parallel tasks. For example, if you are running a query to build an aggregate table, no one else is using the system, and you want the query to complete as quickly as possible even if it greatly increases resource consumption.

Note: The FORCE_AGGREGATION_TASKS parameter only has an effect if partitioned parallelism is enabled with the PARTITIONED PARALLEL AGGREGATION parameter. For more information, refer to “Enabling Partitioned Parallelism for Aggregation” on page 9-23.
**FORCE_TASKS Syntax**

To specify the number of tasks for all sessions, enter a line in the rbw.config file using the following syntax:

```
TUNE FORCE_SCAN_TASKS OFF value
```

```
TUNE FORCE_FETCH_TASKS OFF value
```

```
TUNE FORCE_JOIN_TASKS OFF value
```

```
TUNE FORCE_HASHJOIN_TASKS OFF value
```

```
TUNE FORCE_AGGREGATION_TASKS OFF value
```

To specify the number of tasks for a single session, enter a SET statement using the following syntax:

```
SET FORCE_SCAN_TASKS OFF value
```

```
SET FORCE_FETCH_TASKS OFF value
```

```
SET FORCE_JOIN_TASKS OFF value
```

```
SET FORCE_HASHJOIN_TASKS OFF value
```

```
SET FORCE_AGGREGATION_TASKS OFF value
```
Explicit control of parallelism by the specified task limit is not enabled. The default value is OFF.

A value

An integer value that explicitly limits the maximum number of tasks; however, this value does not guarantee the specified number. For the SCAN, FETCH, JOIN, and HASHJOIN parameters, the actual number of tasks used will have an upper bound of the lowest of these three values:

- The specified FORCE_TASKS value.
- The number of PSUs over which the table is distributed.
- The number of tasks that can be allocated from the QUERYPROCS/TOTALQUERYPROCS pool.

For the FORCE_AGGREGATION_TASKS parameter, the actual number of tasks used will have an upper bound of the lower of these two values:

- The specified FORCE_AGGREGATION_TASKS value.
- The number of tasks that can be allocated from the QUERYPROCS/TOTALQUERYPROCS pool.

No argument

If OFF or value is not specified, the SET command returns the current setting for that parameter. For example:

```
set force_scan_tasks;
** INFORMATION ** (1433) FORCE_SCAN_TASKS is currently set to 6.
```
FORCE_SCAN_TASKS

The value set for FORCE_SCAN_TASKS controls the number of parallel tasks for relation scans of tables.

However, the FORCE_SCAN_TASKS value does not guarantee that a certain number of parallel tasks will be used. The actual number of tasks used will be the lowest of these three values:

- The FORCE_SCAN_TASKS value.
- The number of PSUs over which the table is distributed.
- The number of tasks that can be allocated from the QUERYPROCS/TOTALQUERYPROCS pool.

Example

Assume the following settings:

<table>
<thead>
<tr>
<th>FORCE_SCAN_TASKS</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSUs in table</td>
<td>18</td>
</tr>
<tr>
<td>QUERYPROCS</td>
<td>18</td>
</tr>
<tr>
<td>TOTALQUERYPROCS</td>
<td>24</td>
</tr>
</tbody>
</table>

Whether the FORCE_SCAN_TASKS value is used in this case depends on the number of tasks available from the TOTALQUERYPROCS pool. If only 6 tasks are already allocated, 18 tasks will be available so the FORCE_SCAN_TASKS value of 16 will be used.

Note: After 50% of the TOTALQUERYPROCS pool has been allocated, subsequent queries are allocated fewer tasks per query.

Usage Notes

Also note the following points regarding task allocation for relation scans:

- If FORCE_SCAN_TASKS is set, the ROWS_PER_SCAN_TASK value is ignored.
- If FORCE_SCAN_TASKS is set to a value that is greater than the number of disk groups, some disk groups will simply be allocated more than one task. When FORCE_SCAN_TASKS is set, the number of disk groups does not influence the behavior of parallel processing.
FORCE_FETCH_TASKS and FORCE_JOIN_TASKS

The values set for FORCE_FETCH_TASKS and FORCE_JOIN_TASKS control the number of parallel tasks for fetching rows and joining tables in queries that use a STAR index. If either of these values is greater than or equal to 1, it will override the corresponding value set for ROWS_PER_FETCH_TASK or ROWS_PER_JOIN_TASK.

However, the FORCE_FETCH_TASKS and FORCE_JOIN_TASKS values do not guarantee that a certain number of parallel tasks will be used. The actual number of tasks used to fetch rows will be the lowest of these three values:

- The FORCE_FETCH_TASKS value.
- The number of PSUs over which the table is distributed.
- The number of tasks available from the QUERYPROCS/TOTALQUERYPROCS pool.

The actual number of tasks used to join tables will usually be the lowest of these two values:

- The FORCE_JOIN_TASKS value.
- The number of tasks available from the QUERYPROCS/TOTALQUERYPROCS pool.

Note: In rare cases, the FORCE_JOIN_TASKS value might be greater than the number of STAR index rows that match the constraints in the query; therefore, it will not be possible to logically divide and process the query by the specified number of tasks. Instead, the number of matching rows will be used to set the limit on parallel join tasks.

Examples

FORCE_FETCH_TASKS

Assume the following settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE_FETCH_TASKS</td>
<td>16</td>
</tr>
<tr>
<td>PSUs in table</td>
<td>18</td>
</tr>
<tr>
<td>QUERYPROCS</td>
<td>18</td>
</tr>
<tr>
<td>TOTALQUERYPROCS</td>
<td>24</td>
</tr>
</tbody>
</table>

In this case, whether the FORCE_FETCH_TASKS value will be used depends on the number of tasks available from the TOTALQUERYPROCS pool. If only 6 tasks are already allocated, 18 tasks will be available and the FORCE_FETCH_TASKS value of 16 will be used.
**FORCE_JOIN_TASKS**

Assume the following settings:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE_JOIN_TASKS</td>
<td>8</td>
</tr>
<tr>
<td>QUERYPROCS</td>
<td>12</td>
</tr>
<tr>
<td>TOTALQUERYPROCS</td>
<td>30</td>
</tr>
</tbody>
</table>

If there are 9 or more tasks available from the TOTALQUERYPROCS pool, the FORCE_JOIN_TASKS value will be used.

**Usage Notes**

Also note the following points regarding task allocation for fetching rows and joining tables:

- You do not have to force both fetch and join tasks. For example, you can force join tasks but allow fetch tasks to be computed dynamically.
- If FORCE_FETCH_TASKS is set, the ROWS_PER_FETCH_TASK value is not used; similarly, if FORCE_JOIN_TASKS is set, the ROWS_PER_JOIN_TASK value is not used.
- Although the number of PSUs over which the table is distributed affects the allocation of parallel fetch tasks, the number of disk groups does not.
- The number of PSUs used to partition the STAR index does not affect the allocation of parallel join tasks.
- For multi–fact table joins, one fact table is selected to control the partitioning. If 10 PSUs are used to distribute the chosen fact table, 10 tasks are available for fetch-task partitioning.
- If there are fewer than the requested number of tasks available from the QUERYPROCS/TOTALQUERYPROCS pool and both FORCE options are set, the system tries to preserve the ratio of FORCE_JOIN_TASKS to FORCE_FETCH_TASKS values.
**FORCE_HASHJOIN_TASKS**

The value set for FORCE_HASHJOIN_TASKS controls the number of parallel tasks for hybrid hash joins.

However, the FORCE_HASHJOIN_TASKS value does not guarantee that a certain number of parallel tasks will be used. The actual number of tasks used will be the *lowest* of these following values:

- The FORCE_HASHJOIN_TASKS value.
- The number of tasks that can be allocated from the QUERYPROCS/TOTALQUERYPROCS pool.

**Example**

Assume the following settings:

```plaintext
FORCE_HASHJOIN_TASKS 8
QUERYPROCS 12
TOTALQUERYPROCS 30
```

If there are 10 or more tasks available from the TOTALQUERYPROCS pool, the FORCE_HASHJOIN_TASKS value will be used.

**Usage Notes**

Also note the following points regarding task allocation for parallel hybrid hash joins:

- The PARALLEL_HASHJOIN option must be set to ON, either with a SET PARALLEL_HASHJOIN ON command or a TUNE PARALLEL_HASHJOIN ON parameter, in order to get any parallelism from hybrid hash joins.
- You must have at least 2 more than the value you specify in FORCE_HASHJOIN_TASKS available from the QUERYPROCS/TOTALQUERYPROCS pool in order to achieve that level of parallelism. For example, in order to get 8 parallel hash join tasks, you must specify FORCE_HASHJOIN_TASKS to 8 and have at least 10 tasks available from the QUERYPROCS/TOTALQUERYPROCS pool.
Enabling Partitioned Parallelism for Aggregation

Queries that involve aggregation (SUM, MIN, MAX, COUNT) over groups specified in the GROUP BY clause can benefit from parallelism partitioned on the grouping columns, especially if there are a large number of groups (for example, hundreds of thousands or millions).

Partitioned Parallelism Syntax

To enable or disable partitioned parallel aggregation, enter a line in the rbw.config file using the following syntax:

```
TUNE - PARTITIONED_PARALLEL_AGGREGATION OFF ON
```

To enable or disable partitioned parallel aggregation, enter a SET statement using the following syntax:

```
SET - PARTITIONED_PARALLEL_AGGREGATION OFF ON ;
```

OFF

Disables partitioned aggregation parallelism. The default value is OFF.

Usage Notes

Once partitioned parallelism is enabled (PARTITIONED_PARALLEL_AGGREGATION set to ON), the value of the FORCE_AGGREGATION_TASKS parameter is enforced. If partitioned parallelism is disabled (PARTITIONED_PARALLEL_AGGREGATION set to OFF), the FORCE_AGGREGATION_TASKS parameter is ignored.

When PARTITIONED_PARALLEL_AGGREGATION is set to OFF, there is still a potential performance gain with parallel aggregation, but where the parallelism is not partitioned by the grouping columns. For relatively small numbers of groups, this should provide better performance than the partitioned parallelism.

When you set partitioned parallel aggregation on, the amount of tasks used on your machine potentially doubles—the amount of tasks needed for the parallel aggregation plus the amount of tasks needed for the rest of the query processing; therefore, to ensure the same resources to the other parts of the parallel query, you should increase the values of the QUERYPROCS and TOTALQUERYPROCS parameters.
Partitioned parallelism is also effective when populating a table with an INSERT INTO...SELECT...GROUP BY operation, and is particularly effective when there are a large number of groups. This might improve the performance of these types of operations when populating aggregate tables for use with the Red Brick Vista option. For details about the Red Brick Vista option, refer to the *Red Brick Vista User's Guide*. 
Other Limitations to Parallelism

The equations on page 9-14 determine upper limits on the number of parallel tasks applied for specific operation; however, other system considerations also limit these numbers.

The number of tasks applied to index join tasks is also limited by the number of file groups in which the segments for the selected STAR index reside. In the example on page 9-15, in order for two tasks to be used to process the join tasks, the index must be spread over at least two file groups. If it is fewer than two file groups, only one task will be applied (unless you force parallelism with the FORCE_JOIN_TASKS parameter or specify more than 1 disk group task with the GROUP parameter). In general, if you examine the segments in which the STAR index resides, the limit on the amount of parallelism used in the join tasks is the number of file groups covered by those segments.

The number of tasks assigned to fetch tasks is also limited in the same manner by the number of groups in which the data resides.

Another limitation is the possibility that the system might be unable to allocate the number of tasks desired. In the previous example, the equations indicated that five tasks should be allocated to process the query. If, however, the system is able to allocate only four tasks (other users are using some of the tasks allocated for processing parallel queries), the system must allocate the available tasks between the join and fetch phases. The allocation of limited tasks to the fetch phases is based on the following equation:

\[
\text{Number of fetch processes} = \text{MAX} \left( 1, \frac{\text{requested for fetch}}{\text{total requested}} \times \text{total available} \right)
\]

The remaining available tasks are then allocated for join processing.

In the example, with only four tasks available, the number of tasks allocated to fetch processing is 2:

\[
\text{Number of fetch processes} = \text{MAX} \left( 1, \frac{3}{5} \times 4 \right) = 2
\]

The remaining two tasks are allocated to join processing.

The most important thing to note is that parallelism is bounded by the number of groups in which the index and data reside. In general, the system does not allocate more tasks than there are groups affected by the query.
Example

This example illustrates how the number of tasks allocated is affected by the distribution of the index and data across file groups. Assume a table is implemented as follows:

```sql
create segment idx1 ... (two PSUs);
create segment idx2 ... (two PSUs);
create segment data1 ... (three PSUs);
create segment data2 ... (three PSUs);

create table fact...
data in (data1, data2) segment by ...
primary index in (idx1, idx2) segment by references of (prodkey)
ranges (min:1000, 1000:max)
```

Assume that each PSU is in a disk group by itself.

The maximum number of tasks that will ever be allocated for join processing is four, because the index covers four file groups. If the constraints cover only one segment, the maximum number of join tasks actually would be two, because the segment contains two PSUs, each in a separate disk group.

The row data accesses are unpredictable; however, the calculated maximum number of fetch tasks that might be allocated to fetch processing is six because the data is distributed across six PSUs, each in a separate file group.
Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms
Analysis of System Resources and Workload

Analysis of System Resources and Workload

Parallelism takes advantage of available system resources by scheduling more work concurrently to speed up query processing. Parallelism also introduces additional factors that affect performance gains. Getting the best performance on parallel queries depends on the degree to which you can exploit concurrency and load balancing across a system, a task that requires careful planning, even before you load the database. The main questions concern system resource allocation and usage.

Disk Usage

To reduce disk contention and increase I/O activity, I/O load should be distributed evenly across as many physical disks as there are parallel tasks. The object is to improve disk service time and improve I/O concurrency in order to reduce the time a query waits because its tasks are blocked on I/O.

Ideally, no more than one disk group, as defined by the FILE_GROUP parameter, should be assigned to one physical disk. Disk arrays or disks grouped together using a logical volume manager facility are exceptions; in these cases, there are always multiple physical disks grouped together as one logical disk. In the following discussions, a disk group is treated as interchangeable with a physical disk. If there are multiple files (PSUs) per physical disk, these files should normally be organized into a single disk group using the FILE_GROUP parameter. Such an organization ensures that I/O can be scheduled evenly and reduces excessive head movements on the disk, especially important for parallel scans where it is desirable to have an orderly sequential access on the file and to take advantage of any available read-ahead capability as much as possible.

In some cases where disk arrays or disks are grouped together, such as striped disks or RAID systems, or where the query workload is CPU-intensive rather than I/O-bound and there is excess CPU capacity, better performance might result from allowing more than one task per disk group. If you want to allow parallel tasks within a disk group for a query, use the TUNE GROUP parameter to specify the number of tasks.
For Data

If you are not sure how uniformly the data will be distributed across segments, or if you want to spread the data across more physical disks because you anticipate the queries will tend to cluster on a single disk or only a few disks, you can use the SEGMENT BY HASH option in creating tables. Hashing segments helps to ensure that data will be evenly distributed across all segments and thereby helps to balance processing across parallel tasks. However, before deciding to use the hash option, consider the space management issues associated with hashing; for example, hashed segments cannot be dropped individually or taken offline for loads.

For STAR Indexes

It is not always necessary or desirable to segment the STAR index. Consider whether the benefit merits the extra administration overhead. Parallelism for join tasks is derived based on PSUs and disk groups. As long as there are multiple PSUs and they are not grouped into a disk group, parallelism can be invoked even if the STAR index is in a single segment.

If the STAR index is to be loaded across multiple segments for parallel join processing, load the index on separate disks if possible. As a minimum, allow as many index PSUs or segments as the planned number of join tasks. If the CPUs are very fast, you might need to allow more index and join tasks to keep the CPUs busy. If not enough disks are available, the next best option is to load the index segments on the same disks as the table segments, as long as the disks are not too busy.

Memory Usage

Memory requirements increase with parallel processing, even if the number of users remains the same. Because parallel tasks are spawned from the “parent” server, the child tasks inherit many of the characteristics of the parent, including memory requirements. So be aware of the additional memory demands in parallel processing, particularly important in a multi-user environment.

To determine memory requirements at your site, you should first determine how many users are actively executing queries at the same time (concurrently active users). The number of concurrently active users will determine the memory demand and paging/swapping activities in the system at any one time. For example, if 100 users are connected, but 80 of them only submit a query (10 minutes long) once a day, 18 of them only twice a day (5 minutes long), and 2 of them all the time, then you can estimate about 3 concurrently
active users (assuming the 98 users submit their queries evenly throughout the day). Three users probably will not introduce heavy paging/swapping activities in the system. However, if there are 50 concurrently active users, examine carefully whether there is enough memory to support them. If there is not enough memory, thrashing will occur, resulting in very heavy paging/swapping activities.

If the number of concurrently active users is high, consider either adding more memory or reducing the amount of parallelism. You can reduce parallelism either by reducing the number of query tasks per user (QUERYPROCS) or by ensuring that only large queries use parallel processing. You can prevent trivial or small queries from using parallel processing by choosing a large value (thousands to tens-of-thousands for STARjoin queries and tens- to hundreds-of-thousands for scans) in the ROWS_PER_JOIN/FETCH/SCAN_TASK parameters.

When the system is up and running, monitor the paging/swapping activities. Acceptable values for these two activities vary depending on system size and speed. Rather than using generic values that may or may not apply to your system, a better approach is to monitor the system when it is not performing optimally; the paging/swapping disks will indicate when things are not right. Use Performance Monitor or another performance monitoring tool to monitor the percentage of time spent waiting for I/O and the disk service time. You can also evaluate disk usage based on busy percentage, disk request queue, and disk waiting time. If all these indicators are high, users are probably kept waiting while the system is busy paging or thrashing in memory. To decrease the wait time, you can add more paging/swapping devices, reduce parallelism, or add memory.
**CPU Allocation**

The degree of concurrency in parallel processing largely depends on how many CPUs are available. Although allocating all the CPUs in the system might yield the best results, this option is often not practical because other work in the system could be competing for CPU resources. If all the CPUs are used for parallel query processing, other users would experience a slow-down because CPU resources are less available to them. Therefore, each administrator must decide how the CPU resources are to be distributed. Of course, if all the CPUs are already saturated (that is, 100% busy), there will be no gain and perhaps even a slight degradation from parallel processing. After you have determined the number of CPUs to use for parallel query processing, you can derive the number of parallel tasks (QUERYPROCS) as follows:

- For a query that is quite CPU-intensive, set the QUERYPROCS parameter to the “number” of CPUs divided by the number of concurrently active users.
- For a query that is not very CPU-intensive but is I/O-intensive, set the QUERYPROCS parameter to two or three times the value derived for CPU-intensive queries. Monitor the CPU-busy statistic to determine whether more parallel tasks are needed.

For example, assume you have a 12-CPU system and you want to allocate about 65% of CPU resources (8 CPUs) to parallel processing. If the system is 45% busy overall, it is reasonable to add more parallel tasks until the system reaches at least 65% busy. And if other tasks are executing in the system at the same time, it is reasonable to add even more parallel tasks, depending on the distribution of resource consumption. (If there are other tasks, parallel processing probably was not consuming 65% of the CPU resources.)

The FILE_GROUPS setup also affects concurrency. For I/O-intensive queries, specify as many disk groups (physical disks) as there are parallel tasks (QUERYPROCS). One factor that the server uses to determine how many parallel tasks to create is the number of disk groups. If the number of disk groups is less than the QUERYPROCS value, the number of parallel tasks to create will be reduced to the number of disk groups.

If there are more CPUs than disk groups and there is excessive CPU capacity, use the TUNE GROUP parameter to provide more parallelism than is normally allocated per disk group.
Tuning for Specific Query Types

This section describes how you can fine-tune parallel processing for specific query types. Before tuning for parallel processing, however, make sure that the queries themselves are fine-tuned with well-written SQL.

Note: The FORCE_TASKS parameters override the corresponding ROWS_PER_TASK parameters and are not intended for use as general tuning parameters. The following discussion focuses on tuning with the ROWS_PER_TASK parameters.

For information on how Red Brick Warehouse processes queries, refer to “Understanding Red Brick Query Processing” in Chapter 9 of the Warehouse Administrator’s Guide.

Parallel STARjoin Queries

The speedup expected from parallel processing of STARjoin queries (those queries that use a STAR index) depends primarily on:

- Density of the query (number of rows selected from the referencing table).
- Number of parallel tasks.
- Mix of parallel (join and fetch) tasks.
- Number of disk groups, or file groups, as defined by the TUNE_FILE_GROUPS entries in the rbw.config file.
- Amount of parallelism per disk group, as defined by the TUNE GROUP entries in the rbw.config file. Keep in mind that allowing parallelism within disk groups might increase disk contention and degrade performance.

Density

In general, the higher the density of a query, the better the potential for speedup.

Number of Parallel Tasks

The number of tasks can depend on the distribution of the data. If data for the query is clustered in the referencing table, it is possible that not all parallel tasks will get work. In this case, hashing the referencing table segments helps distribute the data more evenly across disks. For some cases where STARjoin queries are slow, alternate STAR indexes might provide more speedup than parallelism; depending on which columns are constrained, consider alternate STAR indexes before resorting to parallel processing.
Mix of Parallel Tasks and File Groups

Choosing the right mix between join and fetch tasks is also important in speedup gains. For example, assume a query requires relatively much more post processing than join processing but has only one fetch task allocated; the parallel speedup is much less than if more fetch tasks were allocated.

Use the following guidelines to finely tune the mix for specific queries:

- For queries dominated by post processing, allocate more fetch than join tasks.
- For queries dominated by join processing, allocate more join than fetch tasks.
- If there is a mix of join and post processing, start by specifying an equal number of join and fetch tasks. Then alter the ratio in both directions by 50% to determine which ratio is best. Usually, ratios of n/1 or 1/n are not the best choices, although there are exceptions.

Use the ROWS_PER_JOIN/FETCH_TASK parameters to control the ratio between join and fetch tasks. Note that the values for these two parameters are inversely proportional to the ratio desired. For example, for six join and two fetch tasks, specify:

rows per join task = 2000
rows per fetch task = 6000

The actual allocation of tasks is more complicated:

1. The number of tasks is calculated by dividing the rows per join/fetch task into the estimated number of rows. If the join or fetch value is less than the estimated number of rows, no parallel tasks are created.
2. The number of tasks is evaluated against the number of file groups and group limits; the smaller number becomes the new number of tasks. For join tasks, the number of file groups for index segments is used. For fetch tasks, the number of file groups for the table is used.
3. The QUERYPROCS value is compared with the sum of the join and fetch tasks; the smaller number becomes the new number of tasks.

If the QUERYPROCS value is smaller, the ratio between the join and fetch tasks (as specified by the ROWS_PER_JOIN/FETCH_TASK values) is preserved. If, however, the number of file groups for either joins or fetches is less than the number of tasks allocated by the ratio, the ratio is not preserved: Maximum parallelism is offered by allocating join and fetch tasks up to the QUERYPROCS value.
4. If the remaining number of tasks specified by the QUERYPROCs value is less than 50% of TOTALQUERYPROCS, a graceful degradation process begins, assigning only a portion of the remaining QUERYPROCS.

**Considerations for Multi-User Environments**

In a multi-user environment, it is often not worthwhile to offer parallelism for trivial queries (queries that take less than one minute). Parallel queries consume much more memory and the user-perceivable speedup improvement is relatively small. You must consider the corresponding resource trade-off to decide which query types (trivial, medium, or large) merit parallel processing. To prevent trivial queries from using parallel processing, specify high values for the ROWS_PER_JOIN/FETCH_TASK parameters.

Additionally, use the TOTALQUERYPROCS parameter to limit the number of users of parallel processing; a reasonable value for the TOTALQUERYPROCS parameter is two to three times the QUERYPROCS value. If there is enough memory, you might find values of five times or higher acceptable. Fairly high values for the ROWS_PER_JOIN/FETCH_TASK parameters ensure that only large queries use parallel processing.

**Parallel Table Scans**

The speedup provided by parallel processing of table-scan operations is determined by the number of parallel scan tasks, and the number of file (disk) groups available, and the TUNE GROUP parameter value.

Depending on the speed of the CPUs, several parallel scan tasks might be required to keep the “number” of CPUs busy. The recommended procedure is to assign no more than one disk group to one physical disk and to allow parallel scan processing to assign no more than one task per file (disk) group, thus facilitating sequential contiguous disk access whenever no more than one user is accessing the disk at a time. Otherwise, disk service times per block might increase by orders of magnitude. You can, however, use the TUNE GROUP parameter to allow more than one task per group in cases where the a query is CPU-bound on a multi-processor system with additional CPU capacity.

As described for parallel STARjoin queries, the recommendations for limiting trivial queries (those that are small enough to return in one minute or less) and multi-user environments apply to ROWS_PER_SCAN_TASK as well.
SuperScan Technology

SuperScan technology is used whenever multiple users are scanning the same table at the same time or overlap some of the time. The perceived improvement varies greatly from time to time depending on system activities.

SuperScan technology takes advantage of the operating system’s file buffer cache to reduce physical I/O activity. If the scanning tasks find most of the I/O blocks in the file buffer cache, there will be a large decrease in I/O activity. If few blocks are found in the file buffer cache, there will be a smaller decrease. Because the file buffer cache is used by everyone in the system, its state (that is, the hit rate) depends on the system activities. The interval between additional users beginning to scan the table also affects the decrease in I/O activity. Because users probably do not have the same time arrival pattern from day to day, improvement from SuperScan technology might vary from day to day.

About Reasonable Values

No one set of tuning values applies to all customers across all queries. As discussed in the previous sections, many considerations affect parallelism and they are different for each customer, depending on the hardware platform, configuration, query mix, and number of users. The information presented here highlights general areas of concern and provides guidelines for tuning query performance for your environment.

The ROWS_PER_JOIN/FETCH/SCAN_TASK parameters have two main uses. First, they control parallelism and determine when it is invoked; use these parameters to set the minimum estimated number of rows required before parallel processing is used. Secondly, the ROWS_PER_JOIN/FETCH_TASK parameters determine the ratio between join and fetch tasks for STARjoin queries.

Previous sections discuss factors relevant to the number of parallel tasks; the primary one is the “number” of CPUs. Then disk groups, memory, and users are considered, with TOTALQUERYPROCS limiting the number of users for parallel query processing.
Summary: Basic Guidelines

The following suggestions provide some basic guidelines for setting "reasonable" values; refer back to the previous sections for more detailed information. Remember, however, each system and workload is unique, so you must experiment to determine what works best at your site.

1. Set the ROWS_PER_JOIN/FETCH/SCAN_TASK values fairly high. For more information, refer to “Memory Usage” on page 9-28 and “Setting Minimum Row Requirements with ROWS_PER_TASK Parameters” on page 9-10.

2. Set the values for ROWS_PER_JOIN_TASK and ROWS_PER_FETCH_TASK to the same value. For more information, refer to “Tuning for Specific Query Types” on page 9-31.


4. Set TOTALQUERYPROCS to at least two to three times QUERYPROCS, or even higher if memory is available. For more information, refer to “Limiting Available Tasks” on page 9-8.

5. If multiple PSUs for the same table are on the same disk device, define a disk group (file group) for those PSUs to limit disk contention. For more information, refer to “Disk Usage” on page 9-27.

6. In most cases, you do not need to set the TUNE GROUP parameter: generally, only one task per disk group is best. For more information, refer to “Increasing Disk Group Tasks with the GROUP Parameter” on page 9-6.

These basic settings will provide users with a reasonable environment for parallelism on large queries, but they will not process every query optimally. If you want to fine-tune parallel processing for more optimal processing of specific queries, refer to the suggestions on page 9-31.
Tuning a Warehouse for Parallel Query Processing on Windows NT Platforms

Summary: Basic Guidelines
This chapter contains new and changed messages, along with their causes and responses, and supplements the Messages and Codes Reference Guide. The following chapters are affected:

- Chapter 1, “Event Log Messages”
- Chapter 2, “Informational, Warning, and Error Messages”
Chapter 1, “Event Log Messages”

In the message format for the event log messages, incorrect severity indicators are used in the message identifiers. The following table identifies the correct severity mappings:

<table>
<thead>
<tr>
<th>Incorrect Severity Identifier</th>
<th>Correct Severity Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
</tr>
<tr>
<td>W</td>
<td>U</td>
</tr>
</tbody>
</table>

For example:

<table>
<thead>
<tr>
<th>Incorrect Severity Identifier</th>
<th>Correct Severity Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBE076I</td>
<td>RBE0076R</td>
</tr>
<tr>
<td>RBE079S</td>
<td>RBE079A</td>
</tr>
<tr>
<td>RBO077W</td>
<td>RBO077U</td>
</tr>
</tbody>
</table>

The following messages have been added or modified.

RBO150A  Database starter daemon started.

  Reason:  The database starter daemon was started.

RBO151A  Database starter daemon completed.

  Reason:  The database starter daemon has completed.

RBO152A  Vacuum cleaner daemon started.

  Reason:  The vacuum cleaner daemon was started.

RBO153A  Vacuum cleaner daemon completed.

  Reason:  The vacuum cleaner daemon has completed.
RBO154R  \textit{stat_typetime_string cp time, time_string tot cp time, time_string agg tot cp time, time_string elapsed time, time_string agg elapsed time, Logical Reads/Writes:numbernumberstat}

\textbf{Reason:} Full statistic messages have been enabled. This message contains statistical information about the command. There might be platform-specific information following the Logical Reads/Writes information.

RBO155A  Process checker daemon started.

\textbf{Reason:} The process checker daemon was started.

RBO156A  Process checker daemon completed.

\textbf{Reason:} The process checker daemon has completed.

RBO157A  Abnormally terminated process has been detected. PID: \textit{pid}.

\textbf{Reason:} The process checker daemon detected an abnormally terminated process. Some cleanup will be performed internally.

RBO158A  Abnormally terminated process has been detected. Process ID: \textit{pid}, Thread ID: \textit{tid}.

\textbf{Reason:} The process checker daemon detected an abnormally terminated thread. Some cleanup will be performed internally.

RBA621A  User \textit{user_name} logged in.

\textbf{Response:} Records that a user that is a member of any user-defined role logged in.

RBO1004A  Timer daemon shutdown requested.

\textbf{Reason:} The timer daemon received a command to quit.

RBO1005A  Timer daemon started.

\textbf{Reason:} The timer daemon was started.

RBE6001U  Corrupt entry found at \textit{number} block.

\textbf{Reason:} A corrupted entry was found during recovery.

RBE6002A  Recovery completed with \textit{number} blocks recovered and \textit{number} errors during recovery.

\textbf{Reason:} Recovery completed with errors.
Messages and Codes Reference Guide
Chapter 1, “Event Log Messages”

RBO6003A  Recovery completed with number blocks recovered.
Reason:   Recovery completed successfully.

RBE6010U  The shared resources for the database dbname have been marked bad. Please restart the database.
Reason:   Terminate the shared resources for the database and then reconnect to the database.

RBE6050A  Transaction with an assigned revision number number aborted.

RBE6051A  Transaction aborted.

RBO6052R  Transaction with an assigned revision number number committed.

RBO6053A  All the shared resources cleanup has completed.

RBE6054A  Encounter an internal error : number during the cleanup.

RBE6055A  Transaction with an assigned revision number number has been committed.

RBE6056A  Encounter an internal error : number during the database shared memory cleanup.

RBO6061R  Committed the segment segname with revision number.

RBO6062R  Cleaned the segment segname with revision number from version log.

RBO6063A  Database 'dbname' started.

RBO6064A  Database stopped.

RBS6065R  Version log created in segname.

RBS6066R  Version log dropped.

RBO6067R  Versioning started.

RBO6068R  Versioning stopped.

RBO6071U  Version log utilization reached a critical level of number percent.
Reason:   Either complete some transactions or create the version log again with more space.
Chapter 2, “Informational, Warning, and Error Messages”

The following messages have been added or modified.

**RBA003E** Operation not completed because of lock conflict and session has set locks nowait.

**Cause:** The session has set locks nowait, and the operation requires a lock that cannot be obtained immediately. The lock may be a system table lock, a table lock or a segment lock.

**Response:** If the database is versioning, the lock conflict may be caused by the vacuum cleaner daemon, which reads the system tables from time to time during cleaning. Try the operation again. If an ALTER TABLE operation has failed, you may have to use the ALTER TABLE RESUME or RESET operation to complete or undo the original operation.

**RBA057E** A scalar or table subquery in a WHERE or SET clause must have a single item in the select list.

**RBS123E** Duplicate constraint names are not allowed. Duplicated name is 'constraint_name'.

**Cause:** The indicated constraint name is already defined in the database or it is already used in the statement. Constraint names must be unique within the database.

**Response:** Change the name of the constraint and submit the statement again.

**RBA154E** A database file has been corrupted.

**RBA229E** Segment segment_name is not present in the database.

**Cause:** A DROP SEGMENT, ALTER SEGMENT, CHECK TABLE, CHECK INDEX statement or a segment-related TMU operation was specified for a segment that does not exist.

**Response:** Correct the statement and submit it again.

**RBS268W** Rows were not updated because null values violated constraints: number.

**Cause:** UPDATE operation attempted to insert NULL values into one or more columns that do not allow NULLs.
RBL409I  Reorganizing table table_name in parallel mode.

**Cause:** This information message indicates that the TMU has begun a REORG operation on the named table.

**Response:** Information only.

RBL419I  Reorganizing table table_name in serial mode.

**Cause:** This information message indicates that the TMU has begun a REORG operation on the named table.

**Response:** Information only.

RBA435E  The DUPLICATESPILLPERCENT value number is not between 0 and 100.

**Cause:** A SET INDEX TEMPSPACE DUPLICATESPILLPERCENT statement specified an illegal value. The DUPLICATESPILLPERCENT must be a value between 0 and 100, inclusive.

**Response:** Specify a value within valid limits.

RBR825E  Cannot retrieve numcols columns (maximum maxcols).

**Cause:** Your query attempted to retrieve more than the maximum number of columns supported by the program.

**Response:** Shorten the query's SELECT clause so that it produces fewer columns.

RBS1039E Function 'function_name' was called with an invalid argument value number.

**Cause:** The value of the specified argument of the specified function is invalid. If the function is SQRT, valid argument values are zero and positive numeric values. If the function is LN, valid argument values are positive numeric values.

**Response:** You can either use the ARITHIGNORE/ARITHABORT SET commands or rbw.config-file options to control the behavior when SQRT or LN processes an invalid value, or you can rewrite the query to include only valid values for the function (zero and positive numeric values for SQRT; positive numeric values for LN).
RBL1040E  Temporary index space exhausted.

Cause: The processing of the TMU statement terminated because temporary index space was exhausted. Either the file system(s) containing the temporary index files was full or the "redbrick" user's quota on that file system(s) was exceeded.

Response: To resolve this problem, try one or more of the following suggestions:

1. Use the INDEX_TEMPSPACE_DIRECTORIES parameters to specify that alternative or additional directories on file systems with additional disk space be used by the TMU.

2. Ensure that there are no temporary index files from prior invocations of Red Brick Warehouse in the temporary index directories. You can specify an appropriate cleanup script in the rbw.config file to be run each time the warehouse daemon starts.

3. Expand the available disk space by removing unnecessary files from the file system(s) containing the directories specified for temporary index space.

4. Ensure that disk quotas for the "redbrick" user on the file system(s) containing the directories specified for temporary index space are not too restrictive.

RBA1099F  Unable to fork daemon process: (errno) reason.

Cause: A service daemon was unable to initialize itself by forking a child process. The operating-system error code and reason are indicated.

Response: Review the operating-system error and take appropriate corrective action.

RBS1103E  The specified database name dbname was not found.

Cause: The ALTER SYSTEM or ALTER DATABASE command specified a database name that was not recognized by the administration daemon, which has knowledge only of databases that have had some activity since the daemon started.

Response: Select the DBNAME column from any Dynamic Statistic Table to make sure that a valid database name is specified.
RBL1305I  Starting duplicate removal phase with number duplicates.

Cause: Indicates that the TMU has started the final phase of duplicate removal during a LOAD DATA or REORG operation. This message is issued only when an index is built using the standard OPTIMIZED indexing mode. In some cases, this duplicate-removal phase can take substantial time.

Response: Information only.

RBS1413F  Unable to fork process, operating system error: string.

Cause: An attempt to fork a process failed. The operating system returned the indicated error, which might be the result of an operating system resource being exhausted at the system or user (redbrick) level. Often the resource that has been exhausted is swap space or processes.

Possible causes are an incorrectly configured kernel (such as the swap space or the number of processes per system being set too low), incorrect configuration parameters in the rbw.config file (such as TOTALQUERYPROCS being set too high), or other applications or subsystems consuming the available resources.

Response: Examine the rbw.config file to verify that the configuration options are set as desired (paying particular attention to the MAX_SERVERS, TOTALQUERYPROCS, and QUERY_MEMORY_LIMIT parameters).

Verify that the kernel is configured as recommended by Red Brick Systems for the configuration parameters specified in the rbw.config file (refer to the appropriate Red Brick Installation and Configuration Guide). Verify that if applications other than Red Brick Warehouse are running on the server machine, then the kernel parameters have been adjusted to reflect this additional load.

Examine the operating system logs for messages logged at or shortly before the time this error was reported.

Based on this analysis, adjust the kernel parameters, rbw.config parameters, and/or concurrent application load as appropriate. If necessary, contact the Red Brick Customer Support Center for assistance.
RBA 1437E  Attempt to configure more than number temp_space_type DIRECTORIES.

**Cause:** An attempt was made to configure more directories for the indicated type of temporary space than the maximum number allowed, either specified with an rbw.config file entry or with a SET [QUERY | INDEX]_TEMPSPACE DIRECTORIES command.

**Response:** Check the rbw.config file or SET commands to determine how many directories you have configured. The maximum number is 64 directories for INDEX_TEMPSPACE directories and 64 directories for QUERY_TEMPSPACE directories.

RBA1502E  Failed to create message queues. Internal error number number.

**Cause:** An error occurred while creating a message queue.

**Response:** Check for lack of IPC resources on the machine. If resources are sufficient, report the error, including the error number, to the Red Brick Customer Support Center.

RBA1503E  Message queue operation on qid msgqid failed with errno errno.

**Cause:** An error occurred while trying to send or receive on a message queue.

**Response:** This error indicates an unexpected error while trying to send or receive a message on a message queue. Check for lack of IPC resources on the machine. If resources are sufficient, report the error, including the error number, to the Red Brick Customer Support Center.

RBS1629E  Subqueries in ON clause of outer joins are not supported.

**Cause:** The ON clause of an outer join cannot contain subqueries.

**Response:** Rewrite the outer join to eliminate the need for a subquery in the ON clause.

RBS1693E  Model column name exceeds maximum length of len bytes.

**Cause:** The column name for a model column is too long.

**Response:** Shorten the name of the column and resubmit the command.
RBS1694E  Cannot update the primary key of table table_name because there is no index on the primary key.

    **Cause:** The index on the primary key of the specified table has been dropped. Updating the primary key of a table requires that a B-TREE or STAR index be defined on the primary key of that table.

    **Response:** Issue a CREATE INDEX statement to define an index on the primary key columns of the designated table.

RBS1695E  MAXROWS PER SEGMENT should be in the range 1 to 4,294,967,295.

    **Cause:** A value outside the valid range for MAXROWS PER SEGMENT was specified.

    **Response:** Specify a value in the range 1 to 4,294,967,295 (inclusive) for MAXROWS PER SEGMENT.

RBS1696E  Missing or invalid primary key index on table table_name.

    **Cause:** Attempted an operation on a table which requires a primary key index on the table, or referenced table, and the index is missing or invalid.

    **Response:** Create, or drop and recreate, a primary key index on the given table.

RBA1933I  Marking precomputed views invalid.

    **Cause:** A REORG statement has deleted rows from the table, which invalidates the precomputed views defined on the table being reorganized. This message indicates that REORG is trying to automatically mark the affected precomputed views invalid.

    **Response:** None required.
RBA1934W  Failed to invalidate all precomputed views.

Cause:     A REORG operation deleted rows from the table being reorganized, which invalidated all precomputed views defined on the table. However, an error occurred that prevented REORG from automatically marking all of the precomputed views invalid. The most likely error is that REORG was unable to obtain the necessary database locks.

Response: Either mark all precomputed views invalid with the SET PRECOMPUTED VIEWS FOR <detailtable_name> command. Or update all precomputed views, even if they are marked valid, to reflect the rows deleted from the detail table. Then mark all precomputed views valid with the SET PRECOMPUTED VIEWS FOR <detailtable_name> VALID command.

RBS1951E  Only one predicate on column colname is allowed per query.

Cause:     When an Advisor system table is queried, some columns can have only a single predicate placed upon them. More than one predicate was provided on the column in this query.

Response: Correct the Advisor query and submit the statement again.

RBS1964E  Table creation date must be less than or equal to END_DATE.

Cause:     The table specified in the advisor query did not exist during the specified date range.

Response: Change or drop the START_DATE or END_DATE constraint and submit the statement again.

RBS3000E  CHECK TABLE not allowed on system tables.

Cause:     User submitted a CHECK TABLE command on a system table for execution.

Response: Check Table on system tables are not supported.

RBS3001E  CHECK TABLE must reference a base table.

Cause:     A CHECK TABLE statement specified an object that is not a base table. Check Table can only operate on base tables, not temporary tables, views, other synonyms, or model tables.

Response: Rewrite the CHECK TABLE statement to reference a base table and submit it again.
RBS3002E  A directory for logging output must be specified.

Cause:  A CHECK TABLE or CHECK INDEX statement must specify a directory for logging output.

Response:  Rewrite the CHECK statement to include a DIRECTORY <log-directory> clause so that output is logged to the log directory. It again.

RBS3003E  The logging directory dir_name must exist and be writeable.

Cause:  The directory for the CHECK TABLE and CHECK INDEX statements must exist and be writeable.

Response:  Ensure that the specified directory exists and is writeable and submit the statement again.

RBS3004E  The segment seg_name is not associated with seg_name obj_name.

Cause:  A segment was specified as part of the CHECK INDEX or CHECK TABLE statement but the segment is not part of the index/table.

Response:  Specify a segment that is part of the index/table and submit the statement again.

RBS3005E  Insufficient authority to CHECK object.

Cause:  The user who issued the CHECK statement did not have the required authority. To check a table or an index, a user must be a member of the DBA system role.

Response:  Notify the warehouse administrator and request the required authority.

RBA6500E  Cannot start a new transaction.

Cause:  The transaction manager is not ready or has terminated.

RBA6505E  The active revision table is full.

Cause:  A long-running transaction is holding up the revision.

Response:  Stop the long running transaction or increase the size of the active revision table. To increase the size of the active revision table, you need to drop and recreate the version log with the new revision table size.
### RBA6506E

**Cannot start a VERSIONING transaction because the database is not enabled for versioning.**

**Cause:** Versioning capability has been turned off by an ALTER DATABASE STOP VERSIONING statement; however, the user is expecting to use versioning through the use of SET VERSIONING ON command or an OPTION VERSIONING ON entry in the rbw.config file.

**Response:**

### RBS6507F

**The current database session and your connection have been terminated.**

**Cause:** You issued the ALTER SYSTEM TERMINATE statement.

**Response:** Use the CONNECT statement to connect to the database again.

### RBS6508E

**Cannot execute statement because other users are connected to the database.**

**Cause:** You can execute this statement only if you are the only user of the database.

**Response:** Terminate all other user sessions. If necessary, quiesce the database. Then enter the statement again.

### RBS6509F

**Database refusing connection; cannot connect to database.**

**Cause:** Another user is doing something to the database that causes it to refuse connections.

**Response:** Try to connect again later.

### RBS6510E

**Segment segment_name is currently designated as the version log segment.**

**Cause:** A database can have only one version log segment. An attempt was made to add a version log segment to a database that already has a version log segment.

**Response:** If there is an error during an ALTER DATABASE CREATE VERSION LOG command, the system may create a version log segment but it cannot be enabled as a version log. If this is the case, use an ALTER DATABASE DROP VERSION LOG command to remove the current version log segment before creating a new version log.
RBS6511E  Segment segment_name is in use.

Causes:  You attempted to use a segment that is already in use.

Response:  Check the status of the segment in the RBW_SEGMENTS table.

RBS6512E  Not enough disk space for the log segment.

Causes:  When the log segment is created, all of its PSUs are expanded to their maximum size (MAXSIZE). This error occurs if there is not enough room on disk for the entire log segment.

Response:  Drop the version log, redefine the log segment PSUs so that there is enough disk space for them to be expanded to their maximum size, and then specify the segment as the log segment with the ALTER DATABASE CREATE VERSION LOG command.

RBS6513E  Database does not have a version log.

Causes:  You attempted to enable or drop the version log when it does not exist.

Response:  Create version log and enter the statement again. If there is an error during an ALTER DATABASE CREATE VERSION LOG command, a version log segment might exist but it cannot be enabled as the version log. If this is the case, use an ALTER DATABASE DROP VERSION LOG command, and then create the version log again.

RBS6514E  Cannot drop version log because versioning is enabled or the log is not empty.

Causes:  The version log cannot be dropped because versioning is enabled or the log is not empty.

Response:  Check that versioning is disabled; then wait until the version log is empty and enter the DROP statement again.

RBS6515E  Cannot stop vacuum cleaning because it may be cleaning for a blocking transaction.

Causes:  The vacuum cleaner is currently helping a blocking transaction to clean the version log for the blocks it is going to modify.

Response:  After the blocking transaction has finished, enter the STOP CLEANING command again.
Messages and Codes Reference Guide
Chapter 2, “Informational, Warning, and Error Messages”

RBA6516F  RBWAPID is not running or not available.

Cause:  The RBWAPID daemon is not running or it is being shut down.

Response:  Make sure that the RBWAPID daemon is properly started.

RBS6517E  Cannot perform operation on segment segment_name, which contains the version log.

Cause:  A DROP SEGMENT or ALTER SEGMENT statement with an option that is valid only for non-VERSION-LOG segments was issued for the VERSION LOG segment.

Response:  Carefully review what you are trying to do.

RBS6518E  Cannot start vacuum cleaning because stop cleaning is in progress.

Cause:  Another user issued a STOP CLEANING command, which has not yet finished.

Response:  Wait for the STOP CLEANING command to finish and then enter the START CLEANING command again.

RBS6520F  Error initializing lock file 'file_name': error_text.

Cause:  The database lock file could not be initialized.

Response:  Check the file permissions on the file and check the owner and setuid bit on the Red Brick Warehouse server executable file (rbwsvr). The owner of the Red Brick Warehouse server executable file must be able to create, remove, read, and write the specified lock file.

RBS6521F  Operating system error setting or releasing lock on 'file_name': error_text.

Cause:  The operating system returned an error when Red Brick Warehouse attempted to set a lock on the database lock file.

Response:  The indicated error might be the result of a problem with the configuration of the platform on which Red Brick Warehouse is executing. If so, fix the problem and start the warehouse daemon again. If unable to determine the problem, contact the Red Brick Customer Support Center for assistance.
RBS6522E  Insufficient authority to perform ALTER DATABASE operation.

Cause: The user who issued the ALTER DATABASE statement did not have the required authority. To use an ALTER DATABASE command, a user must have ALTER_SYSTEM task authority.

Response: Notify the warehouse administrator and request the required authority.

RBA6523F  Reached the maximum number of active databases.

Cause: The limit on the number of active databases has been reached.

Response: Either use an ALTER DATABASE TERMINATE command to stop databases that are not required to be active or else increase the value of the MAX_ACTIVE_DATABASES parameter in the rbw.config file and start the RBWAPID daemon again.

RBA6524F  Reached the maximum number of database sessions.

Cause: The limit on the total number of database sessions has been reached.

Response: Review the DST_SESSIONS table and kill any idle database sessions.

RBA6525E  Vacuum cleaner daemon is not running.

Cause: The backup process is started when the vacuum cleaner daemon is not running and the version log is not empty.

Response: Start vacuum cleaner daemon and run backup again.
RBA6526F  The shared state of the current database may be corrupt. Please logout immediately!

Cause: An error has been discovered in the shared state of the current database. Some possible causes are: - A process died abnormally while modifying the shared memory; - the version log is corrupted (should see message 6543); or - cannot start a database during the first connection—that is, the database starter daemon got a fatal error (such as not enough shared memory, Locale is invalid (message 1800)) during the startup.

Response: Use the following procedure to recover:

1. Issue an ALTER SYSTEM QUIESCE DATABASE <database> command from the ADMIN database to stop new connections.
2. Look at the RBWLOG file (the last error before seeing the message 6526) to identify what corrupted the database shared memory.
3. Shutdown the database by issuing an ALTER DATABASE TERMINATE DATABASE <database> command from the ADMIN database.
4. Fix the problem found in step (2) if possible.
5. Issue an ALTER SYSTEM RESUME DATABASE <database> command from the ADMIN database to allow new connection to the database.
6. Connect to the database again from the RISQL Entry Tool, which executes the recovery mechanism.

RBA6527F  The current database is being closed down.

Cause: The current database is being closed down by another user.

Response: Wait until the database has been completely closed down. Then try to connect again, which will restart the database.

RBS6528E  The maximum number of active revisions should be at least min_number.

Cause: The value specified for the maximum number of active revisions is too small.

Response: Issue the command again with a larger value.
**Messages and Codes Reference Guide**  
*Chapter 2, “Informational, Warning, and Error Messages”*

**RBS6529E** This command cannot be issued from the ADMIN database.  
**Cause:** A system management operation was attempted in the ADMIN database.  
**Response:** Connect to the appropriate database and enter the command again.

**RBS6530E** This command must be issued from the ADMIN database.  
**Cause:** You can only administer another database from the ADMIN database.  
**Response:** Connect to the ADMIN database and enter the command again.

**RBA6531E** Version log full. Statement aborted.  
**Cause:** The version log is full so the statement has been aborted.  
**Response:** Either make the version log bigger or else wait until other versioning statements have completed, and then enter the statement again.

**RBA6532E** Out of vacuum cleaner shared resources.  
**Cause:** The vacuum cleaner infrastructure has run out of shared resources. Either the vacuum cleaner is not running or else it is not able to keep up with the changes.  
**Response:** Wait for a few minutes and try the operation again.

**RBA6533F** This program is not compatible with the RBWAPID daemon.  
**Cause:** This current executable file is not compatible with the rest of the Red Brick software.  
**Response:** Check your path and ensure that you are using the correct version of the current program.

**RBA6534F** Not able to allocate enough shared memory.  
**Cause:** The shared memory limit has been reached.  
**Response:** Increase the operating system shared memory segment limit or the system pagefile size.
RBA6535F  Unable to find the SHMEM declaration for the given RB_HOST in the config file.

Response: Cannot find the shared memory declaration (SHMEM) for the warehouse logical name. Add a SHMEM declaration for the desired RB_HOST entry in the rbw.config file.

RBA6540F  Session terminated, shared memory may be corrupt.

Cause: A process died while it was modifying shared memory. It is not safe to use the contents of the current shared memory.

Response: Stop the RBWAPID daemon which will terminate use of the current shared memory, then restart the RBWAPID daemon.

RBL6541F  Version log exists. Cannot perform UPGRADE unless version log is dropped.

Cause: An UPGRADE operation was attempted on a database with a version log.

Response: Drop the version log and then run the UPGRADE operation again.

RBA6542W  All previous changes have been rolled back.

Cause: All changes for this operation have been rolled back because a transaction was aborted.

Response: None necessary.
RBA6543F  The version log is damaged. The shared resources of the database have been marked corrupted.

**Cause:** The version log segment is damaged. The database cannot be used until this problem has been fixed.

**Response:** Use the following procedure:

1. Use an `ALTER SYSTEM QUIESCE DATABASE <database>` command from the ADMIN database to stop new connections.
2. Shut down the database with an `ALTER DATABASE TERMINATE` command from the ADMIN database.
3. Fix the damaged version log segment.
4. Issue an `ALTER SYSTEM RESUME DATABASE <database>` command from the ADMIN database to re-allow new connection to the database.
5. Connect to the database again from the RISQL Entry Tool, which executes the recovery mechanism.

RBA6544F  The database entry has inconsistent information.

**Cause:** This condition can occur when a previous database directory got removed by the normal file deletion command, rather than the Redbrick database deletion tool.

**Response:** Either issue an `ALTER DATABASE TERMINATE DATABASE <database>` command from the ADMIN database to remove the database entry or restart APID. Then reconnect to the database again.

RBA6545E  Command not allowed on a database with a version log.

**Cause:** This command cannot be performed on a database with a version log.

**Response:** Drop version log and proceed with the operation.

RBA6546E  Not able to start a vacuum cleaner daemon.

**Cause:** An error occurred during an attempt to start a vacuum cleaner daemon.

**Response:** Find the error in the rbwlog file and correct the problem that caused the error. Then submit the command again.
Messages and Codes Reference Guide
Chapter 2, “Informational, Warning, and Error Messages”

RBA6547E The size of the version log PSU id psu_num1 must be a multiple of the operating system page size.

Cause: The specified PSU is memory mapped by the version log, and the size of the PSU must be a multiple of the Operating System Page Size.

Response: Drop the version log, and change the MAXSIZE of the specified PSU so that its size is a multiple of the operating system page size. Then create the version Log again.

RBA6991I Number of active connections ...: connections

Response: None necessary.

RBA6992I Number of active database ......: databases

Response: None necessary.

RBA6993I Active connections to database ‘dbname’.

Response: None necessary.

RBA6994E rbshow: Missing argument for -dbc option.

Response: None necessary.

RBA6995W Cannot delete database 'dbname'. Database is in use.

Response: None necessary.

RBS7008E SQL Backtrack API version btapi_version is not compatible with RBRAPI version rbrapi_version.

Cause: The backtrack command used is not compatible to the RBR API version installed.

Response: Install the correct Backtrack or the RBR API library.

RBL8520F Error on removal of temp discardfile filename. OS Errno = errno.

Response: The unlink() system call of a temp file failed. Check the OS code and correct the problem.
RBL8550I  User Cancelled TMU operation.

Cause: The TMU received a cancellation event from the user on an NT system.

Response: The TMU will clean up the LOAD operation in progress and then terminate.

RBL8551W  TMU row level messages mode is NONE.

Cause: The TMU_ROW_MESSAGES is NONE. Row level messages will not be printed.

Response: Turn TMU_ROW_MESSAGES to FULL if the messages are desired.

RBL8552F  Current installation has been upgraded past the current offline load.

Cause: The offline load was done in a version pre-dating the sync. The sync cannot be performed.

Response: Do the SYNC using the old version.

RBL8553W  Insufficient resources to memory map all the primary keys of referenced tables.

Cause: To enhance performance, an attempt was made to access the primary key index of a referenced table by mapping its index files into the process memory space. This attempt failed due to resource limitations, most likely limitations on virtual memory usage. The primary key indexes will instead be accessed using the buffer cache which may degrade performance somewhat. This is most likely to happen when a referenced table has a very large primary key index.

Response: If the primary key indexes on tables referenced by the operation are not very large, you may be able to increase resource limits to allow more and/or larger files to be accessed via memory mapping. For example, the C shell command 'limit vmemoryuse unlimited' or the Bourne shell command 'ulimit' can be used on some UNIX platforms to increase the amount of virtual memory available to a process.

RBL8600E  Index index_name was specified more than once.

Cause: The list of indexes specified in the statement contains a duplicate index name.

Response: Correct the statement and submit it again.
RBL8601I Referential integrity checked only for the foreign keys used in the STAR indexes being reorganized.

Cause: A TMU REORG statement specified REFERENCE CHECKING OFF and included one or more STAR indexes to be rebuilt. Referential integrity is checked for each foreign key used in the index key of a STAR index being rebuilt. Referential integrity is not checked for any other foreign key.

Response: Information only.

RBL8602I Reorganization of table table_name terminating early because table does not have any indexes to rebuild and no referential integrity checking was requested.

Cause: The REORG statement is terminating without rebuilding any indexes and checking any rows for referential integrity violations. No indexes are rebuilt because there are no indexes defined on the table. Referential integrity checking is not done because either it has been disabled in the REORG statement or the table does not have any foreign keys to check.

Response: None necessary.

RBL8603W Task usertask interrupted.

Cause: The specified REORG task terminated before completing its operation because an error or interrupt occurred in a related REORG task. Another message will be issued that describes the error in the other task.

Response: Examine the error message issued by the other task and correct that problem.

RBL8604E An error was encountered during the reorganization of table table_name.

Cause: The parallel REORG operation cannot be completed because of an interrupt or error in one of its worker tasks. The interrupt or error caused the task to terminate prematurely. Another message will be issued that gives the reason for the termination of the task.

Response: Examine the message that describes the termination of the task and correct the problem. Check whether the indexes rebuilt by the REORG are marked valid. If they are marked valid, the indexes were rebuilt correctly and can be used. If they are marked invalid, the indexes must be rebuilt.
RBL8605E  Task usertask terminated after receiving signal number sig_number.

Cause: The TMU received a termination signal such as SIGINT(2), SIGHUP(1), SIGQUIT(3), or SIGTERM(15). SIGINT and SIGQUIT are usually generated by the terminal as the responses to the interrupt and quit characters (usually Control-C and Control-\). SIGHUP is usually the result of a disconnected terminal.

Response: The TMU will terminate the reorganization of the table and mark all indexes being rebuilt invalid.

RBL8606E  Parallel REORG received operating system error os_error while trying to create task usertask.

Cause: A parallel REORG command was not able to create the indicated task due to resource limits in the operating system.

Response: Evaluate the operating system configuration to determine why the tasks could not be created. Then try one of the following:

1. Alter the operating system configuration to allow the tasks to be created. 2. Reorganize the table at a different time when there is less contention for operating system resources. 3. Limit the parallelism of the REORG. For example, run REORG in serial mode.

RBL8607W  Row number number in segment 'segmentname' contains a duplicate key value for index 'indexname'.

Cause: The key value for the specified row already exists in the named index. The index is a unique index, so the key value for this row cannot be inserted into the index. If the ON DISCARD DELETE ROW option was selected for the REORG then the row with the duplicate key value will be deleted from the table. Otherwise, the index named in the message will be marked invalid at the end of the REORG.

Response: If the index is invalid at the end of the REORG, delete the row from the table and REORG the table again.

RBL8608F  The DISCARDS limit has been reached.

Cause: The maximum number of DISCARDS allowed for the REORG has been reached. Rows can be discarded either because they fail the checks for referential integrity or because they contain duplicate index key values.

Response: Check the discard file to determine the source of excess errors.
RBL8609F  REORG failed to rebuild any of the specified indexes.

Cause:   Errors were encountered while trying to rebuild each of the indexes specified in the REORG statement. Other messages preceding this one explain the specific errors encountered for each index. REORG could not complete any useful work, and so it terminated.

Response: Fix the errors identified in the previous messages or run the REORG with the ON DISCARDS DELETE ROWS option specified.

RBL8610F  DISCARD detected. REORG terminating.

Cause:   The REORG terminated prematurely because a DISCARD condition was encountered and the ON DISCARD option was set to ABORT. The DISCARD condition was either a referential integrity violation, a duplicate index key in a unique index, or a insufficient space available to build an index. The actual DISCARD condition encountered is reported in a previous message.

Response: Fix the error identified in the previous message.

RBL8611E  DISCARDFILES are not allowed with the ON DISCARD ABORT or INVALIDATE INDEX options.

Cause:   A DISCARDFILE clause was specified in a REORG statement that selected the ON DISCARD ABORT or ON DISCARD INVALIDATE INDEX option. The DISCARDFILE clause may have been specified in the OPTIMIZE clause, in its own separate clause, or in both places. Neither clause is allowed. DISCARD files are allowed only with the ON DISCARD DELETE ROW option.

Response: Remove the DISCARDFILE clause from the REORG statement.

RBL8612E  REORG terminated because the portion of the INDEX TEMPSPACE MAXSPILLSIZE allocated to duplicates has been exceeded.

Cause:   A REORG operation has run out of spill space on disk to store rows with duplicate index key values.

Response: Increase the INDEX TEMPSPACE DUPLICATESPILLPERCENT or the INDEX TEMPSPACE MAXSPILLSIZE and run the REORG operation again. Or as an alternative, DELETE the rows containing duplicate index key values and run the REORG operation again.
RBL8613W  Total number of rows deleted: number.

Cause: The table being reorganized contained rows that violated referential integrity constraints and/or contained duplicate index key values. The REORG statement was run with the ON DISCARD DELETE ROW option selected, so the discarded rows were deleted from the table. This message indicates the number of rows deleted. If a discard file was specified in the REORG statement, the individual deleted rows are recorded in the file.

Response: None needed.

RBL8614I  Index name successfully rebuilt.

Cause: Index successfully reorganized. The REORG operation was successful in rebuilding the named index.

Response: None needed.

RBL8615W  Failed to rebuild index name.

Cause: The reorganization of the index failed. The REORG operation was not able to rebuild the named index. The cause of the failure is given in an earlier message. The index is marked invalid and cannot be used in queries until it has been successfully rebuilt.

Response: Fix the problem identified in the earlier message and submit the REORG command again.

RBL8619E  MAX TASKS value too small.

Cause: The user specified value for MAX TASKS must be greater than the sum of SCAN TASKS and INDEX TASKS values if these values are specified, and at least 2. The MAX TASKS value is the maximum number of SCAN and INDEX tasks combined which will be used by REORG.

Response: Make sure that the MAX TASKS value meets these requirements, and submit the REORG command again.
RBL8620I  Rows reorganized: totalcount. RI violations: ricount. Duplicate keys: dupcount. Reorganization continues...

Cause:  This message reports intermediate status of a REORG. The first value reports the number of rows that have been read and processed without any referential integrity violations being detected. If reference checking is completely disabled, this value is a count of all rows read. The second value reports the number of rows that have failed an explicit referential integrity check. If the ON DISCARD DELETE ROW option was selected, these rows have been deleted. The third value indicates the number of duplicate index key values that have been detected; the rows with the duplicate keys have not yet been deleted.

Response:  None needed.

RBL8621I  Duplicate rows discarded: number. Reorganization continues...

Cause:  This message reports intermediate status of a REORG during its duplicate removal phase. It reports the number of rows that have been deleted so far due to duplicate index key values.

Response:  None needed.

RBL8622I  Total number of rows processed: number.

Cause:  This message reports the total number of rows read from the reorganized table.

RBL8623I  Number of rows that passed the referential integrity check: number.

RBL8624W  Number of rows that failed the referential integrity check: number.
RBL8625W  Number of duplicate index key values detected: number.

Cause: A REORG operation detected the indicated number of duplicate key values in the unique indexes being reorganized. Unique indexes include the index defined on the primary key of the table, as well as indexes whose key columns are defined as unique. If a single row of the table contains duplicate key values for more than one index, each duplicate value will be counted separately.

Response: The rows with duplicate key values are either reported in previous messages or written to the discard file specified in the REORG statement. If either the ON DISCARD INVALIDATE INDEX option or the ABORT option was specified, REORG may not report all rows with duplicate key values. If the ON DISCARD DELETE ROW option was specified, the rows with duplicate key values will be deleted from the table.

RBL8626I  Starting duplicate removal phase.

Cause: Indicates that the TMU has started the final phase of duplicate removal during a REORG operation. In some cases, this duplicate removal phase can take substantial time.

Response: Information only.

RBL8627E  Cannot delete a row with a referential integrity violation because some indexes are not being rebuilt. Run a REORG on all indexes of table table_name.

Cause: A REORG statement specified a partial list of indexes to be reorganized (that is, not all indexes associated with the table were specified), but a referential integrity violation was detected during the operation. REORG operations cannot correct referential integrity violations if any STAR indexes are defined on the table and are not being rebuilt. So the REORG operation was terminated before it finished. The table is left in an invalid state until a REORG of all indexes defined on the table is completed.

Response: Perform a REORG operation that does not list specific indexes or that lists all indexes associated with the target table.
RBL8628W Cannot delete duplicate row from index index_name because of a referential integrity violation. REORG this index.

Cause: A referential integrity violation was detected while trying to delete a duplicate row from the named STAR index. A row was not found in the referenced table with a primary key value that matches the foreign key value in the duplicate row, so the row could not be deleted from the STAR index. The row was, however, deleted from the table and the other indexes defined on the table.

Response: Perform a REORG operation on the named index.

RBL8657E Cannot specify the RECALCULATE RANGES option when versioning is enabled.

Cause: Versioning has been enabled for a REORG operation in which the RECALCULATE RANGES option has been specified. REORG must be run as a blocking transaction when the RECALCULATE RANGES option is specified.

Response: Correct the statement and submit it again.

RBL8700W Truncating constant string specified for column column_name.

Cause: Constant string is longer than the maximum specified for the column in the CREATE TABLE statement.

Response: Warning only. Review constant string specified in load command.

RBL8701W Unsupported EBCDIC character char found in row row_number at position byte_pos.

Cause: When TMU_DISCARD_EXTENDED_EBCDIC mode is set, an attempt was made to load EBCDIC data which is outside the supported character set. Red Brick TMU supports EBCDIC to ASCII conversion only for characters in the IBM CS 640 character set plus exclamation point (!).

Response: The input row containing the unsupported EBCDIC character will be discarded.
RBL8702E Discard in ASCII for format IBM load is incompatible with TMU DISCARD EXTENDED EBCDIC ON.

Cause: The TMU_DISCARD_EXTENDED_EBCDIC mode discards rows which do not have a supported conversion to ASCII in Red Brick TMU. A row discarded due to this condition cannot meaningfully be converted to ASCII in the discard file. CREATE TABLE statement.

Response: Either discard in EBCDIC or set TMU_DISCARD_EXTENDED_EBCDIC OFF.

RBL8703E Format IBM cannot be specified together with NLS_LOCALE.

Cause: Format IBM in the format clause is used to indicate that the input data is EBCDIC. NLS_LOCALE in the locale clause is an alternate method to specify the codepage of the input data. Both FORMAT IBM and NLS_LOCALE may not be specified together in a load data statement. To load EBCDIC data, choose an NLS_LOCALE with an EBCDIC codepage OR specify FORMAT IBM.

Response: Choose FORMAT IBM or NLS_LOCALE to characterize the input data.

RBL8704E Invalid hex literal specified for NULLIF field on line \line_number.

Cause: A NULLIF keyword was included in a LOAD DATA field specification, but the literal string or hex string was invalid because it contained a character other than '0' to '9' or 'a' to 'f' or 'A' to 'F'.

Response: Correct the invalid hex literal specification in the LOAD DATA statement.

RBL8705E The value for COMMIT RECORD INTERVAL must be a positive integer or OFF.

Cause: An invalid value was specified for COMMIT RECORD INTERVAL set command in the load control file. The set command take the keyword "OFF" or a positive integer as valid input.

Response: Correct the invalid value in the SET TMU command.
RBL8706E  The value for COMMIT TIME INTERVAL must be a positive integer or OFF.

**Cause:** An invalid value was specified for COMMIT TIME INTERVAL set command in the load control file. The set command take the keyword "OFF" or a positive integer as valid input.

**Response:** Correct the invalid value in the SET TMU command.

RBL8707I  Versioning is ON.

RBL8708I  Performing interval commit

**Cause:** The TMU operation was set to perform interval commits by the TMU commit record interval or the TMU commit time interval options. One of the intervals has expired and the TMU is pausing to commit the accumulated changes.

RBL8709I  Commit complete. Loading continues...

RBL8710I  Interval commit set to record_count records.

RBL8711I  Interval commit set to minute_count minutes.

RBL8712F  Terminating PTMU because of failure to start all child threads.

**Response:** One or more child threads could not initialize without error.

RBL8713F  Child thread terminating due to exception exception_code

**Cause:** Child thread encountered fatal exception.

RBO10070E  Invalid argument specified value.

RBO10150E  UNIQUE_ID buffer too small, size: size

RBO10171W  Unrecognized table type 'type'.

**Cause:** The application passed an unsupported table type to SQLTables. The driver will ignore this table type when constructing its query.

**Response:** Ensure that the application's functionality does not depend on enumerating tables of the unsupported type.

RBO10403E  No such service.

**Response:** Check services file and modify it if necessary.
RBO10405E Internal error: If used, both send_data and haveSection must be present; or missing haveResult function.

RBO10406E Host not found: host_name.

**Cause:** Check hosts file and modify it if necessary. The specified host name is neither a valid dotted-decimal IP address nor a host name string recognized by gethostbyname().

**Response:** Make sure that you have specified the server's host:port string correctly. Make sure that DNS is working properly on the client machine. Try to 'ping' the host name.

RBO10407E Could not create socket. os error:errno.

RBO10408E Could not connect to the server. os error:errno.

**Response:** Check the host and the service and run the application again.

RBO10410E Could not receive from the server. OS error:errno.

RBO10411E Internal error calling SCFE have_result:errno.

RBO10412E Internal error calling SCFE have_section:errno.

RBO10415E Internal error: Bad protocol: OS error:errno.

RBO10416E Protocol error: incorrect handshake with server.

RBO10417E Attempt to signal server process failed. OS error:errno.

RBO10419E Could not open local connection to the daemon. OS error:errno.

RBO10420E Could not write to daemon thru local connection. OS error:errno.

RBO10425E Error checking if the host is same as local machine.

RBO10426E Invalid server specification. Correct format is '<host>:<port>' or '<port>'

RBO10429E Error in setting socket option. os error:errno.
RBO10430W Could not connect to server via UDS "UDS name", OS error: OS message (errno). Retrying with inet socket.

Cause: The ODBC library is unable to establish a connection with the local server via the stated UNIX-domain socket (UDS), for the reason stated.

Response: Check to ensure that the server is running. If it is not, the UDS will not exist. Check to ensure that RB_CONFIG is correct for the server of interest. The UDS is assumed to be located in the directory named in RB_CONFIG. If the connection string passed to SQLDriverConnect contains an explicit RB_CONFIG=value, that will be used; else if there is an RB_CONFIG item in the data-source definition, that will be used; else the value of the environment variable RB_CONFIG will be used. (Note: risql and risqlrpt explicitly pass the value of the environment variable RB_CONFIG to SQLDriverConnect, which prioritizes the environment variable over the other two choices for those applications.) The client will retry by attempting to connect via an inet socket instead.

RBO10431F Cannot read local host name. OS error: OS message (errno).

Cause: The gethostname function failed for the reason stated in the message.

Response: Make sure that the protocol stack you are using is one of the ones supported for Red Brick’s Client Connectivity Pack.

RBO10432F Error waiting for socket. OS error: OS message (errno).

Cause: The TCP/IP select() function failed for the reason stated in the message.

Response: Make sure that the server to which you were connected is still running. Make sure that the protocol stack you are using is one of the ones supported for Red Brick’s Client Connectivity Pack.

RBO10433F Internal error: select() returned inconsistent results.

Cause: The TCP/IP select() function implies that the socket is in an unexpected state.

Response: Contact the Red Brick Customer Support Center.
RBO10434F The server is no longer available.

**Cause:** The TCP/IP select() function reports that an exception has occurred on the socket. This usually means that the server has shut down.

**Response:** Contact the Red Brick Customer Support Center.

RBO10435F Internal error: getCIUs callback returned error.

**Cause:** An internal function has returned an unrecognized code.

**Response:** Contact the Red Brick Customer Support Center.

RBO10436W RB_CONFIG directory "RB_CONFIG" does not exist.

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because RB_CONFIG does not identify a directory.

**Response:** Correct the RB_CONFIG setting and retry.

RBO10437W Cannot access RB_CONFIG directory "RB_CONFIG": OS message (errno).

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because RB_CONFIG could not be accessed for the reason given.

**Response:** Correct the RB_CONFIG setting and retry.

RBO10438W The file identified by RB_CONFIG "RB_CONFIG" is not a directory.

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because RB_CONFIG names something that is not a directory.

**Response:** Correct the RB_CONFIG setting and retry.
Messages and Codes Reference Guide
Chapter 2, “Informational, Warning, and Error Messages”

RBO10439W There is no UNIX-domain socket named "UDS name".

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because there is no socket with the correct name in the RB_CONFIG directory. The specified port number might be incorrect, or the server might not be running at this time.

**Response:** Check that RB_CONFIG identifies the correct directory. Check that the port number (embedded in the UDS name) is correct. Check that the server’s API daemon is running.

RBO10440W Cannot access the UNIX-domain socket named "UDS name": OS message (errno).

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because although there seems to be an entry with the correct name in the RB_CONFIG directory, it cannot be accessed for the reason given.

**Response:** Check that RB_CONFIG identifies the correct directory. Ensure that no other person or program has created a file with the name shown in that directory.

RBO10441W The file named "UDS name" is not a UNIX-domain socket.

**Cause:** When attempting to connect to the server with a UNIX-domain socket, the socket could not be opened because although there is an entry with the correct name in the RB_CONFIG directory, it does not identify a UNIX-domain socket.

**Response:** Check that RB_CONFIG identifies the correct directory. Ensure that no other person or program has created a file with the name shown in that directory.

RBO10442F The server has unexpectedly closed the socket. It may have stopped running.

**Cause:** When the client software attempts a read or write operation to the socket and gets an error, but the error number is zero, it generally means that the other end of the socket has been closed. The server may or may not still be running, but in any case we can no longer talk to it.

**Response:** Examine previous errors to see if any were fatal to the server. If there is no obvious explanation for why the server might have stopped running, contact the Red Brick Customer Support Center.
RBO10443E  An internal error has occurred while attempting to receive incoming data.

  **Cause:** This is an internal error.
  **Response:** Contact the Red Brick Customer Support Center.

RBO10444F  A fatal internal error has occurred while attempting to receive incoming data.

  **Cause:** This is an internal error.
  **Response:** Contact the Red Brick Customer Support Center.

RBO10445F  Error calling SCFE section_buff:errno.

  **Cause:** This is an internal error.
  **Response:** Contact the Red Brick Customer Support Center.

RBL12001S  Maximum connection limit for username has been reached.

  **Cause:** The maximum number of connections allowed for this user has reached. This maximum number is determined by the license for this user.
  **Response:** Either try to connect again later or close some of the sessions.
**Index**

A
- ABORT keyword, TMU REORG 5-11
- ADMIN ADVISOR_LOGGING parameter, modifying 3-12, 3-15
- ADMIN parameters, modifying 3-16
- ALLOW_POSSIBLE_DEADLOCKS parameter, modifying 3-13, 3-16
- ALTER DATABASE command
  - description 4-2
  - syntax description 4-3
- ALTER DATABASE CREATE BACKUP DATA command 4-3
- ALTER DATABASE DROP BACKUP DATA command 4-3
- ARITHABORT parameter, modifying 3-13, 3-16
- AUTO_INVALIDATE_PRECOMPUTED_VIEW parameter, modifying 3-13, 3-16
- AUTOROWGEN parameter
  - modifying 3-14, 3-16

B
- BackOffice, Microsoft 3-6
- backup segment
  - creating 4-2
  - dropping data 4-2
- BIN directory, warehouse 6-9

C
- cases, tracked by technical support xviii
- CLEANUP_SCRIPT parameter, modifying file 3-12, 3-15
- commit intervals 5-27
- CONCAT function 4-19
- connections limit, changing 3-12
- control-c coordination thread 3-9
- conventions
  - syntax diagrams xiv
  - syntax notation xiii
- costs, of version log 2-2
- CPUs, allocation for parallel queries 9-30
- CREATE INDEX command, syntax description 4-8
- CROSS_JOIN parameter, modifying 3-13
- Customer Support Center xvii

D
- daemon processes
  - general discussion 3-4
  - See also rbwapid, rbw.servermon, rbwlogd, rbwadmd daemon
- data processing 5-5 to 5-8
- datetime literals, nonstandard 4-22
- DEC function 4-12
- DEFAULT_DATA_SEGMENT parameter, modifying 3-13, 3-16
- DEFAULT_INDEX_SEGMENT parameter, modifying 3-13, 3-16
- DEFERRED INDEXES keyword, TMU REORG 5-10
- DELETE operations, versioned 2-9
- DELETE ROW keyword, TMU REORG 5-12
- Digital UNIX, kernel
  - parameters 6-2 to 6-3
- dimension table cleansing 2-4
- discard files, TMU, referential integrity
  - discards 5-13
- discardfile clause, syntax 5-12
- DISCARDFILE keyword, TMU REORG 5-12
Index

DISCARDS keyword, TMU REORG 5-14
disk groups
  defining 9-4
  effect on parallel queries 9-25
  specifying processes 9-6
disk spill files
INDEX TEMPSPACE parameters 5-18, 5-19
disk usage, parallel queries 9-27
documentation
  list of Red Brick Systems xi
  support xix
DST_COMMANDS table 3-29
DST_DATABASES table 3-32
DST_LOCKS table 3-35
DST_SESSIONS table 3-36
dynamic statistic tables
  DST_COMMANDS 3-29
  DST_DATABASES 3-32
  DST_LOCKS 3-35
  DST_SESSIONS 3-36
F
file groups, See disk groups
FILE_GROUP parameter
  described 9-2
  modifying 3-13
  syntax and usage 9-4
  See also disk groups
FILLFACTOR parameters,
  modifying 3-14, 3-16
FORCE_AGGREGATION_TASKS parameter
  described 9-2
  modifying 3-13
FORCE_FETCH_TASKS parameter
  described 9-2
  modifying 3-13
FORCE_HASHJOIN_TASKS parameter
  described 9-22
  modifying 3-13
FORCE_JOIN_TASKS parameter
  described 9-2
  modifying 3-13
FORCE_SCAN_TASKS parameter
  described 9-2
  modifying 3-13
G
GROUP parameter
  described 9-2
  syntax and usage 9-6
H
HP-UX, kernel parameters 6-4 to 6-5
I
  I/O contention, parallel queries 9-27
  IGNORE_PARTIAL_INDEXES parameter,
    modifying 3-13, 3-16
  index name clause, TMU REORG 5-10
INDEX_TEMPSPACE parameters
  DIRECTORY 5-19
  MAXSPILLSIZE 5-20
  modifying 3-14, 3-16
  THRESHOLD 5-19
  TMU control 5-18
INDEX_TEMPSPACE_DUPLICATE
  SPILLPERCENT parameter
  rbw.config file entry 3-19
  syntax 5-19
INDEX_TEMPSPACE_MAXSPILLSIZE parameter, TMU control 5-18
INDEX_TEMPSPACE_THRESHOLD parameter, TMU control 5-18
indexes
  creating 4-8
  default names 5-10
  DEFERRED 5-2
  rebuilding with REORG 5-2
  INT function 4-13, 4-15
  INTERVAL parameter, modifying 3-12
  invalid STAR indexes, cause of 5-3
  INVALIDATE INDEX keyword, TMU REORG 5-11
IRIX, kernel parameters 6-8
isolation level 2-15

K
kernel parameters
  Digital UNIX 6-2 to 6-3
  HP-UX 6-4 to 6-5
  IRIX 6-8
  Solaris 6-6 to 6-7

keywords, in syntax diagrams xvi

L
licensed options, enabling 3-12, 3-15
listener thread 3-9
load window
  benefit with versioned databases 5-22
  increasing 2-3
load_processes parameter, defined 6-3, 6-5, 6-7
loading data
discard files 5-13
processing flow 5-7
LOCALE parameter, modifying 3-12, 3-15
LOGFILE_SIZE parameter,
  modifying 3-12, 3-15

M
MAPFILE parameter, changing 3-12
MAX_ACTIVE_DATABASES parameter
  changing 3-12, 3-15
  rbw.config file entry 3-18
max_active_databases parameter,
  defined 6-3, 6-4, 6-5, 6-6, 6-7, 6-8
max_nprocs parameter 6-6
max_nuniq_idx parameter, defined 6-3, 6-5, 6-7
max_parallel_tasks parameter,
  defined 6-3, 6-5, 6-6, 6-8
max_PSUs parameter, defined 6-4
max_query_tasks parameter, defined 6-3, 6-5, 6-7
max_rbw_users parameter, defined 6-3, 6-4, 6-6, 6-8
MAX_SERVERS parameter,
  changing 3-12, 3-15
maxproc parameter 6-2
MAXROWS_PER_SEGMENT parameter,
  effect on STAR indexes 5-3
MAXSPILLSIZE value 5-20
memory use, parallel queries 9-28
MESSAGE_DIR parameter,
  modifying 3-12, 3-15
metavariables, in syntax diagrams xvi
Microsoft BackOffice 3-6

N
notation conventions xiii
nproc parameter 6-4
num_cpus parameter, defined 6-3, 6-5, 6-7, 6-8
numeric scalar functions
DEC 4-12
INT 4-13, 4-15

O
ON_DISCARD keyword, TMU
  REORG 5-11
OPTIMIZE keyword, TMU REORG 5-11
OPTION ADVISOR_LOGGING parameter,
  modifying 3-13, 3-16
OPTION parameters
  MAX_ACTIVE_DATABASES 3-18
  TMU_COMMIT_RECORD_INTERVAL 3-18
  TMU_COMMIT_TIME_INTERVAL 3-18
  TMU_VERSIONING 3-18
  TRANSACTION_ISOLATION_LEVEL 3-19
  VERSIONING 3-18
OTHER keyword, TMU REORG 5-13

P
parallel queries, allocating tasks
  for 9-20 to 9-21
parallel query processing
  limiting processes 9-8
  minimum row requirements 9-10
  reducing disk contention 9-4
  system limitations 9-25
  tuning 9-1 to 9-35
PARALLEL_HASHJOIN parameter,
  enabling parallel hash joins 9-22
parallelism on demand 9-2
PARTIAL_AVAILABILITY parameter,
  modifying 3-13, 3-16
PARTITIONED_PARALLEL_AGGREGATION parameter
  described 9-2
  modifying 3-13
PASSWORD parameters, modifying 3-14, 3-16
performance
  parallel query tuning 9-1 to 9-35
  parallel table scan 9-33
  resource and workload
  analysis 9-27 to 9-30
  STARjoin queries 9-31
  SuperScan technology 9-34
physical storage units, See PSUs 3-25
PRECOMPUTED_VIEW_QUERY_REWRITE parameter,
  modifying 3-13, 3-16
process checker daemon 3-5
process checker thread 3-9
PROCESS_CHECKING_INTERVAL parameter
  modifying 3-12, 3-15
  rbw.config file entry 3-18
  processes for parallel queries,
  allocating 9-20 to 9-21
  PSUs, sequence ID 3-25

Q
queries, parallel processes for 9-3 to 9-21
query performance, See also performance
  and parallel query processing
QUERY_MEMORY_LIMIT parameter,
  modifying 3-16
QUERY_TEMPSPACE parameters,
  modifying 3-13, 3-16
query-priority concurrency, defined 1-2
QUERYPROCS parameter
  definition 9-2
  described 9-3 to 9-20
  modifying 3-12
  syntax and usage 9-8

R
RAID devices, with version log 2-10
  .RB_HOST.semaphore file, defined 6-9
  .RB_HOST.sharedmemory file,
    defined 6-9
  rbshow file, defined 6-9
  rbshow.exe program 3-8
  rbw.config file
    FORCE_FETCH_TASKS 9-3
    FORCE_JOIN_TASKS 9-3
    FORCE_SCAN_TASKS 9-3
    modifying 3-12, 3-15
  RBW_LOADINFO table 3-20
  RBW_SEGMENTS table 3-22
  RBW_STORAGE table 3-25
  RBW_USERAUTH table 3-26
  rbwadmd daemon
    description 3-4
    rbwadmd thread
      described 3-8
  RBWAPI parameters,
    PROCESS_CHECKING_INTERVAL
      3-18
  rbwapid daemon, description 3-4
  rbwapid thread, described 3-8
  rbwconc thread 3-9
  rbwlogd thread, described 3-9
  rbwlsnr thread 3-9
  rbwpchk daemon 3-5
  rbwpchk thread 3-9
  rbwsvr process 3-4
  rbwvcda daemon 3-5
  rbwvcda thread 3-10
  RECALCULATE RANGES keyword, TMU REORG 5-10
Red Brick Warehouse
  overview of NT threads 3-6 to 3-10
  process overview 3-2 to 3-4
Red Brick Warehouse listener thread 3-9
redbrick directory, files and directories 6-9
redbrick_dir, See redbrick directory
REFERENCE_CHECKING keyword, TMU REORG 5-11
referential integrity
  maintaining with TMU REORG 5-3
  overriding of 5-14
Index

relation scans 9-2, 9-19
REORG
discardfile format 5-16
disk space 5-16
locking behavior 5-15
online/offline operation 5-16
parallel 5-5
partial-index, limitations of 5-15
partial-index, options 5-3 to 5-4
serial 5-5
syntax 5-9
tasks sequence 5-7
usage 5-3
REPORT_INTERVAL parameter, modifying 3-16
RESET, TEMPSPACE parameters 5-20
RESULT BUFFER FULL ACTION parameter, modifying 3-13, 3-16
RESULT BUFFER parameter, modifying 3-13, 3-16
RI_DISCARDFILE keyword, TMU REORG 5-13
ROWS_PER_FETCH_TASK parameter described 9-2, 9-10
modifying 3-13
syntax and usage 9-13
ROWS_PER_JOIN_TASK parameter described 9-2, 9-10
modifying 3-13
syntax and usage 9-13
ROWS_PER_SCAN_TASK parameter described 9-2, 9-10
modifying 3-13
syntax and usage 9-11

S
scalar functions
CONCAT 4-19
DEC 4-12
INT 4-13, 4-15
segment name clause, TMU REORG 5-10
segments, backup 4-2
SEGMENTS parameter, modifying 3-13, 3-16
sem-mni parameter 6-2
semmni parameter 6-4, 6-8
sem-mns parameter 6-2
semms parameter 6-4, 6-8
semssys:seminfo parameters 6-6
SERVER parameter
changing 3-15
modifying 3-12
server, warehouse
described 3-8
See also rbwsvr file
SERVER_NAME parameter, modifying 3-12, 3-15
SET commands, SQL
FIRST DAYOFWEEK 4-21
FORCE_FETCH_TASKS 9-20
FORCE_HASHJOIN_TASKS 9-22
FORCE_JOIN_TASKS 9-20
FORCE_SCAN_TASKS 9-19
TRANSACTION ISOLATION LEVEL 2-15, 4-11
VERSIONING 4-10
SET commands, SQL TEMPSPACE 5-19
SET commands, TMU
INDEX TEMPSPACE MAXSPILLSIZE parameter 5-18
INDEX TEMPSPACE parameters 5-18
INDEX TEMPSPACE THRESHOLD parameter 5-18
syntax 5-17
TEMPSPACE
DUPLICATESPILLPERCENT parameters 5-18
TMU COMMIT RECORD INTERVAL 5-28
TMU COMMIT TIME INTERVAL 5-29
TMU INDEX TASKS 5-17
TMU INPUT TASKS 5-17
TMU MAX TASKS 5-17
SET FIRST DAYOFWEEK command
syntax description 4-21
SET TRANSACTION ISOLATION LEVEL command
examples of modes 2-15
syntax description 4-11
SET VERSIONING command, syntax description 4-10
SHMEM parameter, modifying 3-12
shm-max parameter 6-2
shmmx parameter 6-4
shm-mni parameter 6-2
Index

shmmni parameter 6-4, 6-8
shmsys:shminfo parameters 6-6
Solaris, kernel parameters 6-6 to 6-7
SQL commands
ALTER DATABASE 4-2
CREATE INDEX 4-8
SQL Server date formats 4-22
STARjoin, parallel query performance 9-31
string scalar functions, CONCAT 4-19
SuperScan technology 9-4, 9-34
support
documentation xix
technical xvii
syntax, REORG statement 5-9
syntax diagrams
conventions for xiv
keywords in xvi
metavariabes in xvi
syntax notation xiii
sysconfigtab file, for kernel parameters 6-2
Systems Management Server support 3-6

T
table name clause, TMU REORG 5-10, 5-13
tables
reorganizing 5-3 to 5-17
task_max parameter 6-2
tasks for parallel queries,
allocating 9-16 to 9-21
technical support xvii
temporary space management 5-18
TEMPORARY_DATA_SEGMENT parameter, modifying 3-13, 3-16
TEMPORARY_INDEX_SEGMENT parameter, modifying 3-13, 3-16
TEMPSPACE_DUPLICATE SPILLPERCENT parameters, TMU control 5-18
threads
described 3-7
See also rbwapid, rbwlogd, rbwsvr, rbwconc, rbwlsnr, or rbwadmd

TMU, REORG operations 5-2 to 5-17
TMU_BUFFERS parameter,
modifying 3-14, 3-16
TMU_COMMIT_RECORD_INTERVAL parameter
modifying 3-14, 3-16
rbw.config file entry 3-18
TMU_COMMIT_TIME_INTERVAL parameter
modifying 3-14, 3-16
rbw.config file entry 3-18
TMU_CONVERSION_TASKS parameter,
modifying 3-14
TMU_INDEX_TASKS parameter,
modifying 3-14
TMU_OPTIMIZE parameter,
modifying 3-14, 3-16
TMU_SERIAL_MODE parameter,
modifying 3-14
TMU_VERYIONING parameter
modifying 3-14, 3-16
rbw.config file entry 3-18
TOTALQUERYPROCS parameter
definition 9-2
described 9-3 to 9-20
modifying 3-12
syntax and usage 9-8
TRANSACTION_ISOLATION_LEVEL parameter
modifying 3-13, 3-16
rbw.config file entry 3-19
transactions
defined 2-15
single statement 2-15
trickle feed applications 2-20, 5-27
troubleshooting, general problems xix
TUNE parameters
FORCE_AGGREGATION_TASKS 9-23
FORCE_FETCH_TASKS 9-17
FORCE_JOIN_TASKS 9-17
FORCE_SCAN_TASKS 9-17
INDEX_TEMPSPACE_DUPLICATE SPILLPERCENT 3-19
PARTITIONED_PARALLEL_AGGREGATION 9-23
tuning parameters, parallel query 9-2
Index

**U**
USE_INVALID_PRECOMPUTED_VIEWS
parameter, modifying  3-13, 3-16

**V**
vacuum cleaner daemon  3-5
vacuum cleaner thread  3-10
version log
  adding space  2-11
  allocating space  2-10
  costs  2-2
  tuning for performance  2-10
versioned transactions
  defined  2-15
VERSIONING parameter
  modifying  3-13, 3-16
  rbw.config file entry  3-18

**W**
warehouse administrator, defined ix
warehouse processes, overview  3-2
warehouse service, defined  3-6
World Wide Web address, for Red Brick Systems xvii
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