



StorHouse Database Administration Guide

StorHouse/RM Release 3.4

Publication Number
900108 Rev. M

September 17, 2008





All rights reserved. No part of this publication may be reproduced, translated, stored in any electronic retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of FileTek, Inc.

Copyright © 1996-2008 FileTek, Inc. As an Unpublished Licensed Work.
Publication Number: 900108 Rev. M

NOTICE: U.S. GOVERNMENT USERS

This notice applies to all acquisitions of this work by or for the U.S. Government ("Government"), or by any prime contractor or subcontractor (at any tier) under any contract, cooperative agreement or other activity with the Government. By accepting delivery of this work, the Government agrees that this work and the Licensed Program(s) described herein qualify as "commercial" computer software within the meaning of the acquisition regulation(s) applicable to this procurement. The terms of conditions of the license for the Licensed Program(s) shall pertain to the Government's use and disclosure of this work and the Licensed Program(s), and shall supersede any conflicting contractual terms or conditions. If the license for this work and the Licensed Program(s) fails to meet the Government's need or is inconsistent in any respect with Federal law, the Government agrees to return this work and the Licensed Program(s), unused, to FileTek, Inc. The following additional statement applies only to acquisitions governed by DFARS Subpart 227.4 (October 1988) "Restricted Rights - Use, duplication and disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (OCT. 1988)." Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited. All rights reserved. A copyright notice on this work and/or on the Licensed Program(s) by itself does not constitute publication or public disclosure of this work or the Licensed Program(s). The contractor/manufacturer is:

FileTek, Inc.
9400 Key West Avenue
Rockville, Maryland 20850

Information in this document is subject to change without notice and does not represent a commitment on the part of FileTek, Inc. Further, FileTek, Inc. reserves the right to supplement the document with information not available at the time of creation of the document. FILETEK, INC. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND CANNOT WARRANT THE RESULTS YOU MAY OBTAIN USING THE DOCUMENT. IN NO EVENT SHALL FILETEK, INC. BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OR DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, EVEN IF FILETEK, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES ARISING FROM ANY DEFECT OR ERROR IN THIS PUBLICATION. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

FileTek and StorHouse are registered U.S. trademarks of FileTek, Inc. VRAM is a U.S. trademark of FileTek, Inc. All other brand or product names are trademarks or registered trademarks of their respective owners.

Documentation for FileTek's StorHouse product. Protected by the following U.S. Patents: 4,864,572; 5,247,660; 5,727,197; 6,049,804. Other patents pending.

Contents

Welcome	xix
StorHouse family of products	xix
StorHouse/SM	xix
StorHouse/RM	xx
StorHouse/Control Center	xx
Purpose of this document	xx
Intended audience	xxi
Contents	xxii
Related documentation	xxiv
Other StorHouse/RM publications	xxiv
StorHouse publications	xxv
StorHouse/Control Center publications	xxvi
Conventions	xxvii
 Chapter 1: Introduction	 1-1
What is StorHouse/RM?	1-1
StorHouse/RM features	1-2
Standard SQL	1-2
System managed storage	1-2
Remote database interfaces	1-2
Federation	1-3
Application program interface	1-3
Bulk data loading and unloading	1-3

Large object (LOB) support	1-3
Concurrency	1-3
Indexing	1-3
High-speed extraction	1-3
Database integrity	1-4
Controlled access/database security	1-4
StorHouse/RM software components	1-4
Data loaders	1-4
FileTek MVS Data Loader utility	1-5
FileTek FTP Data Loader	1-6
Data unloader	1-6
StorHouse engines	1-7

Chapter 2: StorHouse database concepts2-1

StorHouse database components	2-1
Database user components	2-2
User tablespaces	2-3
User tables	2-3
User table indexes	2-4
Views	2-4
Synonyms	2-5
Database system components	2-5
System tablespace	2-5
System tables	2-6
System table indexes	2-7
System table logs	2-7
System table index logs	2-7
Segmentation	2-7
Table data files	2-9
Index files	2-9
LOB storage and subsegment files	2-10
In-line LOBs	2-11
Out-of-line LOBs	2-13
Extents	2-15

Storage management on StorHouse	2-15
Naming conventions	2-16
Database names	2-17
Database user component names	2-17
Delimited names	2-18
Unique names	2-18
Database user component identifiers	2-19
Segment and LOB subsegment identifiers	2-19
Segment tags	2-20
Segment file names	2-20
StorHouse/RM 3.0 (and higher) segment file names	2-21
StorHouse/RM 2.3 segment file names	2-23
StorHouse/RM 2.1 to 2.2A segment file names	2-24
Database system component file names	2-26
Database system component identifiers	2-26
Summary of StorHouse database concepts	2-27

Chapter 3: StorHouse database administration basics3-1

People who manage StorHouse databases	3-1
Tools to help you manage StorHouse databases	3-2
StorHouse SQL statements	3-2
StorHouse Command Language	3-4
StorHouse utilities	3-5
Resources to help you manage StorHouse databases	3-7
System parameters	3-7
User log	3-7
System tables	3-8
How to perform administration tasks	3-9
Submitting StorHouse SQL statements	3-9
Issuing StorHouse commands	3-9
Running StorHouse utilities	3-10
Account for performing database and system administration tasks	3-11

Chapter 4: Accounts and privileges4-1

About StorHouse accounts	4-1
Account IDs	4-2
Account passwords	4-2
About StorHouse privileges	4-3
Access privilege	4-3
Command privileges	4-4
SQLEXECUTE privilege	4-4
SQLCOMMAND privilege	4-4
Database privileges	4-4
DBA privilege	4-5
RESOURCE privilege	4-5
SCAN privilege	4-6
Database component privileges	4-6
Types of accounts and their privileges	4-7
SYSADM account	4-7
PUBLIC account	4-8
General database user accounts	4-8
Loader accounts	4-8
Unloader accounts	4-9
StorHouse utility accounts	4-9
Local database application accounts	4-9
Host language application accounts	4-10
Ways to assign privileges	4-10
The WITH GRANT OPTION	4-11
When you grant to PUBLIC	4-12
System tables with privilege information	4-13
Summary of StorHouse privileges	4-13
Creating a StorHouse account	4-14
Granting database privileges	4-16
Granting database component privileges	4-17
Revoking database privileges	4-18
Revoking database component privileges	4-19

Changing a StorHouse account password	4-20
Adding an access or command privilege	4-21
Removing an access or command privilege	4-22
Listing StorHouse account IDs	4-23
Listing access and command privileges	4-23
Listing the privileges granted to PUBLIC	4-24
Listing database privileges	4-25
Listing database component privileges	4-26
Removing a StorHouse account	4-27

Chapter 5: User tablespaces5-1

About user tablespaces	5-1
Subspaces	5-2
Storage specifications	5-4
Volume set	5-4
File set	5-5
Object type	5-5
Access Time Factor	5-7
Vulnerability Time Factor	5-8
Error Detection Code	5-9
File access group	5-10
Maximum data extent size	5-10
Extent holding period in the performance buffer	5-11
User tablespace examples	5-12
Example 1: Minimal volume mounts	5-13
Storage specifications	5-13
Tablespace assignments	5-13
Example 2: Simultaneous removal	5-14
Storage specifications.	5-14
Tablespace assignments	5-14
Example 3: Load concurrency	5-15

Storage specifications	5-15
Tablespace assignments	5-15
Example 4: Access concurrency	5-16
Storage specifications	5-16
Tablespace assignments	5-18
Subspace selection during data load, index load, and segment merge operations ..	5-18
Default user tablespaces	5-19
Default user tablespace for a user table	5-19
Default user tablespace for a LOB column	5-19
Default user tablespace for an index	5-20
User tablespace naming conventions	5-20
User tablespace names	5-20
Tablespace IDs	5-21
Subspace numbers	5-21
System tables with user tablespace information	5-21
What's different about StorHouse user tablespaces?	5-22
Setting up a user tablespace	5-22
Creating volume sets and file sets	5-23
Creating a user tablespace	5-25
Assigning a default user tablespace	5-28
Listing accounts and their default user tablespaces	5-29
Changing a default user tablespace	5-29
Displaying default storage specifications	5-30
Altering a user tablespace	5-32
Listing user tablespace names and IDs	5-33
Displaying storage specifications for a user tablespace	5-34
Dropping a user tablespace	5-35

Chapter 6: User tables6-1

About StorHouse user tables	6-1
-----------------------------------	-----

Process of creating a user table	6-2
Owners and creators of user tables	6-3
User table naming conventions	6-3
User table names	6-3
Table IDs	6-4
System tables with user table information	6-4
What's different about StorHouse user tables?	6-5
Designing a column of a user table	6-5
Deciding the order of columns	6-6
Choosing a column name	6-6
Determining the data type	6-6
Using null values in a column	6-10
Specifying defaults for a column	6-11
Specifying LOB storage options	6-12
To set the maximum length for in-line LOB data	6-12
To store LOB values in a separate LOB subsegment file	6-13
To share storage with another LOB column	6-14
To store LOB values in a different user tablespace	6-14
Creating a user table	6-15
Displaying information about tables	6-18
Listing table names and IDs	6-19
Displaying a column definition	6-19
Displaying volume set, file set, and segment information	6-21
Renaming a user table	6-25
Dropping a user table	6-25
Undropping a user table	6-26
Purging a user table	6-28
Chapter 7: Indexes for user tables	7-1
About StorHouse indexes	7-1

Value indexes	7-2
Queries that use value indexes	7-3
Value indexes and join processing	7-3
How value indexes are stored	7-4
Hash indexes	7-4
Queries that use hash indexes	7-4
How hash indexes are stored	7-5
Hash indexes and join processing	7-5
Range indexes	7-5
Queries that use range indexes	7-6
Range indexes and extractor processing	7-8
How range indexes are stored	7-8
Simple and compound indexes	7-8
Default index type	7-9
Index owners	7-10
Index naming conventions	7-10
Index names	7-10
Index IDs	7-10
System tables with index information	7-11
What's different about StorHouse indexes?	7-11
Creating an index for a user table	7-11
Displaying information about indexes	7-13
Listing index names and IDs	7-14
Displaying volume set, file set, and segment information	7-16
Dropping an index	7-20

Chapter 8: Views8-1

About views	8-1
How views work	8-1
Owners and creators of views	8-3
View naming conventions	8-4
View restrictions	8-4

View privileges	8-5
System tables with view information	8-5
Creating a view	8-6
Creating views from multiple tables	8-7
Naming columns in views	8-9
Displaying information about views	8-10
Listing view names	8-11
Renaming a view	8-12
Dropping a view	8-12
 Chapter 9: Synonyms	9-1
About synonyms	9-1
Private and public synonyms	9-2
Synonym owners	9-2
Synonym naming conventions	9-2
System table with synonym information	9-3
Creating a synonym	9-3
Displaying information about synonyms	9-4
Listing synonym names	9-5
Renaming a synonym	9-6
Dropping a synonym	9-6
 Chapter 10: Metadata backup	10-1
About metadata backup	10-1
Files that are backed up	10-2
When to run a backup	10-2
Privileges required to run a backup	10-3
Before backing up metadata	10-3

Metadata backup process	10-4
Ways to run a metadata backup	10-4
Metadata backup output	10-4
After backing up metadata	10-5
Where metadata backup files are stored	10-5
Metadata backup file names	10-6
Metadata backup file versions	10-7
Backing up metadata	10-8
Scheduling a metadata backup	10-10
Troubleshooting	10-12
Listing metadata backup files	10-16

Chapter 11: Metadata restore11-1

About metadata restore	11-1
Metadata restore process	11-2
Before restoring metadata	11-2
After restoring metadata	11-3
How to run the metadata restore utility	11-3
Metadata restore output	11-3
Restoring metadata	11-4
Bringing a StorHouse database down	11-7
Bringing a StorHouse database up	11-9
Troubleshooting	11-11
Metadata restore error messages	11-11
Problem determination checklist	11-16

Chapter 12: Redo journaling12-1

About redo journaling	12-1
Journal file	12-2

Primary and secondary journal files	12-2
Journal file status	12-3
Journal file names	12-3
Redo journaling utilities	12-4
Journal chains	12-4
Replay checkpointing	12-6
Ways to run redo journaling utilities	12-7
Privileges for running redo journaling utilities	12-7
Location of the redo journal	12-8
Locking	12-9
Enabling journaling for a new database	12-9
Enabling journaling for an unjournaled database	12-10
Cycling a redo journal	12-11
Archiving and purging journal files	12-12
Replaying a journal file	12-15
Auditing a redo journal	12-19
Resetting the journaling environment	12-22

Chapter 13: Explain facility 13-1

About the explain facility	13-1
Execution tree	13-2
Nodes	13-2
Node operations	13-3
Project	13-4
Restrict.	13-4
Join	13-4
Sort.	13-5
Union.	13-5
Table scan	13-5
Index scan	13-5
Expression tree	13-5

Explain facility SQL statements	13-7
Explain facility result tables	13-7
Explain privileges	13-8
Explain examples	13-9
Example 1: Query without a predicate	13-9
Step 1: Run the explain facility	13-9
Step 2: Obtain the execution plan ID	13-10
Step 3: Display the execution plan	13-10
Step 4: Display the expression for the project node	13-11
Example 2: Query with a predicate	13-12
Step 1: Run the explain facility	13-12
Step 2: Obtain the execution plan ID	13-12
Step 3: Display the execution plan	13-13
Step 4: Display the operator for the restrict node	13-14
Step 5: Display the expressions for the restrict node	13-14
Example 3: Join without a predicate	13-15
Step 1: Run the explain facility	13-15
Step 2: Obtain the execution plan ID	13-15
Step 3: Display the execution plan	13-16
Step 4: Display the join predicate	13-17
Step 5: Display the expressions for the join predicate	13-17
Example 4: Query with a complex predicate	13-18
Step 1: Run the explain facility	13-18
Step 2: Obtain the execution plan ID	13-18
Step 3: Display the execution plan	13-19
Step 4: Display the operators for the restrict node	13-19
Step 5: Display the expressions for the restrict node	13-21
Example 5: Query with a function	13-21
Step 1: Run the explain facility	13-22
Step 2: Obtain the execution plan ID	13-22
Step 3: Display the execution plan	13-22
Step 4: Display the expression for the root project node	13-23
Step 5: Display the argument for the function	13-23
Example 6: Query that uses an index	13-24
Step 1: Run the explain facility	13-24
Step 2: Obtain the execution plan ID	13-24

Step 3: Display the execution plan	13-25
Step 4: Display the operators of the index scan node	13-26
Step 5: Display the expressions of the index scan node	13-26
Step 6: Display the operators for the restrict node	13-27
Step 7: Display the expressions for the restrict node	13-28
Example 7: Join with a complex predicate	13-28
Step 1: Run the explain facility	13-29
Step 2: Obtain the execution plan ID	13-29
Step 3: Display the execution plan	13-29
Step 4: Display the operators in the index scan node.	13-30
Step 5: Display the expressions of the index scan node	13-31
Accessing the StorHouse/Admin ISQL window	13-32
Creating explain tables	13-33
Running the explain facility	13-34
Obtaining an execution plan ID	13-36
Displaying a query in an execution plan	13-37
Displaying an execution plan	13-38
Displaying expressions in an execution plan	13-39
Displaying operators in an execution plan	13-40
Dropping explain tables	13-42

Chapter 14: Segment delete 14-1

About segment delete	14-1
Before deleting segments	14-1
Ways to run the segment delete utility	14-2
Privileges required to run the utility	14-2
Locking	14-3
Running multiple delete processes	14-3
Restarting an interrupted process	14-3
After deleting segments	14-4

Running the segment delete utility	14-4
Scheduling the segment delete utility	14-6

Appendix A: StorHouse system tables A-1

About system tables	A-1
System table names	A-2
System table privileges	A-3
List of system tables	A-4
SYSCOLAUTH	A-6
SYSCOLUMNS	A-7
SYSDBAUTH	A-8
SYSDROP_PEND	A-10
SYSINDEXES	A-11
SYSPACKAGE	A-13
SYSPACKSTMT	A-15
SYSRANGES	A-17
SYSSMUSERS	A-20
SYSSTAT_COL	A-21
SYSSTAT_HIST	A-22
SYSSTAT_IDX	A-22
SYSSTAT_SMATRIX	A-24
SYSSTHFILES	A-25
SYSSTHSEGMENTS	A-26
SYSSTHSPACES	A-27
SYSSYNONYMS	A-30
SYSTABAUTH	A-31

SYSTABLES	A-33
SYSTBLSPACES	A-35
SYSVIEWS	A-36

Appendix B: System limits B-1

Appendix C: System parameters for StorHouse/RM C-1

About system parameters	C-1
System parameter types	C-1
System parameter access	C-2
System parameter list	C-3
System parameter descriptions	C-4
LOG_SQL_STMT	C-4
LOG_SQL_TRANS	C-4
SQL_BKUP_ACCOUNT	C-5
SQL_BKUP_FSET	C-5
SQL_BKUP_GROUP	C-5
SQL_BKUP_LIMIT	C-6
SQL_BKUP_VSET	C-6
SQL_BLD_INDX_MEM	C-6
SQL_DROP_HOLD	C-7
SQL_HOLD_DATA	C-7
SQL_HOLD_INDX	C-8
SQL_HOLD_SPECIAL	C-8
SQL_INDX_TYPE	C-9
SQL_LDR_ENGINES	C-9
SQL_LDR_MAXINTO	C-9
SQL_LDR_MAXLOAD	C-10
SQL_MAX_EXT_DATA	C-10
SQL_MAX_EXT_HASH	C-11
SQL_MAX_EXT_VAL	C-11
SQL_SESSIONS	C-12
STORHOUSE_REL	C-12

SYSTEM_ID C-12

Displaying a system parameter value C-13

Setting or changing a system parameter value C-14

Appendix D: StorHouse explain tables D-1

About explain tables D-1

STH_EXPLAIN_ID D-2

STH_EXPLAIN_PLAN D-3

STH_EXPLAIN_STMT D-5

STH_EXPLAIN_EXPR D-5

STH_EXPLAIN_OPR D-7

Index Index-1

Welcome

The *StorHouse Database Administration Guide* describes how to create and manage the components of a StorHouse database.

StorHouse family of products

StorHouse® is the FileTek® enterprise-wide solution for managing the capture, storage, movement, and access of gigabytes to petabytes of relational and non-relational detail data. StorHouse technology combines industry-leading, scalable storage devices and Open System processors with specialized storage management and relational database management system (RDBMS) software components. The basic StorHouse software components—StorHouse/SM, StorHouse/RM, and StorHouse/Control Center—provide features to manage and protect your data.

StorHouse/SM

StorHouse/SM, the storage management component, controls a hierarchy of storage devices comprised of cache, redundant array of independent disks (RAID), Serial Advanced Technology Attached (SATA) disks, massive array of idle disks (MAID), erasable and write-once-read-many (WORM) optical disk jukeboxes, and erasable and WORM automated tape libraries. StorHouse/SM is also responsible for automating critical system management tasks, like data migration, backup, and recovery. StorHouse/SM comes standard with all StorHouse systems.

Welcome

Purpose of this document

StorHouse/RM

StorHouse/RM, the RDBMS component, works in conjunction with *StorHouse/SM* to specifically administer the storage, access, and movement of relational data. *StorHouse/RM* provides row-level SQL access to high volumes of detail data on any layer—including tape—in the *StorHouse* storage hierarchy. SQL access is available from different platforms through a variety of industry-standard protocols. *StorHouse/RM* runs on Sun™ Solaris™, Hewlett-Packard HP-UX, and IBM® AIX platforms.

StorHouse/Control Center

StorHouse/Control Center (CC) is the FileTek Windows®-based network computing system for providing administrative control of the *StorHouse* family of products. *StorHouse/Control Center* works with *StorHouse/SM* release 4.2 and higher and consists of one or more *StorHouse/Control Center* servers that communicate with *StorHouse/Control Center* clients over an IP network.

- The *StorHouse/Control Center server*, which runs on Windows NT, XP Pro, and 2000 platforms, provides network connectivity to *StorHouse*.
- The *StorHouse/Control Center clients*, which run on Windows 95, 98, 2000, XP Pro, and NT platforms, consist of one or more graphical user interface (GUI) modules for performing *StorHouse* system and database administration tasks, configuring and managing *StorHouse/Control Center* servers, and analyzing and monitoring *StorHouse* activity and performance.

Purpose of this document

The *StorHouse Database Administration Guide* describes how to manage *StorHouse* databases. Database administration tasks include creating database user tables and indexes, managing privileges for accessing database data, and

assigning volume sets and file sets and other storage specifications to user tablespaces.

This manual also describes the StorHouse system administration tasks that are associated with StorHouse/RM. These tasks include managing the StorHouse signon accounts that access StorHouse databases, creating volume sets and files sets for database data, and setting various StorHouse system parameters for the efficient operation of StorHouse/RM.

This manual focuses on StorHouse database administration instead of system administration. It does not describe StorHouse hardware configurations, software features, storage allocation and control concepts, or StorHouse Command Language syntax. Refer to the manuals in the StorHouse User Document Set for more information about these topics.

This manual does not describe how to create databases. Contact FileTek Customer Support or your account system engineer for more information about creating databases. You can also create databases with the StorHouse/Admin module of StorHouse/Control Center.

You can manage StorHouse databases with a command-based interface (SQL statements and StorHouse Command Language commands) or with a GUI interface (StorHouse/Admin). This manual shows the command-based interface. Refer to the StorHouse/Control Center User Document Set for more information about using the StorHouse/Admin GUI interface to manage StorHouse databases.

Intended audience

The *StorHouse Database Administration Guide* is intended for StorHouse database administrators (DBAs). A chief DBA typically oversees the operation and maintenance of all StorHouse databases. A departmental DBA typically manages one or more StorHouse databases. Depending on your organization, you or a separate StorHouse system administrator may manage StorHouse system tasks.

This manual assumes that you understand Structured Query Language (SQL) and StorHouse concepts. It also assumes that you have successfully completed these two FileTek training courses:

- StorHouse Overview and Command Language
- StorHouse System Administration

Contact your FileTek customer support representative for information about how to register for these courses if you have not already completed them. It is critical that you satisfy these prerequisites so that you can work effectively and knowledgeably with the StorHouse system administrator at your site.

Note: In this guide, *you* refers to both the StorHouse system administrator and to StorHouse DBAs.

Contents

This document is organized as follows:

- Chapter 1, “Introduction,” describes StorHouse/RM features and software components.
- Chapter 2, “StorHouse database concepts,” describes database components and naming conventions and how StorHouse stores and manages database data.
- Chapter 3, “StorHouse database administration basics,” describes the tools and resources available for managing StorHouse databases.
- Chapter 4, “Accounts and privileges,” describes StorHouse accounts and privileges that enable users to access StorHouse database data.
- Chapter 5, “User tablespaces,” defines user tablespaces and explains how to maintain them.

- Chapter 6, “User tables,” defines user tables and explains how to manage them.
- Chapter 7, “Indexes for user tables,” defines indexes and explains how to manage them.
- Chapter 8, “Views,” defines views and explains how to manage them.
- Chapter 9, “Synonyms,” defines synonyms and explains how to manage them.
- Chapter 10, “Metadata backup,” explains how to back up the metadata and range indexes for one or multiple databases.
- Chapter 11, “Metadata restore,” explains how to restore metadata using metadata backup files.
- Chapter 12, “Redo journaling,” describes the redo journal and how to manage it.
- Chapter 13, “Explain facility,” describes how to analyze the execution plan created by the optimizer for a given query.
- Chapter 14, “Segment delete,” explains how to run the segment delete utility to delete invalidated segments and associated objects.
- Appendix A, “StorHouse system tables,” describes the system tables used by StorHouse/RM.
- Appendix B, “System limits,” lists the maximum limits for various components of StorHouse/RM.
- Appendix C, “System parameters for StorHouse/RM,” describes the StorHouse system parameters that are used to configure StorHouse/RM.

- Appendix D, “StorHouse explain tables,” describes the tables containing explain data.

Related documentation

The following FileTek publications can help you use and understand StorHouse/RM.

Other StorHouse/RM publications

In addition to the *StorHouse Database Administration Guide*, the StorHouse/RM User Document Set consists of the following publications:

- The *FileTek MVS Data Loader Utility Manual*, publication number 900109, describes how to load data into StorHouse user tables from an MVS environment.
- The *FileTek FTP Data Loader Manual*, publication number 900115, describes how to load data into StorHouse user tables from a UNIX, VAX, or other host environment with File Transfer Protocol (FTP).
- The *FileTek FTP Data Unloader Manual*, publication number 900137, explains how unload data from StorHouse user tables with FTP.
- The *StorHouse SQL Reference Manual*, publication number 900111, describes the SQL statements, predicates, and functions supported by StorHouse.
- The *StorHouse/RM SQL and Utility Quick Reference*, publication number 900122, contains formats and examples of SQL statements, predicates, functions, data loader statements, data unloader statement, and utilities.

- The *StorHouse ESQL Manual*, publication number 900121, explains how application programs can use the StorHouse ESQL interface to submit SQL statements to retrieve data in StorHouse user tables.
- The *StorHouse/RM Glossary*, publication number 900112, defines terms used in the StorHouse/RM User Document Set.
- The *StorHouse/RM Metadata Conversion Manual*, publication number 900142, explains how to convert metadata from one StorHouse/RM release to another.
- The *StorHouse/UDB Link Installation and Configuration Manual*, publication number 900163, explains how to install the StorHouse/UDB Link and how to configure IBM® DB2® Universal Database (DB2 UDB) to work with StorHouse. It also discusses security management, data modeling, and DB2 components that support federation.

StorHouse publications

Some of the StorHouse publications that can help you understand StorHouse/RM are:

- The *StorHouse Glossary*, publication number 900027, defines the terminology used in FileTek StorHouse publications.
- The *StorHouse Concepts and Facilities Manual*, publication number 900026, defines the basic concepts, structures, and functions of StorHouse.
- The *Command Language Reference Manual*, publication number 900005, describes StorHouse Command Language commands, system parameters, and command privileges.
- The *System Administrator's Guide*, publication number 900007, describes system recovery, user file backup, account administration, and storage management procedures and concepts for StorHouse.

- The *Messages and Codes Manual*, publication number 900032, lists all StorHouse system and host software messages.

StorHouse/Control Center publications

The following StorHouse/Control Center publications describe how to use StorHouse/Admin to complete StorHouse system and database administration tasks.

- *Getting Started with StorHouse/Admin*, publication number 900135, contains an introduction to the software features, windows, menus, tool bars, and selected system and database management tasks.
- The *StorHouse/Admin System Administrator's Quick Reference*, publication number 900147, contains procedures for performing system administration tasks with StorHouse/Admin. Some tasks include setting system parameters, creating file access groups, changing volume attributes, and printing labels.
- The *StorHouse/Admin Database Administrator's Quick Reference*, publication number 9001487, contains procedures for performing database administration tasks with StorHouse/Admin. Some tasks include creating databases, granting and revoking privileges, using ISQL, archiving and backing up segments, and backing up and restoring metadata.
- The *StorHouse/Admin System Operator's Reference*, publication number 900149, contains procedures for performing system operator tasks with StorHouse/Admin. Some tasks include bringing up a StorHouse device, reserving the system, monitoring user activity, and shutting down the system.



Conventions

This book uses the following conventions for illustrating statement and command formats, presenting examples, and identifying special terms:

Convention	Meaning
UPPERCASE HELVETICA	SQL statement and StorHouse command examples
<i>Italics</i>	New terms, emphasized text, and publication titles
▼	Procedures



Welcome

Conventions

Introduction

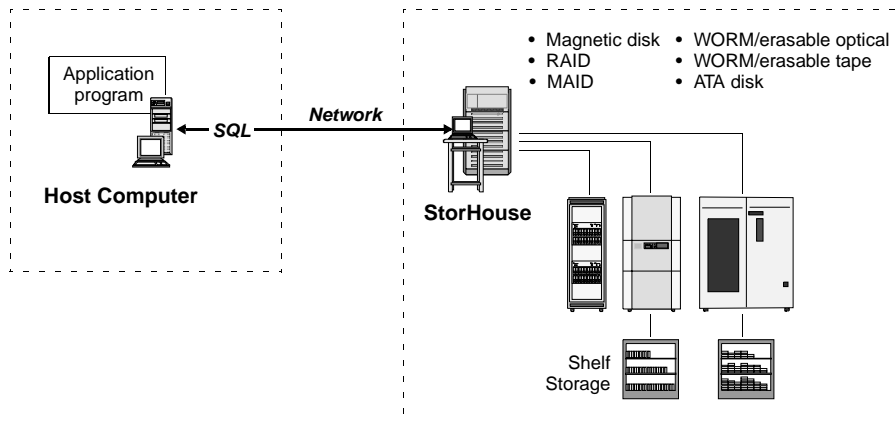
This chapter provides an overview of StorHouse/RM. It describes the basic functions, features, and software components of StorHouse/RM.

What is StorHouse/RM?

StorHouse/RM is specialized relational database management software for managing the storage, access, and movement of relational data across the enterprise. StorHouse/RM works in conjunction with StorHouse/SM to provide a storage repository for high volumes of detail data.

With StorHouse/RM, you load database data from your host computer into StorHouse database tables using FileTek data loading processes. Once data is loaded into StorHouse, client applications and database tools running on a variety of host platforms can access selected rows of information on any layer in the StorHouse storage hierarchy using industry-standard Structured Query

Language (SQL). The following graphic shows an example of how StorHouse/RM provides SQL access to the StorHouse storage hierarchy.



StorHouse/RM features

StorHouse/RM provides the features you would expect of an advanced RDBMS, as well as other features that uniquely suit it to storing and retrieving vast amounts of data on StorHouse.

Standard SQL. The SQL you use to access StorHouse databases is a subset of the American National Standards Institute (ANSI) SQL-99 standard plus extensions defined by FileTek to support additional capabilities.

System managed storage. Database tables and indexes are stored on an automatically managed hierarchy of media and devices including RAID, ATA disks, MAID, erasable and WORM optical disk jukeboxes, and erasable and WORM automated tape libraries, and shelf storage.

Remote database interfaces. Database gateways enable applications based on an existing RDBMS, such as IBM DB2 and Oracle®, to interface with StorHouse.

In this case, StorHouse is a *remote database* and the RDBMS is a *local database*. Gateways with network communications support are a class of software called *middleware*.

Federation. StorHouse can serve as a data source in a federated system. The *StorHouse/UDB Link* is the FileTek software that implements the connection between DB2 UDB and StorHouse databases. This software gives DB2 UDB users near-transparent access to the terabytes to petabytes of data supported by StorHouse/RM.

Application program interface. StorHouse SQL provides an *Embedded SQL (ESQL) interface* that supports coding SQL statements in C and C++ programs. By embedding SQL statements in a host program, you can develop applications that are more flexible than those developed in just the host language or SQL.

Bulk data loading and unloading. Comprehensive data loader programs developed by FileTek transfer large amounts of relational data from host environments to StorHouse. The FileTek data unloader utility copies data from StorHouse database tables to a local file on a client system.

Large object (LOB) support. StorHouse/RM supports the storage and access of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs). BLOBs include data such as video, photos, and voice up to 2 gigabytes (GB) in size. CLOBs include large character-based data up to 2 GB in size.

Concurrency. The StorHouse/RM architecture facilitates concurrent access by multiple users of local databases, multiple application programs, and multiple loading and unloading processes.

Indexing. Comprehensive database indexing speeds access to data in large complex tables. StorHouse supports three index types: hash, value, and range.

High-speed extraction. The StorHouse extractor provides a practical solution for retrieving massive amounts of database data. It is a performance enhancement

feature for queries that result in a *full table scan* (a search in every row in a database table without using an index) or a *full segment select* (a query that returns data from one or more entire segments with the use of a range index).

Database integrity. A reliable system of constraints, logging, recovery, journaling, and metadata backup ensure database integrity.

Controlled access/database security. A complete set of privileges controls user access to StorHouse databases and database data.

StorHouse/RM software components

The major StorHouse/RM software components are:

- Data loaders
- Data unloader
- StorHouse engines

Data loaders

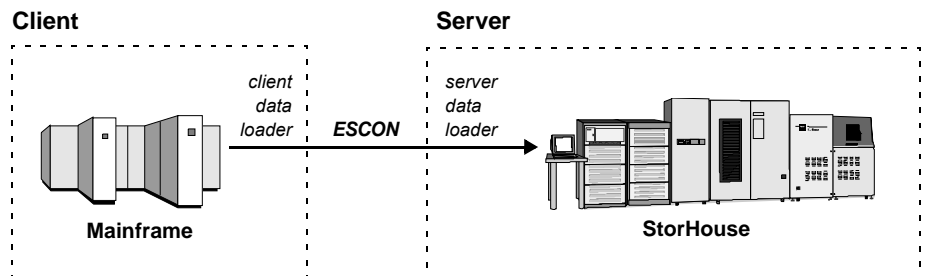
FileTek uses bulk data loading processes to load table data into StorHouse. You can load data from an MVS environment with the FileTek MVS Data Loader utility, and you can load data from an FTP-enabled host with the FileTek FTP Data Loader.

FileTek MVS Data Loader utility

The *FileTek MVS Data Loader utility* is an MVS batch program that initiates the loading of a sequential dataset from a host computer into a StorHouse user table. This process requires cooperation between two separate FileTek data loader programs:

- The *client data loader*, which runs on your host computer, prepares your data for loading and sends it to StorHouse. The FileTek MVS Data Loader utility is the FileTek-supplied client data loader.
- The *server data loader*, which runs on StorHouse, loads your data into StorHouse user tables and builds and stores any indexes.

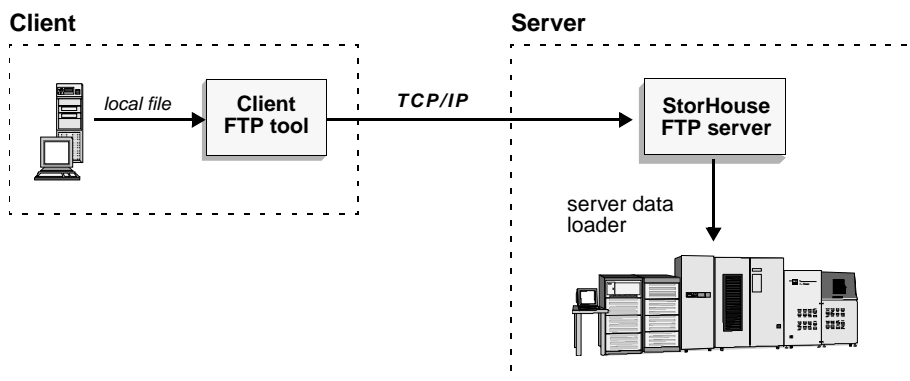
The FileTek MVS Data Loader utility bypasses the gateways when transferring data from the host to the server, relying instead on an existing channel connection and the StorHouse application program interface (API) to achieve maximum data transfer rates.



Refer to the *FileTek MVS Data Loader Utility Manual* for information about loading data from an MVS host.

FileTek FTP Data Loader

The *FileTek FTP Data Loader* is a tool for loading data into StorHouse user tables from a host that's enabled with FTP. With the FileTek FTP Data Loader, you use your standard client FTP client software to communicate with the FileTek StorHouse FTP server. These two programs communicate over a TCP/IP connection to transfer files from your local file system on your host to a remote file system on StorHouse.



Refer to the *FileTek FTP Data Loader Manual* for information about loading data via FTP.

Data unloader

The FileTek FTP Data Unloader is a tool for copying data from StorHouse database tables to your client system. It executes a SELECT statement, then formats and transfers the result data according to your specifications. With the FileTek FTP Data Unloader, you use your standard FTP client software to

communicate with the StorHouse FTP server. Some of the features of the FileTek FTP Data Unloader include the following:

- You can unload an entire table, specific columns or rows of a table, or multiple tables (join).
- Depending on your host environment, you can pipe the output to another program, such as a data loading utility.
- You can receive the result data in one of three record formats: text, fixed-length, or variable length.
- You can receive the result data in a sequential (or flat) file on your computer or in a Virtual Record Access Manager (VRAM™) file on StorHouse.
- You can receive LOB data in separate result files (one LOB value per file) on your client computer or on a remote system.

Refer to the *FileTek FTP Data Unloader Manual* for information about unloading data via FTP.

StorHouse engines

A StorHouse *engine* is the main StorHouse/RM process. The main functions of a StorHouse engine are as follows.

Parsing and optimizing SQL. A StorHouse engine checks your privileges, parses, then optimizes an SQL statement before executing it. The *parser* analyzes the request for syntax and semantics, then breaks it into basic components for the optimizer. The *optimizer* develops an access plan that most efficiently completes the query.

Retrieving data from StorHouse. When you submit an SQL SELECT statement, StorHouse/RM assigns an engine to execute the query. The StorHouse engine parses and optimizes the request, identifies the table and rows within the

table that it wants to access, retrieves the information, and forwards it to the requesting host.

Assisting a data loader. During a load, a StorHouse engine checks your StorHouse privileges and obtains the metadata to store the table, index, and LOB data. When a load completes, a StorHouse engine updates the metadata. One StorHouse engine is assigned to each load.

StorHouse database concepts

This chapter contains an overview of StorHouse databases. It explains:

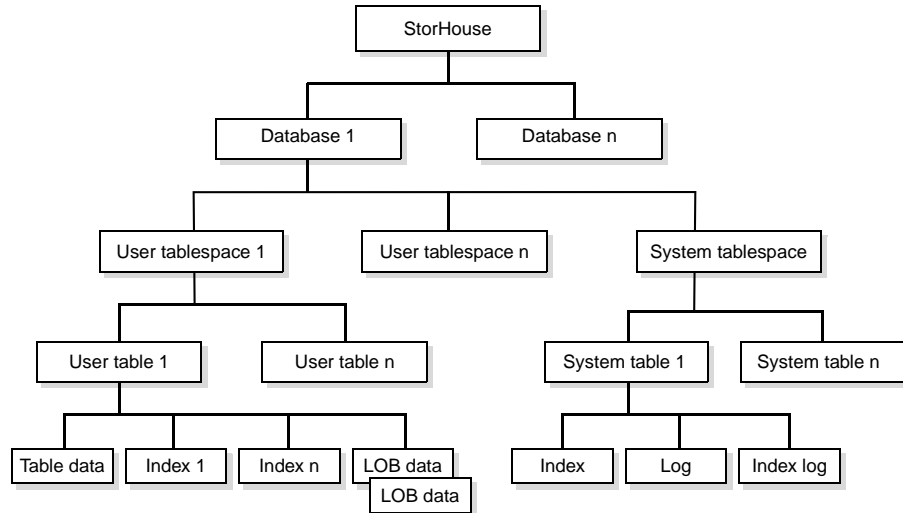
- Types of database components, both system and user
- Segments and their relationship to table data, indexes, and LOB data
- Conventions for naming databases, database components, and segments

StorHouse database components

A *StorHouse database* is a collection of user and system components used for storing and accessing read-only, detail data. In general, database user components are the database information that you store, access, and view while database system components describe a database. User components reside on and are managed by the StorHouse/SM file system. System components reside on and are managed by the UNIX file system.

The following graphic illustrates the user and system components of a StorHouse database. It shows all user table components assigned to the same user tablespace.

You can also assign table data to one user tablespace and indexes and LOB data to different user tablespaces, as shown in the graphic on page 2-3.



StorHouse provides both a logical and a physical structure to system and user data. The logical structure (data representation) is accomplished with tables. The physical structure (data storage) is accomplished with files. Tables represent data, and files contain data. Users view and access data as relational tables. StorHouse/RM accesses data stored in files.

Database user components

Database user components are structures you work with to manage the storage and retrieval of data. StorHouse database user components are:

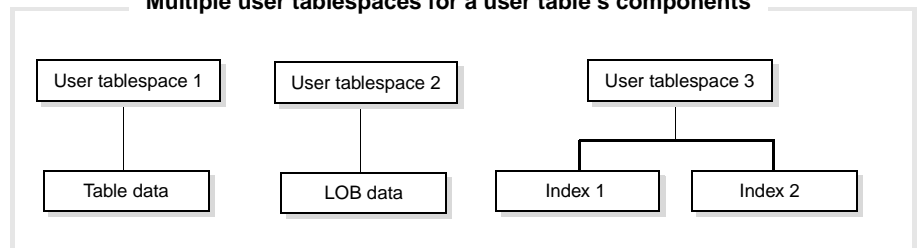
- User tablespaces
- User tables
- User table indexes
- Views
- Synonyms

User tablespaces

A *user tablespace* is a logical structure that defines how to store table, index, and LOB data on StorHouse. Before creating and loading user tables, you create one or more user tablespaces. Each user tablespace consists of one or more *subspaces* that define the storage specifications for table data, hash index entries, value index entries, and LOB data.

When you create a user table, you assign it to a user tablespace. The table data then belongs to that user tablespace. If the user table contains a LOB column, you can assign that LOB column to a different user tablespace or use the default, which is the same user tablespace as the table. And when you create an index for the user table, you can assign that index to the same user tablespace as the table or to a different one. The components then of a user table can belong to multiple user tablespaces, as shown in the following graphic.

Multiple user tablespaces for a user table's components



If you don't explicitly assign user tables, LOB columns, and indexes to a user tablespace, StorHouse/RM attempts to assign them to a default tablespace.

See Chapter 5, "User tablespaces," for more information about user tablespaces, subspaces, and storage specifications.

User tables

A *user table* holds user-accessible data. Logically, a user table consists of columns, rows, and data values. A row is also called a *tuple*. A column is also called a *field*. You define the columns of a user table with a CREATE TABLE statement. Then

you load data into that user table with a FileTek data loader. *User table components* are all of the data—table data, index entries, LOB data—that refer to the same user table.

Physically, user tables are stored in one or more *segments*. When you load data, StorHouse/RM saves the table data as a table data file in a segment. See “Segmentation” on page 2-7 for more information about how table data is stored on StorHouse.

See Chapter 6, “User tables,” for more information about user tables.

User table indexes

A *user table index* provides quick and efficient access to table data. You can create an index on a column (*simple index*) or combination of columns (*compound index*) in a user table. StorHouse supports three index types: range, value, and hash. When you create an index, you indicate the index type.

Physically, hash indexes and value indexes are stored in one or more index files in segments, and range indexes are stored in system tables. See “Segmentation” on page 2-7 for more information about how hash and value indexes are stored. See “SYSRANGES” on page A-17 for a description of the range index system tables.

See Chapter 7, “Indexes for user tables,” for an explanation of index types, when StorHouse/RM uses different indexes, and how to create them.

Views

A *view* is a virtual table that appears and acts like a table but draws its contents from one or more existing tables and/or views. When you query a view, you actually query the tables referenced by the view.

For instance, you might construct a two-column view that provides a column of customer names drawn from one table and a column of associated sales representatives from another table. Or you might construct a view from an employee record table that excludes the column containing salary information.

Views are useful for tailoring or limiting user access to data. They provide convenience, security, or both by letting you determine which data in which tables is available to which users.

See Chapter 8, “Views,” for more information about working with views.

Synonyms

A *synonym* is another name that you can assign to a table, view, or synonym. You can create a *private synonym* for your own use or a *public synonym* for all accounts who can access the database. You use synonyms in place of the original table, view, or synonym name in SQL statements,

See Chapter 9, “Synonyms,” for more information about working with synonyms.

Database system components

Database system components are structures that StorHouse/RM uses to manage a database. These system components—also called *metadata*—reside on the UNIX file system. Like user components, system components are represented as tables and indexes and stored as files. StorHouse database system components are:

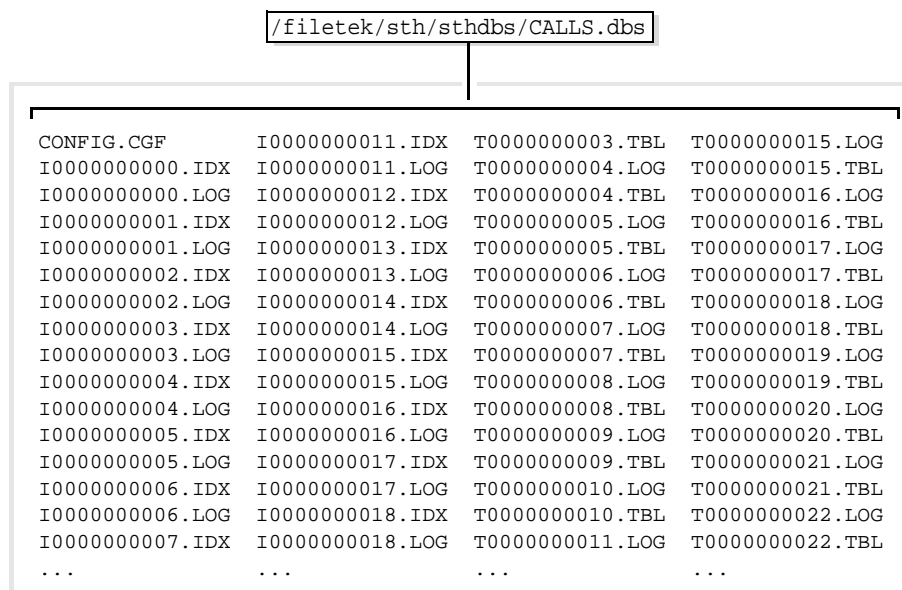
- System tablespace
- System tables
- System table indexes
- System table logs
- System table index logs

System tablespace

A *system tablespace* is a logical structure that contains the system components for a database. Each database has a system tablespace. On UNIX, a system tablespace is a directory, also called *database directory*. For each new database, StorHouse/RM

creates a database directory and a set of metadata under the default path `/filetek/sth/sthdb`.

In the following graphic, for example, the `CALLS.dbs` directory in the UNIX file system contains the system component files for the `CALLS` database.



See page 2-26 for more information about UNIX file names for database system components.

System tables

A *system table* contains information about the contents of a database as well as account access to database components. System tables also contain range index entries. For each new database, StorHouse/RM creates a set of system tables and stores them as UNIX files in the system tablespace. StorHouse/RM then maintains all system tables except the `SYSSMUSERS` system table. Only accounts with the appropriate privileges should use SQL to access system tables.

See Appendix A, “StorHouse system tables,” for descriptions of all system tables. Chapter 10 explains how to back up system tables and other database system components.

System table indexes

A *system table index* helps StorHouse/RM access information in a system table. Some, but not all, system tables have an index. When you create a database, StorHouse/RM creates the system table indexes and stores them as UNIX files in the system tablespace.

System table logs

Each system table has a corresponding *system table log* that’s used to recover changes to system tables. Before StorHouse/RM updates a system table, it first copies a “before image” of any record being updated to the system table log and then makes the change in the system table. If the transaction fails or is rolled back, StorHouse/RM copies the before image (or *undo record*) back to the system table, removing the change. If the transaction completes or is committed, StorHouse/RM empties the system table log.

System table index logs

Each system table index has a log that’s used to recover changes to system table indexes. The operation of the index log is the same as the system table log.

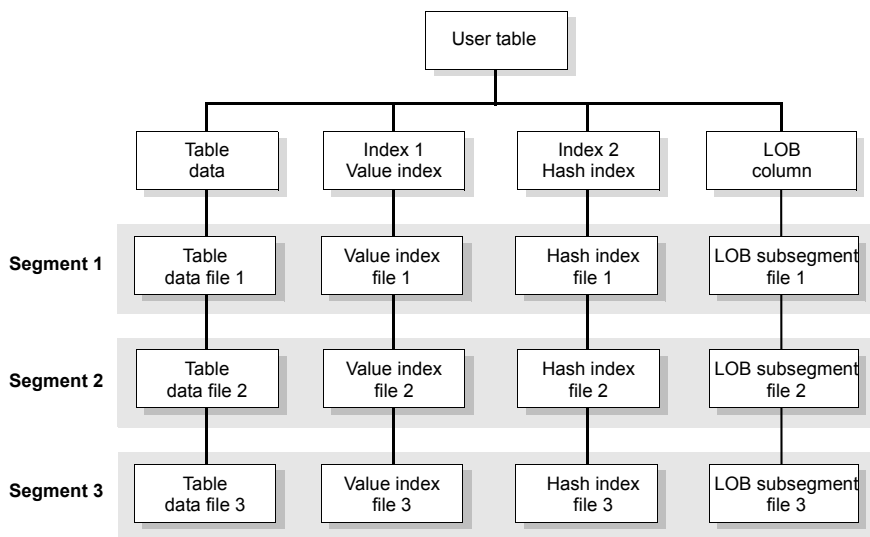
Segmentation

StorHouse user tables consist of one or more segments. A *segment* is a set of StorHouse files containing table data, index entries, and LOB data. Each time you load data into a user table, StorHouse creates a segment with one file for the table data, one file for each hash index, one file for each value index, and one or more subsegment files for each LOB column. The term *segment files* refers to the

table data files, hash index files, value index files, and LOB subsegment files that make up StorHouse segments.

Note: Depending on the actual size or by user request, LOB values may be stored in the table data file, or LOB values in different columns may be stored in the same LOB subsegment file, or LOB values in the same column may be stored in multiple LOB subsegment files.

The following user table consists of three segments. This user table has one value index, one hash index, and one LOB column.



Whether a user table is composed of one or multiple segments depends on the following:

- Size of the table data – The maximum size of a table data file is approximately 100 GB. If input data for a load exceeds 100 GB, you must load that data into multiple segments.
- Subsequent loads – When you load more data into a table, you create at least one (depending on the size of the data) new segment for each load.

- User request – You have the option of creating multiple segments during a load. For example, if you're loading data consisting of a year of transactions, you can create one segment for the January transactions, another segment for the February transactions, and so on.

Table data files

The data of a table is stored in a *table data file*. Each segment contains one table data file. The maximum size of a row in a table data file is 32,705 bytes. The maximum size of a table data file is approximately 100 GB. Table data files may contain:

- LOB values that fit within the row
- Object identifiers (OIDs) for LOB values that do not fit within the row

See “LOB storage and subsegment files” on page 2-10 for more information about object identifiers and ways to store LOB data.

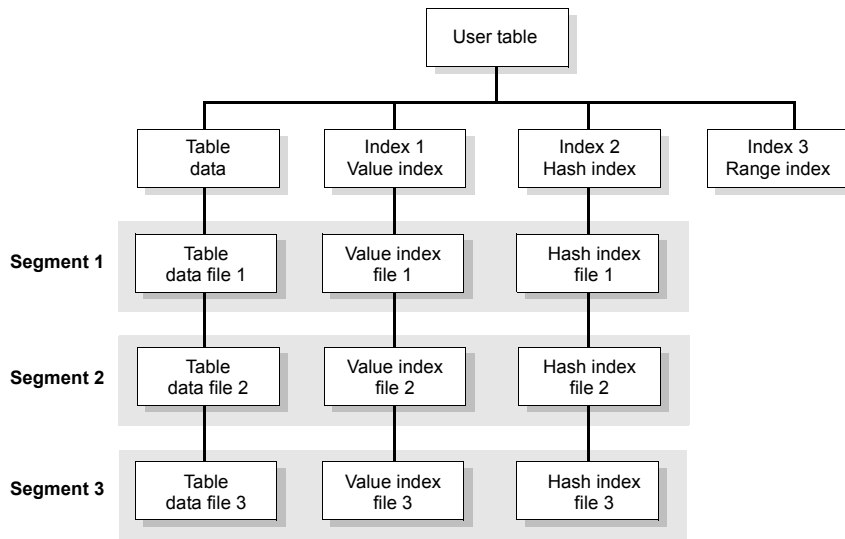
Index files

During a load, a FileTek data loader builds and stores any index entries for value, hash, and range indexes. The data loader stores:

- Value index entries in a *value index file*
- Hash index entries in a *hash index file*
- Range index entries in a set of system tables

Each segment contains one value index file for each value index on a table and one hash index file for each hash index on a table. A range index applies to all segments of a user table. When you load more data into a user table, a FileTek data loader creates a new segment with index files and inserts the range index entries into the system tables.

For instance, if you had created one value, hash, and range index for a user table, and you loaded data in January, February, and March, then after the March load, the user table would consist of three segments. Each segment would contain a table data file, a value index file, and a hash index file. The range index would contain entries for the three segments.



LOB storage and subsegment files

A user table can consist of multiple LOB columns defined as BLOB and CLOB data types. The maximum size of a LOB column is 2 GB. When creating a table, you can choose to store LOB values:

- In the table data file (space permitting)
- In a separate file, called a *LOB subsegment file*, with other LOB values in the same column
- In a separate LOB subsegment file with other values in other LOB columns

An *in-line LOB* is a LOB value that's stored with the table data. An *out-of-line LOB* is a LOB value that's stored in a separate file from the table data. StorHouse/RM determines in-line and out-of-line storage on a row-by-row basis.

In-line LOBs

If space permits—a LOB value does not exceed any user-defined in-line limit and the row does not exceed 32 KB—you can store a LOB value in a table data file. StorHouse/RM treats in-line LOB values like other variable-length, nullable fields and determines in-line storage by the actual size of the LOB value, not by the maximum size defined for the LOB column. In-line storage is the default, that is, unless you specify otherwise, a FileTek data loader attempts to store LOB values with the table data.

You can set the maximum length for in-line LOB data, for instance, 15 KB. If you omit the length, the default is 32 KB. A FileTek data loader stores a LOB value in a row as long as the value does not exceed the maximum in-line length (like 15 KB) or the maximum row length (32 KB). If a LOB value exceeds either length, the data loader stores the value in a separate LOB subsegment file. For instance, if other columns in a table are 20 KB and a LOB value is 13 KB (within the 15 KB limit), the data loader stores the LOB value out-of-line because the total row size (in this example, 33 KB) exceeds the maximum (32 KB).

For example, assume you created a table for bank account information with check images that require approximately 28 KB of storage. You defined the check image column as BLOB data type and intend to store the images in-line with the table data when possible. Row 1 contains 31 KB of data and row 2 contains 33 KB. The FileTek data loader stores all table data and the row 1 check image in the

table data file and stores the row 2 check image in a separate LOB subsegment file. Row 2 contains an OID that identifies the LOB subsegment file.

Row 1	Table data 3 KB	In-line check image 28 KB
Row 2	Table data 5 KB	OID

↓

Out-of-line check image 28 KB

Note: In the preceding example, the size of row 2 is 5 KB plus 22 bytes for the OID.

If a user table contains multiple LOB columns, a FileTek data loader determines in-line storage in column order. In other words, the data loader attempts to store the first LOB column value with the table data, and if space permits, attempts to store the second LOB column value with the table data, and so on.

For instance, if you defined a check image column before a photo ID column, the data loader first attempts to store the check image with the table data before the photo ID. In the following example, the data loader stores the row 1 check image in-line with the table data and the row 1 photo ID out-of-line in a LOB subsegment file. For row 2, the data loader stores both the check image and the photo ID in-line with the table data because both LOBs fit within the row.

Out-of-line photo 3 KB

Row 1	Table data 3 KB	In-line check image 28 KB	OID
Row 2	Table data 1 KB	In-line check image 27 KB	In-line photo 3 KB

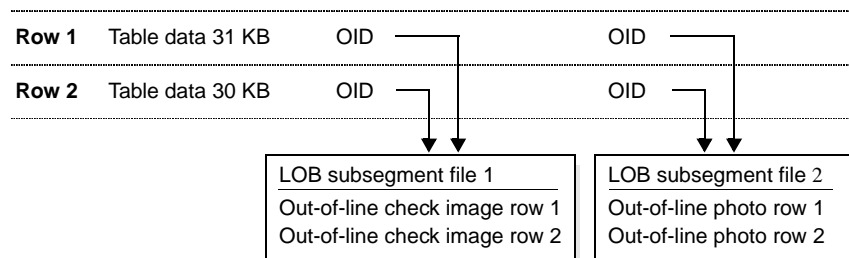
Note: In the preceding example, the size of row 1 is 31 KB plus 22 bytes for the OID.

Out-of-line LOBs

Out-of-line LOBs are stored in LOB subsegment files. For out-of-line LOBs, a FileTek data loader generates and stores *object identifiers* (OIDs) in rows in the table data file. StorHouse/RM uses OIDs to locate out-of-line LOB values. An OID, 22 bytes in size, contains the following:

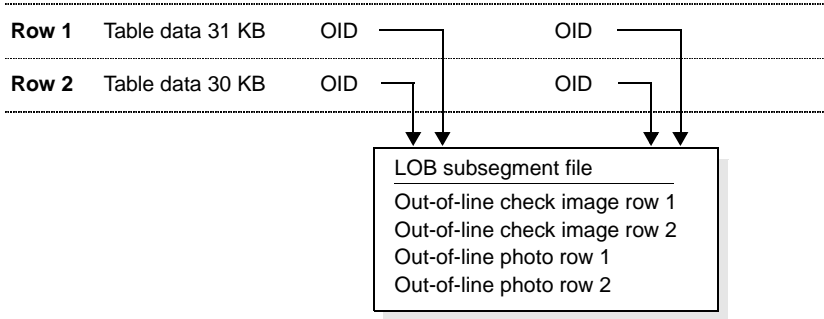
- LOB length (in bytes)
- LOB subsegment ID
- Starting frame number within the LOB subsegment file
- Offset within the LOB subsegment file

When defining a LOB column, you can choose to always store LOB values out-of-line or you can use the default—store LOB values in-line when possible and out-of-line when necessary. By default, a FileTek data loader attempts to store a column's values in the same LOB subsegment file. For instance, in the following example, the check images would reside in one LOB subsegment file and the photo IDs in a different LOB subsegment file.

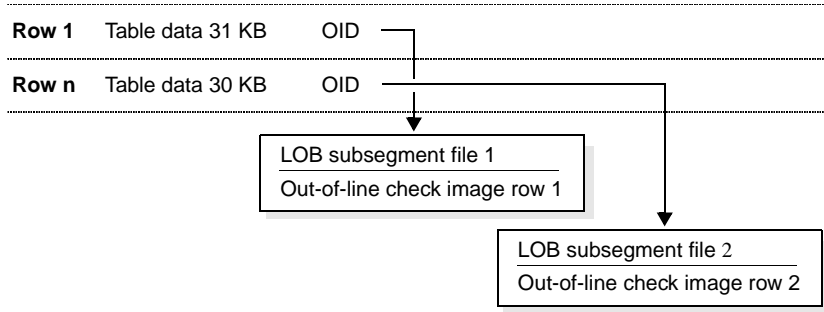


You can also choose to store values of different LOB columns in the same LOB subsegment file. In other words, multiple LOB values in multiple columns can

share the same LOB subsegment file. For instance, the check images and photo IDs can share one LOB subsegment file.

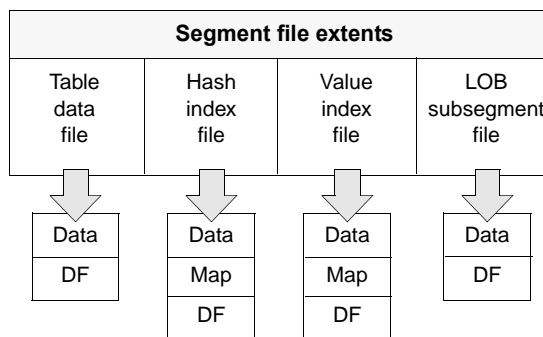


If a LOB subsegment file reaches the maximum file size (approximately 100 GB), a FileTek data loader creates a new file as needed. This means that values for a LOB column may actually be stored in multiple LOB subsegment files.



Extents

StorHouse files consist of *extents*, which are contiguous data or control information that StorHouse treats as a unit. The following graphic shows the different types of extents for segment files.



All files have data and DF extents. Index files also have map extents. Definitions of these extents are as follows:

- A *data extent* contains user data and/or control data.
- A *definitions (DF) extent* contains information necessary to retrieve the data.
- A *map extent* is the high-level index that StorHouse always reads first when doing index lookups.

A user tablespace defines the maximum size of extents and the number of days extents are held in the performance buffer. See “Maximum data extent size” on page 5-10 and “Extent holding period in the performance buffer” on page 5-11 for more information about managing extents.

Storage management on StorHouse

StorHouse/SM provides a file type, called STORHOUSE, for segment files. The StorHouse/SM software automatically manages the movement of these files

through the storage hierarchy based on the storage specifications in a user tablespace. You can also:

- Back up and/or archive files to create duplicate copies for recovery
- Recover files should they become unreadable
- Invalidate segments in a user table, making the files inaccessible
- Merge segments, for instance, consolidate a group of small segments into one segment

Refer to the *StorHouse System Administrator's Guide* for more information about backing up, archiving, and recovering files. Refer to the *FileTek FTP Data Loader Manual* or the *FileTek MVS Data Loader Utility Manual* for more information about invalidating and merging segments.

Naming conventions

This section describes naming conventions for StorHouse databases and database components. In general:

- Database user components have user-assigned names and system-assigned identifiers.
- Segments have user-assigned tags and system-assigned identifiers and StorHouse file names.
- Database system components have system-assigned identifiers and UNIX file names.

Database names

A StorHouse database name:

- Must be unique
- Must start with a letter
- Cannot exceed 32 characters
- Is case sensitive
- Can consist of any combination of a–z (lowercase), A–Z (uppercase), 0–9, and underscore (_)

Some considerations for naming databases are as follows:

- If you use StorHouse in conjunction with a local database, follow the naming rules for your local database.
- StorHouse uses the database name (stored in uppercase) as part of the file name for segment files. For database names longer than 16 characters, StorHouse uses a database alias instead of the actual database name. See “Segment and LOB subsegment identifiers” on page 2-19 for more information about database aliases.
- If you plan to export database data from one StorHouse system and import that data into another StorHouse system, the database name cannot be longer than 16 characters.
- You cannot use case to differentiate StorHouse database names. For instance, you can’t name one database CUSTOMER and a second database customer or Customer. Database names must be unique.

Database user component names

When you create a database user component, you give it a name. You assign names to user tablespaces, user tables, columns, user table indexes, views, and

synonyms. A database user component name should follow these SQL identifier conventions:

- Start with a letter
- Cannot be a StorHouse SQL reserved word (refer to the *StorHouse SQL Reference Manual* for a list of reserved words)
- Can contain up to 32 contiguous characters (no blanks) including a-z (lowercase), A-Z (uppercase), 0-9, and underscore (_)

Delimited names

If a database user component name does not follow SQL identifier conventions, you must delimit it with double quotes in SQL statements. You must also use the appropriate case because delimited SQL identifiers are case sensitive.

For example, if you create a user tablespace with this delimited name:

```
CREATE TABLE SPACE "oct1999"
```

then you must delimit the name and use the same case in other SQL statements:

```
ALTER TABLE SPACE "oct1999"
```

Unique names

Database user component names must be unique as follows:

- User tablespace names must be unique within a database.
- For user tables, the combination of owner and table name must be unique from all table, view, and synonym names in a database.
- Column names must be unique within a user table, but they do not have to be unique within a database.

- For indexes, the combination of owner and index name must be unique from all other index names in a database.
- For views, the combination of owner and view name must be unique from all table, view, and synonym names in a database.
- For synonyms, the combination of owner and synonym name must be unique from all table, view, and synonym names in a database.

Database user component identifiers

StorHouse/RM assigns numeric component identifiers (IDs) to user tablespaces, user tables, and indexes. You can use these IDs to access information in system tables. The following rules apply to system-assigned identifiers.

- Tablespace IDs – Tablespaces are numbered sequentially within a database. User tablespace IDs start with 2.
- Table IDs – Tables are numbered sequentially within a database, meaning no two tables have the same component ID. User table IDs start with 100.
- Index IDs – Indexes are numbered sequentially within the database, meaning no two indexes have the same component ID. User table index IDs start with 100.

Segment and LOB subsegment identifiers

StorHouse/RM assigns each segment a *segment ID*, numbered sequentially by user table. The first segment ID for each user table is 0, the second segment ID is 1, and so on.

StorHouse/RM assigns each LOB subsegment file a *subsegment ID*, numbered sequentially within a segment. The first subsegment ID is 1. If a user table contains one LOB column, and that LOB column requires multiple subsegment

files, then subsegment IDs increment sequentially, that is, the first subsegment ID is 1, the second subsegment ID is 2, and so on.

If a user table contains multiple LOB columns, and each LOB column requires one LOB subsegment file, then subsegment ID 1 identifies the first LOB column, subsegment ID 2 identifies the second LOB column, and so on.

If a user table contains multiple LOB columns (for instance, two) and each LOB column requires multiple LOB subsegment files (for instance, two), then subsegment ID 1 identifies the first file for the first LOB column, subsegment ID 2 identifies the first file for the second LOB column, subsegment ID 3 identifies the second file for the first LOB column, and subsegment ID 4 identifies the second file for the second LOB column.

Segment tags

During a load, you can optionally name the current segment you're loading in case you need to replace or merge the segment in the future. This name is called a *segment tag*. If you don't supply a segment tag, the load ID is the default. Segment tags are stored in the SYSSTHSEGMENTS system table. Refer to the *FileTek MVS Data Loader Utility Manual* or the *FileTek FTP Data Loader Manual* for more information about naming, replacing, and merging segments.

Segment file names

Segment files have StorHouse file names. You can use file names in StorHouse Command Language commands. The file naming convention for these files may differ by StorHouse/RM release. A file always retains the original file name.

StorHouse/RM 3.0 (and higher) segment file names

The file naming convention for table data files, index files, and LOB subsegment files created in StorHouse/RM 3.0 and higher is as follows:

database_name.system_id.typecomponent_id.segment_id.lob_subsegment_id

File naming convention for StorHouse/RM 3.0 and higher

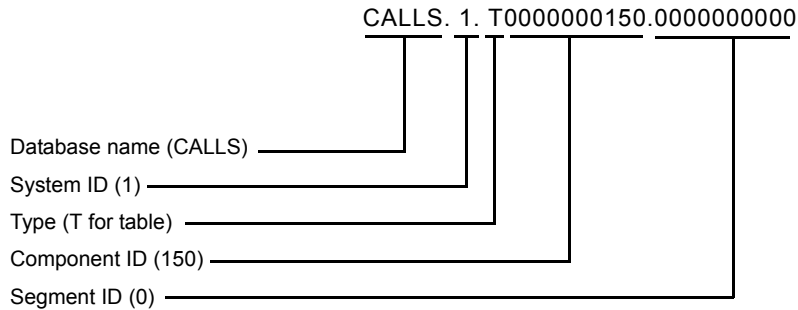
File name item	Length	Description
database_name	up to 16 characters	Name of the StorHouse database. If the database name exceeds 16 characters, StorHouse uses a database alias consisting of the first 12 characters of the database name, followed by the @ character and three decimal digits. For example, StorHouse replaces the database name BILLDETAILWAREHOUSE with the database alias BILLDETAILWA@001.
system_id	up to 6 decimal digits	Value of the StorHouse SYSTEM_ID system parameter
type	1 character	Type of component: <ul style="list-style-type: none"> ■ H for hash index ■ L for LOB ■ T for table ■ V for value index
component_id	10 decimal digits	Table ID or index ID. If the type is L (for LOB), the component_id is the table ID.
segment_id	10 decimal digits	Segment ID containing the file
lob_subsegment_id	9 decimal digits	Subsegment ID containing the LOB

2

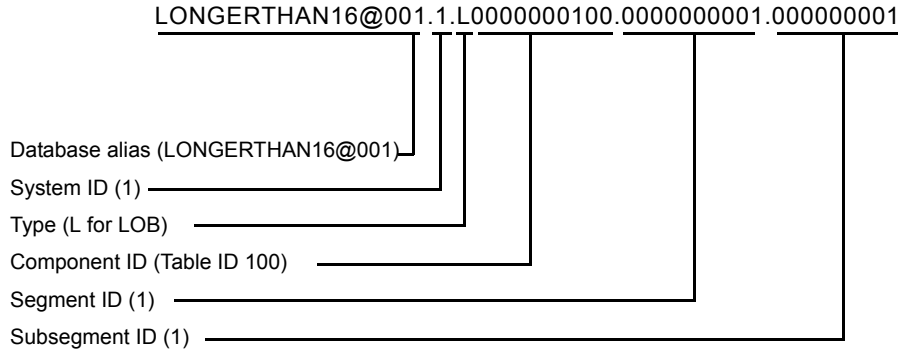
StorHouse database concepts

Naming conventions

For example, the following StorHouse file name identifies a table data file in the first segment (segment ID 0) of a user table (table ID 150) in the CALLS database.



The following StorHouse file name identifies the first LOB subsegment file in the second segment (segment ID 1) of a user table (table ID 100) in the LONGERTHAN16CHARACTERSDBNAME database.



StorHouse/RM 2.3 segment file names

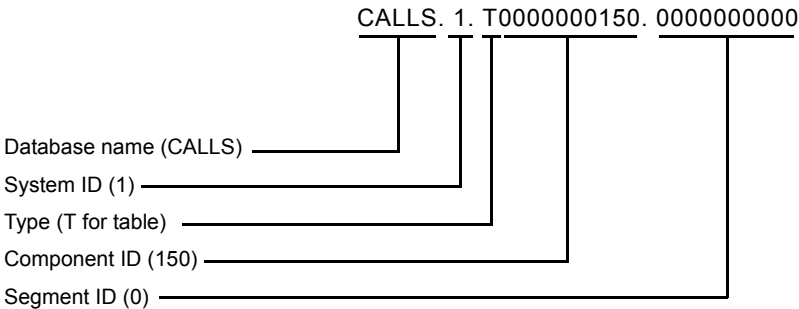
The file naming convention for table data and index files created in StorHouse/RM 2.3 is as follows:

database_name.system_id.typecomponent_id.segment_id

File naming convention for StorHouse/RM 2.3

File name item	Length	Description
database_name	up to 16 characters	Name of the StorHouse database. If the database name exceeds 16 characters, StorHouse uses a database alias.
system_id	up to 6 decimal digits	Value of the StorHouse SYSTEM_ID system parameter
type	1 character	Type of component: <ul style="list-style-type: none">■ H for hash index■ T for table■ V for value index
component_id	10 decimal digits	Table ID or index ID
segment_id	10 decimal digits	Segment ID containing the file

For example, the following StorHouse file name identifies a table data file in the first segment (segment ID 0) of a user table (table ID150) in the CALLS database.

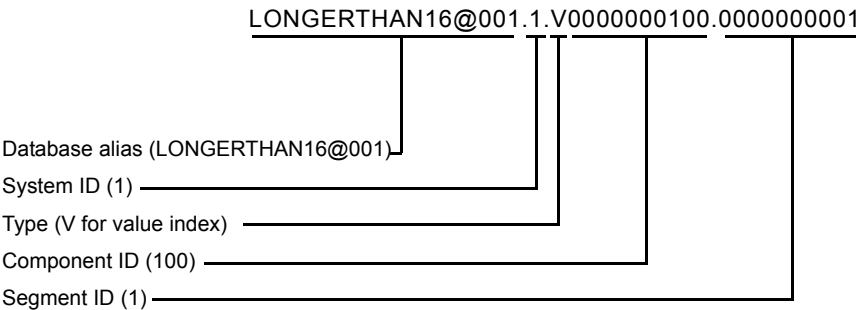


2

StorHouse database concepts

Naming conventions

The following StorHouse file name identifies a value index file in the second segment (segment ID1) of a user table for value index ID 100 in the LONGERTHAN16CHARACTERSDBNAME database.



StorHouse/RM 2.1 to 2.2A segment file names

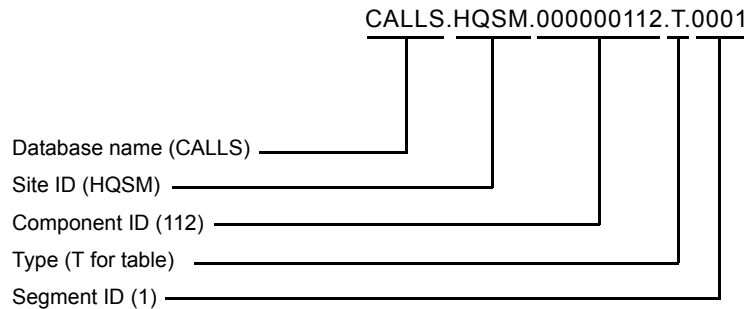
The file naming convention for table data and index files created in StorHouse/RM release 2.1 to 2.2A is as follows:

database_name.site_id.component_id.type.segment_id

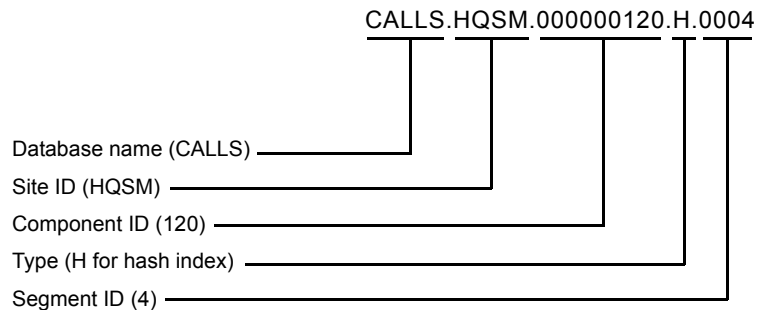
File naming convention for StorHouse/RM 2.1 to 2.2A

File name item	Length	Description
database_name	up to 32 characters	Name of the StorHouse database
site_id	14 characters	Value of the StorHouse SITE_ID system parameter
component_id	9 decimal digits	Table ID or index ID
type	1 character	Type of component: <ul style="list-style-type: none">■ H for hash index■ T for table■ V for value index
segment_id	4 decimal digits	Segment ID containing the file

For example, the following StorHouse file name (in the StorHouse/RM release 2.1 to 2.2A format) identifies a table data file in table ID 112. This segment (ID 1) is the second segment in the user table (the first segment has segment ID 0).



The following StorHouse file name identifies a hash index file in index ID 120. This segment (ID 4) is the fifth segment in the user table.



Database system component file names

Each system component in a system tablespace has a UNIX file name. The following table lists the UNIX file naming convention for StorHouse system components. The letter n indicates the table ID or index ID.

UNIX file name format for system components

Component	Name format	Example
System table	Tnnnnnnnnnn.TBL	T0000000012.TBL
System table index	Innnnnnnnnn.IDX	I0000000005.IDX
System table log	Tnnnnnnnnnn.LOG	T0000000012.LOG
System table index log	Innnnnnnnnn.LOG	I0000000005.LOG

Note: When you access system tables through SQL, you use the system table name rather than the UNIX file name. See Appendix A, “StorHouse system tables,” for more information about using system table names in SQL statements.

Database system component identifiers

StorHouse/RM assigns numeric component identifiers (IDs) to system tablespaces, system tables, and system table indexes.

- Tablespace ID 0 is the system tablespace and tablespace ID 1 is the temporary tablespace.
- System table IDs start with 0.
- System table index IDs start with 0.

Summary of StorHouse database concepts

StorHouse databases consist of user components and system components. The following general rules apply to database user components:

- User tables represent data.
- User tables consist of one or more segments.
- A segment is a set of files containing table data, index entries, and LOB data.
- Each file is a STORHOUSE-type file on StorHouse.
- StorHouse/RM supports three types of indexes: value, hash, and range.
- Value and hash indexes are stored in index files.
- Range indexes are stored in system tables.
- You can assign user tables, indexes, and LOB columns to user tablespaces.
- User tablespaces contain subspaces that define how to store table files, index files, and LOB subsegment files on StorHouse.

The following general rules apply to database system components:

- System tables contain data describing a database as well as range index entries.
- Each system table is a separate UNIX file.
- System components are stored in the system tablespace.
- A system tablespace is a database directory in UNIX.
- Each database has a system tablespace.

2

StorHouse database concepts

Summary of StorHouse database concepts

StorHouse database administration basics

This chapter describes the basics of StorHouse database administration, such as:

- The roles of people who manage StorHouse databases
- Tools and resources for managing StorHouse databases
- How to perform database administration tasks
- An account for performing database administration tasks

People who manage StorHouse databases

Working with StorHouse databases requires planning and coordination among several people. Because organizations define job responsibilities differently, each person's exact role in StorHouse database administration may vary from site to site. A list of typical roles and responsibilities follows.

- StorHouse chief database administrator (DBA) – Manages all StorHouse databases, creates database user components, and establishes security plans to control access to StorHouse
- StorHouse departmental DBA – Designs and maintains a single StorHouse database, creates database user components, and controls access to database data
- StorHouse system administrator – Works with StorHouse DBAs to develop a storage strategy and to maximize the capabilities of StorHouse

- Application developer – Analyzes data and develops application programs that access data stored in StorHouse
- FileTek customer support representative – Assists with the installation and configuration of StorHouse/RM and provides other support functions as contracted

You in this manual refers to the StorHouse system administrator and to StorHouse DBAs.

Tools to help you manage StorHouse databases

Managing a StorHouse database involves a mix of database administration tasks and system administration tasks. You perform these tasks with StorHouse SQL statements, StorHouse Command Language commands, and StorHouse utilities.

StorHouse SQL statements

The SQL you use for StorHouse database administration is compatible with the ANSI SQL-99 standard. There are two important differences:

- StorHouse SQL includes extensions to the standard. (An *extension* is an extra option or feature.)
- StorHouse does not support all of the ANSI statements.

Because of these exceptions, the SQL you use to manage a StorHouse database may differ slightly from the SQL you use to manage your local database.

The following table lists the SQL statements you use to manage a StorHouse database.

SQL statements for StorHouse database administration

Use	To
ALTER TABLE SPACE	Change storage specifications in a user tablespace
CREATE EXPLAIN TABLE	Create an explain table
CREATE INDEX	Create an index for a user table
CREATE SYNONYM	Create a synonym
CREATE TABLE	Create a user table or re-create a dropped user table
CREATE TABLE SPACE	Create a user tablespace
CREATE VIEW	Create a view
DELETE	Remove a default tablespace entry for an account
DROP EXPLAIN TABLE	Remove an explain table
DROP INDEX	Remove an index for a user table
DROP SYNONYM	Remove a synonym
DROP TABLE	Prepares a user table for purging
DROP TABLE SPACE	Remove a user tablespace
DROP VIEW	Remove a view
EXPLAIN PLAN	Run the explain facility to post data to explain tables
GRANT	Grant privileges to a StorHouse account
INSERT	Assign a default tablespace to an account or database
PURGE TABLE	Removes metadata and segment files for a dropped user table
RENAME	Renames user tables, views, and synonyms
REVOKE	Remove privileges from a StorHouse account

SQL statements for StorHouse database administration (continued)

Use	To
SELECT	View information in system and user tables
UPDATE	Change the default tablespace for an account or database

Refer to the *StorHouse SQL Reference Manual* for formats and descriptions of these SQL statements. This guide contains examples.

StorHouse Command Language

You or the StorHouse system administrator use StorHouse Command Language commands to complete system administration tasks required for StorHouse/RM. Just as SQL is a set of statements you use to work with a relational database, the *StorHouse Command Language* is a set of commands for managing StorHouse resources. For instance, you use StorHouse commands to create volume sets and file sets for segment files and to create StorHouse accounts for users who will access StorHouse databases. The following table contains some of the StorHouse Command Language commands that you or the StorHouse system administrator may use.

Commands for StorHouse system administration

Use	To
ARCHIVE	Create an archive copy of a segment file or a metadata backup file
CHECKPOINT	Create a checkpoint file of the StorHouse directory after backing up metadata
CREATE ACCOUNT	Create StorHouse accounts
CREATE BACKUP	Create a backup copy of a segment file or a metadata backup file
CREATE FSET	Create file sets for segment files

Commands for StorHouse system administration (continued)

Use	To
CREATE VSET	Create volume sets for segment files
REMOVE ACCOUNT	Delete StorHouse accounts
RUN	Invoke the metadata backup utility, metadata restore utility, redo journaling utilities, and segment delete utility
SCHEDULE	Schedule utilities to run automatically
SET ACCOUNT	Change StorHouse account passwords and change access and command privileges
SET SYSTEM	Set or change StorHouse system parameters
SHOW ACCOUNT	Display StorHouse account IDs and access and command privileges
SHOW FILE	Display StorHouse file information for table files, index files, LOB subsegment files, and metadata backup files
SHOW FSET	Display information about file sets created for user tablespaces
SHOW SYSTEM	Display values for StorHouse system parameters
SHOW VSET	Display information about volume sets created for user tablespaces

Refer to the StorHouse *Command Language Reference Manual* for complete descriptions of these and other StorHouse commands.

StorHouse utilities

You or the StorHouse system administrator run specific StorHouse utilities to perform database administration tasks. The following table lists the utilities.

Utilities for StorHouse database administration

Use	To
sthdb_backup	Back up metadata and optionally start journaling for an existing database
sthdb_down	Take a StorHouse database offline manually and force it into a down state
sthdb_restore	Recover metadata for one or more StorHouse databases from metadata backup files
sthjou_archive	Archive a redo journal on StorHouse
sthjou_audit	Verify that a redo journal chain is readable
sthjou_cycle	Close the current redo journal and start a new one
sthjou_replay	Apply a redo journal chain to finish restoring a StorHouse database
sthseg_delete	Delete invalidated segments from a StorHouse database and purges dropped user tables
sthdb_up	Reset a downed StorHouse database to an up (online) state
sthtbl_audit	Validate the structure of a StorHouse system table
syscreate	Create a StorHouse database and optionally enable journaling for the database

Resources to help you manage StorHouse databases

StorHouse provides the following resources to support database administration: system parameters, user log, and system tables.

System parameters

StorHouse *system parameters* enable you to configure how StorHouse manages various resources or operations. A specific set of system parameters affects StorHouse/RM. For instance, you can specify the maximum number of StorHouse engines that can run concurrently, and you can control the maximum number of concurrent StorHouse engines used to load data.

Some system parameters are informational and others are user-definable. FileTek supplies default values for the user-definable system parameters. You and the StorHouse system administrator should decide whether to use or change the supplied default values for the system parameters that affect StorHouse/RM. You use the StorHouse Command Language SET SYSTEM and SHOW SYSTEM commands to work with system parameters.

See Appendix C, “System parameters for StorHouse/RM,” for a description of the system parameters you use to tune StorHouse/RM and for instructions on displaying and setting StorHouse system parameters. Refer to the StorHouse *Command Language Reference Manual* for a complete description of all StorHouse system parameters.

User log

To help you monitor system activity, StorHouse records information in the *user log*—a file that provides statistical and administrative information about certain

events and various StorHouse activities. You can capture the following StorHouse/RM information in the user log:

- Database connects and disconnects
- Unsuccessful connects (possible security violations)
- SQL statements submitted
- Transaction statistics for each completed transaction, including number of bytes retrieved
- Segment files queried, loaded successfully, and at checkpoint (due to load failure)

You must set various system parameters to enable logging in general. Additionally, whether StorHouse logs SQL, transaction, and file information depends on how you set these system parameters:

- LOG_SQL_STMT
- LOG_SQL_TRANS
- LOG_FILE

Note that the default setting for LOG_SQL_STMT and LOG_SQL_TRANS is TRUE (for logging). Refer to the StorHouse *User Log Format* manual for more information about the user log and the types of events recorded. Refer to the StorHouse *System Administrator's Guide* for more information about how to enable logging.

System tables

StorHouse/RM creates and maintains system tables for each StorHouse database. System tables contain useful information for administration. For example, you can query the SYSTABLES system table to list user table and view names in a database or the SYSSTHSPACES system table to display storage specifications for

a user tablespace. You can also maintain the SYSSMUSERS system table to assign default user tablespaces to accounts and a database. See Appendix A, “StorHouse system tables,” for more information about StorHouse system tables.

How to perform administration tasks

You perform StorHouse database administration tasks by submitting StorHouse SQL statements and running StorHouse utilities. You perform StorHouse system administration tasks by submitting StorHouse Command Language commands.

Submitting StorHouse SQL statements

You can submit SQL to StorHouse with any of the following tools:

- C or C++ programs through StorHouse ESQ
- Local database programs through a gateway or a federator
- FileTek data loaders (MVS and FTP)
- StorHouse/Control Center Interactive SQL (ISQL)

After you submit an SQL statement, StorHouse/RM generates an SQL status code. Refer to the *StorHouse SQL Reference Manual* for a list of SQL status codes.

Issuing StorHouse commands

You can issue StorHouse Command Language commands with any terminal or workstation that enables you to start a StorHouse session via the Interactive Interface. The StorHouse *Interactive Interface* lets you communicate directly with StorHouse. From your workstation you can sign on to StorHouse, issue StorHouse Command Language commands, and receive StorHouse messages.

You can also issue StorHouse Command Language commands with the StorHouse/Admin module of StorHouse/Control Center. Refer to the *StorHouse Command Language Reference Manual* or the *StorHouse/Control Center User Document Set* for more information about issuing StorHouse commands.

Running StorHouse utilities

You can run the following StorHouse utilities with the StorHouse Command Language RUN and SCHEDULE commands.

- Metadata backup (sthdb_backup)
- Segment delete (sthseg_del)
- Journal cycle (sthjou_cycle)
- Journal archive (sthjou_archive)

You must run the following utilities at the StorHouse operating system (UNIX) command prompt:

- Database create (syscreate)
- Database down (sthdb_down)
- Database up (sthdb_up)
- Metadata restore (sthdb_restore)
- Journal audit (sthjou_audit)
- Journal replay (sthjou_replay)

In order to run a utility from the StorHouse operating system prompt, you must log in to the StorHouse server using the UNIX operator account. You enter the entire command and its arguments on a single line.

In order to run a utility with the RUN or SCHEDULE command, you must sign on to StorHouse (start a StorHouse session) with a StorHouse account that has OPERATOR, SYSTEM, or SERVICE privilege. Note that you can't direct the output of a RUN command to a file.

Most of the utilities have an online help option (-h), which displays a description of the command. The following example displays help for the sthdb_up utility:

```
$STHROOT/bin/sthdb_up -h
```

Account for performing database and system administration tasks

StorHouse includes one pre-defined account—*SYSADM*—that you can use to perform all StorHouse system and database administration tasks. The initial password for this account is *sysadm*. Be sure to change the password after installation and periodically as needed. See page 4-7 for more information about *SYSADM*. See page 4-20 for instructions on changing the initial *SYSADM* account password. Do not delete or disable the *SYSADM* account.

3

StorHouse database administration basics

Account for performing database and system administration tasks

Accounts and privileges

This chapter describes StorHouse accounts and privileges and explains how to:

- Create an account
- Assign privileges to an account
- Remove privileges from an account
- Change an account password
- List account IDs
- List privileges assigned to accounts
- Remove an account

About StorHouse accounts

A *StorHouse account* with a specified set of StorHouse privileges enables a user to access StorHouse databases. Users need a StorHouse account to:

- Perform StorHouse system and database administration tasks
- Submit StorHouse SQL statements
- Load and unload StorHouse data
- Back up metadata
- Access StorHouse data from a host language application (using ESQL) or a local database application (such as a DB2 UDB or Oracle application)

One StorHouse account can be dedicated to a single user or shared by many users. You create, update, and remove StorHouse accounts with the StorHouse

Command Language CREATE ACCOUNT, SET ACCOUNT, and REMOVE ACCOUNT commands.

Account IDs

When you create a StorHouse account, you assign an account ID. A *StorHouse account ID* consists of 1 to 12 characters that may include A–Z (uppercase), 0–9, \$ (dollar sign), and _ (underscore). StorHouse always converts account IDs to uppercase. FileTek recommends that account IDs follow StorHouse SQL identifier naming conventions, but you can delimit account IDs that do not follow those conventions. Refer to the *StorHouse SQL Reference Manual* for more information about specifying delimited account IDs in SQL statements.

Account passwords

You assign an account password when you create a StorHouse account. An *account password* can contain up to 32 characters consisting of the following ASCII characters: A–Z (uppercase), 0–9, \$ (dollar sign) and _ (underscore). StorHouse always converts account passwords to uppercase.

When you assign a password, that account must always specify it when:

- Logging into the StorHouse FTP server to load data with the FileTek FTP Data Loader or to unload data with the FileTek FTP Data Unloader
- Connecting to a StorHouse database (in the USING clause of a CONNECT statement)
- Supplying the ACCT value on the SMDEF control statement to load data with the FileTek MVS Data Loader utility

If you don't assign a password, then the account cannot access StorHouse databases.

About StorHouse privileges

StorHouse provides a complete set of privileges—access, command, database, and database component—that defines different functions an account can perform. You assign access and command privileges when creating a StorHouse account. These privileges define the functions an account can perform in all StorHouse databases. You can then grant database and database component privileges to that account. These privileges define the functions an account can perform in a specific StorHouse database.

While there are many StorHouse access and command privileges, a subset of these privileges affects StorHouse/RM. Refer to the StorHouse *Command Language Reference Manual* for a complete list and definition of all access and command privileges. The following table summarizes the access and command privileges and the database and database component privileges for StorHouse/RM.

StorHouse privileges

Type of privilege	Privileges
Access	SQLADMIN
Command	SQLEXECUTE and SQLCOMMAND
Database	DBA, RESOURCE, and SCAN
Database component	ALL, DELETE, INDEX, INSERT, SELECT, and UPDATE

Access privilege

The *SQLADMIN access privilege* enables an account to perform database administration tasks in *all* StorHouse databases. Specifically, SQLADMIN implicitly grants an account the DBA database privilege in all StorHouse databases. However, to actually perform database administration tasks, the account must also have the SQLEXECUTE command privilege. A chief DBA, for example, might have this SQLADMIN privilege.

Command privileges

Command privileges enable accounts to submit certain StorHouse commands. While there are many StorHouse command privileges, two command privileges are specific to StorHouse databases: `SQLEXECUTE` and `SQLCOMMAND`.

`SQLEXECUTE` privilege

The *`SQLEXECUTE` command privilege* enables an account to submit StorHouse SQL statements in all StorHouse databases. An account must have this privilege at a minimum to access StorHouse databases. Although accounts with this privilege can submit SQL statements in *all* StorHouse databases, database and database component privileges restrict access to components in specific databases.

`SQLCOMMAND` privilege

The *`SQLCOMMAND` command privilege* enables an account to run a FileTek data loader and unloader in all StorHouse databases. Specifically, this privilege permits an account to invoke a data loader and unloader, which then submits a StorHouse Command Language `EXECUTE STH_LOAD` command. The `SQLCOMMAND` privilege is just one of several privileges required to load and unload data. See “Loader accounts” on page 4-8 for a list of the privileges required to load data. See “Unloader accounts” on page 4-9 for a list of the privileges required to unload data.

Database privileges

Database privileges control the functions an account can perform in a specific database. There are three StorHouse database privileges:

- `DBA`
- `RESOURCE`
- `SCAN`

If an account has none of these database privileges, then that account is a *general database user account* and has database component privileges on the database only.

DBA privilege

DBA privilege is the highest privilege an account can have on a specific database. Typically, this privilege is reserved for a departmental DBA. (An account with the SQLADMIN access privilege has DBA privilege on all databases.) An account with DBA privilege can perform the following functions for a specific database:

- Create database user components (user tables, indexes, views, and synonyms)
- Create database user components for other accounts
- Access and maintain system tables
- Grant database and database component privileges to accounts
- Revoke database and database component privileges from accounts
- Select data from any table, view, or synonym
- Load all user tables in a database
- Undrop user tables

DBA privilege includes all the privileges provided by the RESOURCE privilege. It does not include SCAN privilege; however, an account with DBA can grant SCAN privilege to other accounts.

RESOURCE privilege

An account with RESOURCE privilege has more database privileges than a general database user account but fewer privileges than an account with DBA privilege. Such an account can do the following for a specific database:

- Create database user components
- Select information from these database user components
- Grant other accounts SELECT and INDEX privileges on these components

SCAN privilege

An account with SCAN privilege can read all rows in any user table (in other words, perform a full table scan on a user table) on which it has SELECT privilege. The purpose of SCAN privilege is to control which accounts can do full table scans. Because full table scans can be very time consuming, you may want to limit their use. Neither RESOURCE nor DBA privilege automatically confers SCAN privilege. This is a separate privilege that must be independently granted to an account.

Accounts that use the StorHouse *extractor* feature to perform full table scans require the SCAN privilege. Accounts that use the FileTek FTP Data Unloader also require SCAN when unloading an entire table or when the unload operation must read all rows in a table without the use of an index.

Database component privileges

Database component privileges determine an account's access to specific database components, such as tables and views or columns within tables and views. Column level privileges apply to system tables only because StorHouse does not allow updates of user tables. StorHouse database component privileges are as follows.

Database component privileges

Privilege	Authorizes the account to
ALL	Have all privileges on a specific table or view
DELETE	Delete rows from a system table or system table view
INDEX	Create an index on a user table
INSERT	Load data into a user table or insert rows into a system table or view
SELECT	Access data in a table or view
UPDATE	Update one or more columns in a system table or system table view

The following rules apply to database component privileges:

- An account with DBA privilege on a database has all available database component privileges on all components.
- An account with RESOURCE privilege on a database has all available database component privileges on any components created by that account.
- General database user accounts have those database component privileges granted to them by other accounts and those available to PUBLIC. If they've been granted no database component privileges by other accounts, they have PUBLIC privileges only.
- No matter what an account's database component privileges, the account has ALL component privileges on any table or view that it owns.

Types of accounts and their privileges

This section describes two special accounts—SYSADM and PUBLIC—as well as the combination of privileges required to create other types of accounts.

SYSADM account

SYSADM, a pre-defined StorHouse account, has all StorHouse privileges. This account provides immediate full access to StorHouse, including every StorHouse database. Using SYSADM, you can:

- Perform StorHouse system administration tasks
- Perform all StorHouse database administration tasks on all databases
- Run StorHouse utilities
- Submit SQL statements in all StorHouse databases
- Do full table scans on all user tables in all StorHouse databases
- Grant SCAN database privilege to other accounts

- Run a FileTek data loader to load data into user tables
- Unload data with the FileTek FTP Data Unloader

PUBLIC account

PUBLIC is a term for the collective group of all StorHouse accounts. You can grant only database component privileges to PUBLIC. You can't grant access, command, and database privileges to PUBLIC. By granting a database component privilege to PUBLIC, you grant it to every account with SQLEXECUTE privilege.

The privileges available to PUBLIC vary from database to database. See page 4-12 for more information about granting and revoking privileges to and from PUBLIC. You can also use PUBLIC to designate a default user tablespace for a database. See page 5-28 for more information about designating a default user tablespace for a database.

General database user accounts

StorHouse accounts with SQLEXECUTE privilege and without database privileges (DBA, RESOURCE, or SCAN) are general database user accounts. These accounts have all privileges granted to PUBLIC and any database component privilege granted to the account.

Loader accounts

To load data into StorHouse user tables, load deferred indexes, and merge segments, a StorHouse account requires a combination of StorHouse command and database component privileges. The minimum command privileges needed to run a FileTek data loader are as follows:

- | | | |
|----------|------------|--------------|
| ■ ATF | ■ PUT | ■ SQLCOMMAND |
| ■ DELETE | ■ RECORD | ■ SQLEXECUTE |
| ■ GET | ■ SETGROUP | ■ VTF |

The account must also have the INSERT database component privilege on the user tables it is loading or have been granted the DBA database privilege. An account with the SQLADMIN or ALLPRIV privilege must have the INSERT or DBA privilege to load user tables. See “Loader accounts” on page 4-16 for an example loader account.

Unloader accounts

To unload data using the FileTek FTP Data Unloader, a StorHouse account requires these privileges:

- SELECT database component privilege on the user table(s) to be unloaded
- SCAN database privilege (if the query results in a full table scan)
- SQLEXECUTE command privilege to submit StorHouse SQL
- SQLCOMMAND command privilege to invoke the unloader

StorHouse utility accounts

To run StorHouse utilities, such as metadata backup and redo journaling, a StorHouse account requires one of the following StorHouse command privileges:

- OPERATOR
- SERVICE
- SYSTEM

The SYSADM account can run the utilities because that account has all command privileges. You must assign the OPERATOR, SERVICE, or SYSTEM privilege to authorize any other account to run utilities. Refer to the *StorHouse Command Language Reference Manual* for more information about these command privileges.

Local database application accounts

To access StorHouse from a local database application (for example, a DB2 or Oracle application), you must create a StorHouse account for the application

user. For instance, DB2 UDB users require StorHouse accounts to access StorHouse databases. Those accounts must be mapped to DB2 authorization IDs. Refer to the *StorHouse/UDB Link Installation and Configuration Manual* for more information about managing account security and privileges in a federated system.

At a minimum, a local database application account must have SQLEXECUTE privilege. Assign other StorHouse privileges as appropriate. For instance, if the account will create user tables and indexes, then assign the RESOURCE privilege.

Host language application accounts

To access StorHouse from a host language application, you create a StorHouse account and specify the account ID in the USER clause and the account password (expressed as a host variable) in the USING clause of your ESQL program CONNECT statement. At a minimum, a host language application account must have SQLEXECUTE privilege. Assign other StorHouse privileges as appropriate. For instance, if the account may perform full table scans, assign the SCAN privilege and the SELECT privilege on all applicable user tables.

Ways to assign privileges

You assign access and command privileges when you create or update accounts with the StorHouse Command Language CREATE ACCOUNT or SET ACCOUNT commands. You assign database and database component privileges with the StorHouse SQL GRANT and REVOKE statements. The GRANT statement grants a database or database component privilege to an account, while the REVOKE statement removes such a privilege from an account.

The following rules apply to granting and revoking database and database component privileges:

- An account with DBA privilege on a given database can grant privileges on that database to any StorHouse account.

- An account with RESOURCE privilege on a database can grant privileges on only those database components that it owns to any StorHouse account.
- If you are authorized to grant or revoke privileges, you can do so for any StorHouse account except your own (you can't grant to or revoke from yourself).
- New privileges take effect with the next request following execution of the GRANT or REVOKE statement.
- Only accounts with the SQLEXECUTE privilege can use their privileges. For example, if you grant SCAN privilege to an account but that account does not have SQLEXECUTE, then the account cannot scan user tables.

As mentioned, you grant database and database component privileges using the GRANT statement. The following example grants SELECT privilege on the SALARY table to USER1:

```
GRANT SELECT ON SALARY  
TO USER1
```

The WITH GRANT OPTION

The WITH GRANT OPTION argument on the GRANT statement enables accounts to grant their privileges, or a subset of their privileges, to other accounts. If you omit the argument, the account to which you're granting privileges (grantee) cannot grant those privileges to other accounts.

For example, to grant USER1 the SELECT privilege on the SALARY table with the WITH GRANT OPTION argument, issue the following statement:

```
GRANT SELECT ON SALARY  
TO USER1  
WITH GRANT OPTION
```

Now USER1 can grant the SELECT privilege on the SALARY table to other accounts.

When you grant to PUBLIC

You can grant and revoke database component privileges to and from PUBLIC. By doing this, you can grant or revoke privileges to all accounts with just one GRANT or REVOKE statement.

For instance, suppose you just created a new table called RECORDS and you'd like to grant SELECT privilege on that table to all StorHouse accounts. Instead of submitting individual GRANT statements for each account, you can grant SELECT privilege on the RECORDS table to PUBLIC in one statement as follows:

GRANT SELECT ON RECORDS TO PUBLIC

Now all StorHouse accounts with SQLEXECUTE privilege can select information from RECORDS.

You can also revoke privileges from PUBLIC. For instance, should you need to restrict access to RECORDS from all accounts but three, you'd revoke SELECT privilege from PUBLIC and then grant SELECT privilege to the three accounts that require the privilege. You'd issue the following statement to revoke SELECT from PUBLIC:

REVOKE SELECT ON RECORDS FROM PUBLIC

Remember the following:

- PUBLIC privileges vary from database to database. For each database there is a separate set of privileges available to PUBLIC.
- Any StorHouse account with SQLEXECUTE has PUBLIC access to all StorHouse databases.

- An account's privileges are the combination of those granted to PUBLIC and those granted to the account.
- If PUBLIC has SELECT privilege on a table, then all accounts have SELECT privilege on the table. Revoking SELECT from individual accounts when PUBLIC has SELECT has no effect.

System tables with privilege information

StorHouse/RM records database and database component privileges in three system tables. StorHouse/RM updates these system tables when you grant and revoke privileges.

- SYSDBAUTH – Contains accounts and their database privileges (DBA, RESOURCE, and SCAN).
- SYSCOLAUTH – Contains accounts with the UPDATE database component privilege on system table columns.
- SYSTABAUTH – Contains accounts and their database component privileges (DELETE, INDEX, INSERT, SELECT, and UPDATE) on tables and views.

Summary of StorHouse privileges

StorHouse privileges define the functions an account can perform and the database components an account can access. In summary:

- Four types of StorHouse privileges are access, command, database, and database component.
- Access (SQLADMIN) and command (SQLEXECUTE and SQLCOMMAND) privileges enable an account to perform specific functions in all StorHouse databases.

- Database (DBA, RESOURCE, and SCAN) and database component (ALL, DELETE, INDEX, INSERT, SELECT, and UPDATE) privileges enable an account to perform specific functions and access certain components in specific StorHouse databases.
- You assign an access or command privilege when you create an account. You create accounts using the StorHouse Command Language CREATE ACCOUNT command.
- You assign a database or database component privilege by using the StorHouse SQL GRANT statement.
- You remove or change an access or command privilege by using the StorHouse Command Language SET ACCOUNT command.
- You remove a database or database component privilege by using the StorHouse SQL REVOKE statement.

Creating a StorHouse account

You can create a StorHouse account by submitting the StorHouse Command Language CREATE ACCOUNT command. You must assign one or more of the access and command privileges—SQLADMIN, SQLEXECUTE, and SQLCOMMAND—to the account. You must also assign an account password.

Keep in mind the following when creating a StorHouse account:

- Assign an account ID that conforms to SQL identifier naming conventions. See “Account IDs” on page 4-2 for more information about naming conventions.
- If you don’t assign an account password, then the account cannot access StorHouse databases.

- If you disable an account with the /DISABLED parameter modifier, the account cannot access StorHouse databases.
- To give the account complete DBA access to all StorHouse databases, assign the SQLADMIN privilege. An account with SQLADMIN, however, also needs SQLEXECUTE to perform database tasks.
- To enable any account to access StorHouse databases, assign the SQLEXECUTE privilege. The account's access to individual databases is then determined by database and database component privileges granted to the account and database component privileges granted to PUBLIC.
- To enable an account to run the FileTek MVS Data Loader utility, the FileTek FTP Data Loader, or the FileTek FTP Data Unloader, assign the SQLCOMMAND privilege. You must also assign other command privileges and grant database component privileges as listed at "Loader accounts" on page 4-8 and "Unloader accounts" on page 4-9.
- To enable an account to run the metadata backup utility, assign the OPERATOR, SERVICE, or SYSTEM privilege.
- To enable an account to perform full table scans or to use the extractor feature, assign the SCAN privilege.

▼ **To create a StorHouse account**

Required privileges: ACCOUNT, ANYACCOUNT, and SETGROUP

1. Sign on to StorHouse.
2. At the command prompt (?), submit a StorHouse Command Language CREATE ACCOUNT command and press Enter. Some examples follow.

- To create the account **USER1** with a password of **SPRING** and with **SQLEXECUTE** command privilege, type:

```
CREATE ACCOUNT USER1 SPRING /PRIVILEGE=SQLEXECUTE
```

- To create the account **USER2** with a password of **SUMMER** and with **SQLADMIN** access privilege and **SQLEXECUTE** command privilege, type:

```
CREATE ACCOUNT USER2 SUMMER /PRIVILEGE=(SQLADMIN,  
SQLEXECUTE)
```

- To create the account **LOADER** with a password of **FALL** that can load data and run the metadata backup utility, type:

```
CREATE ACCOUNT LOADER FALL /PRIVILEGE=(OPERATOR,  
SQLEXECUTE, SQLCOMMAND, RECORD, PUT, GET, SETGROUP,  
DELETE, ATF, VTF)
```

Granting database privileges

You can grant the **DBA**, **RESOURCE**, or **SCAN** database privilege by submitting a **GRANT** statement. StorHouse accounts require **SQLEXECUTE** privilege to use their database privileges.

▼ To grant database privileges to a StorHouse account

Required privileges: **SQLEXECUTE** and **DBA**

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a **GRANT** statement to grant a database privilege. Some examples follow.

- To grant DBA and SCAN privileges to USER1, type:

```
GRANT DBA, SCAN  
TO USER1
```

- To grant RESOURCE privilege to USER2, type:

```
GRANT RESOURCE  
TO USER2
```

Granting database component privileges

You can grant database component privileges to accounts by submitting a GRANT statement. StorHouse accounts require SQLEXECUTE privilege to use their database component privileges. Database component privileges are:

- ALL
- DELETE (only for system tables and system table views)
- INDEX
- INSERT (only for system tables, system table views, and loading user tables)
- SELECT
- UPDATE (only for system tables and system table views)

▼ To grant a database component privilege to a StorHouse account

Required privileges: SQLEXECUTE and DBA or component owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a GRANT statement to grant a database component privilege. Some examples follow.

- To grant INDEX privilege on the CUSTOMERS table to USER1, type:

```
GRANT INDEX  
ON CUSTOMERS  
TO USER1
```

- To grant SELECT privilege on the CUSTOMERS table to USER1 and grant USER1 the WITH GRANT OPTION to grant SELECT privilege to other accounts, type:

```
GRANT SELECT  
ON CUSTOMERS  
TO USER1  
WITH GRANT OPTION
```

Revoking database privileges

You can revoke the DBA, RESOURCE, or SCAN database privilege from an account by submitting a REVOKE statement. See “Listing database privileges” on page 4-25 to verify an account’s database privileges. If you’re removing all database privileges for an account, you can specify all the privileges in one REVOKE statement.

▼ To revoke database privileges from a StorHouse account

Required privileges: SQLEXECUTE and DBA

1. Follow your site’s procedure for connecting to your StorHouse database.
2. Submit a REVOKE statement to revoke a database privilege from an account. Some examples follow.

- To revoke the DBA privilege from USER1, type:

```
REVOKE DBA  
FROM USER1
```

- To revoke all three database privileges from USER2, type:

```
REVOKE DBA, RESOURCE, SCAN  
FROM USER2
```

Revoking database component privileges

You can revoke database component privileges—ALL, DELETE, INDEX, INSERT, SELECT, and UPDATE—from an account by submitting a REVOKE statement. See “Listing database component privileges” on page 4-26 to verify an account’s database component privileges. If you’re removing all database component privileges for an account, you can revoke all privileges in one REVOKE ALL statement.

Note: You cannot revoke a database component privilege from that component’s owner.

▼ To revoke database component privileges from a StorHouse account

Required privileges: SQLEXECUTE and DBA or component owner

1. Follow your site’s procedure for connecting to your StorHouse database.
2. Submit a REVOKE statement to revoke a database component privilege from an account. Some examples follow.

- To revoke the INDEX privilege on the CUSTOMERS table from USER1, type:

```
REVOKE INDEX  
ON CUSTOMERS  
FROM USER1
```

- To revoke all privileges on the CUSTOMERS table from USER1, type:

```
REVOKE ALL  
ON CUSTOMERS  
FROM USER1
```

Changing a StorHouse account password

You can change an account password by using the StorHouse Command Language SET ACCOUNT command. The new password takes effect the next time the account accesses StorHouse.

▼ To change an account password

Required privileges: ACCOUNT, ANYACCOUNT, PASSWORD, and SETGROUP

1. Sign on to StorHouse.
2. At the command prompt (?), submit the StorHouse Command Language SET ACCOUNT command with the /NEWPASSWORD parameter modifier and press **Enter**. An example follows.

- To change USER1's password, type the following command:

```
SET ACCOUNT USER1 /NEWPASSWORD
```


The system responds:

Enter new password or RETURN for none?

Reenter new password or RETURN for none?

Enter the new password in response to each prompt. The passwords entered do not display on the screen.

Adding an access or command privilege

You can add an access or command privilege to a StorHouse account by using the StorHouse Command Language SET ACCOUNT command. The new privilege takes effect the next time the account accesses StorHouse.

▼ To add an access or command privilege to a StorHouse account

Required privileges: ACCOUNT, ANYACCOUNT, and SETGROUP

1. Sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SET ACCOUNT command and press **Enter**. Some examples follow.

- To add SQLCOMMAND to USER1's StorHouse privileges, type:

```
SET ACCOUNT USER1 /PRIVILEGE=SQLCOMMAND
```

- To add SQLADMIN and SQLCOMMAND to USER3's StorHouse privileges, type:

```
SET ACCOUNT USER3 /PRIVILEGE=(SQLADMIN,SQLCOMMAND)
```

- To add OPERATOR to USER2's StorHouse privileges, type:

```
SET ACCOUNT USER2 /PRIVILEGE=OPERATOR
```

Removing an access or command privilege

You can remove an access or command privilege from a StorHouse account by using the StorHouse Command Language SET ACCOUNT command and preceding the privilege to be removed with NO. Removed StorHouse access and command privileges take effect the next time the account accesses StorHouse.

▼ To remove an access or command privilege from a StorHouse account

Required privileges: ACCOUNT, ANYACCOUNT, and SETGROUP

1. Sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SET ACCOUNT command and press **Enter**. Some examples follow.

- To remove SQLCOMMAND from USER1's StorHouse privileges, type:

```
SET ACCOUNT USER1 /PRIVILEGE=NOSQLCOMMAND
```

- To remove SQLADMIN and SQLCOMMAND from USER2's StorHouse privileges, type:

```
SET ACCOUNT USER2 /PRIVILEGE=(NOSQLADMIN,  
NOSQLCOMMAND)
```

Listing StorHouse account IDs

You can list all StorHouse accounts that can access StorHouse databases by using the StorHouse Command Language SHOW ACCOUNT command.

▼ **To list all StorHouse accounts that can access StorHouse databases**

Required privilege: SHOW

1. Sign on to StorHouse.
2. At the command prompt (?), submit the following StorHouse Command Language SHOW ACCOUNT command, and then press **Enter**.

```
SHOW ACCOUNT * /PRIV=(SQLADMIN, SQLEXECUTE,  
SQLCOMMAND, ALLPRIV)
```

Listing access and command privileges

You can list access and command privileges for one or more StorHouse accounts by using the StorHouse Command Language SHOW ACCOUNT command.

▼ **To list access and command privileges for a StorHouse account**

Required privilege: SHOW

1. Sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW ACCOUNT command and press **Enter**. Some examples follow.

- To list access and command privileges for a specific account, such as USER1, type:

```
SHOW ACCOUNT USER1 /PRIVILEGE
```

- To list access and command privileges for all StorHouse accounts, type:

```
SHOW ACCOUNT * /PRIVILEGE
```

Listing the privileges granted to PUBLIC

StorHouse accounts with SQLEXECUTE privilege automatically get all the database component privileges granted to PUBLIC for each StorHouse database. You can query the SYSTABAUTH system table to determine the privileges granted to PUBLIC.

▼ To list the privileges granted to PUBLIC

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSTABAUTH

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSTABAUTH system table. Remember to precede the system table name with SYSADM (the system table owner name). Use uppercase when specifying the grantee PUBLIC. For example:

```
SELECT *  
FROM SYSADM.SYSTABAUTH  
WHERE GRANTEE = 'PUBLIC'
```

Listing database privileges

You can list database privileges—DBA, RESOURCE, or SCAN—for one or more StorHouse accounts by submitting a SELECT statement on the SYSDBAUTH system table.

▼ To list StorHouse database privileges for a StorHouse account

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSDBAUTH

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement to list database privileges. Some examples follow.

- To list all StorHouse database privileges for all accounts, type:

```
SELECT *  
FROM SYSADM.SYSDBAUTH
```

A y in a column indicates the account has the privilege. A g indicates the account has the privilege with the WITH GRANT OPTION.

- To list all StorHouse database privileges for grantee (account) USER1, type:

```
SELECT DBA_ACC, RES_ACC, SCN_ACC  
FROM SYSADM.SYSDBAUTH  
WHERE GRANTEE='USER1'
```

A y in a column indicates USER1 has the privilege. A g indicates USER1 has the privilege with the WITH GRANT OPTION.

- To list all StorHouse accounts with DBA privilege, type:

```
SELECT GRANTEE  
FROM SYSADM.SYSDBAUTH  
WHERE DBA_ACC='y'
```

Listing database component privileges

You can list database component privileges—ALL, DELETE, INDEX, INSERT, SELECT, and UPDATE—for one or more StorHouse accounts by submitting a SELECT statement on the SYSCOLAUTH and SYSTABAUTH system tables, respectively.

The following rules apply to database component privileges:

- The INDEX and SELECT table privileges apply to both user tables and system tables.
 - The DELETE and UPDATE privileges apply to system tables only.
 - The INSERT privilege applies to system tables only; however, the account used by a FileTek data loader to sign on to StorHouse requires INSERT privilege to load data into user tables.
- ▼ **To list StorHouse database component privileges for a StorHouse account**

Required privileges: SQLEXECUTE and either DBA or SELECT on both SYSTABAUTH and SYSCOLAUTH

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement to list account database component privileges. Some examples follow.

- To list all columns for which the grantee (account) USER1 has UPDATE privilege and to list the system table in which each column resides, type:

```
SELECT TBL, COL
FROM SYSADM.SYSCOLAUTH
WHERE GRANTEE='USER1'
AND UPD='y'
```

- To list all StorHouse table and view privileges for grantee (account) USER1 and to list the table or view on which each privilege was granted, type:

```
SELECT TBL, INS, DEL, UPD, SEL, NDX, ALT
FROM SYSADM.SYSTABAUTH
WHERE GRANTEE='USER1'
```

A y in a column indicates the account has the appropriate privilege. A g indicates the account has the privilege with the WITH GRANT OPTION.

Removing a StorHouse account

You can remove a StorHouse account by performing the following tasks:

- Task 1 – For each database, revoke all database component privileges granted to the account.
- Task 2 – For each database, revoke all database privileges granted to the account.
- Task 3 – Delete the StorHouse account.

Because you're removing the account, you can revoke all privileges on a database component with one REVOKE ALL statement and all database privileges with

one REVOKE statement. You must issue these statements for all databases that the account can access.

▼ **To remove a StorHouse account**

Required privileges: ACCOUNT, ANYACCOUNT, SQLEXECUTE, and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a REVOKE ALL statement for each database component. For example, if USER1 has database component privileges on the CUSTOMERS table and the SALARIES table, then to revoke all database component privileges on those tables, submit this statement:

```
REVOKE ALL  
ON CUSTOMERS  
FROM USER1
```

and then submit this statement:

```
REVOKE ALL  
ON SALARIES  
FROM USER1
```

3. Submit a REVOKE statement to revoke all database privileges from the account. For example, to revoke all database privileges from USER1, type:

```
REVOKE DBA, RESOURCE, SCAN  
FROM USER1
```

4. Sign on to StorHouse.

5. At the command prompt (?), submit the StorHouse Command Language REMOVE ACCOUNT command and press **Enter**. For example, to remove the account USER1, type:

```
REMOVE ACCOUNT USER1
```

If USER1 is signed on when the command is executed, the current session is not affected. However, after USER1's session is terminated, the account ID USER1 cannot be used again to access StorHouse.

4

Accounts and privileges

Removing a StorHouse account

User tablespaces

This chapter describes user tablespaces and explains how to:

- Create a user tablespace
- Assign and change a default user tablespace
- List accounts and their default user tablespaces
- Display default storage specifications
- Alter a user tablespace
- List user tablespace names and IDs
- Display storage specifications for a user tablespace
- Drop a user tablespace

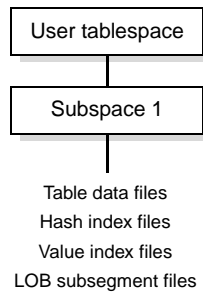
About user tablespaces

Before creating and loading user tables in a database, you create one or more user tablespaces and can designate a default user tablespace for the database. A *user tablespace* is a logical database component that defines where to store segment files on StorHouse and how to manage them. When you create a user table, you assign it to a user tablespace. You can assign LOB columns to the same user tablespace as the table or to different user tablespaces. And when you create indexes for that user table, you can assign them to the same user tablespace as the table or to different ones. The table files, index files, and LOB subsegment files are stored according to the specifications of their assigned user tablespace.

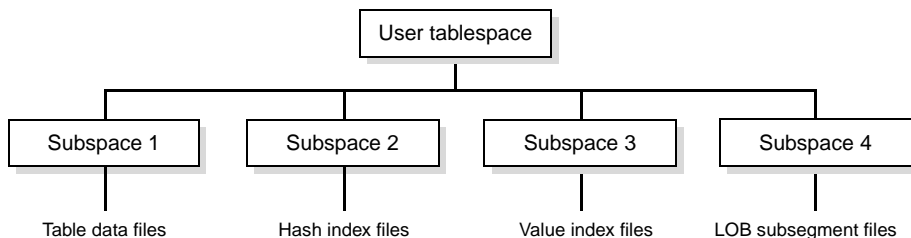
You create user tablespaces with the CREATE TABLE SPACE statement. See page 5-26 for a summary of the CREATE TABLE SPACE arguments.

Subspaces

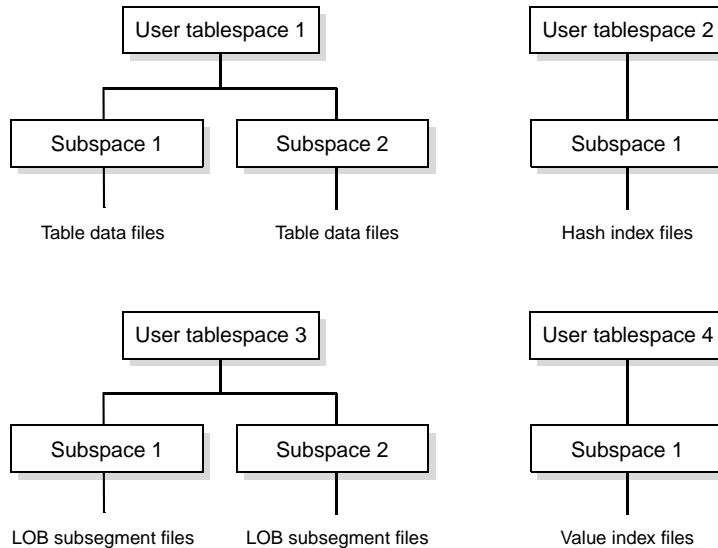
A user tablespace consists of one or more subspaces. A *subspace* contains a set of storage specifications for a specific type of user table component—table data, indexes, LOB data—or for all components. For instance, to manage the storage of all components in the same way, you can create a user tablespace with one subspace—or one set of storage specifications—for all components.



Or to manage the storage of different components in different ways, you can define multiple subspaces in a user tablespace. For instance, you can define one subspace for table data files, a second subspace for hash index files, a third subspace for value index files, and a fourth subspace for LOB subsegment files, defining different storage specifications for each subspace.



You can also assign all components to different user tablespaces. Each user tablespace can consist of one or more subspaces.



Whether you use multiple user tablespaces and/or multiple subspaces depends on your storage strategy. The following section, “Storage specifications,” defines the storage options that you set in a subspace. See “User tablespace examples” on page 5-12 for examples.

Note: When space permits or by user request, StorHouse/RM stores in-line LOBs with the table data, ignoring any LOB subspace or user tablespace. Subspaces for LOB subsegment files apply to out-of-line LOBs only. See “LOB storage and subsegment files” on page 2-10 for more information about in-line and out-of-line LOBs.

Storage specifications

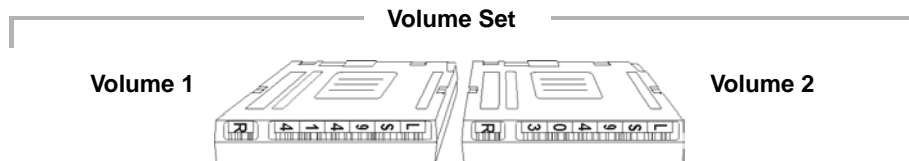
You define *storage specifications* for each subspace in a user tablespace. You can change those specifications when needed. Storage specifications are:

- Volume set
- File set
- Object type
- Access Time Factor
- Vulnerability Time Factor
- Error Detection Code
- File access group
- Maximum extent size
- Extent holding period

The volume set and file set are required for each subspace. You can use default values, if applicable, for the remaining storage specifications.

Volume set

A *volume set* (VSET) is one or more physical volumes that are treated as a logical unit of storage. A *volume* is a unit of media, such as an optical disk cartridge or a tape cartridge, on which data can be recorded and read. For example, the following volume set consists of two tape cartridges.

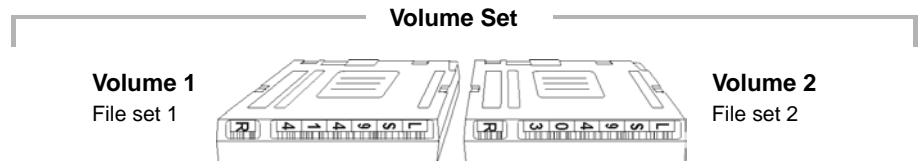


When you create a volume set, you specify a library device and the media type. You also specify volume set properties, such as the initial size to allocate and whether there's a limit, if any, to how large the volume set can grow. Refer to the *StorHouse Concepts and Facilities Manual* and the *StorHouse System*

Administrator's Guide for more information about volume sets and their properties.

File set

A *file set* (FSET) is a collection of files within a volume set. Typically, a volume set contains a single file set, but it can contain multiple file sets. For example, the following volume set consists of two file sets. You could, for instance, store table data files and LOB subsegment files in file set 1 and index files in file set 2.



Whether you use multiple volume sets and file sets for segment files depends on your data and your access and performance requirements. Refer to the *StorHouse System Administrator's Guide* for guidelines on using one or more volume sets and file sets.

Object type

The *object type* defines the type of component to be stored in a subspace. The object type values are:

Object type values in a subspace

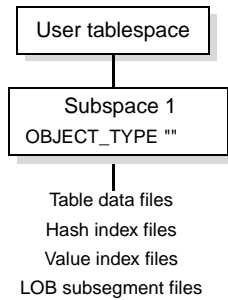
OBJECT_TYPE values	Description
"" or " "	All component types (default)
T	Table data only
H	Hash indexes only
V	Value indexes only
L	LOB data only

5

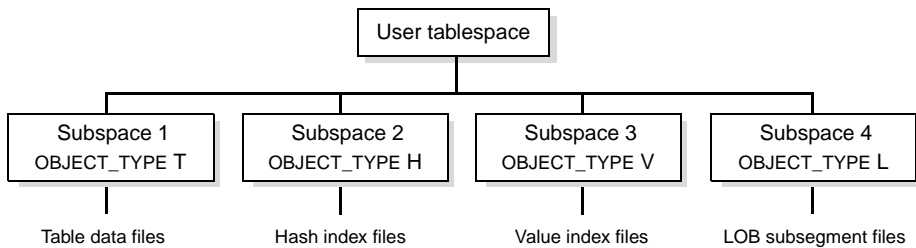
User tablespaces

About user tablespaces

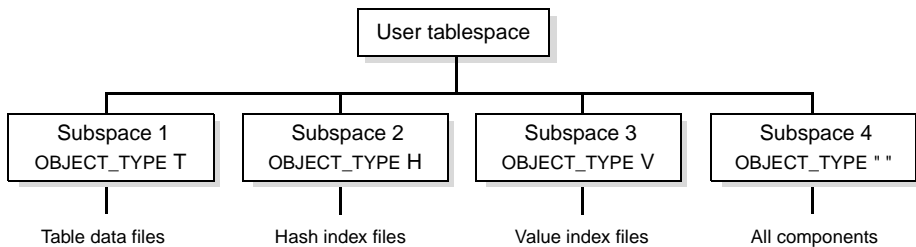
For instance, you can allow all components in a subspace.



Or you can allow only certain components in a subspace.



Or you can define a mix of subspaces that allows all components and specific components.



When a user tablespace contains multiple subspaces for a component type, a FileTek data loader automatically selects the lowest-numbered subspace for each component type. For instance, in the preceding example:

- Subspace 1 and subspace 4 allow table data files and in-line LOBs. The data loader automatically selects subspace 1 for table data and in-line LOBs.
- Subspace 2 and subspace 4 allow hash index files. The data loader automatically selects subspace 2 for hash indexes.
- Subspace 3 and subspace 4 allow value index files. The data loader automatically selects subspace 3 for value indexes.
- Subspace 4 is the only subspace that allows out-of-line LOBs. The data loader automatically selects subspace 4 for out-of-line LOBs.

During a data load, index load, or segment merge operation, however, you can select subspaces or request to rotate among subspaces if you do not want to use the lowest-numbered subspace for a component type. In this example, for instance, you can select subspace 4 for all components, or you can select subspace 4 for just table data and LOB data and continue to use the applicable lowest-numbered subspace for each index type. Refer to the *FileTek FTP Data Loader Manual* or the *FileTek MVS Data Loader Utility Manual* for more information about subspace selection and rotation during data load, index load, and merge operations.

Access Time Factor

The *Access Time Factor* (ATF) indicates the importance of access time for segment files. This parameter works with the StorHouse migrate function to keep data that is most likely to be accessed in the StorHouse performance buffer while

maintaining a supply of free space. You can assign one ATF value for each subspace. The ATF values are:

ATF values in a subspace

ATF values	Description
0	Use the value of the StorHouse ATF system parameter, which sets the ATF value to the system default.
1	Short access time is very important.
2	Short access time is moderately important.
3	Short access time is less important.

Refer to the *StorHouse Concepts and Facilities Manual* for more information about ATF and migration.

Vulnerability Time Factor

The *Vulnerability Time Factor* (VTF) determines when StorHouse writes files to their resident file sets as well as whether to create performance copies of files in the performance buffer. You can assign one VTF value for each subspace. The VTF values are:

VTF values in a subspace

VTF values	Description
DIRECT	Bypass the performance buffer and directly write the files to their file sets. (DF and map extents, however, are always written to the performance buffer as well as to their resident file set.)
NOW	Write extents to the performance buffer first and copy them to their file sets second. StorHouse/RM copies each extent to the resident file set after creating the extent on the performance buffer. So for a table file, StorHouse/RM creates the data extent on the performance buffer and then copies the data extent to its resident file set. Then StorHouse/RM creates the DF extent on the performance buffer and then copies it to the file set. Both of these operations (create and copy) take place during the load.

VTF values in a subspace (continued)

VTF values	Description
NEXT	Write extents to the performance buffer during a load and copy them to their file sets during the next StorHouse write-back operation.
DEFAULT	Use the value of the StorHouse VTF system parameter, which sets the VTF value to the system default.

Refer to the *StorHouse Concepts and Facilities Manual* for more information about VTF, the performance buffer, and write-back operations.

Error Detection Code

The *Error Detection Code* (EDC) parameter determines whether to use the StorHouse error detection feature for segment files. StorHouse generates EDCs during data loads and uses EDCs to detect errors during data movement in StorHouse. FileTek recommends that you always use EDC. You can assign one EDC value for each subspace. The EDC values are:

EDC values in a subspace

EDC values	Description
Y	Use EDC.
N	Do not use EDC.
D	Use the value of the StorHouse EDC system parameter, which sets the EDC value to the system default.

Refer to the *StorHouse System Administrator's Guide* for more information about EDC checking and all of the StorHouse system parameters that control EDC.

Note how the following EDC system parameters affect StorHouse/RM:

EDC system parameters

EDC system parameter	When set to TRUE
EDC_INTERNAL	StorHouse performs an EDC check during query activity.
EDC_COPY	StorHouse performs an EDC check during file backup, archive, and write-back operations. A write-back occurs during a data load when VTF is NOW. In this case, when a FileTek data loader creates an extent, StorHouse does a write-back and checks EDCs. The load completes after all extents have been created and written back successfully.

File access group

A *file access group* provides a way to group files logically. For instance, you can assign files to be accessed by the same application or users to the same file access group. You can assign one group to each subspace. This means that table data, related indexes, and LOB data may be stored in the same or different groups. The default group name for StorHouse/RM is STH. Refer to the *StorHouse Concepts and Facilities Manual* and the *StorHouse System Administrator's Guide* for more information about file access groups and how to create them.

Maximum data extent size

You can set the maximum data extent size (in megabytes) for segment files. The range is 1 MB to the maximum surface size for the VSET. If you don't set the

maximum data extent size in a subspace, StorHouse/RM uses the following defaults:

Defaults for maximum data extent sizes

Default	Segment file type
Value of SQL_MAX_EXT_DATA system parameter	Table data files
Value of SQL_MAX_EXT_HASH system parameter	Hash index files
Value of SQL_MAX_EXT_VAL system parameter	Value index files
100 MB	LOB subsegment files

See Appendix C, “System parameters for StorHouse/RM,” for more information about these system parameters and recommendations on how to set them.

Extent holding period in the performance buffer

StorHouse automatically migrates extents off the performance buffer based on factors such as extent size, ATF values, and access history. You can hold different types of extents in the performance buffer for a specified number of days (0 to 32,767 days) to enhance access performance. Note, however, that StorHouse may migrate extents sooner if space is needed and no other data is available for migration.

For each subspace, you can assign an extent holding period for data extents and a different extent holding period for DF and map extents (also called *special*

extents). If you don't set extent holding periods for a subspace, StorHouse/RM uses the following defaults:

Defaults for extent holding periods

File type	Extent type	Default
Table data files	Data	Value of the SQL_HOLD_DATA system parameter
Index files	Data	Value of the SQL_HOLD_INDX system parameter
	Map	Value of the SQL_HOLD_SPECIAL system parameter
LOB subsegment files	Data	0 days
All files	DF	Value of the SQL_HOLD_SPECIAL system parameter

See Appendix C, “System parameters for StorHouse/RM,” for more information about these system parameters and recommendations on how to set them.

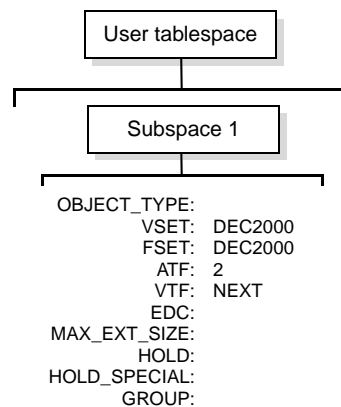
User tablespace examples

The number of user tablespaces you create or the number of subspaces for each user tablespace depends on your data and your access and performance requirements. Be sure to work with the StorHouse system administrator to define the best storage strategy for your data. This section contains examples of some of the ways you can define user tablespaces to meet different requirements.

Example 1: Minimal volume mounts

Assume you don't need to manage the storage of different data in different ways, and you want to minimize volume mounts.

Storage specifications. Create one user tablespace with one subspace for all component types and allocate one volume set and one file set.



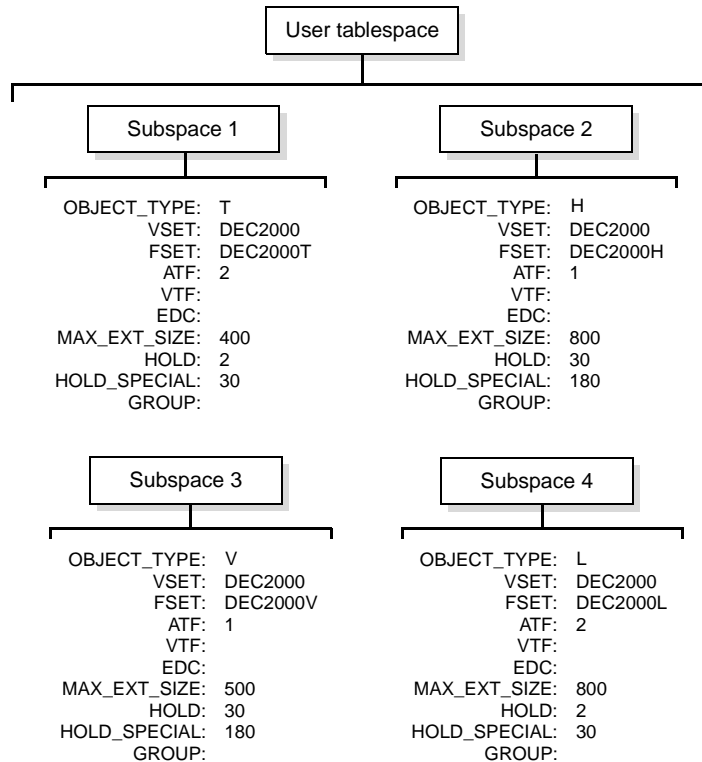
In this example, default values are used for all parameters except ATF and VTF.

Tablespace assignments. Assign the user table to the user tablespace and use the default user tablespace for indexes and LOB columns, that is, omit the TABLESPACE clause on the CREATE INDEX statements and LOB column definitions.

Example 2: Simultaneous removal

Assume you want to manage the storage of table data, indexes, and LOB data in different ways but remove them from StorHouse at the same time. Some reasons for simultaneous removal include the requirement to archive or back up table, index, and LOB data for off-site storage or to erase table, index, and LOB data with the same aging pattern from erasable volumes.

Storage specifications. Create one user tablespace with four subspaces, one subspace for each component type. Allocate four file sets in one volume set.

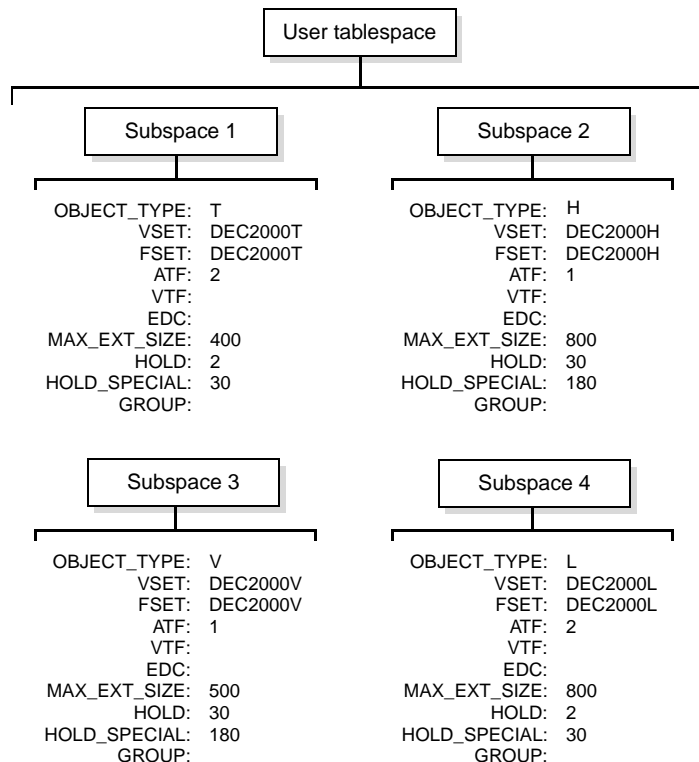


Tablespace assignments. Assign the user table, its indexes, and LOB columns to the same user tablespace.

Example 3: Load concurrency

Assume you want to manage the storage of table data, indexes, and LOB data in different ways but don't need to remove them from StorHouse at the same time. You also want to load table, index, and LOB data concurrently onto removable media.

Storage specifications. Create one user tablespace with four subspaces, one subspace for each component type. Allocate different volume sets and file sets for each subspace.



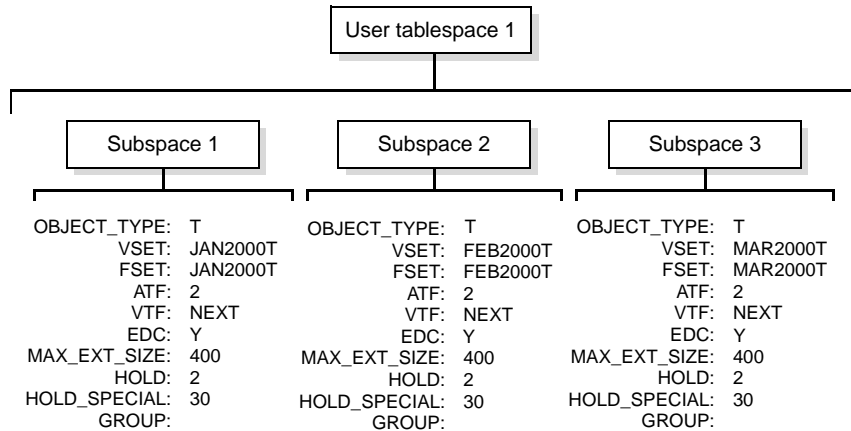
Tablespace assignments. Assign the user table, its indexes, and LOB columns to the same user tablespace.

Example 4: Access concurrency

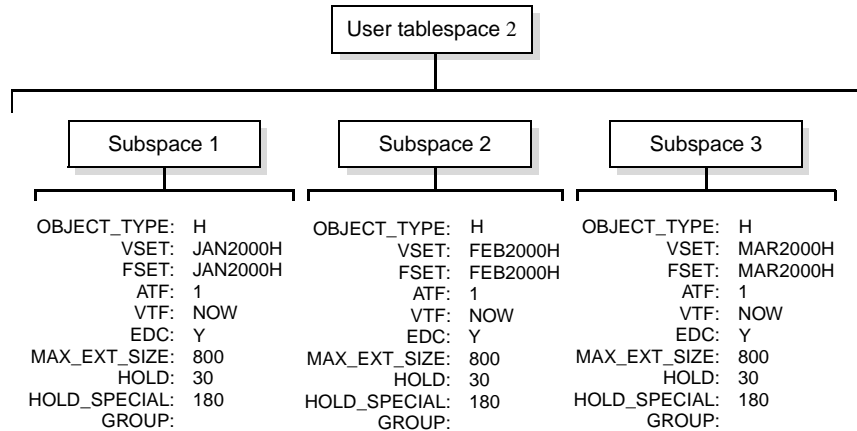
Assume you want to manage the storage of table data, indexes, and LOB data in different ways and don't need to remove them from StorHouse at the same time. You also plan to load data into the user table each month during a quarter and expect users to access multiple segments at a time.

Storage specifications. Create multiple user tablespaces, one for each component. (You can also do this with one user tablespace.) Define three subspaces in each user tablespace, one subspace for each month in a quarter. Allocate different volume sets and file sets for each subspace.

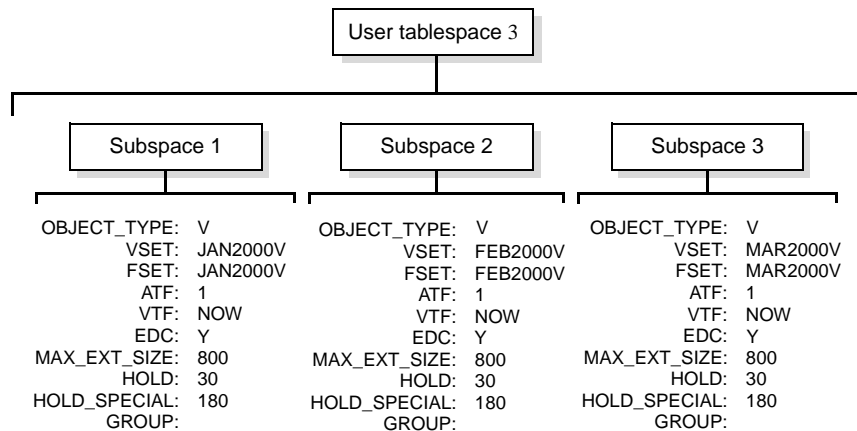
For instance, user tablespace 1 contains storage specifications for table data files only.



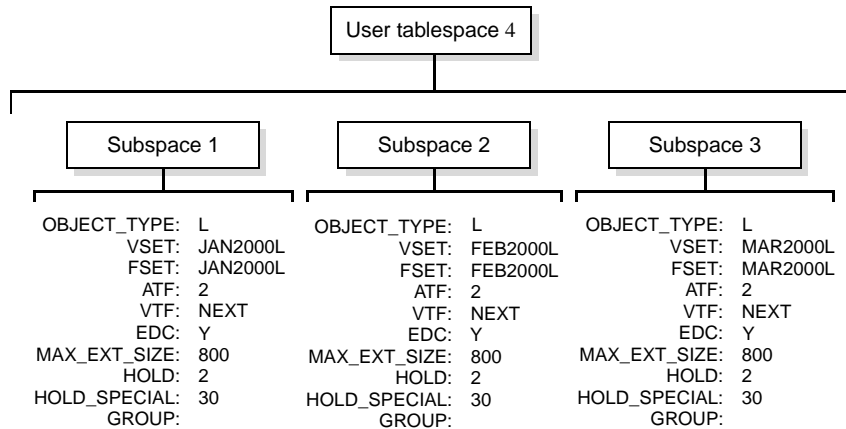
User tablespace 2 contains storage specifications for hash index files only.



User tablespace 3 contains storage specifications for value index files only.



User tablespace 4 contains storage specifications for LOB subsegment files only.



Note: To continue using the same user tablespaces and subspaces for the next quarter, you can create new volume sets and file sets and change the volume set and file set names in each subspace.

Tablespace assignments. Assign the user table to user tablespace 1, hash indexes to user tablespace 2, value indexes to user tablespace 3, and LOB columns to user tablespace 4.

Subspace selection during data load, index load, and segment merge operations

When loading table data, loading deferred indexes, and merging segments, a FileTek data loader determines the correct user tablespace(s) and automatically stores each segment file according to the appropriate subspace specifications. You can also explicitly select subspaces or rotate among eligible subspaces. Refer to the *FileTek FTP Data Loader Manual* or the *FileTek MVS Data Loader Utility Manual* for more information about subspace selection and rotation.

Default user tablespaces

A *default user tablespace* is the default that StorHouse/RM uses when you omit a TABLE SPACE clause on the CREATE TABLE and CREATE INDEX statements. You can assign a default user tablespace to a database and to each account. You make these assignments by inserting the account ID and user tablespace name into the SYSSMUSERS system table. To assign a default user tablespace to a database, use PUBLIC as the account ID.

Default user tablespace for a user table

When you create a user table and omit the TABLE SPACE clause, StorHouse/RM determines the default user tablespace as follows:

1. First, StorHouse/RM attempts to assign the user table to the account default user tablespace.
2. If there is no account default user tablespace, StorHouse/RM attempts to assign the user table to the database default user tablespace.
3. If there is no database default user tablespace, an error occurs.

Note: If you are creating a user table for another account, that is, you are the creator and another account is the owner, then StorHouse/RM uses *your* account default user tablespace if you omit the TABLE SPACE clause.

Default user tablespace for a LOB column

When you define a column definition for a LOB column and omit the optional TABLE SPACE clause, StorHouse/RM assigns the LOB column to the same user tablespace as the user table. In the following example, StorHouse/RM assigns the CLOB column named COLUMN2 to the default user tablespace and assigns the BLOB column named COLUMN3 to the BLOBSpace user tablespace.

```
CREATE TABLE MYTABLE  
(COLUMN1 INTEGER,  
COLUMN2 CLOB,  
COLUMN3 BLOB TABLESPACE BLOBSpace)  
TABLESPACE MYTABLESPACE
```

Default user tablespace for an index

When you create an index and omit the TABLE SPACE clause, StorHouse/RM assigns the index to the same user tablespace as the user table. In the following example, StorHouse/RM assigns the range index named PAY_DATE to the same user tablespace as the CUSTOMERS table.

```
CREATE RANGE INDEX PAY_DATE  
ON CUSTOMERS (PDATE)
```

User tablespace naming conventions

Each user tablespace has a user tablespace name and a user tablespace ID. Each subspace has a subspace number.

User tablespace names

When you create a user tablespace, you give it a name. A user tablespace name must be unique in a database and should follow SQL identifier conventions. You must delimit user tablespace names that do not follow these conventions. Note that if you delimit a user tablespace name and that name contains lowercase characters (for instance, CREATE TABLE SPACE “May2000”), you must type the name in the same case (May2000, not MAY2000 or may2000) when inserting, updating, or deleting rows in SYSSMUSERS. If you do not delimit a user tablespace name, you must type it in uppercase characters on INSERT, UPDATE, or DELETE statements to SYSSMUSERS.

Tablespace IDs

StorHouse/RM assigns a *tablespace identifier* (ID) to each tablespace. This ID is unique within a database. Tablespace ID 0 identifies the system tablespace and tablespace ID 1 identifies the temporary tablespace. User tablespace IDs start with 2 and increment by 1 for each new user tablespace.

Subspace numbers

When you define a subspace, you assign a *subspace number*, which is a value from 0 to 2,147,483,647. You use subspace numbers when altering user tablespaces and when explicitly selecting subspaces during a load. Subspace numbers must be unique for each subspace in a user tablespace.

System tables with user tablespace information

The following system tables contain information about user tablespaces:

- SYSTBLSPACES – Contains the tablespace name and tablespace ID. StorHouse/RM inserts or deletes a row in this system table when you create or drop a user tablespace.
- SYSSTHSPACES – Contains storage specifications for each subspace in a user tablespace. StorHouse/RM inserts or updates one or more rows in this system table when you create or alter a user tablespace.
- SYSSMUSERS – Contains default user tablespace assignments for a database and accounts. You can maintain this system table, that is, insert, update, and delete rows to manage account and database default user tablespaces.

What's different about StorHouse user tablespaces?

The concept of tablespace differs by relational database management system, and how you work with tablespaces may also differ. For instance, in DB2 a tablespace is a physical VSAM dataset that contains table data. In StorHouse, a user tablespace is a logical component that defines storage specifications for table data, indexes, and LOB data.

Keep in mind the following when working with user tablespaces:

- A StorHouse database has one or more user tablespaces.
- Each user table, index, and LOB column belongs to a user tablespace.
- Indexes and LOB columns can belong to the same user tablespace as the table or to a different user tablespace(s).
- A user tablespace consists of one or more subspaces.
- A subspace can define storage specifications for all component types (table data, value index, hash index, or LOB data) or just one component type.
- You can alter a user tablespace to store subsequent segment files differently.
- You can assign a default user tablespace to each database and account.

Setting up a user tablespace

Setting up a new StorHouse user tablespace is a two-task process:

- Task 1 – Create volume sets and file sets for segment files with the StorHouse Command Language CREATE VSET and CREATE FSET commands.

- Task 2 – Create a user tablespace with the StorHouse SQL CREATE TABLESPACE statement.

Creating volume sets and file sets

You can create volume sets and file sets for segment files with the StorHouse Command Language CREATE VSET and CREATE FSET commands. When you create volume sets and file sets, you name them. The following rules apply when choosing these names:

- Volume set and file set names can contain from 1 to 8 characters and consist of a–z, A–Z, 0–9, _ (underscore), and \$ (dollar sign).
- A volume set name must be unique within StorHouse.
- A file set name must be unique within a volume set.
- Volume set and file set names are case insensitive. You cannot use case to distinguish volume set and file set names. For example, the volume set name MYVSET is the same as the volume set name myvset or MYvset.

If possible, choose a volume set or file set name that follows SQL identifier naming conventions. Use double quotes (") to delimit volume set and file set names that start with a number or underscore (_), contain a \$, or are an SQL reserved word. If you delimit a volume set or file set name, StorHouse/RM stores it in the SYSSTHSPACES system table in the same case as entered (lowercase, mixed case, uppercase); however, StorHouse always treats volume set and file set names as uppercase.

Refer to the StorHouse *Command Language Reference Manual* and the StorHouse *System Administrator's Guide* for more information about the CREATE VSET and CREATE FSET commands.

▼ To create a volume set

Required privilege: ALLOCATION

1. Sign on to StorHouse.
2. At the command prompt (?), type the following and press **Enter**:

```
CREATE VSET vset_name modifiers
```

where `vset_name` is the name of the new volume set and `modifiers` are the desired command and parameter modifiers (to assign values other than the defaults). Some examples follow.

- To create a volume set called JAN2000 using all of the command and parameter defaults, type:

```
CREATE VSET JAN2000
```

- To create a volume set called FEB2000H using all of the command and parameter defaults except `/LIBRARY` (to create the volume set on a library device other than the default), type:

```
CREATE VSET FEB2000H /LIBRARY=L02
```

3. Record the volume set name because you need it when you create the user tablespace.
4. If needed, repeat this procedure to create a second volume set for the user tablespace.

▼ To create a file set

Required privilege: ALLOCATION

1. If necessary, sign on to StorHouse.

2. At the command prompt(?), type the following and press **Enter**:

```
CREATE FSET fset_name modifiers
```

where `fset_name` is the name of the new file set and `modifiers` are the desired command and parameter modifiers (to assign values other than the defaults). Below are some examples.

- To create a file set called `MY_TABLE` for the volume set called `JAN2000`, using all of the command and parameter defaults, type:

```
CREATE FSET MY_TABLE /VSET=JAN2000
```

- To create a file set called `FEB2000H` for the volume set called `FEB2000H`, using all of the command and parameter defaults except `/SIZE` (to preallocate 500 megabytes of storage for the file set), type:

```
CREATE FSET FEB2000H /VSET=FEB2000H /SIZE=500M
```

3. Record the file set name because you need it when you create the user tablespace.
4. If needed, repeat this procedure to create a second file set for the user tablespace.

Creating a user tablespace

You can create a user tablespace with the `CREATE TABLE SPACE` statement. (TABLE SPACE can be one or two words.)

Before you begin:

- Obtain the names of the volume set(s) and file set(s) that you or the StorHouse system administrator created for this user tablespace. Each subspace requires a volume set name and a file set name.

Note: Remember to delimit volume set and file set names that do not follow SQL identifier conventions.

- Decide whether to use the default storage specifications. See “Displaying default storage specifications” on page 5-30 for information about displaying the current system default values.
- Choose a name for the user tablespace. See “User tablespace naming conventions” on page 5-20 for naming rules.

The following table summarizes the optional arguments on the CREATE TABLE SPACE statement. Refer to Chapter 4 in the *StorHouse SQL Reference Manual* for information about the CREATE TABLE SPACE statement.

Optional arguments on CREATE TABLE SPACE

Argument	Default	Values
OBJECT_TYPE type_value	one blank " "	" ", "", T, H, V, L
ATF atf_value	0	0, 1, 2, 3
VTF vtf_value	DEFAULT	DEFAULT, NOW, NEXT, DIRECT
EDC edc_value	D	D, Y, N
GROUP group_name	STH	Valid StorHouse group
MAX_EXT_SIZE size_in_megabytes	0	1 to maximum surface size for the VSET
HOLD number_of_days	0	0 to 32767
HOLD_SPECIAL number_of_days	0	0 to 32767

▼ To create a user tablespace

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE TABLE SPACE statement for the user tablespace. Some examples follow.
 - To create a user tablespace called BILLINGTABLE with one subspace that uses all default values, type:

```
CREATE TABLE SPACE BILLINGTABLE
(SUBSPACE 1 VSET JAN2000T FSET JAN2000T)
```

- To create a user tablespace called BILLINGJAN with four subspaces, one for each component, type:

```
CREATE TABLE SPACE BILLINGJAN
(SUBSPACE 1 VSET JAN2000T FSET JAN2000T OBJECT_TYPE T
ATF 2 VTF NOW MAX_EXT_SIZE 400 HOLD 30
HOLD_SPECIAL 180,
SUBSPACE 2 VSET JAN2000H FSET JAN2000H OBJECT_TYPE H
ATF 1 VTF NEXT MAX_EXT_SIZE 800 HOLD 90
HOLD_SPECIAL 365,
SUBSPACE 3 VSET JAN2000V FSET JAN2000V OBJECT_TYPE V
ATF 1 VTF NEXT MAX_EXT_SIZE 500 HOLD 90
HOLD_SPECIAL 365,
SUBSPACE 4 VSET JAN2000L FSET JAN2000L OBJECT_TYPE L
ATF 2 VTF NOW MAX_EXT_SIZE 800 HOLD 30
HOLD_SPECIAL 180)
```

Assigning a default user tablespace

You can assign a default user tablespace to an account by inserting the account ID and user tablespace name into the SYSSMUSERS system table. You can assign a default user tablespace to a database by using PUBLIC as the account ID. StorHouse/RM uses a database default user tablespace when an account without an account default user tablespace omits the TABLE SPACE clause on the CREATE TABLE statement.

Caution: On the INSERT statement, type the account ID in uppercase. Also, if you delimited the tablespace name when you created the user tablespace, and the name contains lowercase letters (for instance, CREATE TABLE SPACE “Mar2000”), type the name in the same case (Mar2000, not MAR2000 or mar2000). If you didn’t delimit the tablespace name, type it in uppercase letters.

▼ To assign a default user tablespace

Required privileges: SQLEXECUTE and either DBA or INSERT on SYSSMUSERS

1. Follow your site’s procedure for connecting to your StorHouse database.
2. Submit an INSERT statement to insert the account ID and tablespace name into the SYSSMUSERS system table. Some examples follow.

- To assign an account default user tablespace called mar2000 to USER1, type:

```
INSERT INTO SYSADM.SYSSMUSERS (ACCOUNTID,  
DEFAULT_TS) VALUES ('USER1','mar2000')
```

- To assign a database default user tablespace called DEFSPACE, type:

```
INSERT INTO SYSADM.SYSSMUSERS (ACCOUNTID,  
DEFAULT_TS) VALUES ('PUBLIC','DEFSPACE')
```

Listing accounts and their default user tablespaces

You can list the account IDs and tablespace names of all accounts with default user tablespaces by submitting a SELECT statement on the SYSSMUSERS system table. The account ID PUBLIC identifies the database default user tablespace.

▼ To list all accounts and their default user tablespaces

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSSMUSERS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit the following SELECT statement on the SYSSMUSERS system table:

```
SELECT *  
FROM SYSADM.SYSSMUSERS
```

Changing a default user tablespace

You can change the default user tablespace for a specific account or for a database by submitting an UPDATE statement on the SYSSMUSERS system table.

Caution: On the UPDATE statement, type the account ID in uppercase. Also, if you delimited the tablespace name when you created the user tablespace, and the name contains lowercase letters (for instance, CREATE TABLE SPACE "Mar2000"), be sure to type the name in the same case (Mar2000, not MAR2000 or mar2000). If you didn't delimit the tablespace name, type it in uppercase letters.

▼ To change a default user tablespace

Required privileges: SQLEXECUTE and either DBA or UPDATE on SYSSMUSERS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit an UPDATE statement to change the default user tablespace. Some examples follow.
 - To change the account default user tablespace to apr2000 for USER1, type:

```
UPDATE SYSADM.SYSSMUSERS
SET DEFAULT_TS='apr2000'
WHERE ACCOUNTID='USER1'
```

- To change the database default user tablespace to DEFSPACE2, type:

```
UPDATE SYSADM.SYSSMUSERS
SET DEFAULT_TS='DEFSPACE2'
WHERE ACCOUNTID='PUBLIC'
```

Displaying default storage specifications

You can display default storage specifications for all but OBJECT_TYPE and GROUP by submitting a StorHouse Command Language SHOW SYSTEM command. If you omit a parameter on the CREATE TABLE SPACE statement, StorHouse/RM uses the default. Note that the default value for OBJECT_TYPE is (for all component types) and the default value for GROUP is STH.

▼ To display default storage specifications

Required privilege: SHOW

1. Sign on to StorHouse.
2. At the command prompt (?), type the applicable SHOW SYSTEM command and press **Enter**.

- To display the default ATF value, type:

SHOW SYSTEM ATF

- To display the default VTF value, type:

SHOW SYSTEM VTF

- To display the default EDC value, type:

SHOW SYSTEM EDC

- To display the default MAX_EXT_SIZE value for a specific component, type one of the following:

For table data files – SHOW SYSTEM SQL_MAX_EXT_DATA

For hash index files – SHOW SYSTEM SQL_MAX_EXT_HASH

For value index files – SHOW SYSTEM SQL_MAX_EXT_VAL

For LOB subsegment files – The default MAX_EXT_SIZE value is 100 MB.

- To display the default HOLD value for a specific component, type one of the following:

For table data files – SHOW SYSTEM SQL_HOLD_DATA

For index files – SHOW SYSTEM SQL_HOLD_INDX

For LOB subsegment files – The default HOLD value is 0 days.

- To display the default HOLD_SPECIAL value, type:

SHOW SYSTEM SQL_HOLD_SPECIAL

Altering a user tablespace

You can change any of the storage specifications in a user tablespace with the ALTER TABLE SPACE statement. (TABLE SPACE can be one or two words.) Note the following:

- You cannot add or delete subspaces.
- When you change storage specifications, any new segment files are stored according to the new specifications. Existing segment files are stored according to the old specifications.

▼ To alter a user tablespace

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit an ALTER TABLE SPACE statement for the user tablespace. Some examples follow.
 - To change the volume set and file set names assigned to SUBSPACE 1 in the BILLINGTABLE user tablespace, type:

```
ALTER TABLE SPACE BILLINGTABLE
(SUBSPACE 1 VSET FEB2000T FSET FEB2000T)
```

- To change the ATF value for SUBSPACE 1, the MAX_EXT_SIZE for SUBSPACE 2, and the number of days to hold special data extents in the performance buffer for SUBSPACE 3 in the BILLINGJAN user tablespace, type:

```
ALTER TABLE SPACE BILLINGJAN
( SUBSPACE 1 ATF 1 ,
  SUBSPACE 2 MAX_EXT_SIZE 400,
  SUBSPACE 3 HOLD_SPECIAL 65535 )
```

- To change the OBJECT_TYPE value for SUBSPACE 1 to allow all component types (no restriction), type:

```
ALTER TABLE SPACE BILLING2000  
(SUBSPACE 1 OBJECT_TYPE " ")
```

Note that there must be a space between the quotes (after OBJECT_TYPE).

Listing user tablespace names and IDs

You can list a user tablespace ID, name, or both by submitting a SELECT statement on the SYSTBLSPACES system table. This procedure is useful if you are choosing a user tablespace name and need to determine whether it is unique.

▼ To list a user tablespace ID, or name, or both

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSTBLSPACES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSTBLSPACES system table. Some examples follow.

- To list all user tablespace IDs and names in a database, type:

```
SELECT *  
FROM SYSADM.SYSTBLSPACES
```

- To list all tablespace names in a database, type:

```
SELECT TSNAME  
FROM SYSADM.SYSTBLSPACES
```

- To list all tablespace IDs in a database, type:

```
SELECT ID
FROM SYSADM.SYSTBLSPACES
```

- To list a tablespace ID for a specific user tablespace (such as BILLING2000), type:

```
SELECT ID
FROM SYSADM.SYSTBLSPACES
WHERE TSNAME='BILLING2000'
```

Displaying storage specifications for a user tablespace

You can display the storage specifications defined for each subspace in a user tablespace by submitting a SELECT statement on the SYSSTHSPACES system table. You need the tablespace ID to display the storage specifications for a specific user tablespace.

▼ To display storage specifications

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSSTHSPACES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSSTHSPACES system table. Some examples follow:
 - To display all information about all user tablespaces in a database, type:

```
SELECT *
FROM SYSADM.SYSSTHSPACES
```

- To display the volume sets and file sets for all subspaces in a specific user tablespace with tablespace ID 3, type:

```
SELECT TSSUBSPACE_NUM,TSVSET,TSFSET  
FROM SYSADM.SYSSTHSPACES  
WHERE TSID='3'
```

- To display the ATF and VTF values for all subspaces in a specific user tablespace with tablespace ID 3, type:

```
SELECT TSSUBSPACE_NUM,TSATF,TSVTF  
FROM SYSADM.SYSSTHSPACES  
WHERE TSID='3'
```

Dropping a user tablespace

You can remove a user tablespace from the current database with the DROP TABLE SPACE statement. (TABLE SPACE can be one or two words.) Dropping a user tablespace removes all subspaces in that user tablespace.

Before you drop a user tablespace, you must drop all the user tables in that tablespace. When you drop the user tables, StorHouse/RM automatically drops the indexes associated with the user tables, even if the indexes are in a different user tablespace.

▼ To drop a user tablespace

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.

5

User tablespaces

Dropping a user tablespace

2. Submit a DROP TABLE SPACE statement. The following example drops a user tablespace called BILLING2000:

```
DROP TABLE SPACE BILLING2000
```

User tables

This chapter describes user tables and explains how to:

- Design a column of a user table
- Create a user table
- Display information about user tables
- List table names and table IDs
- Display a column definition for a user table
- Display volume set, file set, and segment information
- Rename a user table
- Drop a user table
- Undrop a user table
- Purge a user table

About StorHouse user tables

A *user table* is a database component that holds user-accessible data. Logically, user tables consist of columns, rows, and data values. For instance, the following user table contains columns for customer account numbers, last names, and

cities. Each row contains the account number, last name, and city for a specific customer.

Columns represent specific data types or domains.

	ACCOUNT	LAST_NAME	CITY
<i>Rows represent specific database records or tuples.</i>	91256	ANDERSEN	PARIS
	91347	WHITE	MADRID
	91486	MCGUIRE	CHARLOTTESVILLE
	97865	CORNFELD	SARASOTA
	99003	BROWN	COLOGNE

The intersection of a row and a column contains a data value.

Physically, user table data is stored in table data files in segments. In-line LOBs are stored in the table data file, and out-of-line LOBs are stored in LOB subsegment files. See “Segmentation” on page 2-7 for more information about how table data and LOB data are stored.

Process of creating a user table

Before you can load data into a user table, you must define—or create—it. When you create a user table, you:

- Name the table. See “User table naming conventions” on page 6-3 for information about user table names.
- Define the columns of the table. See “Designing a column of a user table” on page 6-5 for more information about column definitions.
- Assign the user table to a user tablespace. See Chapter 5, “User tablespaces,” for more information about user tablespaces. If you don’t assign the user table to a user tablespace, StorHouse/RM assigns the table to a default user tablespace.

Owners and creators of user tables

Each user table has an owner. A *user table owner* has ALL database component privileges for the user table. These privileges include SELECT, INDEX, and INSERT (for loading only). In other words, the owner of a user table can retrieve data from the table, create indexes for the table, and load data into the table. In addition, a user table owner can grant privileges for that user table to other accounts.

Note: To load data into a user table, the owner also needs access and command privileges, as described on page 4-8.

Typically, the owner is the person who creates the user table, but accounts with DBA privilege can create user tables on behalf of other accounts. In this case, the person with DBA privilege is the *user table creator*, and the account for which they create the table is the user table owner. So, if account A has DBA privilege and creates a table for account B, account A is the creator and account B is the owner.

User table naming conventions

Each user table has a table name and a table ID.

User table names

When you create a user table, you give it a name. A table name should follow SQL identifier conventions. If a table name does not follow these conventions, you must delimit it and use the appropriate case in SQL statements.

A *fully qualified table name* is the combination of owner and table name. A table, view, and synonym name must be unique for an owner. For example, CALLSDBA.ACCOUNTS and SYSADM.ACCOUNTS are valid user table names in a database. A table called CALLSDBA.ACCOUNTS and a view called CALLSDBA.ACCOUNTS are not valid names because the combination is not unique.

Table IDs

StorHouse/RM assigns table IDs to user tables when you create them (and to system tables when you create a StorHouse database). Table IDs are unique within a database. The first user table ID in a database is 100. The first system table ID is 0.

System tables with user table information

Various system tables contain information about user tables.

- **SYSTABLES** – Contains one row for each user table (and system table and view) in a database. StorHouse/RM inserts a row into this system table when you create a user table or view and deletes a row when you drop a user table or view.
- **SYSCOLUMNS** – Contains one row for each column of every table in a database. For user tables, StorHouse/RM inserts a row into this system table for each column definition specified on the CREATE TABLE statement.
- **SYSSTHFILES** – Contains the StorHouse file names and group names of segment files. StorHouse/RM updates this system table after a confirmed data load operation, an index load operation, a segment replace and a merge operation, or when you drop a user table or index.
- **SYSSTHSEGMENTS** – Contains information about segments. StorHouse/RM updates this system table after a confirmed data load, a segment replace and a merge operation, or when you drop a user table.
- **SYSDROP_PEND** – Contains information about dropped user tables. StorHouse/RM uses the information should you undrop a user table.

What's different about StorHouse user tables?

Working with a user table in StorHouse may be different from working with one in your local database. In StorHouse:

- You cannot define primary or foreign keys for a user table because StorHouse does not support referential constraints.
- You cannot use an INSERT statement to add a row to a user table. Instead, you load rows into a user table with the FileTek MVS Data Loader utility or the FileTek FTP Data Loader.
- StorHouse does not support the UPDATE and DELETE statements on user tables. You cannot use an UPDATE statement to change a data value in a user table or a DELETE statement to remove a row from a user table.
- StorHouse does not support the ALTER TABLE statement on user tables. Once you create a user table, you cannot add a column, delete a column, or modify a data type in that table.

Designing a column of a user table

When creating a user table, you define the characteristics of each column. When designing a column, you:

- Decide the order of the columns
- Choose a name for the column
- Determine the data type of the column
- Decide whether the column can contain null values
- Specify a default value for the column, if desired
- Specify any LOB storage options

Deciding the order of columns

The order that you define columns affects the sequence in which columns appear in a result set when you issue `SELECT *` statement. For instance, if you define the following columns in this order—`BILL_ACCOUNT`, `BILL_DATE`, `BILL_CATEGORY`—then they appear in that order in the result set.

If a user table contains multiple LOB columns, StorHouse/RM determines in-line storage in column order. In other words, StorHouse/RM attempts to store the first LOB column value with the table data, and if space permits, attempts to store the second LOB column value with the table data, and so on. Therefore, the order you define LOB columns may affect which column values are stored with the table data. See page 6-12 for more information about LOB storage options.

Otherwise, column order does not affect performance. You can define columns that contain null values first or last, or you can define columns that contain variable-length data types first or last. Internally, StorHouse/RM stores column definitions to maximize performance.

Choosing a column name

Column names must be unique within a user table, but they do not have to be unique within a database. This means you can use the same column names in multiple user tables. For example, a `BILLSUMMARY` table can contain a column called `BILL_ACCOUNT` and a `BILLDETAIL` table can also contain a column called `BILL_ACCOUNT`. A column name should also follow SQL identifier conventions. You must delimit column names that do not follow these conventions.

Determining the data type

The *data type* of a column determines the maximum length of data values in the column and the kind of data that is valid for the column. By specifying a data

type you are enforcing an *integrity constraint* on a column. Prohibiting null values in a column is another integrity constraint (see page 6-10). When you load data into a user table or use an INSERT or UPDATE statement on a system table, the value that you provide for a column must be consistent with the length and data type of that column. StorHouse/RM truncates data strings longer than the allowable maximum for the data type.

StorHouse supports the following data types for defining columns. These are called *database data types*. For more information about database data types, refer to the *StorHouse SQL Reference Manual*. Note the following:

- The lengths are optional. StorHouse/RM uses a default length when you omit it.
- StorHouse/RM automatically compresses any VARCHAR and VARBINARY column with a length greater than 4096.

Data types for defining columns

Data type	Description	Fixed/ variable	Stored length
BIGINT	Signed integer from -2 ⁶³ (-9223372036854775808) through +2 ⁶³ -1 (9223372036854775807), inclusive	Fixed	8 bytes
BINARY(length)	Array of bytes (8 bits each) with a length from 1 to 256 bytes	Fixed	Value of length

Data types for defining columns (continued)

Data type	Description	Fixed/ variable	Stored length
BLOB(length [K M G]) BINARY LARGE OBJECT(length [K M G])	Array of bytes with a length from <ul style="list-style-type: none"> ■ 1 to 2147483638 bytes ■ 1 to 2097152 K ■ 1 to 2048 M ■ 1 or 2 G 	Variable	In-line: Size of BLOB plus 4 bytes Out-of-line: Size of BLOB in the LOB subsegment file and 22 bytes for the OID in the table data file
CHAR(length)	Array of characters with a length from 1 to 256 bytes	Fixed	Value of length
CLOB(length [K M G]) CHARACTER LARGE OBJECT(length [K M G])	Array of characters measured in bytes with a length from: <ul style="list-style-type: none"> ■ 1 to 2147483638 bytes ■ 1 to 2097152 K ■ 1 to 2048 M ■ 1 to 2 G 	Variable	In-line: Size of CLOB plus 4 bytes Out-of-line: Size of CLOB in the LOB subsegment file plus 22 bytes for the OID in the table data file
DATE	Date value representing a month, day, and year	Fixed	4 bytes
DOUBLE PRECISION	Double precision, floating- point number	Fixed	8 bytes
INTEGER	Signed integer from -2147483648 through 2147483647, inclusive	Fixed	4 bytes

Data types for defining columns (continued)

Data type	Description	Fixed/ variable	Stored length
NUMERIC(P[,S]) DECIMAL(P[,S])	Decimal fixed-point number with a precision (P) and a scale (S) up to 31 digits	Variable	(P/2)+1
REAL	Single precision, floating-point number	Fixed	4 bytes
SMALLINT	Signed small integer with values from -32768 through 32767, inclusive	Fixed	2 bytes
TIME	Time value representing an hour, minutes, seconds, and optionally milliseconds	Fixed	4 bytes
TIMESTAMP	Date and time combination representing a month, day, year, hour, minutes, seconds, and optionally milliseconds and microseconds	Fixed	8 bytes
VARBINARY(length)	Variable-length array of bytes (8 bits each) with a length from 1 to 32705 bytes (maximum length of 256 for indexed fields)	Variable	Actual size of data + 2 bytes
VARCHAR(length)	Variable-length array of characters with a length from 1 to 32705 bytes (maximum length of 256 for indexed fields)	Variable	Actual size of data + 2 bytes

Using null values in a column

StorHouse/RM has a value indicator, called a *null value*, to stand for an unknown, not applicable, or missing value. If a column allows null values, then you can load null values into that column.

For example, assume a column in a customer table contains middle initials. Not everyone has a middle initial, so you must be able to load data into the user table without requiring a middle initial for a customer record. In this case, you would allow null values in this column.

Caution: You cannot use the INSERT, UPDATE, or ALTER TABLE statements with user tables. If you define a column as null and load data without data values, then those data values will *always* be null. You cannot provide new or different values later.

It's important to note that a null value is not a data value in a column; rather, it's a null indicator. Each row contains a null bit for each column that allows nulls. This bit indicates whether null values are allowed. When you define a column, you can choose to allow or disallow null values. The default is to allow null values. You can disallow null values in two ways:

- Define a column as *not null*, which means you must provide a value for the column during a load operation.
- Define a default definition, which means that the column cannot be null and will have a default value if none is provided. See the next section, “Specifying defaults for a column,” for more information about column defaults.

Defining a column as not null imposes an integrity constraint on the column, because not null means that you must *always* provide a data value for the column. StorHouse/RM rejects the request if you try to load, insert, or update data without data values for a column defined as not null and without a default definition.

Specifying defaults for a column

You can optionally specify a default value—called a *default definition*—for a column. A FileTek data loader inserts the default value into the column when there’s no data value in the input data for that column. The maximum length of a default definition is 247 bytes.

Caution: You cannot use the INSERT, UPDATE, or ALTER TABLE statements with user tables. If you define a column with a default value and do not provide a specific data value, that data value will *always* be the default value. You cannot provide new or different data values later.

You can choose from a variety of default values, but some of your choices depend on the data type of the column, as described in the following table.

Column default values

Value	Data type	Description
Literal	Any	An integer, numeric string, or character string for columns of like data type. For example, for a column called STATE defined as CHAR(2), you could specify a default definition of VA. Data values in this column will contain the default value VA if you don't provide another value.
NULL	Any	A null value. This is the default if you omit a default definition.
SYSDATE	DATE	The current date.
SYSTIME	TIME	The current time.
USER	CHAR or VARCHAR	The account ID of the account performing the load. The column must have a minimum length of 12, for example, CHAR(12).

The default for a column is to allow null values. In other words, if you do not specify a default definition for a column, the column can be null.

Specifying LOB storage options

By default, a FileTek data loader attempts to store LOB data with the table data (in-line) in the same user tablespace as the table. For BLOB and CLOB column definitions, you can specify the following optional clauses:

LOB column definition options

To	Use this clause
Set the maximum length for in-line LOB data	INLINE [(length[K])]
Always store LOB data in a different file from the table data	NOT INLINE
Share storage with another LOB column	STORE WITH column_name
Use a different user tablespace from the table	TABLESPACE tablespace_name

To set the maximum length for in-line LOB data

You can set the maximum length for in-line LOB data by including the **INLINE** clause with a length (in K), for instance:

```
CREATE TABLE LOBTABLE  
( CLOB_COLUMN CLOB INLINE(5K) )  
TABLESPACE LOBSpace
```

A FileTek data loader stores the LOB value in the row as long as the value does not exceed the maximum in-line length (5K for example) or the row does not exceed the maximum row length (32,705 bytes). If either length is exceeded, the data loader stores the value in a separate LOB subsegment file.

If you include the `INLINE` clause but omit the length, the default maximum length is 32,705 bytes. For instance, 32,705 bytes is the maximum length of the `CLOB_COLUMN` in the `LOBTABLE` in the following example:

```
CREATE TABLE LOBTABLE  
( CLOB_COLUMN CLOB INLINE )  
TABLESPACE LOBSpace
```

If you omit both the `INLINE` clause and the `NOT INLINE` clause on a column definition, the default is to store the LOB data in-line as space permits. In other words, `INLINE` without a length specification is the default.

To store LOB values in a separate LOB subsegment file

You can store LOB column values in a separate file (out-of-line) from the table data by specifying the `NOT INLINE` clause on a column definition. In this case, the FileTek data loader stores an OID in the table data file and the LOB values in a LOB subsegment file. LOB values for a column are stored in the same LOB subsegment file; however, if a LOB subsegment file reaches the maximum size, (approximately 100 GB), the FileTek data loader creates another LOB subsegment file.

In the following example, the FileTek data loader stores `CHAR_COLUMN` and `DATE_COLUMN` values in the table data file, `CLOB_COLUMN` values in one LOB subsegment file, and `BLOB_COLUMN` values in another LOB subsegment file.

```
CREATE TABLE LOBTABLE2  
( CHAR_COLUMN CHAR(32),  
  DATE_COLUMN DATE,  
  CLOB_COLUMN CLOB(1G) NOT INLINE,  
  BLOB_COLUMN BLOB NOT INLINE )  
TABLESPACE LOBSpace
```

To share storage with another LOB column

You can store the data of multiple LOB columns in the same LOB subsegment file by specifying the `STORE WITH` clause on a column definition. This enables multiple LOB values in multiple columns to share the same LOB subsegment file.

In the following example, the FileTek data loader stores `CLOB_ONE` values out-of-line with the `CLOB_TWO` values and attempts to store the `BLOB_SMALL` values in-line with the `CHAR_COLUMN` data.

```
CREATE TABLE LOBTABLE3
( CHAR_COLUMN CHAR(32),
  CLOB_ONE CLOB NOT INLINE,
  CLOB_TWO CLOB(1M) STORE WITH CLOB_ONE,
  BLOB_SMALL BLOB(100K) )
TABLESPACE LOBSpace
```

To store LOB values in a different user tablespace

You can assign a LOB column to a different user tablespace from the table by specifying the `TABLESPACE` clause on a column definition. If you omit this clause, the FileTek data loader assigns the LOB column to the same user tablespace as the table.

Note: For in-line LOBs, a FileTek data loader always uses the table's subspace and user tablespace, ignoring any LOB subspace or user tablespace.

In the following example, the FileTek data loader uses the `CLOBSPACE` user tablespace for the `CLOB_ONE` column and the `LOBSpace` user tablespace for the `PRIMARY_KEY`, `BLOB_ONE`, and `BLOB_TWO` columns.

```
CREATE TABLE LOBTABLE4
( PRIMARY_KEY INTEGER,
  BLOB_ONE BLOB INLINE(1K) ,
  CLOB_ONE CLOB(1M) NOT INLINE TABLESPACE CLOBSPACE,
  BLOB_TWO BLOB(100K) )
TABLESPACE LOBSpace
```

Creating a user table

You can create a user table in a database by submitting a CREATE TABLE statement. You then load data into that user table with a FileTek data loader.

When you create a user table, you must provide:

- The table name
- At least one column definition

You can optionally provide:

- A default definition for each column
- The user tablespace name (FileTek recommends that you always provide this)

Note that if you don't provide:

- An owner name in the table name, then StorHouse/RM makes you the owner
- A default definition for a column, then a column can be null
- The user tablespace name, StorHouse/RM assigns a default user tablespace as described on page 5-19.

▼ To create a user table

Required privileges: SQLEXECUTE and either DBA or RESOURCE

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE TABLE statement. Some examples follow.

- To create a four-column table called `LOCATION` for account ID `USER1` in the `ORDERS` tablespace, type:

```
CREATE TABLE USER1.LOCATION
( LOC_NUM SMALLINT,
  CITY_NAME CHAR(25),
  STATE_ABBREV CHAR(2),
  COUNTRY CHAR(25) )
TABLE SPACE ORDERS
```

- To create a four-column table called `LOCATION` for your account ID (`SYSADM`) in your default tablespace, type:

```
CREATE TABLE LOCATION
( LOC_NUM SMALLINT,
  CITY_NAME CHAR(25),
  STATE_ABBREV CHAR(2),
  COUNTRY CHAR(25) )
```

- To create a four-column table called `LOCATION` for your account ID (`SYSADM`) in your default tablespace with a default definition of U.S.A. for the `COUNTRY` column, type:

```
CREATE TABLE LOCATION
( LOC_NUM SMALLINT,
  CITY_NAME CHAR(25),
  STATE_ABBREV CHAR(2),
  COUNTRY CHAR(25) DEFAULT 'U.S.A.' )
```

- To create a table with a CLOB column using the default user tablespace, type:

```
CREATE TABLE LOBEXAMPLE1
( PRIMARY_KEY INTEGER,
  CLOB_ONE CLOB )
```

In the preceding example, the FileTek data loader stores the CLOB_ONE column values in-line as long as the value or the row size does not exceed 32KB. If either exceeds the size limit, the data loader stores the PRIMARY_KEY values in the table data file and the CLOB_ONE values out-of-line in a LOB subsegment file in the default user tablespace.

- To create a table with multiple LOB columns using both in-line and out-of-line storage options, type:

```
CREATE TABLE LOBEXAMPLE2
( PRIMARY_KEY INTEGER,
  CLOB_ONE CLOB INLINE(1K),
  CLOB_SMALL CLOB(1M) NOT INLINE,
  BLOB_SMALL BLOB(100K) )
TABLESPACE LOBSPACE
```

In the preceding example, the FileTek data loader stores CLOB_ONE values in-line for values up to 1K, BLOB_SMALL values in-line as long as they fit in the row, and CLOB_SMALL values out-of-line in a separate LOB subsegment file. If only one of the LOB values fits in the row, the data loader stores the CLOB_ONE values because that column is defined first.

- To create a table with multiple LOB columns that share storage, type:

```
CREATE TABLE LOBEXAMPLE3
( PRIMARY_KEY INTEGER,
  CLOB_ONE CLOB,
  CLOB_SMALL CLOB(1M) STORE WITH CLOB_ONE,
  BLOB_SMALL BLOB(100K) STORE WITH CLOB_ONE )
TABLESPACE LOBSPACE
```

In the preceding example, the FileTek data loader stores the data in two files. The table data file contains the PRIMARY_KEY values plus any LOB values that fit in the table data row. The LOB subsegment file contains the CLOB_ONE, CLOB_SMALL, and BLOB_SMALL values that do not fit in the table row.

Displaying information about tables

You can display information about user and system tables in a database by submitting a SELECT statement on the SYSTABLES system table. This information includes the table creator and owner, the ID of the user tablespace to which the table is assigned, and the next segment ID to be loaded.

▼ To display information about tables

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSTABLES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement for the SYSTABLES system table. Some examples follow.

- To display all information about all user and system tables in a database, type:

```
SELECT *  
FROM SYSADM.SYSTABLES
```

- To display all information about all tables owned by a specific owner, for instance, by SYSADM, type:

```
SELECT *  
FROM SYSADM.SYSTABLES  
WHERE OWNER='SYSADM'
```

- To display all information about a specific table (by table name), type:

```
SELECT *  
FROM SYSADM.SYSTABLES  
WHERE TBL='LOCATION'
```


Listing table names and IDs

You can list a table name or ID by submitting a SELECT statement on the SYSTABLES system table. This task is helpful if you are choosing a user table name and want to determine whether the one you have chosen is unique for a database.

▼ To list a table ID or a table name

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSTABLES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement for the SYSTABLES system table. Some examples follow.

- To display all table IDs and table names of all user and system tables in a database, type:

```
SELECT ID, TBL  
FROM SYSADM.SYSTABLES
```

- To display all table names for a specific owner, such as SYSADM, type:

```
SELECT TBL  
FROM SYSADM.SYSTABLES  
WHERE OWNER='SYSADM'
```

Displaying a column definition

You can display a column definition for a user table by submitting a SELECT statement on the SYSCOLUMNS system table. This task is helpful if you want to

determine the data types of columns or whether a column allows null values or has default values. A column definition includes the following:

- Column name
- Data type
- Length of the data type
- Scale of a NUMERIC or DECIMAL data type
- Null indicator
- Default value, if specified
- LOB storage options

▼ **To display a column definition for a user table**

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSCOLUMNS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSCOLUMNS system table. Some examples follow.

- To display all column definitions for a specific table, such as the LOCATION table, type:

```
SELECT *  
FROM SYSADM.SYSCOLUMNS  
WHERE TBL='LOCATION'
```

- To display a specific column definition for a specific table, such as the STATE column in the LOCATION table, type:

```
SELECT *  
FROM SYSADM.SYSCOLUMNS  
WHERE COL='STATE'  
AND TBL='LOCATION'
```

- To display all column definitions allowing null values, for all tables in a database, type:

```
SELECT *  
FROM SYSADM.SYSCOLUMNS  
WHERE NULLFLAG='Y'
```

- To list the table names with BLOB or CLOB columns and the in-line maximum length for each column, type:

```
SELECT TBL LOB_INLINE_MAX  
FROM SYSADM.SYSCOLUMNS  
WHERE COLTYPE IN ('BLOB' , 'CLOB')
```

Displaying volume set, file set, and segment information

You can display information about volume sets, file sets, and segment files by submitting StorHouse Command Language SHOW VSET, SHOW FSET, and SHOW FILE commands. This is useful if you want to display segment file sizes and volume set and file set properties.

Refer to the StorHouse *Command Language Reference Manual* for more information about these commands. See “Segment and LOB subsegment identifiers” on page 2-19 for more information about file name formats. The file name format differs by StorHouse/RM release.

▼ **To display all available information about a volume set**

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW VSET command and press **Enter**. An example follows.
 - To display all the available information about volume set JAN2000, type:

```
SHOW VSET JAN2000 /FULL
```

The system displays the following:

```
VSET=JAN2000 DIRECTORY=PRIMARY SIZE=20459KB  
SURFACES=2 GENERAL_FREE=18604KB MEDIA=OEB LIMIT=0KB  
GENERAL_ALLOCATED=1854KB CREATED=01-JAN-2000:00:06:04  
MODIFIED=27-JAN-2000:14:03:16 CYCLE=0 DEACTIVATE=0  
EXPIRE=0 HOLD=NOHOLD LIBRARY=L00 ARCFSET=*  
BKPVSET=none BKPFSET=* MEMO="Rack 4"  
Total vsets displayed=1
```

▼ **To display all available information about a file set**

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW FSET command and press **Enter**. An example follows.
 - To display all available information about file set JAN2000 in volume set JAN2000, type:

```
SHOW FSET JAN2000 /VSET=JAN2000 /FULL
```

The system displays the following:

```
FSET=JAN2000 VSET=JAN2000 DIRECTORY=PRIMARY
SIZE=1853KB GENERAL_FREE=2KB UPDATE_FREE=0KB
GENERAL_ALLOCATED=1165KB UPDATE_ALLOCATED=685KB
UPDATE_PERCENT=00 LIMIT=0KB CREATED=01-JAN-
2000:06:10:09
```

Total fsets displayed=1

▼ To display information about segment files

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW FILE command and press **Enter**. Some examples follow.
 - To display file name, group, version, and file ID information for all files in file set JAN2000 on volume set JAN2000, type:

```
SHOW FILE * /VSET=JAN2000 /FSET=JAN2000
```

The system displays the following:

```
FILE="CALLS.00001.T0000000150.0000000000" GROUP=STH
VERSION=0 FID=234.3
```

Total files displayed = 1

- To display file name, group, version, file ID, data, and size information for a table data file named CALLS.00001.T0000000150.0000000000, type:

```
SHOW FILE CALLS.00001.T0000000150.0000000000 /BRIEF
```

The system displays the following:

```
FILE="CALLS.00001.T0000000150.0000000000" GROUP=STH
VERSION=0 FID=234.3 DATE=22-JAN-2000:21:09:51 SIZE=8088
```

Total files displayed = 1

Note: By substituting /FULL for /BRIEF, you can display all available information about files, including the volume set and file set where they reside.

- To display extent information for a table data file named CALLS.00001.T0000000150.0000000000, type:

```
SHOW FILE CALLS.00001.T0000000150.0000000000
/EXTENT
```

The system displays all extents for the table data file, beginning with the most recently written extent. In this example, the DF extent is listed first (extent number 1000001), followed by the data extent (extent number 1000000).

```
FILE="CALLS.00001.T0000000150.0000000000" GROUP=STH
VERSION=0 FID=43726.636
```

```
EXTENT_NUMBER=1000001 EXTENT_SID=43726
EXTENT_DATE=22-JAN-2000:21:09:51
EXTENT_WRITTEN=22-JAN-2000:21:09:59 EXTENT_REVISION=1
EXTENT_SIZE=80 EXTENT_LOCATION=OEB"31F395EF":A
EXTENT_LEVEL=L EXTENT_STATUS=(LAST) EXTENT_MF=none
EXTENT_RETENTION_DATE=25-JAN-2000:11:59:59
```

```
EXTENT_NUMBER=1000000 EXTENT_SID=43726
EXTENT_DATE=22-JAN-2000:21:09:51
EXTENT_WRITTEN=22-JAN-2000:21:09:53 EXTENT_REVISION=1
EXTENT_SIZE=800 EXTENT_LOCATION=OEB"31F395EF":A
```

```
EXTENT_LEVEL=L EXTENT_STATUS=(none) EXTENT_MF=none  
EXTENT_RETENTION_DATE=25-JAN-2000:11:59:59
```

```
EXTENTS_DISPLAYED=2
```

```
Total files displayed=1
```

Renaming a user table

You can rename a user table by submitting a RENAME statement. When you rename a user table, the old name becomes obsolete. You must have DBA or RESOURCE privilege to rename a user table. A DBA may rename a user table for another owner. You do not have to provide the owner name if you are the table's owner.

▼ To rename a user table

Required privileges: SQLEXECUTE and either DBA or RESOURCE

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a RENAME statement. For example, to rename the USER1.LOCATION table to USER1.REGIONWEST, type:

```
RENAME USER1.LOCATION TO USER1.REGIONWEST
```

Dropping a user table

You can drop a user table by submitting a DROP TABLE statement. When you drop a user table, StorHouse/RM adds a row to the SYSDROP_PEND system table, but it does not delete any metadata or invalidate segment files. The user table is assigned (renamed to) an internally generated name that cannot be used

in any CREATE TABLE statement. You subsequently use the PURGE TABLE statement when you're ready to delete metadata and invalidate segment files for a user table. See "Purging a user table" on page 6-28 for more information about deleting metadata and invalidating segment files after dropping a user table.

▼ **To drop a user table**

Required privileges: SQLEXECUTE and either DBA or table owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a DROP TABLE statement. Some examples follow.
 - To drop a table you own called LOCATION, type:

```
DROP TABLE LOCATION
```

- To drop a table called CUSTOMER owned by account ID USER 1, if you have DBA privilege, type:

```
DROP TABLE USER1.CUSTOMER
```

Undropping a user table

You (an account with DBA database privilege) can undrop a user table that was previously dropped by submitting a CREATE TABLE statement with the FROM DROPPED clause. When you undrop a user table, StorHouse/RM re-creates the user table with all of the data intact. You can rename the new user table and/or indexes or keep the original name(s).

If a user table is repeatedly created and dropped with the same owner and table name, then multiple instances of the user table may exist. For example, if you create a table, drop it, create it again with the same name and drop it, then two table instances exist (assuming you didn't purge the tables). You can specify a

drop time (BEFORE clause) to indicate which table instance to undrop. For example, if you dropped the first table instance on July 1, 2006 and you dropped the second table instance on July 2, 2006, then you could specify '7/2/2006' to undrop the first table instance. If you omit the BEFORE clause, StorHouse/RM undrops the most recent table instance.

▼ To undrop a user table

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE TABLE statement with the FROM DROPPED clause and optionally the BEFORE clause. Some examples follow.

- To undrop a table called LOCATION keeping the original table and index names, type:

```
CREATE TABLE FROM DROPPED LOCATION
```

- To undrop the LOCATION table dropped before 07/30/2006 and to keep the original table and index names, type:

```
CREATE TABLE FROM DROPPED LOCATION BEFORE '07/30/2006'
```

- To undrop the LOCATION table dropped before 07/30/2006 and to rename the new table and indexes, type:

```
CREATE TABLE NEWLOCATION FROM DROPPED LOCATION  
BEFORE '07/30/2006'  
INDEX NAMES NEWLOCATION_IDX1, NEWLOCATION_IDX2
```

Purging a user table

You can purge a dropped user table by submitting a `PURGE TABLE` statement. When you purge a user table, StorHouse/RM deletes all metadata for the user table and associated indexes and invalidates the segment files. StorHouse/RM does not purge any views dependent on the table, but it does invalidate them.

If multiple instances of a user table exists (when a table with the same owner name and table names is created and dropped repeatedly), you can include a `BEFORE` clause to indicate which table instance to purge. If you omit the `BEFORE` clause, StorHouse/RM purges the most recent user table.

The `SQL_DROP_HOLD` system parameter specifies the minimum delay, in days, between a drop and a purge operation. An error occurs if you attempt to purge a user table before that time has expired.

After purging a user table, you can run (1) the segment delete utility to remove the segment files from StorHouse and (2) the StorHouse `REMOVE` command to remove StorHouse directory entries to recover the space used by the segment files (if reusable). See Chapter 14, “Segment delete,” for more information about running the segment delete utility.

▼ To purge a user table

Required privileges: `SQLEXECUTE` and either `DBA` or table owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a `PURGE TABLE` statement. Some examples follow.
 - To purge a table called `CUSTOMER` owned by account ID `USER 1`, type:

```
PURGE TABLE USER1.CUSTOMER
```

- To purge a table called LOCATION that was dropped before 07/30/2006, type:

```
PURGE TABLE LOCATION BEFORE '07/30/2006'
```

6

User tables

Purging a user table

Indexes for user tables

This chapter describes indexes and explains how to:

- Create an index
- Display information about indexes
- List index names and IDs
- Display volume set, file set, and segment information
- Drop an index

About StorHouse indexes

User table *indexes* provide quick and efficient access to table data. StorHouse supports three types of indexes: value, hash, and range. You define the indexes, and a FileTek data loader creates the index entries during a load. You can display information about indexes or delete them when they are no longer needed.

The following general rules apply to indexes:

- You can create any index type for any column data type except BLOB and CLOB.
- You can create different types of indexes for the same column. For instance, you can create a range index and a value index for a column.
- The maximum length of a VARCHAR and VARBINARY column that you can index is 256.

- You must create a user table before you can create an index for it.
- The name of an index (combination of owner and index name) must be unique within a database.
- You can create an index of any type before or after you've loaded data into the table.
- A *deferred index* is an index created after a table has been loaded.
- You perform an *index load operation* to load a deferred index (create index entries for existing segments). Refer to the *FileTek FTP Data Loader Manual* and the *FileTek MVS Data Loader Utility Manual* for more information about loading deferred indexes.

Value indexes

A *value index* is a type of index based on a list (in ascending order) of all the values in a column (or group of columns for a compound value index). For each column value, a value index contains an index map to the table row containing that value. By searching the value index rather than the table, then matching column values to row IDs (also called *tuple IDs*), StorHouse/RM can more efficiently find a range of rows.

For example, the following value index on the BILL_ACCT column contains an ascending list of column values and their corresponding row IDs.

Value index

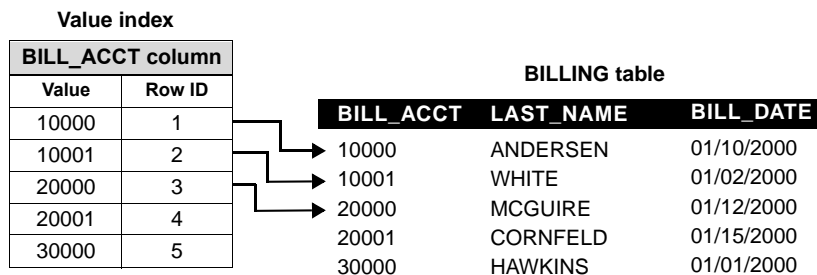
BILL_ACCT column	
Value	Row ID
10000	1
10001	2
20000	3
20001	4
30000	5

Queries that use value indexes

Value indexes work effectively with queries that return multiple rows based on a range of values. For instance, assume you created a value index on a column called `BILL_ACCT` in a user table called `BILLING` and then requested information for all account numbers between 10000 and 20000 with the following query:

```
SELECT BILL_ACCT, LAST_NAME, BILL_DATE
FROM BILLING
WHERE BILL_ACCT BETWEEN 10000 AND 20000
```

In this example, StorHouse/RM uses the value index on the `BILL_ACCT` column to locate the rows containing the range of values.



Value indexes and join processing

Value indexes can enhance join processing. For instance, a join may qualify for hybrid IN join processing when:

- The query is an equi-join
- The inner table in the join SQL has a value index on the join column

Refer to the *StorHouse SQL Reference Manual* for more information about joins, including hybrid IN.

How value indexes are stored

StorHouse/RM stores each value index as a *value index file* in a segment. For each value index on a table, there is always one value index file associated with each table data file. Each value index file is a separate STORHOUSE-type file. See “Index files” on page 2-9 for more information about value index files.

When you create a value index, you can assign it to a user tablespace. The value index files then are stored according to the specifications of the user tablespace. If you don't assign the index to a user tablespace, that is, you omit the TABLESPACE clause on the CREATE INDEX statement, StorHouse/RM assigns the index to the same user tablespace as the user table. See Chapter 5, “User tablespaces,” for more information about user tablespaces.

Hash indexes

A *hash index* is a two-part index based on an index map extent and a hit list that uses a proprietary StorHouse algorithm to effectively locate individual table rows based on individual index values. For hash indexes (unlike for value indexes), StorHouse/RM must access the data row to determine that the row's column content meets the selection criteria.

Queries that use hash indexes

Hash indexes work for queries that return specific rows based on a specific value. In other words, StorHouse/RM uses a hash index for queries that contain only equals-type predicates. StorHouse/RM uses a compound hash index for queries that contain equals-type predicates for all columns in the compound index.

For instance, StorHouse/RM might use a hash index for the following query because the query contains an equals-type predicate:

```
SELECT LAST_NAME, BILL_DATE
FROM BILLING
WHERE BILL_ACCT = 10000
```


In this example, you could create both a value index and a hash index on the `BILL_ACCT` column to satisfy both single row queries and queries that request a range of values. StorHouse/RM decides which index to use to satisfy a specific query.

How hash indexes are stored

StorHouse/RM stores each hash index as a *hash index file* in a segment. For each hash index on a table, there is always one hash index file associated with each table data file. Each hash index file is a separate `STORHOUSE`-type file. See “Index files” on page 2-9 for more information about hash index files.

When you create a hash index, you can assign it to a user tablespace. The hash index files then are stored according to the specifications of the user tablespace. If you don't assign the index to a user tablespace, that is, you omit the `TABLESPACE` clause on the `CREATE INDEX` statement, StorHouse/RM assigns the index to the same user tablespace as the user table. See Chapter 5, “User tablespaces,” for more information about user tablespaces.

Hash indexes and join processing

StorHouse/RM does not use compound hash indexes to process joins. See “Simple and compound indexes” on page 7-8 for more information about compound indexes.

Range indexes

For a specific column (or columns in the case of a compound index) within a user table, a *range index* contains the lowest and highest column data values for

each segment. For instance, the following range index on the BILL_DATE column contains the low and high dates of a user table with five segments.

Range index

BILL_DATE column		
Segment	Low Value	High Value
1	01/01/1999	01/31/1999
2	02/01/1999	02/28/1999
3	03/01/1999	03/31/1999
4	04/01/1999	04/30/1999
5	05/01/1999	05/31/1999

Each time you load new data into a user table, StorHouse/RM enters the low and high data values into the range index along with the segment ID of the segment that contains them. If there's one segment, there's one low and high value pair in the range index. If there are 400 segments, there are 400 low and high value pairs in the range index.

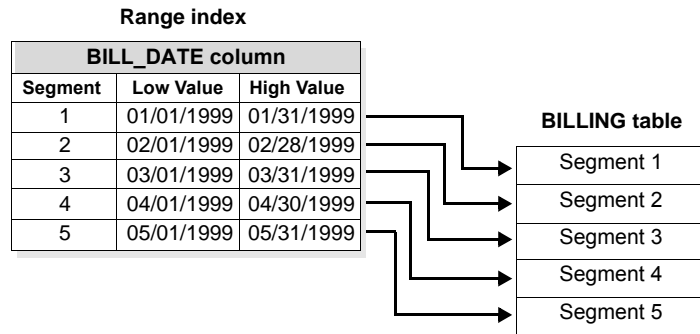
Queries that use range indexes

Range indexes are most effective for tables with multiple segments and when the data is partitioned on an indexed column. For instance, assume:

- Each month you load customer billing data into a table.
- Each row of data contains a billing date.
- The table contains a BILL_DATE column.

This data is partitioned by billing date because each segment contains a different billing date. By creating a range index for the BILL_DATE column, you can effectively search the table by date range and quickly find the segment containing the data for the billing month you want.

For example, assume the BILLING table consists of five segments and a range index is built on a BILL_DATE column. Segment 1 contains January data, segment 2 contains February data, and so on.



When responding to a query based on a specific date, StorHouse/RM can search the range index to determine which segment contains that date. It then searches the correct segment for the date, rather than searching sequentially through each segment.

For instance, if you issued the following query, StorHouse/RM uses the range index on the BILL_DATE column to quickly determine that segment 1 contains January data:

```
SELECT *  
FROM BILLING  
WHERE BILL_DATE='01/01/1999'  
AND BILL_ACCT=30000
```

If the BILL_ACCT and BILL_DATE columns have hash indexes, StorHouse/RM can use those hash indexes to search for the specific account and date values in segment 1.

Range indexes and extractor processing

An *extraction* is a type of query that returns selected columns from all rows in a table. A query may qualify for extraction when:

- The query selects one or more entire segments
- All predicates in the WHERE clause are based on columns with range indexes
- The query meets all other extractor requirements

Refer to the *StorHouse ESQL Manual* for more information about extractor processing.

How range indexes are stored

StorHouse/RM stores range index entries in a set of system tables. There is one system table for each data type. When you load more data into a user table or when you load a deferred range index, StorHouse/RM updates the range index with the new segment's low and high data values for the indexed column or columns. See "SYSRANGES" on page A-17 for more information about how range indexes are stored.

Simple and compound indexes

You can base an index on an individual column or on a combination of columns. An index based on one column is a *simple index*. An index based on multiple columns is a *compound* (or *composite*) index. You can create a compound index of any type, but as with simple indexes, you must indicate value, hash, or range when you create the index. For instance, you can create a compound value index called CUSTOMER based on the columns LAST_NAME, BILL_ACCT, and BILL_DATE in the BILLING table.

StorHouse/RM uses compound indexes when the comparison operator on all but the last column in a query is an equal sign (=) and all logical operators are AND. For example:

```
SELECT *  
FROM BILLING  
WHERE LAST_NAME='SMITH' AND BILL_ACCT='1940339' AND  
BILL_DATE < '01/01/2000'
```

When you create a compound index, the column values are stored in the order you list them in the CREATE INDEX statement. For instance, if you base a compound index on DATE and TIME fields specifying DATE first, the index values are sorted on ascending times within ascending dates. If you specify TIME first, the index values are sorted on ascending dates within ascending times.

The following rules apply to compound indexes:

- A compound index can include columns of different data types.
- A compound index is either a range, value, or hash compound index.
- The maximum number of columns allowed in a compound index is 16.
- The maximum size of any column in a compound index is 256 bytes.
- The maximum size of a compound index is 4096 bytes. This means you can index 16 columns, each with a maximum size of 256 bytes.

Default index type

The StorHouse system parameter SQL_INDX_TYPE specifies the default index type—value, hash, or range—that StorHouse/RM creates when you omit the index type on a CREATE INDEX statement. You can change the default index type by changing the SQL_INDX_TYPE system parameter. See page C-9 for more information about the SQL_INDX_TYPE system parameter.

Index owners

The owner of a user table owns all of the indexes for that table, even if someone else creates the indexes. For instance, if you create a table for JACK, and then you create an index for that table, JACK owns the index, even though you created it.

Index naming conventions

Each index has an index name and an index ID.

Index names

When you create an index, you give it a name. An index name should follow SQL identifier conventions. If an index name does not follow these conventions, you must delimit it and use the appropriate case in SQL statements.

A *fully qualified index name* is the combination of owner name and index name. This combination must be unique from all other combinations in a database. In other words, more than one index can have the same descriptive index name as long as the owner names are unique. For example, JACK.BILL_DATE and SANDY.BILL_DATE are valid index names within a database. Remember that the owner of the user table is automatically the owner of the index.

Index IDs

StorHouse/RM assigns index IDs to indexes when you create them. Index IDs are unique within a database. The first user table index ID in a database is 100, and the first system table index ID is 0.

System tables with index information

The following system tables contain information about indexes:

- **SYSINDEXES** – Contains one row for each indexed column of every table in a database. StorHouse/RM inserts a row (or multiple rows in the case of a compound index) into this system table when you create an index and deletes a row(s) when you drop an index. StorHouse/RM updates this system table after a deferred index load when all segments have indexes.
- **SYSSTHFILES** – Contains the StorHouse file names and group names of segment files, including index files. StorHouse/RM updates this system table after a confirmed data load, index load, segment replace and merge operation, or when you drop a user table or index.

What's different about StorHouse indexes?

Note the following considerations when working with indexes:

- StorHouse/RM does not support unique indexes for user tables.
- Index values are always stored in ascending order.

Creating an index for a user table

You can create a hash, value, or range index for one or more columns in a user table by submitting a `CREATE INDEX` statement. If you omit the `TABLE SPACE` clause, StorHouse/RM assigns the index to the same user tablespace as the table. If you omit the index type, StorHouse/RM uses the value of the `SQL_INDX_TYPE` system parameter. If the table contains data (has been loaded), you must create a deferred index.

▼ To create an index for a user table that hasn't been loaded

Required privileges: SQLEXECUTE and either DBA or RESOURCE or INDEX or table owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE INDEX statement for the column or columns you want to index. Some examples follow.

- To create a simple value index called CUSTACCT on the ACCTNO column in the CUSTOMERS table, type:

```
CREATE VALUE INDEX CUSTACCT  
ON CUSTOMERS (ACCTNO)
```

- To create a compound (multi-column) hash index called CUSTACCTREP on the ACCTNO and SALESREP columns in the CUSTOMERS table, and to assign the hash index to the HASHINDEX user tablespace, type:

```
CREATE HASH INDEX CUSTACCTREP  
ON CUSTOMERS (ACCTNO, SALESREP)  
TABLE SPACE HASHINDEX
```

- To create a simple range index called PAY_DATE on the PDATE column in the CUSTOMERS table, type:

```
CREATE RANGE INDEX PAY_DATE  
ON CUSTOMERS (PDATE)
```


▼ **To create a deferred index for a user table that has been loaded**

Required privileges: SQLEXECUTE and either DBA or RESOURCE or INDEX or table owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE INDEX statement for the column or columns you want to index, using the DEFERRED keyword. For example, to create a range index called BILLING_END on the INVOICE_DATE column in the CUSTOMERS table, type:

```
CREATE RANGE INDEX BILLING_END  
ON CUSTOMERS (INVOICE_DATE)  
DEFERRED
```

Displaying information about indexes

You can display information about an index by submitting a SELECT statement on the SYSINDEXES system table. This information includes index owners, indexed columns, user table name and owner, index type (value, hash, range), and tablespace ID.

▼ **To display information about indexes**

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSINDEXES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement for the columns you want to display. Some examples follow.

- To display the index ID of the CUSTACCT index owned by SMITH, type:

```
SELECT ID
FROM SYSADM.SYSINDEXES
WHERE IDXNAME='CUSTACCT'
AND IDXOWNER='SMITH'
```

- To display the name of the column used for the CUSTACCT index for the ACCOUNTS table, type:

```
SELECT COLNAME
FROM SYSADM.SYSINDEXES
WHERE IDXNAME='CUSTACCT'
AND TBL='ACCOUNTS'
```

- To display all the indexes defined for the ACCOUNTS table with their index types, type:

```
SELECT IDXNAME, IDXTYPE
FROM SYSADM.SYSINDEXES
WHERE TBL='ACCOUNTS'
```

Listing index names and IDs

You can list the names and IDs of system and user table indexes by submitting a SELECT statement on the SYSINDEXES system table.

▼ To list index names and IDs

Required privileges: SQLEXECUTE and either DBA or SELECT on SYSINDEXES

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSINDEXES system table. Some examples follow.

- To list the names and IDs of all indexes in the database, type:

```
SELECT IDXNAME, ID
FROM SYSADM.SYSINDEXES
```

- To list the names of all indexes on user tables and their associated user table names in the database, type:

```
SELECT IDXNAME, TBL
FROM SYSADM.SYSINDEXES
WHERE TBL NOT LIKE 'SYS%'
```

Because all system tables begin with the letters SYS, you are searching for all tables that do *not* begin with SYS, which are the user tables.

- To list the names and owners of all indexes in a database, type:

```
SELECT IDXNAME, IDXOWNER
FROM SYSADM.SYSINDEXES
```

- To list the IDs and names of all indexes for CUSTOMERS table owned by SMITH, type:

```
SELECT ID, IDXNAME
FROM SYSADM.SYSINDEXES
WHERE TBL='CUSTOMERS'
AND TBLOWNER='SMITH'
```

Displaying volume set, file set, and segment information

You can display volume set, file set, and segment information for indexes by submitting StorHouse Command Language SHOW VSET, SHOW FSET, and SHOW FILE commands. This is useful if you want to display segment sizes, segment create dates, extent information, and volume set and file set properties.

Refer to the StorHouse *Command Language Reference Manual* for more information about these commands. See “Segment file names” on page 2-20 for more information about file name formats. Remember that the file name format differs by StorHouse/RM release.

▼ To display all available information about a volume set

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW VSET command and press **Enter**. An example follows.
 - To display all the available information about volume set JAN1997, type:

```
SHOW VSET JAN1997 /FULL
```

The system displays the following:

```
VSET=JAN1997 DIRECTORY=PRIMARY SIZE=20459KB
SURFACES=2 GENERAL_FREE=18604KB MEDIA=OEB LIMIT=0KB
GENERAL_ALLOCATED=1854KB CREATED=01-JAN-1997:00:06:04
MODIFIED=29-JAN-1997:19:07:14 CYCLE=0 DEACTIVATE=0
EXPIRE=0 HOLD=NOHOLD LIBRARY=L00 ARCFSET=*
BKPVSET=none BKPFSET=* MEMO="Rack 4"
```

Total vsets displayed=1

▼ To display all available information about a file set

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW FSET command and press **Enter**. An example follows.
 - To display all available information about file set JAN1997 in volume set JAN1997, type:

```
SHOW FSET JAN1997 /VSET=JAN1997 /FULL
```

The system displays the following:

```
FSET=JAN1997 VSET=JAN1997 DIRECTORY=PRIMARY  
SIZE=1853KB GENERAL_FREE=2KB UPDATE_FREE=0KB  
GENERAL_ALLOCATED=1165KB UPDATE_ALLOCATED=685KB  
UPDATE_PERCENT=00 LIMIT=0KB CREATED=01-JAN-  
1997:06:10:09 MODIFIED=29-JAN-1997:19:07:14
```

Total fsets displayed=1

▼ To display information about an index file

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the appropriate StorHouse Command Language SHOW FILE command and press **Enter**. Some examples follow.

- To display file name, group, version, and file ID information for all files on file set JAN1997 on volume set JAN1997 (the * is a wildcard character), type:

```
SHOW FILE * /VSET=JAN1997 /FSET=JAN1997
```

The system displays the following (note that the file names are in 2.1 to 2.2A format):

```
FILE="CUSTOMERS.HQSM.000000130.H.0001" GROUP=STH  
VERSION=0 FID=234.2
```

```
FILE="CUSTOMERS.HQSM.000000130.H.0002" GROUP=STH  
VERSION=0 FID=234.333
```

Total files displayed = 2

- To display file name, group, version, file ID, data, and size information for all index files named CUSTOMERS.HQSM.000000130.H.* on StorHouse, type:

```
SHOW FILE CUSTOMERS.HQSM.000000130.H.* /BRIEF
```

The system displays the following:

```
FILE="CUSTOMERS.HQSM.000000130.H.0001" GROUP=STH  
VERSION=0 FID=234.2 DATE=22-JAN-1997:21:09:51 SIZE=7488
```

```
FILE="CUSTOMERS.HQSM.000000130.H.0002" GROUP=STH  
VERSION=0 FID=234.333 DATE=29-JAN-1997:19:03:25 SIZE=398
```

Total files displayed = 2

Note: By substituting /FULL for /BRIEF, you can display all available information about the files, including the volume set and file set where they reside.

- To display extent information for a value index file named STORHSE1ACCTNG97.SYSTEMA.000000161.V.0000, type:

```
SHOW FILE STORHSE1ACCTNG97.SYSTEMA.000000161.V.0000
/EXTENT
```

The system displays all extents, beginning with the most recently written extent. In this example, the extents are listed in this order: map extent (number 1000002), DF extent (number 1000001), and data extent (number 1000000).

```
FILE="STORHSE1ACCTNG97.SYSTEMA.000000161.V.0000"
GROUP=STH VERSION=0 FID=43726.225
```

```
EXTENT_NUMBER=1000002 EXTENT_SID=43726
EXTENT_DATE=21-JAN-1997:14:02:47
EXTENT_WRITTEN=21-JAN-1997:14:02:47 EXTENT_REVISION=1
EXTENT_SIZE=88 EXTENT_LOCATION=OEB"312C89B7":A
EXTENT_LEVEL=L EXTENT_STATUS=(BUFFERED, LAST, NEW)
EXTENT_MF=800000 EXTENT_RETENTION_DATE=21-JAN-
1999:11:59:59
```

```
EXTENT_NUMBER=1000001 EXTENT_SID=43726
EXTENT_DATE=21-JAN-1997:14:02:43
EXTENT_WRITTEN=21-JAN-1997:14:02:44 EXTENT_REVISION=1
EXTENT_SIZE=80 EXTENT_LOCATION=OEB"312C89B7":A
EXTENT_LEVEL=L EXTENT_STATUS=(BUFFERED, LAST, NEW)
EXTENT_MF=400000 EXTENT_RETENTION_DATE=21-JAN-
1999:11:59:59
```

```
EXTENT_NUMBER=1000000 EXTENT_SID=43726
EXTENT_DATE=21-JAN-1997:14:02:42
EXTENT_WRITTEN=21-JAN-1997:14:02:42 EXTENT_REVISION=1
EXTENT_SIZE=750 EXTENT_LOCATION=OEB"312C89B7":A
EXTENT_LEVEL=L EXTENT_STATUS=(BUFFERED, NEW)
```

```
EXTENT_MF=100000 EXTENT_RETENTION_DATE=21-JAN-1999:11:59:59
```

```
EXTENTS_DISPLAYED=3
```

```
Total files displayed=1
```

Dropping an index

You can drop an index on a user table by submitting a DROP INDEX statement. If the index is a value or hash index, this statement removes all index files (StorHouse primary files) for all segments in the user table. If the index is a range index, this statement removes all of the index entries from the range index system tables.

Note: The DROP TABLE statement performs a soft drop on all of the associated user table indexes. The PURGE TABLE statement automatically deletes all of the associated indexes.

▼ To drop an index on a user table

Required privileges: SQLEXECUTE and either DBA (to drop an index owned by another account) or RESOURCE or INDEX or table owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a DROP INDEX statement for the index you want to delete. For example, to remove the CUSTINDEX index on the CUSTOMER table, type:

```
DROP INDEX CUSTINDEX ON CUSTOMER
```


Views

This chapter describes views and explains how to:

- Create a view
- Display information about views
- List view names
- Rename a view
- Drop a view

About views

A *view* is a temporary database component that draws its contents from one or more existing tables and/or views. When you query a view, you actually query the tables referenced by the view.

Views tailor or limit access to information. They provide convenience, security, or both by letting you determine who can access what data. For convenience, you might construct a two-column view that derives customer names from the CUSTOMER table and their associated sales representatives from the SALESREPS table. For security reasons, you might construct a view that excludes commission information from SALESREPS.

How views work

You create views by pairing the CREATE VIEW statement with a SELECT statement. CREATE VIEW names the view. The SELECT statement contains a

query that defines the view. For example, suppose you have the following four-column table called SALESREPS:

SALESREPS table

SALES_REP	EMP_NUM	CITY	COMMISSION
ANDERSEN	0001000	PARIS	.03
WHITE	0002000	MADRID	.07
HOHMAN	0003000	COLOGNE	.03
REID	0004000	PHILADELPHIA	.05
MCGUIRE	0005000	CHARLOTTESVILLE	.04
SHOVLIN	0006000	PHILADELPHIA	.10
MCKENNA	0007000	LONDON	.09
CORNFELD	0008000	SARASOTA	.02

To create a view that displays only those sales representatives who are assigned to Philadelphia, you would issue the following SQL statement:

```
CREATE VIEW PHILLYSTAFF
AS SELECT *
FROM SALESREPS
WHERE CITY = 'PHILADELPHIA'
```

Whenever you access the PHILLYSTAFF view, StorHouse/RM executes the associated SELECT statement on the SALESREPS table to find those table entries that contain PHILADELPHIA in the CITY column.

For instance, if you issue this query to display all information in the PHILLYSTAFF view:

```
SELECT *
FROM PHILLYSTAFF
```

In response, StorHouse/RM executes the following SQL:

```
SELECT *  
FROM SALESREPS  
WHERE CITY = 'PHILADELPHIA'
```

The result of this query is:

PHILLYSTAFF View

SALES_REP	EMP_NUM	CITY	COMMISSION
REID	0004000	PHILADELPHIA	.05
SHOVLIN	0006000	PHILADELPHIA	.10

In summary, the PHILLYSTAFF view is a stored query that accesses the SALESREPS table whenever you access the view. When information in SALESREPS changes, the PHILLYSTAFF view may also change because the stored query may return different results. For example, if you load more data into the SALESREPS table, the next time you query the PHILLYSTAFF view, any new sales representatives are included in the results.

Owners and creators of views

Each view has an owner. A view *owner* has ALL table privileges for the view. (This does not include the INDEX privilege because you cannot add an index to a view.) In addition, a view owner can grant privileges for that view to other accounts. Typically, an owner is the account that creates the view, but accounts with DBA privilege can create views on behalf of other accounts by specifying an owner name in the CREATE VIEW statement. In this case, the account with DBA privilege is the view *creator*, and the account with the specified owner name is the view owner.

For example, in the following statement, account SSC is the owner of the view LONDON_STAFF:

```
CREATE VIEW SSC.LONDON_STAFF
AS SELECT *
FROM SALESREPS
WHERE CITY = 'LONDON'
```

View naming conventions

When you create a view, you give it a name. A view name should follow SQL identifier conventions. If a view name does not follow these conventions, you must delimit it and use the appropriate case in SQL statements.

A fully qualified view name is the combination of owner name and view name. A view, table, and synonym name must be unique for an owner. For example, CALLSDBA.OCTBILL and SYSADM.OCTBILL are valid view names in a database. A view called CALLSDBA.OCTBILL and a synonym called CALLSDBA.OCTBILL are not valid names because the combination of owner name and view name is not unique. If you omit the owner name when creating a view, then StorHouse/RM makes you the owner.

View restrictions

The following restrictions apply to views:

- You must base a view on a single query. Subqueries are not allowed.
- The query may not contain an ORDER BY clause.
- You cannot use DELETE, INSERT, or UPDATE statements with views derived from user tables. These statements are restricted to system tables and views based on system tables.

View privileges

The following rules apply:

- To create a view, you must have DBA or RESOURCE privilege.
- If you have RESOURCE privilege, you must also have SELECT privilege on any tables that the view references.
- When you create a view, you acquire a set of default privileges. These may include privileges you don't have on the underlying tables. However, when you attempt to access the view, StorHouse/RM will not let you perform operations on the view that you're not allowed to perform on the underlying tables.
- If your privileges on underlying tables were issued using the WITH GRANT OPTION of the GRANT command, you can grant privileges on your view to other accounts.
- When you grant privileges on the view to other accounts, the other accounts need not have privileges on the underlying tables.

System tables with view information

The following system tables contain information about views:

- SYSTABLES – Contains one row for each view (and system table and user table) in a database. StorHouse/RM inserts or deletes rows in this system table when you create or drop views (and user tables).
- SYSVIEWS – Contains one row for each view in a database. StorHouse/RM inserts or deletes rows in this system table when you create or drop views.

- SYSTABAUTH – Contains account privileges for views (and tables). StorHouse/RM maintains this system table when you grant and revoke privileges.

Creating a view

You can create a view by submitting a CREATE VIEW statement. This statement contains the view name and the query constructs the view.

Note the following:

- You can create views from tables, views, or both. The procedure is the same in every case.
- By default, the column names of a view are the same as the column names of underlying tables unless you explicitly rename them when creating the view.

▼ To create a view

Required privileges: SQLEXECUTE and either DBA or RESOURCE and SELECT on referenced tables

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE VIEW statement for the view you want to create. For example, to create a view named PHILLYSTAFF that only displays the Philadelphia sales representatives from the SALESREPS table, type:

```
CREATE VIEW PHILLYSTAFF
AS SELECT *
FROM SALESREPS
WHERE CITY = 'PHILADELPHIA'
```

Creating views from multiple tables

One of the most useful features of views is that you can combine tables, tables and views, or views and views to create useful virtual tables containing information that you could not get from any of the tables or views alone.

For instance, suppose that in addition to the SALESREPS table that matches representatives with cities and commission rates, you also have a SALARY table that includes everyone in your company, their positions, and their annual salaries.

SALARY table

EMPLOYEE	EMP_NUM	POSITION	SALARY
ANDERSEN	0001000	SALES_REP	\$35,500
CORNFELD	0008000	SALES_REP	\$19,500
MCGUIRE	0005000	SALES_REP	\$42,000
HEMINGWAY	0011000	COPY_WRITER	\$55,000
HOLT	0003000	SALES_REP	\$480,000
MATISSE	0010000	DESIGNER	\$36,000
MCKENNA	0007000	SALES_REP	\$87,000
REID	0004000	SALES_REP	\$30,500
SHOVLIN	0006000	SALES_REP	\$36,000
WHITE	0002000	SALES_REP	\$235,000
WILKINS	0009000	ACCOUNTANT	\$75,000

To create a view called `SALES_PAY` displaying all sales representatives, their commission rates, and their annual salaries, you could issue the following SQL statement to combine this information from the `SALESREPS` and `SALARY` tables:

```
CREATE VIEW SALES_PAY
AS SELECT SALES_REP, SALESREP.EMP_NUM, COMMISSION, SALARY
FROM SALESREPS, SALARY
WHERE POSITION = 'SALES_REP'
AND SALES_REP = EMPLOYEE
```

Notice that within the query, you must precede the column name `EMP_NUM` with the table name `SALESREPS`. This is because `EMP_NUM` is a column name in both the `SALESREPS` and `SALARY` tables. Whenever such an ambiguity occurs, you must resolve it by providing the fully qualified name of the table or view that you intend to use.

To see all rows of the `SALES_PAY` view, issue the query:

```
SELECT *
FROM SALES_PAY
```

The query produces the following:

SALES_PAY View

SALES_REP	EMP_NUM	COMMISSION	SALARY
ANDERSEN	0001000	.03	\$35,500
WHITE	0002000	.07	\$235,000
REID	0004000	.05	\$30,500
MCGUIRE	0005000	.04	\$42,000
SHOVLIN	0006000	.10	\$36,000
MCKENNA	0007000	.09	\$87,000
CORNFELD	0008000	.02	\$19,500
HOLT	0003000	.03	\$480,000

▼ To create a view based on multiple tables (or views)

Required privileges: SQLEXECUTE, DBA or RESOURCE, and SELECT on referenced tables

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE VIEW statement for the view you want to create. For example, to create a view named SALES_PAY that combines the SALESREPS and SALARY tables to display all sales representatives, their commission rates, and their annual salaries, type:

```
CREATE VIEW SALES_PAY
AS SELECT SALES_REP, SALESREP.EMP_NUM, COMMISSION,
SALARY
FROM SALESREPS, SALARY
WHERE POSITION = 'SALES_REP'
AND SALES_REP = EMPLOYEE
```

Naming columns in views

By default, when you create a view, the names of the columns in the view are the same as those in the base tables. However, there may be times when you want to rename the columns. For instance, you must rename columns when creating a view from two tables with columns that have the same name.

You can rename columns in the CREATE VIEW statement by following the name of the view with new column names in parentheses. For instance, you could have created the SALES_PAY view to display the output in columns labeled SALES_PERSONS, NUM, COM, and SALARY as shown in the following example.

Although you don't intend to change every column name, you must specify all four column identifications. This is because if you name one output column, you must name all output columns.

▼ **To create a view with column names that differ from the column names in the base tables**

Required privileges: SQLEXECUTE, DBA or RESOURCE, and SELECT on referenced tables

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE VIEW statement name that includes new column names in parentheses for the view you want to create. For example, to create a view with columns named SALES_PERSONS, NUM, COM, AND SALARY, type:

```
CREATE VIEW SALES_PAY (SALES_PERSONS,NUM,COM,SALARY)
AS SELECT SALES_REP, SALESREP.EMP_NUM, COMMISSION,
SALARY
FROM SALESREPS, SALARY
WHERE POSITION = 'SALES_REP'
AND SALES_REP = EMPLOYEE
```

Displaying information about views

You can display information about views by submitting a SELECT statement on the SYSVIEWS system table. View information includes view name, view owner and creator, and the query that constructs the views.

▼ **To display information about views**

Required privileges: SQLEXECUTE and DBA or SELECT on SYSVIEWS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement on the SYSVIEWS system table. Some examples follow.

- To list all views in a database with the SQL statements that define them, type:

```
SELECT *  
FROM SYSADM.SYSVIEWS
```

- To display information about all views owned by a specific account, for instance, by SYSADM, type:

```
SELECT *  
FROM SYSADM.SYSVIEWS  
WHERE OWNER='SYSADM'
```

- To display the SQL definition of a specific view (by view name), for instance, SALESREPS, type:

```
SELECT VIEWTEXT  
FROM SYSADM.SYSVIEWS  
WHERE VIEWNAME='SALESREPS'
```

Listing view names

You can list the view names in a database by submitting a SELECT statement on the SYSVIEWS system table.

▼ To list view names

Required privileges: SQLEXECUTE and DBA or SELECT on SYSVIEWS

1. Follow your site's procedure for connecting to your StorHouse database.

2. Submit the following SELECT statement on the SYSVIEWS system table.

```
SELECT VIEWNAME,OWNER  
FROM SYSADM.SYSVIEWS
```

Renaming a view

You can rename a view by submitting a RENAME statement. When you rename a view, the old name becomes obsolete. You must have DBA or RESOURCE privilege to rename a view. A DBA may rename a view for another owner. You do not have to provide the owner name if you are the view's owner.

▼ To rename a view

Required privileges: SQLEXECUTE and either DBA or RESOURCE

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a RENAME statement. For example, to rename the SALES_PAY View to SALES_PAY_REGIONWEST, type:

```
RENAME SALES_PAY TO SALES_PAY_REGIONWEST
```

Dropping a view

You can drop a view by submitting a DROP VIEW statement. Views that reference a dropped view become invalid but are not automatically removed from StorHouse.

▼ **To drop a view**

Required privileges: SQLEXECUTE and DBA (to drop a view owned by another account) or view owner.

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a DROP VIEW statement for the view you want to drop. Some examples follow.

- To drop a view named SALES_SAL that you own, type:

```
DROP VIEW SALES_SAL
```

- To drop a view named CORP_SALES owned by account USER1, type:

```
DROP VIEW USER1.CORP_SALES
```

8

Views

Dropping a view

Synonyms

This chapter describes synonyms and explains how to:

- Create a synonym
- Display information about synonyms
- List synonym names
- Rename a synonym
- Drop a synonym

About synonyms

A *synonym* is another name for a table or view. You can create a synonym and then use it instead of the original component name in the following SQL statements:

- SELECT
- GRANT
- REVOKE
- INSERT (for system tables and loading user tables)
- UPDATE (for system tables only)
- DELETE (for system tables only)

Synonyms are convenient because you don't have to use fully qualified table names. For instance, you can submit this query with the synonym **BILLS**:

```
SELECT BILL_ACCT,BILL_AMOUNT  
FROM BILLS
```

instead of this query with the fully qualified table name:

```
SELECT BILLACCT,BILL_AMOUNT  
FROM CALLSDBA.BILLSUMMARY
```

Private and public synonyms

A synonym may be private or public. A *private synonym* belongs to the account who creates it. Synonyms are private by default. To use a private synonym, an account must meet one of the following requirements:

- Own the synonym
- Have DBA privilege
- Have SELECT privilege on the component the synonym represents

A *public synonym* belongs to PUBLIC, which means any StorHouse account can use the synonym. Only accounts with DBA privilege can create and drop public synonyms.

Synonym owners

The account who creates the synonym is also the synonym owner. A synonym owner can drop the synonym.

Synonym naming conventions

When you create a synonym, you give it a name. A synonym name should follow SQL identifier conventions. If a synonym name does not follow these conventions, you must delimit it and use the appropriate case in SQL statements.

A *fully qualified synonym name* is the combination of owner name and synonym name. A table, view, and synonym name must be unique for an owner. For

instance, a synonym called SYSADM.BILLS and a table called CALLSDBA.BILLS are valid names in a database. A synonym called SYSADM.BILLS and a view called SYSADM.BILLS are not unique and therefore not valid names.

System table with synonym information

The SYSSYNONYMS system table contains the name of each synonym, its owner and creator, and the name and owner of the database component the synonym represents. StorHouse/RM inserts and deletes rows in this system table when you create and drop synonyms.

Creating a synonym

You can create a private or public synonym with the CREATE SYNONYM statement. Synonyms are private by default. When you create a synonym, you specify the name of the synonym and the name of the database component the synonym represents.

▼ To create a private synonym

Required privileges: SQLEXECUTE and DBA or RESOURCE

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE SYNONYM statement for the table or view you want to represent with a synonym. For example, to create a private synonym called NEW for the table SYSADM.NEWEMPLOY, type:

```
CREATE SYNONYM NEW  
FOR SYSADM.NEWEMPLOY
```

▼ To create a public synonym

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a CREATE PUBLIC SYNONYM statement for the table or view you want to represent with a synonym. For example, to create a public synonym called EMP for the table JACK.NEWEMPLOY, type:

```
CREATE PUBLIC SYNONYM EMP  
FOR JACK.NEWEMPLOY
```

Displaying information about synonyms

You can display information about synonyms by submitting a SELECT statement on the SYSSYNONYMS system table. Synonym information includes synonym name and the component it represents, owner and creator, and type (public or private).

▼ To display information about synonyms

Required privileges: SQLEXECUTE and DBA or SELECT on SYSSYNONYMS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a SELECT statement for the SYSSYNONYMS system table. Some examples follow.
 - To display information about all synonyms in a database, type:

```
SELECT *  
FROM SYSADM.SYSSYNONYMS
```

- To display information about all synonyms created by a specific account, for instance, by SYSADM, type:

```
SELECT *  
FROM SYSADM.SYSSYNONYMS  
WHERE SOWNER='SYSADM'
```

- To display just the synonyms and the tables or views they represent, type:

```
SELECT SNAME, STBL  
FROM SYSADM.SYSSYNONYMS
```

- To display all synonyms for a specific user table (such as NEWEMPLOY), type:

```
SELECT *  
FROM SYSADM.SYSSYNONYMS  
WHERE STBL='NEWEMPLOY'
```

- To display all of the public synonyms in a database, type:

```
SELECT SNAME  
FROM SYSADM.SYSSYNONYMS  
WHERE ISPUBLIC='1'
```

Listing synonym names

You can list the synonym names by submitting a SELECT statement on the SYSSYNONYMS system table.

▼ To list synonym names

Required privileges: SQLEXECUTE and DBA or SELECT on SYSSYNONYMS

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit the following SELECT statement on the SYSSYNONYMS system table:

```
SELECT SNAME,SOWNER  
FROM SYSADM.SYSSYNONYMS
```

Renaming a synonym

You can rename a synonym by submitting a RENAME statement. When you rename a synonym, the old name becomes obsolete. You must have DBA privilege or own the synonym. A DBA may rename a public synonym.

▼ To rename a synonym

Required privileges: SQLEXECUTE and either DBA or synonym owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a RENAME statement. For example, to rename the EMP synonym to NEWEMP, type:

```
RENAME EMP TO NEWEMP
```

Dropping a synonym

You can drop a private or public synonym by submitting a DROP SYNONYM statement. When you drop a synonym, any views or other synonyms that reference the synonym remain, but they become invalid.

▼ **To drop a private synonym**

Required privileges: SQLEXECUTE and DBA or synonym owner

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a DROP SYNONYM statement for the synonym you want to remove. For example, to remove a synonym called NEW, type:

```
DROP SYNONYM NEW
```

▼ **To drop a public synonym**

Required privileges: SQLEXECUTE and DBA

1. Follow your site's procedure for connecting to your StorHouse database.
2. Submit a DROP PUBLIC SYNONYM statement for the public synonym you want to remove. For example, to remove a public synonym called EMP, type:

```
DROP PUBLIC SYNONYM EMP
```

9

Synonyms

Dropping a synonym

Metadata backup

This chapter describes the metadata backup feature and explains how to:

- Back up metadata
- Schedule a metadata backup
- Troubleshoot errors
- List metadata backup files

Additionally, this chapter contains the error messages generated during a metadata backup. See “Running StorHouse utilities” on page 3-10 for basic information about using StorHouse utilities.

About metadata backup

Metadata is information that StorHouse/RM uses to manage a database. Backing up metadata is necessary to restore a database in the event of a catastrophic failure. With the metadata backup utility (sthdb_backup), you can back up metadata for one database or multiple databases at a time. Users can query a database during a metadata backup.

The metadata backup utility also provides redo journaling options to enable redo journaling for an existing StorHouse database and to reset the journaling environment. See “Enabling journaling for an unjournaled database” on page 12-10 and “Resetting the journaling environment” on page 12-22 for more information about these functions.

Files that are backed up

The metadata backup utility backs up all files in a system tablespace, including the following database components:

- System tables (files with .TBL extensions)
- System table indexes (files with .IDX extensions)
- System log files (files with .LOG extensions)

Use StorHouse Command Language CREATE BACKUP or ARCHIVE commands to back up segment files. Refer to the “Protecting User and System Files” section in the *StorHouse System Administrator’s Guide* for more information about backing up StorHouse user files.

When to run a backup

StorHouse SQL Data Definition Language (DDL) statements—ALTER, CREATE, DROP, GRANT, and REVOKE—update metadata. Consider running a metadata backup after committing any DDL statements. Also, FileTek recommends that you back up metadata after each load, or if you run a set of loads during a “load window,” back up metadata after the set completes.

Do not run a backup while a load is in progress. If any loads are active (unconfirmed, still running, failed and not yet restarted or aborted), an error occurs and the metadata backup utility stops. You can, however, force the utility (see “-f or -F” on page 10-8) to ignore the check for active loads and continue anyway. Be aware, however, that if you run loads or submit DDL statements during a metadata backup, you may have difficulty determining which transactions completed and which were queued, especially if your load stream contains DROP and CREATE statements.

Note: You can run a metadata backup when a database is down. If the backup is used to restore the database, the database will be marked down when the restore completes.

Privileges required to run a backup

A StorHouse account with one of the following command privileges can run the metadata backup utility:

- OPERATOR
- SERVICE
- SYSTEM

Note that the SYSADM account has ALLPRIVILEGE (which includes the above privileges). Refer to the StorHouse *Command Language Reference Manual* for more information about command privileges.

Before backing up metadata

Before invoking a metadata backup, set the following StorHouse system parameters for your site or use the default values if those are acceptable. See Appendix C, “System parameters for StorHouse/RM,” for more information about displaying and defining system parameters. Note that you can specify the FSET and VSET as a metadata backup command option to override the values of the SQL_BKUP_FSET and SQL_BKUP_VSET system parameters.

System parameters for metadata backup

System parameter	Description	Default
SQL_BKUP_ACCOUNT	StorHouse account used to write metadata backup files	SYSTEM
SQL_BKUP_FSET	File set to contain metadata backup files	RMMDBKUP
SQL_BKUP_GROUP	File access group for metadata backup files	RMMDBKUP
SQL_BKUP_LIMIT	Maximum number of backup file versions to keep for each database	10
SQL_BKUP_VSET	Volume set to contain metadata backup files	RMMDBKUP

Metadata backup process

The metadata backup utility performs the following:

- Verifies that the specified database(s) exist
- Checks for active or at-checkpoint data load operations
- Locks the database (read locks on system tables)
- Copies all metadata files to StorHouse
- Unlocks the database

Note that any DDL statements submitted or active loads already running during a backup are queued until after the metadata backup utility unlocks the database.

Ways to run a metadata backup

You can run the metadata backup utility four ways:

- StorHouse Command Language RUN command
- StorHouse Command Language SCHEDULE command
- StorHouse/Admin component of StorHouse/Control Center
- UNIX shell scripts and programs

This chapter describes how to back up metadata with the StorHouse RUN and SCHEDULE commands. Refer to the *StorHouse/Admin Database Administrator's Quick Reference*, publication number 900150, or the StorHouse/Control Center online help for more information about backing up metadata and scheduling a metadata backup with StorHouse/Admin.

Metadata backup output

The metadata backup utility writes status messages to standard output (stdout), which is usually defined as your terminal. Errors are reported to both stdout and to the StorHouse administration log. If the utility encounters a severe (fatal)

error, it displays a message describing the error at the StorHouse console as well. See “Troubleshooting” on page 10-12 for a list of error messages and a suggested response.

After backing up metadata

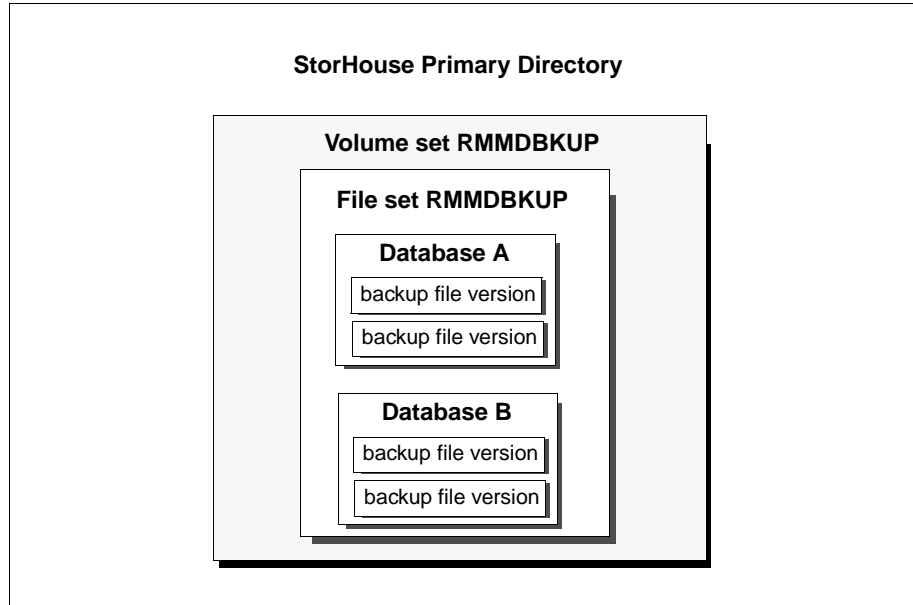
FileTek recommends that you extract StorHouse directory information after a metadata backup. You do this with the StorHouse Command Language `EXTRACT DIRECTORY` command, which copies the StorHouse directory information to a set of StorHouse users files called directory extraction files. This feature increases the speed of recovery and ensures that StorHouse directory entries are in sync with the metadata backup files. This is especially important for loads and DROPs which affect both metadata and directory entries.

Also, consider making a backup or archive copy of each metadata backup file using the `CREATE BACKUP` or the `ARCHIVE` command. This enables you to export those copies from StorHouse and store them off-site for disaster recovery purposes. You should also back up or archive table data files, index files, and LOB subsegment files after a data load. Refer to the *StorHouse System Administrator's Guide* for more information about the StorHouse directory extraction, backup, and archive features.

Where metadata backup files are stored

Metadata backup files are written to a file set in a volume set that is cataloged in the StorHouse primary directory. You specify the volume set and file set names using the `SQL_BKUP_VSET` and `SQL_BKUP_FSET` system parameters. The default name for both system parameters is `RMMDBKUP`. You can also specify the `VSET`, `FSET`, and `VTF` values as a utility option. Each time you run a backup,

the metadata backup utility creates a backup file version for the specified database.



Metadata backup file names

StorHouse files containing metadata have the following naming convention:

`META_BKUP.dbname`

where:

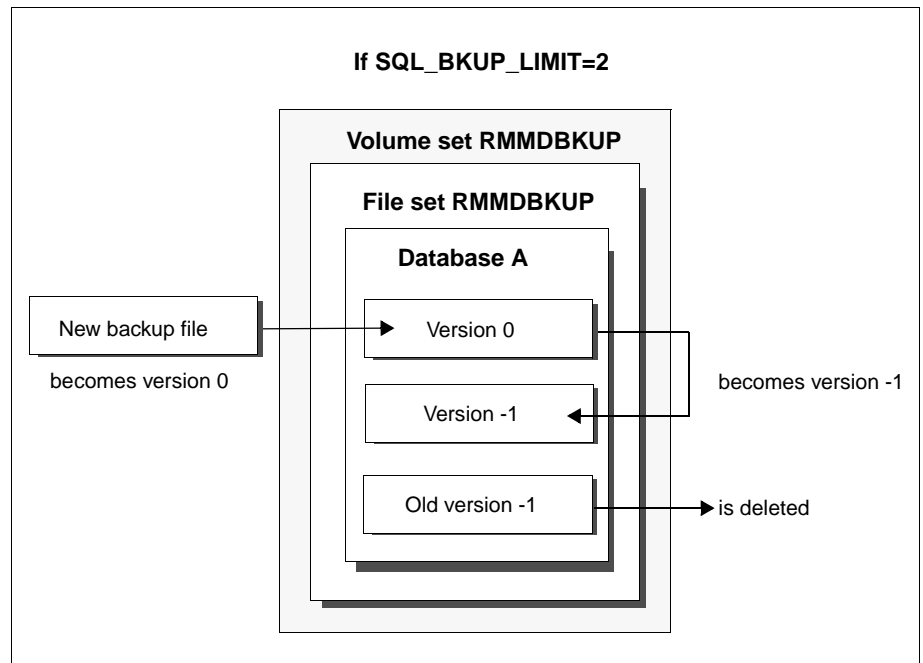
- `META_BKUP` is the file prefix
- `dbname` is the name of the database

For example, `META_BKUP.CUSTOMER`. StorHouse file version facilities maintain version numbers of metadata backup files.

Metadata backup file versions

The `SQL_BKUP_LIMIT` system parameter specifies the maximum number of backup file versions to keep for each database. The maximum setting is 99 backup file versions for each database. The default setting is 10 backup file versions. You can select the file version used to recover metadata.

StorHouse automatically deletes old file versions to make room for new ones, and it rennumbers the relative version numbers. For example, if `SQL_BKUP_LIMIT=2` and the `RMMDBKUP` volume set and file set contain two backup file versions, then the next time you run a metadata backup, StorHouse deletes backup file version -1, rennumbers backup file version 0 to version -1, and names the new backup file version 0.



Refer to the *StorHouse Concepts and Facilities Manual* for more information about file versions.

Backing up metadata

The metadata backup utility (sthdb_backup) makes a backup file of all the files in a system tablespace. You can back up metadata for one database or multiple databases at a time. If an error occurs during a backup, see page 10-12 for a list of error messages, explanations, and suggested user actions.

The format of the metadata backup utility command is:

```
sthdb_backup [options] database_name [ database_name]...
```


Argument	Description
[options]	(optional) Command options.
-f or -F	Option to force the metadata backup utility to run even when loads are active. This option ignores the data loader checkpoint status. You will receive warning messages if the metadata backup utility detects active loads. If you omit this option, in-progress or at-checkpoint data loads cause the metadata backup to fail.
-s or -S {sm_options}	Option to specify the VSET, FSET, and VTF values for the metadata backup file. If you omit this option, the utility uses StorHouse system parameters (SQL_BKUP_VSET and SQL_BKUP_FSET) to obtain the VSET and FSET values, and it uses NEXT as the default for VTF. The format of {sm_options} is: VSET=name,FSET=name,VTF=NOW DIRECT NEXT For example: -s VSET=MYVSET,FSET=MYFSET,VTF=NOW No spaces are permitted between items (sm_options).
-j or -J	Option to enable journaling for an existing database.
-r or -R	Option to reset the journaling environment.
-h or -H	Option to list command usage information.

Argument	Description
-n or -N	Option to run the utility in non-interactive mode. The backup will fail if the utility needs to prompt for direction. If you omit this option, the utility runs in interactive mode.
-v or -V	Option to receive detailed status messages.
database_name	(required) Name of the StorHouse database(s) to back up. When running the utility with the StorHouse RUN command, enclose a database name with lowercase characters in double quotes. You can specify multiple databases. Use a space to separate each database name.

Note: The journaling options (-j and -r) are described in Chapter 12. See “Enabling journaling for an unjournalized database” on page 12-10 and “Resetting the journaling environment” on page 12-22 for more information about the -j and -r options.

▼ **To run a metadata backup using the RUN command**

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the sthdb_backup command and press . Some examples follow.

- To back up a database called Customer and to specify the VSET, FSET, and VTF values, type:

```
run sthdb_backup -s VSET=MYVSET,FSET=MYFSET,VTF=NOW  
"Customer"
```

In this example, the metadata backup utility stops if any loads are active (there's no -f option on the command statement). For instance, the following messages are displayed when the -v option is used

```
Verify correct environment...  
Begin to backup database: Customer...  
Verify the existence of database....  
Check for active or at-checkpoint load operations...  
Data Loading status prevents normal backup of database \Customer\  
Reason code is Data Load checkpoint exists  
Continue with Backup Operation (enter Y or N)? n  
Data Load checkpoint exists  
Backup failed for database \Customer\
```

- To back up two database, SALES and INVENTORY, to force the metadata backup utility to bypass the load check, and to use the default VSET, FSET, and VTF values, type:

```
run sthdb_backup -f SALES INVENTORY
```

In this example, you receive warning messages if the metadata backup utility detects active loads, but the backup continues. Notice there's no comma separating the database names SALES and INVENTORY.

Scheduling a metadata backup

You can schedule a metadata backup to run at specific frequencies and times by using the StorHouse Command Language SCHEDULE command. Refer to the StorHouse *Command Language Reference Manual* for complete information

about the SCHEDULE command. The RUN command must be the last item in the command statement

▼ **To schedule a metadata backup using the SCHEDULE command**

Required privileges: SCHEDULE and OPERATOR or SERVICE or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the SCHEDULE command and press **Enter**. Some examples follow.
 - To schedule an hourly metadata backup of the Customer database, executed by account SYSTEM and password STEM, beginning at 8:00 a.m. on January 1, 2000, type:

```
SCHEDULE /START=01-JAN-2000:08:00:00 /SCHEDULE=HOURLY  
ACCOUNT=SYSTEM /PASSWORD=STEM !RUN STHDB_BACKUP  
"Customer"
```

- To schedule a daily metadata backup of the Customer database, executed by account SYSTEM and password STEM, beginning at 6:00 p.m. on January 1, 2000, type:

```
SCHEDULE /START=01-JAN-2000:18:00:00 /SCHEDULE=DAILY  
ACCOUNT=SYSTEM /PASSWORD=STEM !RUN STHDB_BACKUP  
"Customer"
```

Troubleshooting

This section contains error messages that you may receive at your terminal during a metadata backup. Each message code begins with the prefix RMBR. In the message text, information between backward slashes \ \ represents variables.

No database name specified

Explanation: The RUN command didn't contain a database name. At least one database name is required.

User Response: On the RUN command, specify the name of the database you want to back up. Database names are case sensitive, so if the name contains lowercase characters, enclose it in double quotes.

Invalid command line option\option_letter

Explanation: You entered an option letter (- followed by a character) that is not accepted by the utility.

User Response: Check the command line and be sure to enter one of the options in the table on page 10-8.

SM System Parameter not defined \parameter_name

Explanation: A required StorHouse system parameter doesn't have a value. In order to run the metadata backup utility, the following system parameters require values: SQL_BKUP_ACCOUNT, SQL_BKUP_FSET, SQL_BKUP_GROUP, SQL_BKUP_LIMIT, and SQL_BKUP_VSET.

User Response: Set a value for the identified parameter_name. Refer to Appendix C, "System parameters for StorHouse/RM," for more information about setting system parameters.

Incorrect environment setup, RM not configured

Explanation: A required environment variable—STHDBS or STHROOT—doesn't have a value.

User Response: Contact FileTek Customer Support.

Database does not exist \database_name

Explanation: The database name supplied on the command line refers to a database that doesn't exist in StorHouse.

User Response: Resubmit the RUN command with the correct database name. Be sure to type the exact database name. Database names are case sensitive. Enclose names with lowercase letters in double quotes.

Data loading status prevents normal backup of database \database name

Explanation: The metadata backup utility detected successful load operations that haven't been confirmed or unsuccessful load operations that haven't been restarted or aborted.

User Response: Confirm, restart, or abort all load operations and then resubmit the RUN command. Or, you can use the -f or -F option to ignore the active load operations. Be aware, however, that if you force the utility to continue, you may have difficulty determining which transactions in the load stream completed and which were queued.

Lock Manager error \database_name sqlcode

Explanation: StorHouse could not lock the identified database.

User Response: Call FileTek Customer Support. Be sure to give the representative the sqlcode provided in the message text.

Error writing 'tar' file \filename errno

Explanation: The UNIX utility that copies the metadata files and range index files from UNIX to a temporary file failed while writing the temporary file. The errno is a UNIX error number.

User Response: Contact FileTek Customer Support.

Error opening 'tar' file \filename errno

Explanation: The metadata backup utility couldn't open the identified temporary file. The errno is a UNIX error number.

User Response: Contact FileTek Customer Support.

Error returned from StorHouse/SM operation \return_code

Explanation: StorHouse either couldn't open or write to the metadata backup file, or the StorHouse account specified on the SQL_BKUP_ACCOUNT system parameter is invalid or doesn't have the correct privilege. The return_code is the StorHouse return code.

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information about the return_code, then contact your StorHouse system administrator for help resolving the StorHouse account or file error.

Authentication failed for AccountId \account

Explanation: The metadata backup utility couldn't obtain the password for the account specified on the SQL_BKUP_ACCOUNT system parameter or the account is invalid.

User Response: Contact your StorHouse system administrator.

StorHouse/SM Message text \message_text

Explanation: An error has been reported from a StorHouse API operation. The message_text contains the message returned by StorHouse. The RMBR016 error message is usually accompanied by other messages reporting the failing operation type and return code, for example RMBR012.

User Response: Refer to the User Response for the accompanying message(s).

Error changing to directory \directory errno

Explanation: A UNIX "change directory" (chdir) system call returned the error errno. The target directory is the StorHouse/RM database directory.

User Response: Determine the cause of the error from the UNIX error number.

Error getting temporary directory \directory

Explanation: The file etc/rdbtemp.data in the StorHouse/RM root directory could not be processed. This file is used to locate the directory to be used for the temporary file built during metadata backup processing.

User Response: Contact your FileTek customer support representative.

Backup failed for database \database

Explanation: The metadata backup operation did not complete successfully. The metadata for the indicated database does not have a usable backup.

User Response: Correct the problems indicated by other error messages and re-run the backup.

Listing metadata backup files

Use the StorHouse Command Language SHOW FILE command to list files in the metadata backup volume set and file set.

▼ **To list all metadata backup files**

Required privilege: SHOW

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the SHOW FILE command with the appropriate metadata backup volume set and file set names and press **Enter**. For example, to list all backup file versions in the RMMDBKUP volume set and file set, type:

```
SHOW FILE * /VSET=RMMDBKUP /FSET=RMMDBKUP
```

Metadata restore

This chapter describes the metadata restore feature and it explains how to:

- Restore metadata
- Bring a StorHouse database down
- Bring a StorHouse database up
- Troubleshoot errors

This chapter also contains the error messages generated during a restore. See “Running StorHouse utilities” on page 3-10 for basic information about using StorHouse utilities.

About metadata restore

The *metadata restore utility* (sthdb_restore) recovers metadata for one or more StorHouse databases from metadata backup files. Use this utility to restore the system tables and other system components for a database should the magnetic disks containing the database directories fail or the database become corrupt or damaged. You can restore only metadata that was backed up previously using the metadata backup utility.

Metadata restore process

The metadata restore utility performs the following:

- Verifies that the `$STHROOT` and `$STHDBS` environment variables exist and point to directories.
- Verifies that metadata directories exist for the specified database(s) and if not creates them.
- Verifies that StorHouse is up and that the system parameters `SQL_BKUP_ACCOUNT` and `SQL_BKUP_GROUP` are defined. If these system parameters aren't defined, the utility terminates.
- Verifies that the database is down (set by the `sthdb_down` utility) and if not brings it down. This action ensures that StorHouse engines are unable to use the incomplete metadata during the extraction phase.
- Locates the metadata backup file(s) for the specified database(s), extracts the contents of the metadata backup file, and restores the metadata into `$STHDBS/<dbname>.dbs`.
- Records the successful completion of the utility in the StorHouse administration log and writes a message to standard output (stdout) indicating that the metadata restore utility completed successfully.

Before restoring metadata

Before using the metadata restore utility, read the metadata backup procedures described in Chapter 10 and back up the metadata. Also, you can bring the database down manually using the `sthdb_down` utility described on page 11-4; although the metadata restore utility automatically brings down an online database. Finally, if the restore is part of a general disaster recovery, be sure that

the StorHouse/SM files have already been recovered and the StorHouse/SM system is fully operational.

After restoring metadata

After you run the metadata restore utility, for a journaled database, apply the redo journal to “replay” any metadata changes since the last metadata backup. You use the `sthjou_replay` utility to apply the journal. See “Replaying a journal file” on page 12-15 for more information. For a non-journaled database, you must bring up the database after restoring it. See “Bringing a StorHouse database down” on page 11-7 for more information.

How to run the metadata restore utility

You run the metadata restore utility at the StorHouse operating system (UNIX) prompt with the operator account and password.

Metadata restore output

The metadata restore utility writes status messages and errors to standard output (stdout), which is usually defined as your terminal. If the utility encounters a severe (fatal) error, it displays a message describing the error at the StorHouse console as well. See “Troubleshooting” on page 11-11 for a list of error messages and a suggested response.

Restoring metadata

The metadata restore utility (sthdb_restore) applies the latest version (default) or a specified version of a metadata backup file. Additionally, this utility:

- Verifies the database is currently down and offline and if not brings it down
- Creates a new database directory if one does not exist
- Provides an option to clean out the old database directory (deletes existing .LOG, .IDX, .TBL, .CGF and .arc files)

If an error occurs during a restore, see page 11-11 for a list of error messages, explanations, and suggested user actions.

The format of the sthdb_restore utility command is as follows.

sthdb_restore [options] database_name[:n] [-a alternate_database_name] ...

Argument	Description
[options]	(optional) Command options.
-n -N	Option to run the utility in a non-interactive mode. No prompts are issued. The default is interactive mode.
-r -R	Option to perform a read verification of the metadata backup file. The utility verifies the contents of the metadata backup file. It does not restore the database. The default is to verify the most recent metadata backup file. You can verify a later file version by specifying the version number after the database name. For example: sthdb_restore -r Calls:-1
-v -V	Option to receive detailed status messages. The default is not to receive detailed status messages.
-h	Option to display online help for the utility.

Argument	Description
database_name	(required) Name of the StorHouse database(s) to restore. You can enter multiple database names to restore multiple StorHouse databases at a time.
[:n]	(optional) Version number to use or verify (for the -r option) an earlier version of a metadata backup file. Version 0 is the default and the most recent file version. The next most recent version is -1. For example, Calls:-1 restores the Calls database with the -1 metadata backup file version.
-a -A alternate_database_name	(optional) Name of an alternate database in which to restore the data. If you omit an alternate name, the utility restores the database with the same name. If the alternate database exists, StorHouse/RM prompts (in interactive mode) whether you want to delete the contents of the database directory. If the alternate database does not exist, the utility creates a database directory.

The default is to to restore the metadata to the database directory of the specified database and to run the sthdb_restore utility in interactive mode (with prompting), using the most recent metadata backup file version, and without detailed status messages. Include the appropriate option to run the utility differently.

▼ To restore metadata

1. If necessary, log in to the StorHouse operating system (UNIX) with the operator account and password.
2. Enter the UNIX mv command to rename the database directory to database_name.replaced, where database_name is the database you want to restore. For example:

```
mv Calls.dbs Calls.replaced
```

3. Enter the syscreate command to build a new “pattern” database with the same name as the database to be restored. Do not specify the -j (journal) option,

even if the original database was journaled since the restore operation will correct the journal state. For example:

syscreate Calls

4. At the command prompt, submit the sthdb_restore command and press **Enter**. Some examples follow.

- To restore a database named Calls using all of the utility defaults, type:

```
$STHROOT/bin/sthdb_restore Calls
```

The messages that follow indicate a successful restore

```
Verify correct environment...
Begin to restore database: Calls...
Verify the existence of database....
Verify database is down....
Extract the SM file...
Restore all of the metadata files...
Restore of database \Calls\ Completed Successfully
You may now start up the database with "dbup".
Meta Restore completed successfully.
Database restore operation complete \DBs processed 1 Successful 1 Failed 0\
```

- To restore the Calls database to an existing database called Altcalls using the -1 metadata backup file version, running in interactive mode (-n option omitted) and without detailed messages (-v option omitted), type:

```
$STHROOT/bin/sthdb_restore Calls:-1 -a Altcalls
```

```
Extract the SM file...
Database Altcalls exists.
Do you wish to clean the database directory('Y' or 'N'):
y
Begin cleaning database directory.
170 files deleted.
Restore all of the metadata files...
Restore of database \Altcalls\ Completed Successfully
Journaling active for this database, run sthdb_replay.
```

Meta Restore completed successfully.

Database restore operation complete \DBs processed 1 Successful 1 Failed 0\

In the preceding example:

If the `-n` (non-interactive) option had been specified, the utility would have just deleted the contents of the database directory instead of first displaying the prompt Do you wish to clean the database directory('Y' or 'N').

If the `-v` (verbose) option had been specified, the utility would have listed each deleted file in the Altcalls database directory and provided a detailed account of each file type deleted, for example, the number of .LOG files.

- To restore the db1 database to an alternate database that does not exist, type:

```
sthdb_restore db1 -a altdb1
```

In this example, the utility creates the altdb1 database and brings it down.

Extract the SM file...

DB altdb1 doesn't exist.....

Creating /filetek/sth/sthdb/altdb1.dbs

Database altdb1 needs to be offline.

Sending shutdown message to EAM.

Restore all of the metadata files...

Restore of database \altdb1\ Completed Successfully

Journaling active for this database, run sthdb_replay.

Meta Restore completed successfully.

Database restore operation complete \DBs processed 1 Successful 1 Failed 0\

Bringing a StorHouse database down

The database down utility (sthdb_down) manually takes a StorHouse database offline to prevent a StorHouse engine from connecting to a database and to prevent users from accessing the database. StorHouse displays error message

-20238 when a program or a user attempts to access a down database. You run this command at the StorHouse operating system (UNIX) prompt.

The metadata restore utility automatically brings down an online database. The instructions are included here in case you need to bring down a database manually.

The format of the database down utility command is as follows:

`sthdb_down [options] database_name`

Option/Argument	Description
[options]	(optional) Command options.
-n -N	Option to run the utility in a non-interactive mode. No prompts are issued in this mode.
-h -H	Option to display online help for the utility.
database_name	(required) Name of the StorHouse database that you want to take offline and put into a down state. The -d or -D option letter is no longer required in StorHouse/RM 3.3 and later but may be used for compatibility with previous releases. For example, you can specify -d database_name or just the database_name.

▼ To bring a StorHouse database down

1. If necessary, log in to the StorHouse operating system (UNIX) with the operator account and password.
2. At the operating system prompt, enter the `sthdb_down` command, specify the name of the database you want to bring down, and press Enter.

For example, the following command brings down the Calls database:

```
$STHROOT/bin/sthdb_down Calls
```

3. At the prompt Are you sure? (Y or y) to continue, enter Y or y to bring the database down or press any other key (except `Enter`) to exit the utility without taking the database offline. This prompt does not appear if you specified the `-n` option.

Bringing a StorHouse database up

The database up utility (`sthdb_up`) brings a downed (offline) StorHouse database back up (online). A StorHouse engine can't connect to a database that is in an offline state. A database may be in an offline state due to any of the following reasons:

- StorHouse/RM placed the database in an offline state when it detected that an automatic metadata recovery process failed, which potentially could leave the database in an inconsistent state.
- You took the database offline manually using the database down utility (`sthdb_down`).
- The database has just been restored.

When the state of a database changes to offline as a result of either of the previous actions, StorHouse/RM adds the database name to a list of downed databases. In addition, it creates an empty file named *database_name.LCK* (where *database_name* is the name of the downed database) in the database directory. Together, these actions serve to indicate that a given database is in an offline state.

When you run the database up utility, it performs the following steps:

- Guarantees that the database is in a consistent state (all outstanding transactions have been either committed or rolled back) by invoking a general recovery process to roll back any outstanding transactions. A recovery failure causes the database up utility to fail.

- Removes the .LCK file from the database directory.
- Resets the database to an online state by removing the database name from the list of downed databases.

A force (-f) option is available to force a database up when there is a recovery failure (leaving the database with uncommitted or partially completed transactions) or when there is an inconsistency in the database down indicators (for instance, a missing .LCK file). Cautiously use the force option because accessing a database with transactional inconsistencies can lead to more damage.

The format of the database up utility is as follows:

`sthdb_up [options] database_name`

Option/Argument	Description
[options]	(optional) Command options.
-f -F	Option to force the specified database online even when the database indicators are inconsistent or the database has outstanding transactions.
-n -N	Option to run the utility in a non-interactive mode.
-h -H	Option to display online help for the utility.
database_name	(required) Name of the StorHouse database that you want to reset to an online state. The -d or -D option letter is no longer required in StorHouse/RM 3.3 and later but may be used for compatibility with previous releases. For example, you can specify -d database_name or just the database_name.

▼ To bring or force a database up

1. If necessary, log in to the StorHouse operating system (UNIX) with the operator account and password.

2. At the operating system prompt, enter the sthdb_up command, specify the name of the database you want to bring down, and press **Enter**.

For example, the following command brings up the Calls database:

```
$STHROOT/bin/sthdb_up Calls
```

The following command forces the Calls database to come up even when the database indicators are in an inconsistent state:

```
$STHROOT/bin/sthdb_up -f Calls
```

3. At the prompt Are you sure? (Y or y) to continue, enter Y or y to bring the database up or press any other key (except **Enter**) to exit the utility without taking the database online. This prompt does not appear if you specified the -n option.

Troubleshooting

The metadata restore utility displays a message at the StorHouse console when a severe (fatal) error occurs during execution. Fatal errors result when the metadata restore utility is unable to finish executing due to a serious problem. This section describes the messages and provides a problem determination checklist to assist you in diagnosing and resolving the problems that produce the error messages.

Metadata restore error messages

In the message text, information between backward slashes (\ \) represents variables. Messages that require more extensive analysis and effort to correct include a Problem Determination section, which references one or more items in the “Problem determination checklist” on page 11-16.

StorHouse/RM software not installed

Explanation: You tried to run the metadata restore utility on a StorHouse system on which StorHouse/RM software isn't installed or wasn't set up correctly.

Problem Determination: Item 2

No database name specified

Explanation: The metadata restore command did not contain a database name. At least one database name is required.

User Response: On the metadata restore command, specify the name of the database you want to restore. Enter the name in the correct case. You do not need to enclose the name in quotes.

Invalid command line option \option_letter

Explanation: You entered an invalid command argument.

User Response: Check the command line. Be sure the only arguments following the metadata restore command are the -n , -h, or -f options.

SM System Parameter not defined \parameter_name

Explanation: A required StorHouse system parameter doesn't have a value. In order to run the metadata restore utility, the following system parameters require values: SQL_BKUP_ACCOUNT and SQL_BKUP_GROUP.

User Response: Set a value for the identified parameter_name. See Appendix C, "System parameters for StorHouse/RM" in the *StorHouse Database Administration Guide* for more information about setting system parameters.

Database does not exist \database_name

Explanation: The database name supplied on the command line refers to a database that doesn't exist in StorHouse.

User Response: Verify that the database directory (\$STHDBS/database_name.dbs) exists. Resubmit the metadata restore command with the correct database name. Be sure to use the correct case. You do not need to enclose the database name in quotes.

Problem Determination: Item 3

Error opening 'tar' file \filename errno

Explanation: The metadata restore utility couldn't open the identified temporary file. The errno is a UNIX error number.

User Response: Try to determine the cause of the error based on the text of the UNIX error message. Also, verify whether the temporary file identified in the message text exists. If it doesn't, contact your FileTek customer support representative for assistance.

Error returned from StorHouse/SM operation \return_code

Explanation: StorHouse either couldn't open or read from the metadata backup file, or the StorHouse account specified on the SQL_BKUP_ACCOUNT system parameter is invalid or doesn't have the correct privilege. The return_code is the StorHouse return code.

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information about the return_code.

Authentication failed for Account ID\account

Explanation: The metadata restore utility couldn't obtain the password for the account specified or the account is invalid.

User Response: Check to be sure the account is set up correctly.

Unable to access database directory \directory

Explanation: The metadata restore utility was unable to access the database UNIX directory.

Problem Determination: Items 1, 2, 3, and 4

StorHouse/SM Message text \message_text

Explanation: An error has been reported from a StorHouse API operation. The message_text contains the message returned by StorHouse. This message is usually accompanied by other messages reporting the failing operation type and return code.

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information about these StorHouse messages.

Error getting temporary directory \directory

Explanation: The file etc/rdbtemp.data in the StorHouse/RM root directory could not be processed. This file is used to locate the directory to be used for the temporary file built during metadata backup processing.

User Response: Verify that the \$STHROOT/etc directory exists and can be referenced for reading by the operator account. Verify that the file rdbtemp.data

exists and can be read. List the contents of this file (with `more` or `cat`) and validate that:

- The first line of the file indicates the number of lines in the rest of the file.
- The remaining lines are names of directories that exist and can be referenced for writing by the operator account.

Restore of database failed \database

Explanation: The metadata restore operation did not complete successfully. The metadata for the database is not usable.

User Response: Correct the problems indicated by other `stbdb_restore` error messages and re-run the restore.

Database is still up, use `dbdown` first \database

Explanation: A database must be set to the down state before it can be restored.

User Response: Run the `stbdb_down` utility (directly from a UNIX prompt) to set the database to the downed state. Note that this will stop any current user connections to that database.

Unable to extract backed up database \database sqlcode

Explanation: The StorHouse file containing the tar image of the database could not be retrieved from StorHouse and written to a temporary UNIX file. Prior error messages will indicate the specific failure that occurred.

User Response: Correct the problem indicated by the preceding error messages.

Problem Determination: Item 6

Problem determination checklist

Many of the fatal error messages produced by the metadata restore utility can be attributed to one or more basic problems. These problems, and the suggested actions, are described in the following table. These actions should be performed by your StorHouse system administrator or by your FileTek customer support representative.

Problem determination checklist

Item	Action
1	<p>Ensure that you logged in to the StorHouse server using the UNIX operator account. This account sets up the operating environment that is necessary for a successful execution of the metadata restore utility. If you used an account other than operator, exit the StorHouse system, log in as operator, and re-execute the utility.</p> <p>If you must log in using an account other than operator for security reasons, verify that the account permissions and other data for the account are identical to that of the operator account. In addition, it is recommended that you source the operator account's .cshrc file by executing the following command:</p> <pre>source ~operator/.cshrc</pre> <p>Also, check all files you create to ensure that the operator:sm account can access them.</p>

Problem determination checklist (continued)

Item	Action
2	<p>Verify that the StorHouse/RM software resides on the system and is installed correctly by performing the following steps:</p> <ol style="list-style-type: none"> 1. Ensure that the .cshrc.rm file, which defines environment variables for StorHouse/RM, exists and hasn't been damaged. If necessary, your FileTek customer support representative can recreate the file. 2. Enter the following command at the StorHouse operating system (UNIX) prompt to determine whether the StorHouse/RM environment variables are defined: env 3. Verify that the following environment variables appear in the list (x corresponds to the release level of the software): STH_RELEASE=vxry STHROOT=/filetek/sth/vxry STHDBS=/filetek/sth/sthdb STHLOG=/filetek2/general/sth STHSQLNW=sqlnw Note that the release may also contain a StorHouse/SM release id. For example, STH_RELEASE may have a value such as v3r4_v5r6, where v3r4 is the StorHouse/RM release and v5r6 is the StorHouse/SM release. 4. Ensure that the directory defined for the \$STHROOT environment variable exists and that the release level specified for the software is correct. In addition, ensure that the other environment variables are set up correctly. 5. If one or more of the environment variables hasn't been defined or contains an incorrect value (for example, the software release defined for \$STHROOT is different from the release defined for \$STH_RELEASE), call FileTek customer support.
3	<p>Verify that the database directory (\$STHDBS/database_name.dbs with the correct case and spelling) exists, is owned by operator with UNIX group sm, and has read, write, and execute privilege for the owner.</p> <p>If the directory has been removed or destroyed, use the UNIX mkdir command to recreate it (remember that database names are case sensitive and must have a suffix of .dbs). You do not have to create any files in the directory. If you create a database using this method (UNIX mkdir command), then you must use the (-f) force option to bring it up.</p>

Problem determination checklist (continued)

Item	Action
4	Locate the .cshrc.rm file and verify that it is not damaged or corrupt. If necessary, your FileTek customer support representative can recreate the file.
5	<p>During execution, the sthdb_restore utility locks the specified database(s) to prevent users from accessing “dirty” data during the metadata restore procedure. If an error occurs that prevents the utility from locking the database, do the following:</p> <ol style="list-style-type: none"> 1. Execute stopsm to shut down the StorHouse system. 2. Execute cleanipcs to release StorHouse shared resources (including locks) to UNIX and display error messages at your terminal. 3. If cleanipcs completes successfully, execute startsm to restart the system, and then run the sthdb_restore again. If cleanipcs exits with errors, reboot the system. If necessary, contact your FileTek customer support representative for assistance following the reboot.
6	<p>The sthdb_restore utility executes the UNIX tar utility to extract the contents of the metadata backup tar file. If the tar utility is unable to complete successfully, the error may be due to any of the following problems:</p> <ul style="list-style-type: none"> ■ The account you used to log in to the StorHouse server doesn’t have the necessary permissions. ■ The system has insufficient memory. ■ The file system is full. ■ The hardware (disk) is defective.

Redo journaling

This chapter describes the StorHouse/RM redo journaling feature and explains how to:

- Specify the redo journal location
- Enable journaling for a new database
- Enable journaling for an unjournaled database
- Cycle a redo journal to close the current one and start a new one
- Archive a redo journal to StorHouse
- Replay a redo journal to restore a database
- Purge a redo journal
- Audit a redo journal
- Reset the journaling environment

About redo journaling

Redo journaling is a set of database recovery utilities for capturing, storing, and restoring transactions that affect metadata. Redo journaling is used in conjunction with the metadata backup and restore utilities as follows:

- The metadata backup utility performs a complete backup of the metadata.
- StorHouse/RM redo journaling facilities capture transactions that affect the metadata and save “after” images in journal files.
- The metadata restore utility re-creates the metadata from the last metadata backup or a specified backup file version.

- A redo journaling utility applies the transactions that were captured in the journal files since the last metadata backup or to a specified point in time.

You can start redo journaling when you create a StorHouse database. You can also enable journaling later, but you cannot stop journaling once you start it for a database. FileTek customer support, however, can disable the feature if necessary.

Journal file

A *redo journal*, or *journal file*, contains a record of each transaction that changes a table in a system tablespace. The following committed transactions are captured in a journal file:

- CREATE TABLE
- CREATE INDEX
- DROP TABLE
- DROP INDEX
- INSERT (table rows and index entries)
- UPDATE (table rows and index entries)
- DELETE (table rows and index entries)

Other forms of ALTER, CREATE, and DROP statements are not explicitly captured because the changes they cause in the system tablespace are completely recorded by the capture of the underlying INSERT, UPDATE, and DELETE statements that are generated by those DDL statements.

Primary and secondary journal files

The redo journaling utilities manage primary journal files and secondary journal files. A *primary journal file* is the disk file that is archived to StorHouse or used in the event the redo journal must be applied. A *secondary journal file* is a copy of the primary journal file and is used if the primary is corrupt. Typically, the secondary journal file is located on a different device from the primary journal file.

Journal file status

A journal file may have a status of current, cycled, archived, or purged.

- The *current journal file* is the file presently open and storing journal records. The current journal file is located on UNIX disk.
- A *cycled journal file* is a UNIX file that has been closed in preparation for archiving and deleting.
- An *archived journal file* is a UNIX file written to StorHouse as a StorHouse file. The UNIX file remains on disk until purged.
- A *purged journal file* is a UNIX file deleted from disk only after successful archiving to StorHouse.

Journal file names

The current journal files have the following naming convention:

```
dbname_currenttime.PRI.JOU  
dbname_currenttime.SEC.JOU
```

where:

- dbname is the name of the StorHouse database
- currenttime is the elapsed time in seconds since 00:00:00 Universal Time, January 1, 1970
- PRI is the primary journal file and SEC is the secondary, copy
- .JOU is the file extension

For example, CUSTOMERDB_1042555527.PRI.JOU.

On StorHouse, an archived journal file name does not contain PRI or SEC, for example, CUSTOMERDB_1042666627.JOU.

Redo journaling utilities

You use the following utilities to manage redo journaling.

Task	Utility	See page
Enable journaling when creating a database	syscreate	12-9
Enable journaling for an unjournaled database	sthdb_backup	12-10
Close the current journal file and start a new one	sthjou_cycle	12-11
Validate a journal file, archive a journal file to a StorHouse VSET and FSET, and purge an old journal file from the UNIX directory	sthjou_archive	12-12
Apply journal files to finish restoring a database	sthjou_replay	12-15
Audit a journal file	sthjou_audit	12-19
Reset the journaling environment	sthdb_backup	12-22

Journal chains

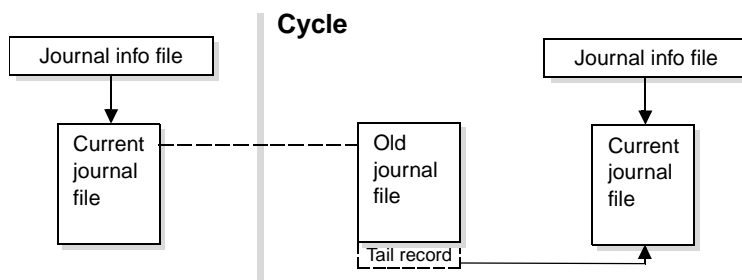
When restoring a database, the journal replay utility may replay multiple journal files—current, cycled, and archived—that are linked together. These linked journal files are called a *journal chain*. The utility can replay an entire journal chain or a partial chain.

A journal chain starts when you create a StorHouse database and enable journaling or when you enable journaling for an existing database (using the -j option on the sthdb_backup utility). In both cases, a metadata backup is performed, creating a *journal info file* in the database directory, a new journal file,

and optionally a secondary journal file. The journal info file contains a pointer to the current journal file. As journal files are cycled:

- The journal info file is updated to point to the new (current) journal file.
- A tail record is written to the old journal file with a pointer to the current journal file.

The current journal file does not have a trailer record and thus terminates the chain. The following graphic illustrates the start of a journal chain.



To use journal recovery, you must first restore the database from a metadata backup file. Then when you run journal replay, the utility determines and replays the first journal file in the chain, and from the chain, finds and replays the next journal file. If a journal file is not on disk, the utility looks for it on StorHouse. Further, the utility can use secondary journal files, if needed. For example, to replay an entire journal chain, the journal replay utility may read current and cycled journal files in the primary directory, then journal files in the secondary directory, then archived journal files on StorHouse. If no journals are lost (including those on disk which have not been archived), the replay can restore the database up to the last committed transaction prior to the failure.

Journal chains are also linked through metadata backups. For example, if the latest metadata backup file is unusable, you could use a prior backup and then the journal replay utility could apply the journal files between the two backups.

Replay checkpointing

The journal replay utility can restore up to the last committed transaction or to a *replay checkpoint*—a point in the journal chain where the replay stops. With replay checkpointing, you perform a partial replay, restoring only those transactions up to a checkpoint.

Each journal file has a creation timestamp expressed as the number of seconds and microseconds that have elapsed since time 0 (midnight January 1, 1970) in Coordinated Universal Time (UTC). You can specify the checkpoint two ways:

- By using the `-l` option to list the journal files and then select the cutoff journal file from the list. See page 12-17 for an example.
- By using the `-d` option to specify a *delta time* in days.

For example, a delta time of 1.5 is:

- 1 full (24 hours) day and 1/2 of a second day
- or 36 hours (24+12)
- or 129600 seconds (36*60*60)

To determine the checkpoint, the journal replay utility determines the current time (now) in seconds from UTC and subtracts the delta time (in seconds) from the current time.

Here's an example of how the delta time is used in replay checkpointing.

- Assume the current time is Monday, June 26 20:47:48 2006. This is 1151369268 seconds since midnight January 1, 1970 (time 0 in UTC).
- You specify a delta time of 1.5, which represents 129600 seconds.
- The current time in seconds (1151369268) minus the delta time in seconds (129600) is 1151239668.

1151239668 (seconds since time 0 UTC) represents Sunday, June 25 08:47:48 2006. Therefore, the journal replay utility replays any file whose creation timestamp is before Sunday, June 25 08:47:48 2006. When the journal replay utility encounters a file with a creation timestamp equal to or later than Sunday, June 25 08:47:48 2006, it ends with a checkpointed replay. The checkpoint will be the next unplayed journal file in the journal chain.

Ways to run redo journaling utilities

You must run the journal replay utility from the StorHouse operating system (UNIX) prompt. You can run the journal cycle and journal archive utilities in the following ways:

- StorHouse Command Language RUN command
- StorHouse Command Language SCHED ULE command
- UNIX command line (StorHouse operating system prompt)
- UNIX cron command

Privileges for running redo journaling utilities

You must use the operator account and password to run the journal replay utility from the StorHouse operating system (UNIX) prompt. A StorHouse account with one of the following command privileges can run the journal archive and journal cycle utilities:

- OPERATOR
- SERVICE
- SYSTEM

Note that the SYSADM account has ALLPRIVILEGE (which includes the above privileges). Refer to the StorHouse *Command Language Reference Manual* for more information about command privileges.

Location of the redo journal

On StorHouse, an archived journal file is stored in a VSET and FSET that you can name when running the journal archive utility (sthjou_archive). If you don't specify the VSET and FSET, the utility uses StorHouse system parameters (SQL_BKUP_VSET and SQL_BKUP_FSET) to obtain the values.

On disk, the current journal files, cycled journal files, and archived but not yet purged journal files are located in directories that you designate in the \$STHROOT/etc/rdbtemp.data file.

Caution: The disks used for journals should never be the same as the disks used for the metadata.

You add an entry to the rdbtemp.data file for each directory to use for journaling and cycling. Specifically, the rdbtemp.data file is organized into three sections:

- Temporary file directories
- Primary journal file directories
- Secondary journal file directories

The first line in a section is the number of entries in the section. Subsequent lines are a full path specification. For example, the following rdbtemp.data file contains two temporary file specifications, four primary journal file specifications, and two secondary journal file specifications.

```
2
/filetek/tmp
/filetek2/tmp

[JOURNAL_PRIMARY]
4
/home/journal/sth3.0/primary1
/home/journal/sth3.0/primary2
/home/journal/sth3.0/primary3
/home/journal/sth3.0/primary4
```



```
[ JOURNAL_SECONDARY ]  
2  
/rm/journal/sth3.0/second1  
/rm/journal/sth3.0/second2
```

When you enable journaling for a new database, StorHouse/RM creates a primary journal file in the first primary directory (for example, `/home/journal/sth3.0/primary1`) and a secondary journal file in the first secondary directory (for example, `/rm/journal/sth3.0/second1`). When you cycle the journal files, the journal cycle utility creates new files in the next directories (for example, `/home/journal/sth3.0/primary2` and `/rm/journal/sth3.0/second2`).

Locking

Journaling uses locks during physical I/O to the current journal file and during journal cycling. These locks affect only the current journal file. Locks are not used for operations on cycled and archived journal files.

Enabling journaling for a new database

When creating a StorHouse database, you can start journaling by using the `-j` or `-J` (journaling) option on the `syscreate` command. When journaling is selected, the `syscreate` utility creates the database, then it creates a backup of the new database. If you don't enable journaling, you can always enable it later. See "Enabling journaling for an unjournaled database" on page 12-10 for more information about starting journaling for an existing database.

▼ To enable journaling for a new database

1. If necessary, sign on to the StorHouse operating system (UNIX) with the operator account and password.

2. At the operating system prompt, enter the syscreate command with the -j or -J option. The syscreate format is:

```
syscreate [-j | -J] database_name
```

For example:

```
$STHROOT/bin/syscreate -j CUSTOMERDB
```

Enabling journaling for an unjournaled database

If you did not enable journaling when you created a database, you can enable it later by using the -j or -J option when running the metadata backup utility (sthdb_backup). The metadata backup utility backs up the metadata and creates the journal files in the primary and secondary directories specified in the rdbtemp.data file.

▼ To enable journaling for an unjournaled database

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the sthdb_backup command with the -j option and press **Enter**. See “Backing up metadata” on page 10-8 for the complete format of the sthdb_backup command. An example follows.

To back up a database called CUSTOMER and to enable journaling, type:

```
run sthdb_backup -j CUSTOMER
```

In this example, the metadata backup utility terminates if any loads are active (there's no -f, or -force, option on the command statement). In this case, no backup is created and journaling is not started for the database.

Cycling a redo journal

The journal cycle utility (`sthjou_cycle`) performs the following functions:

- Creates new files in the next primary and secondary directories specified in the `rdbtemp.data` file
- Writes a record containing the names of the new journal files to the outgoing current journal files
- Closes the current primary and secondary journal files in preparation for archiving

The new files then are the current journal files and the closed files are the cycled journal files. You should cycle journal files when they become large or at a preset interval (cron job).

Note: The metadata backup utility automatically cycles the current journal files.

If the `rdbtemp.data` file contains multiple primary and secondary directory entries, the journal cycle utility will switch to the next directory.

The format of the journal cycle utility is as follows:

`sthjou_cycle [options] database_name`

Argument	Description
[options]	(optional) Command options.
-v -V	Option to display the names of the current primary and secondary journal files, and when <code>sthjou_cycle</code> utility completes, the names of the newly created primary and secondary journal files.

Argument	Description
-h -H	Option to display a description of the sthjou_cycle utility and its command syntax.
database_name	(required) Name of the StorHouse database with journaling enabled. When running the utility with the StorHouse RUN or SCHEDULE command, enclose a database name with lowercase characters in double quotes.

▼ To cycle a redo journal with the RUN command

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the sthjou_cycle command and press **Enter**. For example, to cycle a journal file for the Calls database using the -v option, type:

```
run sthjou_cycle -v "Calls"
```

Archiving and purging journal files

The journal archive utility (sthjou_archive) performs the following functions:

- Verifies the integrity of a cycled primary journal file by reading each journal record. If the utility detects a problem with a primary journal file, it reads the secondary journal file.
- Archives, or copies, journal files from the UNIX directories to a VSET and FSET on StorHouse. This utility archives only those journal files that have been cycled and not yet archived to StorHouse. The utility ignores the current journal file and any disk journal file already archived to StorHouse but not yet deleted.

- Optionally purges archived journal files from disk. You can purge journal files only or archive and purge journal files at the same time.

Running the utility with no options archives every disk journal file (except the current journal file) that has not been previously archived to StorHouse.

The format of the journal archive utility command is as follows.

`sthjou_archive [options] database_name`

Argument	Description
options	(optional) Command options.
-s {sm_options} -S {sm_options}	<p>Option to specify the VSET, FSET, and VTF values for the StorHouse file to contain the journal files. If you omit this option, the utility uses StorHouse system parameters (SQL_BKUP_VSET and SQL_BKUP_FSET) to obtain the values for VSET and FSET, and it uses NEXT as the default for VTF. The format of {sm_options} is:</p> <p>VSET=name,FSET=name,VTF=NOW DIRECT NEXT</p> <p>For example:</p> <p>-s VSET=MYVSET,FSET=MYFSET,VTF=NOW</p> <p>No spaces are permitted between the items (sm_options).</p>
-p N[.nnnnn] -P N[.nnnnn]	<p>Option to purge disk journal files. Only journal files that have been successfully written to StorHouse and verified as safe on StorHouse are purged.</p> <p>N.nnnnn is the age criteria for the disk journal file, where N (required) is days and .nnnnn (optional) is a fraction of a day. The disk journal file must have been archived the specified number of days before it can be purged from disk. For example, -p 1 indicates to purge disk journal files that were archived at least one day ago.</p> <p>You can specify two additional modifiers with the -p option: -o and -c.</p>

Argument	Description
	<p>-o -O</p> <p>Option to only purge disk journal files that have been previously written to StorHouse. No archiving occurs. The utility verifies that a copy of the file exists on StorHouse before it erases the file on disk. This option is valid only if you specify the -p option.</p>
	<p>-c N -C N</p> <p>Option to specify the number of journal file copies that must be stored on StorHouse before the utility deletes the journal files on disk. The N value may be 0, 1, or 2. The utility issues a StorHouse SHOW FILE /SAFE_COPIES command to verify that the number of safe copies exists and are complete and usable. When the number of safe copies exists, the utility purges the disk journal files.</p> <p>The default value is 2. The value 0 (not recommended) indicates at least one copy exists in the performance buffer. This option is valid only if you specify the -p option.</p>
database_name	<p>(required) Name of the StorHouse database with journaling enabled. When running the utility with the StorHouse RUN or SCHEDULE command, enclose a database name with lowercase characters in double quotes.</p>

▼ To archive and/or purge a redo journal with the RUN command

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the sthjou_archive command and press **Enter**. Some examples follow.
 - To archive journal files to a VSET called v2003 and an FSET called f2003, for a database named CUSTOMERDB, type:

```
run sthjou_archive -s vset=v2003,fset=f2003 CUSTOMERDB
```

- To archive journal files to the default VSET and FSET (omit the -s option), to purge journal files that were archived at least seven days ago and have one copy on StorHouse, and to display messages, type:

```
run sthjou_archive -p 7 -c 1 -v CUSTOMERDB
```

The following example illustrates the messages generated for the -v option.

```
Archive file Vfiletek/tmp1/CUSTOMERDB_1114702660.PRI.JOU\  
Vfiletek/tmp1/CUSTOMERDB_1114702660.PRI.JOU\ successfully  
archived to SM.
```

```
Primary Log file Vfiletek/tmp1/CUSTOMERDB_1114702660.PRI.JOU\  
purged.
```

```
Secondary log file Vfiletek2/tmp2/  
CUSTOMERDB_1114702660.SEC.JOU\ purged.
```

```
total files archived \1\  
total files purged   \2\  
total primary files purged \1\  
total secondary files purged \1\  
Wall clock \40.11\  
User      \15.66\  
SYS       \6.93\  
sthdb_archive has completed successfully
```

- To purge journal files that were archived at least five days ago, type:

```
run sthjou_archive -p 5 -O CUSTOMERDB
```

Replaying a journal file

The journal replay utility (sthjou_replay) applies journal files created since the last metadata backup or to a specified checkpoint. You run the journal replay

utility after restoring the database with the metadata restore (sthdb_retore) utility. A database with journaling enabled that has been restored is inaccessible until you run the journal replay utility. The database must be down during both operations. You must run this utility at the StorHouse operating system (UNIX) prompt.

Note: Contact FileTek customer support before restoring a database and replaying a redo journal.

The format of the journal replay utility command is as follows.

sthjou_replay [options] database_name

Argument	Description
[options]	(optional) Command options.
-v -V	Option to print replay statistics to the standard output device at the end of the replay.
-d N[.nnnnn] -D N[.nnnnn]	Option to specify a checkpoint (delta time) to perform a partial replay. N.nnnnn is the age criteria, where N (required) is days and .nnnnn (optional) is a fraction of a day. For example, -d 1.5 replays all of the transactions except for the last day and a half. An alternative to specifying a delta time is to use the -l option to list the journal files in the chain.
-l -L	Option to list the entire journal file chain in order to select a journal file as a cutoff for replay. The list also identifies the creation timestamp and whether a journal file resides on disk or on StorHouse. The sthjou_replay utility will replay up to and including the selected journal file. The -l option is an alternative to the -d option.
-h -H	Option to display a description of the sthjou_replay utility and its command syntax.
database_name	(required) Name of the StorHouse database with journaling enabled.

▼ To replay a redo journal

1. If necessary, sign on to the StorHouse operating system (UNIX) with the operator account and password.
2. At the operating system prompt, submit the sthjou_replay command with the desired option and press **Enter**. Some examples follow.
 - To list the journal files in order to select one as a checkpoint for a partial replay of the Calls database, type:

```
$STHROOT/bin/sthjou_replay -l Calls
```

In the following example listing, six journal files (three archived to StorHouse and three on disk) are available for replay. The fourth journal file is selected as the cutoff for replay (s 4). The journal replay utility will replay up to and including that journal file and then stop.

```
1. SM FILE : Calls_1151345744.PRI.JOU  
creation date: Mon Jun 26, 2006 14:15:44
```

```
2. SM FILE : Calls_1151345788.PRI.JOU  
creation date: Mon Jun 26, 2006 14:16:28
```

```
3. SM FILE : Calls_1151345801.PRI.JOU  
creation date: Mon Jun 26, 2006 14:16:41
```

```
4. DISK FILE : Calls_1151345899.PRI.JOU  
creation date: Mon Jun 26, 2006 14:18:19
```

```
5. DISK FILE : Calls_1151345981.PRI.JOU  
creation date: Mon Jun 26, 2006 14:19:41
```

```
6. DISK FILE : Calls_1151346026.PRI.JOU  
creation date: Mon Jun 26, 2006 14:20:26
```

```
(S)elect, (N)ext, (Q)uit
```

```
s 4
```

>selected 4

4. DISK FILE : Calls_1151345899.PRI.JOU
creation date: Mon Jun 26, 2006 14:18:19

Commencing Redo on select: 4.

STHJOU_REPLAY: Journal checkpointed:
Checkpointed at journal File: Calls_1151345981.PRI.JOU
Checkpoint journal creation timestamp : Mon Jun 26 14:19:41 2006

Wall clock \194.85\
User \0.78\
SYS \0.43\

STHJOU_REPLAY finished with warning: -85101: Warning: Partial
checkpointed replay completion success.

Time at completion: Mon Jun 26 14:26:42 2006

- To replay the entire journal chain with the verbose option for the Calls database, type:

\$STHROOT/bin/sthjou_replay -v Calls

The following sample illustrates output for the verbose option. Only the final four lines are printed without the -v option.

total redo journal files processed \1\
total records processed \988\
total statement records processed \987\
total statements \7210 \
total tables created \0\
total tables dropped \0\
total table inserts \3032\
total table updates \0\
total table deletes \0\
total indexes created \0\

```
total indexes dropped \0\  
total index inserts  \3196\  
total index deletes  \0\  
total config updates \491\  
total committed transactions \491\  
total transaction rollbacks \0\
```

```
Wall clock \33.31\  
User      \1.61\  
SYS       \0.66\  
sthdb_replay has completed successfully
```

Auditing a redo journal

The journal audit utility (sthjou_audit) verifies that a journal chain is readable. By default, the utility verifies journal files created since the most recent metadata backup. However, you can specify a metadata backup file version number to audit journal files further back in time. You can also limit the number of journal files audited.

The journal audit utility reads each journal record and performs a cyclic redundancy check (CRC) to check for errors in stored data. If requested, the journal audit utility also examines each statement (for example, INSERTs) in a journal record and prints a detailed accounting of the statements.

The format of the journal audit utility command is as follows.

```
sthjou_audit [options] database_name[:n]
```

Argument	Description
[options]	(optional) Command options.
-f -F	Option to verify each statement in a journal chain and to print the number and type of statements encountered as well as the number of files processed. In addition, the -f option provides more information about the progress of the audit as it proceeds through the journal chain.
-l N -L N	Option to limit the number of journal files audited. The N must be greater than or equal to 1. A negative value or 0 is invalid and generates an error. For example, -l 15 audits the first 15 journal files in a journal chain.
-h -H	Option to display a description of the sthjou_audit utility and its command syntax.
database_name	(required) Name of the StorHouse database with journaling enabled.
[:n]	(optional) File version number to audit journal files from the current one through a later metadata backup. Version 0 is the default and the most recent metadata backup file version. The next most recent version is -1. For example, Calls:-1 audits journal files from the current one, through the most recent (0 version) backup, back to the -1 backup.

▼ To audit a redo journal

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the sthjou_audit command and press **Enter**. Some examples follow.

- To perform a detailed audit of all the journal files in a chain for the Calls database to the most recent metadata backup, type:

```
$STHROOT/bin/sthjou_audit -f Calls
```

The following sample illustrates the output for the -f option.

```
Audit \rm8\rchaddoc_tmp\sth2.0\tmp_primary4\Calls_1151358485.PRI.JOU\  
Audit complete of \rm8\rchaddoc_tmp\sth2.0\tmp_primary4/  
Calls_1151358485.PRI.JOU\  

```

```
Audit \rm8\rchaddoc_tmp\sth2.0\tmp_primary4\Calls_1151358777.PRI.JOU\  
Audit complete of \rm8\rchaddoc_tmp\sth2.0\tmp_primary4/  
Calls_1151358777.PRI.JOU\  

```

```
Audit \rm8\rchaddoc_tmp\sth2.0\tmp_primary1\Calls_1151358905.PRI.JOU\  
Audit complete of \rm8\rchaddoc_tmp\sth2.0\tmp_primary1/  
Calls_1151358905.PRI.JOU\  

```

```
Audit \rm8\rchaddoc_tmp\sth2.0\tmp_primary4\Calls_1151358921.PRI.JOU\  
Audit complete of \rm8\rchaddoc_tmp\sth2.0\tmp_primary4/  
Calls_1151358921.PRI.JOU\  

```

```
Audit \rm8\rchaddoc_tmp\sth2.0\tmp_primary2\Calls_1151359382.PRI.JOU\  
Audit complete on ///rm8\rchaddoc_tmp\sth2.0\tmp_primary2/  
Calls_1151359382.PRI.JOU\  

```

```
total redo journal files processed \4\  
total records processed \1013\  
total statement records processed \1005\  
total statements \7313 \  
total tables created \1\  
total tables dropped \0\  
total table inserts \3071\  
total table updates \0\  
total table deletes \0\  
total indexes created \1\  
total indexes dropped \0\  
total index inserts \3240\  
total index deletes \0\  
total config updates \500\  

```

```
total committed transactions \500\  
total transaction rollbacks \0\
```

```
Wall clock \2.06\  
User      \0.24\  
SYS       \0.15\  
sthdb_audit has completed successfully
```

- To perform a standard journal audit through the -1 metadata backup file version for the CALLS database and to limit the audit to the first 25 journal files in the chain, type:

```
$sthjou_audit -l 25 CALLS:-1
```

Resetting the journaling environment

The -r option on the metadata backup utility command enables you to reinitialize the journaling environment. This option creates a new journal info file and starts a new journal chain of journal files. Resetting the journaling environment may be useful in cases where journaling was enabled and a partial, or checkpointed, replay was performed.

For example, the following command reinitializes the journaling environment for the CALLS database.

```
run sthdb_backup -r CALLS
```

See Chapter 10, “Metadata backup,” for more information about the metadata backup utility.

Explain facility

This chapter describes the StorHouse/RM explain facility and describes how to:

- Access the StorHouse/Admin ISQL window
- Create explain tables
- Run the explain facility
- Obtain an execution plan ID
- Display a query in an execution plan
- Display an execution plan
- Display expressions in an execution plan
- Display operators in an execution plan

About the explain facility

The StorHouse/RM *explain facility* enables you to examine the strategy, or *execution plan*, implemented by the StorHouse/RM optimizer for a given query. For instance, you can determine whether StorHouse/RM uses an index and which index. Or you can determine the chosen join method and join predicate. You can use the explain facility to analyze SELECT statements only.

Execution tree

An *execution tree* is an internal representation of a query in a form that an engine can process. When you submit an SQL query:

- The *parser* parses the SQL statement, generating an internal execution tree.
- The *optimizer* manipulates and refines the tree, for instance, selects indexes, determines an optimal join order when multiple tables participate in the query, and restructures the execution tree for performance.
- The *engine* then executes the optimized execution tree.

The explain facility enables you to create a representation of the execution tree in the form of relational tables. You can query these tables to view the execution strategies selected by the optimizer.

Nodes

An execution tree is structured as a binary tree. Each *node* in the tree represents a primitive operation in the execution of the query. And, each node in the tree may have one child, two children, or no children. Child nodes are referred to as either left or right child nodes. If a node has one child, that child node is always a left child. A right child node exists only if there are two child nodes.

There are three categories of nodes:

- The *root node* is the entry point into the execution tree. The root node has no parent.
- An *interior node* is a node that has both a parent and one or more children.
- A *leaf node* is a node that has a parent, but no children. Leaf nodes represent operations on base tables.

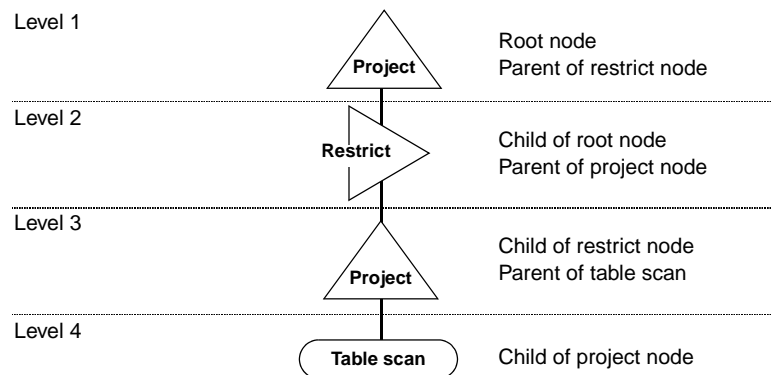
A node requests a result row or rows from its children, manipulates the data, and passes the result to its parent. Execution of the tree begins when the root node is requested to return a row of the final result set. Data (from stored tables) enters the tree only from leaf nodes.

Node operations

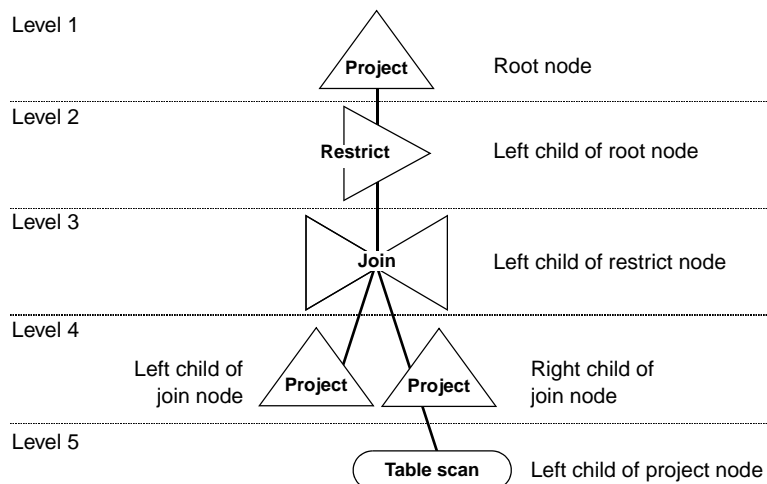
There are six types of nodes, one for each of the six primitive data operations:

- Project
- Restrict
- Join
- Sort
- Union
- Table scan
- Index scan

The root node is always a project operation. An interior node may be a project, restrict, join, sort, or union operation. A leaf node is either a table scan or an index scan operation. The following figure illustrates node operations and parent and child nodes. This execution tree is typical of a query that has a predicate.



The following figure illustrates node operations and a left and right child node of a join node.



Project. A *project node* selects result set *columns* by eliminating or aggregating the columns in its input data. The final result maps to the list of columns in the query result set. The root node of an execution tree is always a project node. A project node has only one child.

Restrict. A *restrict node* selects result set *rows* by eliminating rows from its input data that do not pass a test. For example, in the query `SELECT col1, col2 FROM tbl WHERE col1='x'`, the `WHERE` clause contains a predicate that identifies rows. The restrict node for this example returns a result row only when an input row passes the test (`col1 = 'x'`). A restrict node has only one child.

Join. A *join node* combines the rows from its two child nodes. The fundamental operation used to combine rows is the Cartesian Product, where each row from the left child is combined with *every* row from the right child. Conditional tests can be used to limit the number of rows that are combined by providing a matching condition for the left and right rows. Such a condition is created by predicates in `WHERE` or `ON` clauses that reference columns from two tables.

Joins also can specify the manner of combining rows (inner- or outer-join) and the algorithm employed: nested loop join (uses no index), augmented nested loop (requires an index on the join columns), and hybrid IN, or merge join (requires a value index on the join columns). A join node always has both a left and a right child.

Sort. A *sort node* orders data. An execution plan contains a sort node when the query contains an ORDER BY clause, a GROUP BY clause, or a DISTINCT operator.

Union. A *union node* selects all the rows from its left child node, followed by all or all unique rows from its right child node. An execution plan contains a union node when the query contains a UNION or UNION ALL set operator. A union node always has both a left and a right child.

Table scan. A *table scan node* selects all columns from all rows of one table by reading from the physical data store. The optimizer may choose a table scan when no appropriate index is present or when an index scan is more costly than a table scan. A table scan node is always a leaf node.

Index scan. An *index scan node* uses an index to select all columns of specific rows from one table. An index scan node is always a leaf node.

Expression tree

An *expression tree* is another tree structure that is associated with the execution tree. Expression trees represent relational operators, arithmetic operators, functions, columns, constants, and parameter references. An expression tree can consist of a single node representing a column or it can be a complex tree with many nodes representing a complex predicate. An expression tree is associated with project, restrict, join, and index scan nodes.

With the explain facility, you can list the expressions and operators associated with a node. The following expressions and operators may appear in the explain result data.

Expressions and operators in an expression tree

Type	Description
Logical operators	Logical operators in restrict and join nodes: AND, OR, NOT
Relational operators	<ul style="list-style-type: none">■ Relational operators in restrict nodes and in join predicates: =, <>, <, >, <=, >=■ Relational operators in restrict nodes only: IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE■ Relational operators in join predicates only: EXISTS, NOT EXISTS
Arithmetic operators	+, -, *, /
Aggregate functions	MAX, MIN, COUNT, SUM, and AVG
Scalar functions	ABS, TO_CHAR, SUBSTR, and so on
Column references	Column name reference in a table
Constants	Constant value (literal)
Parameters	Host variable parameter

Explain facility SQL statements

You use the explain facility by submitting a set of SQL statements and querying the result tables. The SQL statements for the explain facility are as follows.

Explain SQL statements

Statement	Description
CREATE EXPLAIN TABLES	Create a set of empty explain tables
DROP EXPLAIN TABLES	Drop a set of explain tables
EXPLAIN PLAN	Specify the query to be explained and run the explain facility to populate the explain tables
SELECT	Query the explain tables

You can submit the explain facility SQL statements using any method supported by StorHouse. See “Submitting StorHouse SQL statements” on page 3-9 for more information about ways to submit StorHouse SQL statements. This chapter explains how to submit the statements using the ISQL facility in StorHouse/Admin.

Explain facility result tables

The explain facility populates the explain tables that you create. You can query these tables to examine the execution plan implemented by the StorHouse/RM optimizer. INSERT, UPDATE, and DELETE statements should not be performed on the explain facility result tables.

The following table briefly describes the StorHouse explain tables. See Appendix D, “StorHouse explain tables” for descriptions of these tables.

Explain facility result tables

Table name	Contains
STH_EXPLAIN_ID	Statement ID that you specify on the EXPLAIN PLAN statement and the associated execution plan ID generated by StorHouse/RM
STH_EXPLAIN_PLAN	Description of the nodes in an execution plan
STH_EXPLAIN_STMT	Query (SELECT statement) specified in your EXPLAIN PLAN statement
STH_EXPLAIN_EXPR	Description of aggregate functions, scalar functions, columns, constants, or parameters in an expression tree
STH_EXPLAIN_OPR	Description of the relational, logical, and arithmetic operators in an expression tree

Explain privileges

At a minimum, you must have the RESOURCE database privilege to create and drop your own explain tables (submit CREATE EXPLAN TABLES and DROP EXPLAN TABLES statements). If you omit the owner name on the statement, the account ID you use to log in to the StorHouse database is the default owner.

An account with the DBA database privilege can create and drop explain tables for other accounts. Only the owner of the explain tables, however, can submit the EXPLAIN PLAN statement. For example, if the SYSADM account creates a set of explain tables for an account named SAC, then only SAC can issue the EXPLAIN PLAN statement for those tables. An account with DBA however, can query all explain result tables.

Explain examples

This section takes you step-by-step through the process of running the explain facility to analyze execution plans for queries. The examples are as follows:

- Example 1: Query without a predicate
- Example 2: Query with a simple predicate
- Example 3: Join without a predicate
- Example 4: Query with a complex predicate
- Example 5: Query with a function
- Example 6: Query that uses an index
- Example 7: Join with a complex predicate

Example 1: Query without a predicate

This example analyzes a query on a user table called CUSTOMERS owned by USER1. No indexes are defined. The table columns are CUSTOMERNO, CUSTOMERNAME, STARTDATE, and STATUS. Assume you, USER1, submit all statements. The query in this example is as follows:

```
SELECT CUSTOMERNO, STATUS FROM CUSTOMERS;
```

Step 1: Run the explain facility. Create the explain tables and then submit an EXPLAIN PLAN statement to specify the query to be explained and to populate the explain tables with information about the query. You can specify a statement ID (up to 32 characters) on the EXPLAIN PLAN statement to identify the execution plan. In the example, however, the STMT_ID clause is omitted. The default statement ID is STH_EXPLAIN_DEFAULT.

```
CREATE EXPLAIN TABLES;
```

```
EXPLAIN PLAN FOR SELECT CUSTOMERNO, STATUS  
FROM CUSTOMERS;
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM. You can use this ID to query the other explain tables.

```
SELECT * FROM STH_EXPLAIN_ID
WHERE STMT_ID='STH_EXPLAIN_DEFAULT';
```

The result table is as follows. In this example, the execution plan ID is 1.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
STH_EXPLAIN_DEFAULT	10/03/2003 16:31:44.000000	1

Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan. It's helpful to order the result table by level (LVL).

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. This example execution plan consists of three nodes. The table scan node indicates an index is not used to execute the query.

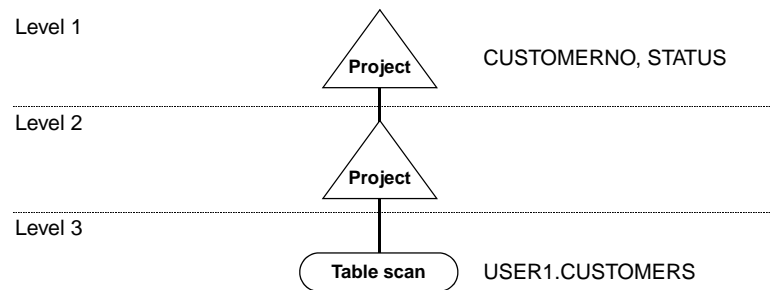
ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	----	-----	---	--	-----
1	12		1	L	PROJECT
1	9	12	2	L	PROJECT
1	10	9	3	L	TABLE SCAN OF USER1.CUSTOMERS

See “STH_EXPLAIN_PLAN” on page D-3 for descriptions of the fields. In summary:

- ID is the execution plan ID.
- NODE is the node number assigned to the node.
- PAR_NODE identifies the node number of the parent node.
- LVL is the node's level in the execution tree. Level 1 is the root node.

- `LR` indicates whether the node is a left child (`L`) or a right child (`R`).
- `EXPLAIN_PLAN` identifies the node operation and any additional information, such as a table name, index name, or join method.

The tree representation of this execution plan looks like this:



Step 4: Display the expression for the project node. Query the expressions for the project nodes. These expressions are located in the `STH_EXPLAIN_EXPR` table. Use the `ID` and `NODE` values in the `STH_EXPLAIN_PLAN` table to display the expressions for a specific node. The following query, for example, displays the `STH_EXPLAIN_EXPR` table for `ID 1` and `NODE 12` (the root node). These entries represent the columns that are projected by `NODE 12`.

```

SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID =1 AND ASSOC_NODE = 12
ORDER BY ENTRY;

```

The result table is as follows. See “STH_EXPLAIN_EXPR” on page D-5 for descriptions of the fields. The `PROJ_T` column indicates the project type, in this example, a column reference (`col`). The `PROJECT_COLUMN` then identifies the column names that are projected up the execution tree.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
---	-----	-----	----	-----	-----
1	12	1	0	col	CUSTOMERS.CUSTOMERNO
1	12	2	1	col	CUSTOMERS.STATUS

Example 2: Query with a predicate

This example uses the same user table as example 1 (`USER1.CUSTOMERS`) to illustrate multiple records in the `STH_EXPLAIN_ID` system table. Assume you, `USER1`, have performed the steps in example 1 and are now going to analyze a query with a predicate. You do not create a new set of explain tables. The query in this example is as follows:

```
SELECT * FROM CUSTOMERS WHERE CUSTOMERNO='1003';
```

Step 1: Run the explain facility. Submit the `EXPLAIN PLAN` statement, setting the statement ID to `EXAMPLE2`.

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE2'
FOR SELECT * FROM CUSTOMERS WHERE CUSTOMERNO='1003';
```

Step 2: Obtain the execution plan ID. Query the `STH_EXPLAIN_ID` table to obtain the execution plan ID. StorHouse/RM generates this ID by counting the number of rows in the `STH_EXPLAIN_ID` table and adding 1 to the value. When you drop your explain tables, the execution plan ID starts at 1.

```
SELECT * FROM STH_EXPLAIN_ID;
```

The result table is as follows. In this example, the execution plan ID for the `EXAMPLE2` statement ID is 2.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
STH_EXPLAIN_DEFAULT	10/03/2003 16:31:44.000000	1
EXAMPLE2	10/03/2003 16:33:48.000000	2

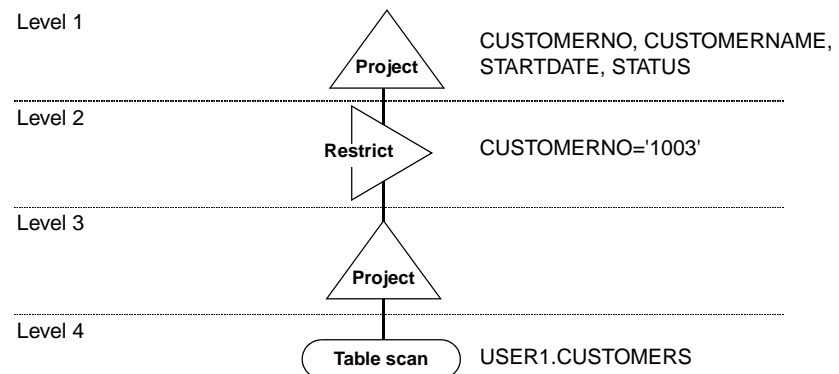
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=2 ORDER BY LVL;
```

The result table is as follows. The addition of the predicate generates a restrict node in the execution tree.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	----	-----	---	--	-----
2	23		1	L	PROJECT
2	7	23	2	L	RESTRICT
2	20	7	3	L	PROJECT
2	21	20	4	L	TABLE SCAN OF USER1.CUSTOMERS

The tree representation of this execution plan looks like this:



Step 4: Display the operator for the restrict node. Query the STH_EXPLAIN_OPR table to display the details of the operator in the predicate. The restrict node (NODE 7) is based on an operator.

```
SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=2 AND ASSOC_NODE=7;
```

The result table is as follows. The LEFT_SIDE and RIGHT_SIDE columns identify entries in the STH_EXPLAIN_EXPR table. StorHouse/RM assigns NODE number 6 to this operator.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TB	LEFT_SIDE	RIGHT_SIDE
---	-----	-----	----	-----	---	-----	-----
2	7	1	6	EQ	E	1	2

Step 5: Display the expressions for the restrict node. Query the STH_EXPLAIN_EXPR table to display the expressions associated with the operator (NODE 6).

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=2 AND ASSOC_NODE=6
ORDER BY ENTRY;
```

The result table is as follows. ENTRY 1 is the LEFT_SIDE in the STH_EXPLAIN_OPR table and represents a column (CUSTOMERNO). ENTRY 2 is the RIGHT_SIDE and represents a constant value (1003). The single row in the STH_EXPLAIN_OPR table along with the two rows in the STH_EXPLAIN_EXPR table represent the WHERE clause in the query.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
--	-----	-----	----	-----	-----
2	6	1	0	col	CUSTOMERS.CUSTOMERNO
2	6	2	5	const	'1003'

Example 3: Join without a predicate

This example analyzes a join of two user tables called INVENTORY and PACKAGES. Both tables are owned by USER2. The INVENTORY table columns are TAGNUM, COMPID, EMPNUM, and LOCATION. The PACKAGES table columns are TAGNUM, COMPID, EMPNUM, and PACKID. A value index is defined for the INVENTORY table on columns TAGNUM, COMPID, and EMPNUM. Assume you, USER2, submit all statements. The query in this example is as follows:

```
SELECT * FROM INVENTORY JOIN PACKAGES
ON INVENTORY.TAGNUM=PACKAGES.TAGNUM;
```

Step 1: Run the explain facility. Create the explain tables and then submit the EXPLAIN PLAN statement.

```
CREATE EXPLAIN TABLES;
```

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE3'
FOR SELECT * FROM INVENTORY JOIN PACKAGES
ON INVENTORY.TAGNUM=PACKAGES.TAGNUM;
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM.

```
SELECT * FROM STH_EXPLAIN_ID;
```

The result table is as follows. In this example, the execution plan ID for EXAMPLE3 is 1.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
EXAMPLE3	10/03/2003 16:31:56.000000	1

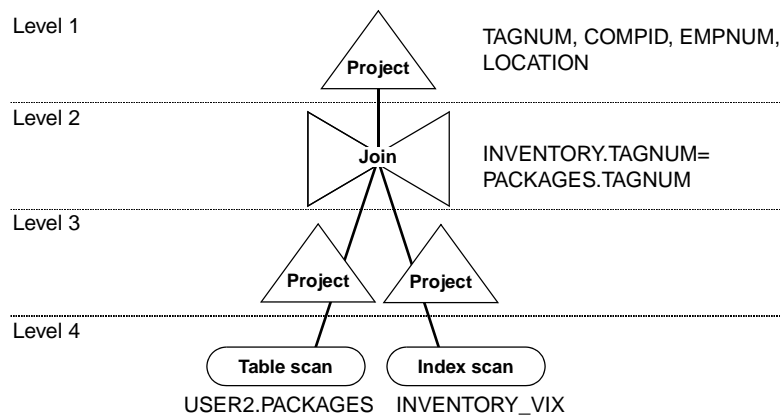
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. The join node indicates the optimizer chose a merge join.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
1	41		1	L	PROJECT
1	39	41	2	L	JOIN[MERGE-JOIN]
1	30	39	3	R	PROJECT
1	36	39	3	L	PROJECT
1	37	30	4	L	INDEX SCAN OF USER2.INVENTORY INDEX INVENTORY_VIX
1	38	36	4	L	TABLE SCAN OF USER2.PACKAGES

The tree representation of this execution plan looks like this:



Step 4: Display the join predicate. Query the STH_EXPLAIN_OPR table to display the details of the join predicate.

```
SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=1 AND ASSOC_NODE=39;
```

The result table is as follows. The LEFT_SIDE and RIGHT_SIDE columns identify entries in the STH_EXPLAIN_EXPR table. The E value under TBL indicates the STH_EXPLAIN_EXPR table.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TBL	LEFT_SIDE	RIGHT_SIDE
1	39	0	39	EQ	E	0-1	2

Step 5: Display the expressions for the join predicate. Query the STH_EXPLAIN_EXPR table to display the expressions that participate in the join predicate (NODE 39).

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=39
ORDER BY ENTRY;
```

The result table is as follows. ENTRY 1 is the LEFT_SIDE in the STH_EXPLAIN_OPR table and ENTRY 2 is the RIGHT_SIDE. The entry from the STH_EXPLAIN_OPR table combined with the two entries in the STH_EXPLAIN_EXPR table represent the join condition (INVENTORY.TAGNUM=PACKAGES.TAGNUM).

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
1	39	1	8	col	PACKAGES.TAGNUM
1	39	2	4	col	INVENTORY.TAGNUM

Example 4: Query with a complex predicate

This example analyzes a query with a complex predicate on a user table called HR owned by USER3. The HR table columns are LOCNUM, DEPTID, EMPNUM, and EMPMGR. No index is defined. (To see an example that analyzes the same query but uses an index, see “Example 6: Query that uses an index” on page 13-24.) Assume you, USER3, submit all statements. The query in this example is as follows:

```
SELECT * FROM HR
WHERE LOCNUM='32808' AND DEPTID='M759' AND EMPNUM='611';
```

Step 1: Run the explain facility. Create the explain tables and then submit the EXPLAIN PLAN statement.

```
CREATE EXPLAIN TABLES;

EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE4'
FOR SELECT * FROM HR
WHERE LOCNUM='32808' AND DEPTID='M759' AND EMPNUM='611';
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM.

```
SELECT * FROM STH_EXPLAIN_ID
WHERE STMT_ID = 'EXAMPLE4';
```

The result table is as follows. In this example, the execution plan ID is 1

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
EXAMPLE4	10/03/2003 16:32:02.000000	1

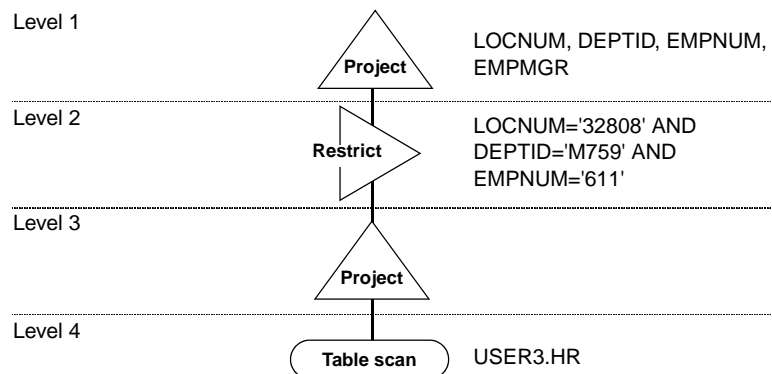
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. The restrict node has an associated expression tree that represents the complex predicate. You can query the STH_EXPLAIN_OPR table to display the expression tree. The NODE number for the restrict node is 15, so the ASSOC_NODE in the STH_EXPLAIN_OPR table is 15.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	---	-----	---	--	-----
1	33		1	L	PROJECT
1	15	33	2	L	RESTRICT
1	30	15	3	L	PROJECT
1	31	30	4	L	TABLE SCAN OF USER3.HR

The tree representation of this execution plan looks like this:



Step 4: Display the operators for the restrict node. Query the STH_EXPLAIN_OPR table to display the details of the operators in the predicate. The restrict node (NODE 15) is based on these operators. It's helpful to specify

ORDER BY ENTRY since the query is a complex predicate and contains multiple operators.

```
SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=1 AND ASSOC_NODE=15
ORDER BY ENTRY;
```

The result table is as follows. The **O** value (for operator) in the **TBL** column indicates the **LEFT_SIDE** and **RIGHT_SIDE** columns refer to entries in the **STH_EXPLAIN_OPR** table. The **E** value (for expression) in the **TBL** column indicates those columns refer to entries in the **STH_EXPLAIN_EXPR** table.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TB	LEFT_SIDE	RIGHT_SIDE
--	-----	-----	----	-----	--	-----	-----
1	15	1	37	AND	O	2	3
1	15	2	6	EQ	E	1	2
1	15	3	36	AND	O	4	5
1	15	4	9	EQ	E	1	2
1	15	5	13	EQ	E	1	2

Note the following:

- ENTRY 1, in the **LEFT_SIDE** column contains 2. This value refers to ENTRY 2 in the **STH_EXPLAIN_OPR** table.
- The **RIGHT_SIDE** column for ENTRY 1 contains 3. This value refers to ENTRY 3 in the **STH_EXPLAIN_OPR** table.
- ENTRY 3, in the **LEFT_SIDE** column contains 4 and the **RIGHT_SIDE** column contains 5. Those values refer to ENTRY 4 and 5 in the **STH_EXPLAIN_OPR** table.

- ENTRY 2, 4, and 5 contain EQ, which indicates an equality operator. The TBL column contains E, which indicates the LEFT_SIDE and RIGHT_SIDE columns represent entries in the STH_EXPLAIN_EXPR table.

Step 5: Display the expressions for the restrict node. Query the STH_EXPLAIN_EXPR table to display the expressions associated with the operators. The NODE numbers are 6, 9 and 13. For example, to display the expressions for ENTRY 5, NODE 13 in the STH_EXPLAIN_OPR table:

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=13
ORDER BY ENTRY;
```

The result table is as follows. ENTRY 1 is the LEFT_SIDE in the STH_EXPLAIN_OPR table and ENTRY 2 is the RIGHT_SIDE. ENTRY 1 represents a column (HR.EMPNUM) and ENTRY 2 represents a constant value ('611'). The single row in the STH_EXPLAIN_OPR table along with the two rows in the STH_EXPLAIN_EXPR table represent the WHERE clause in the query (WHERE EMPNUM='611').

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
--	-----	-----	----	-----	-----
1	13	1	2	col	HR.EMPNUM
1	13	2	12	const	'611'

Example 5: Query with a function

This example shows how the explain facility represents the TO_HEX function in an expression tree. The user table is CDW3V, owned by USER4. The CDW3V table columns are flag, bill_num, from_num, dialed, acct_num, elapsed_time, and switch_data. No indexes are defined. Assume you, USER4, execute the SQL statements. The query is as follows:

```
SELECT TO_HEX(DIALED) FROM CDW3V;
```

Step 1: Run the explain facility. Create the explain tables and then submit the EXPLAIN PLAN statement.

```
CREATE EXPLAIN TABLES;
```

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE5'
FOR SELECT TO_HEX(DIALED) FROM CDW3V;
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM.

```
SELECT * FROM STH_EXPLAIN_ID;
```

The result table is as follows. In this example, the execution plan ID is 1.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
EXAMPLE5	10/03/2003 16:32:08.000000	1

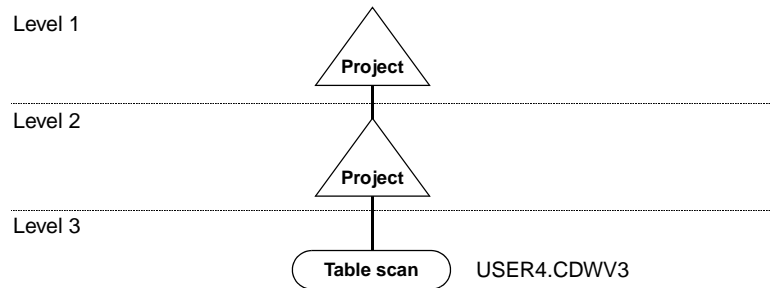
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. To view the representation of the TO_HEX function, you need to look at the list of sub-expressions associated with the root project node.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	----	-----	---	--	-----
1	10		1	L	PROJECT
1	7	10	2	L	PROJECT
1	8	7	3	L	TABLE SCAN OF USER4.CDW3V

The tree representation of this execution plan looks like this:



Step 4: Display the expression for the root project node. Query the STH_EXPLAIN_EXPR table to display the details of the root project node. The NODE number is 10.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=10
ORDER BY ENTRY;
```

The result table is as follows. The one entry represents the scalar TO_HEX function.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
---	-----	-----	----	-----	-----
1	10	1	1	sfunc	function to_hex(1)

Step 5: Display the argument for the function. Query the STH_EXPLAIN_EXPR table again to find the row that represents the argument for the TO_HEX function. You use NODE number 1 as the ASSOC_NODE in the query.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID= 1 AND ASSOC_NODE=1
ORDER BY ENTRY;
```

The result table is as follows. This table identifies the column that serves as the argument for the TO_HEX function.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
---	-----	-----	----	-----	-----
1	1	1	0	col	CDW3V.DIALED

Example 6: Query that uses an index

This example analyzes a query that uses an index. The user table, USER3.HR, is the same one used for example 4; however, a value index is defined on columns LOCNUM and DEPTID. The explain tables are dropped as part of this example. Assume USER3 executes the SQL statements. The query is as follows:

```
SELECT * FROM HR
WHERE LOCNUM='32808' AND DEPTID='M759' AND EMPNUM='611';
```

Step 1: Run the explain facility. Drop the old explain tables, create new explain tables, and then submit the EXPLAIN PLAN statement.

```
DROP EXPLAIN TABLES;
```

```
CREATE EXPLAIN TABLES;
```

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE6'
FOR SELECT * FROM HR
WHERE LOCNUM='32808' AND DEPTID='M759' AND EMPNUM='611';
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM.

```
SELECT * FROM STH_EXPLAIN_ID;
```

The result table is as follows. In this example, the execution plan ID is 1.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
EXAMPLE6	10/03/2003 16:32:28.000000	1

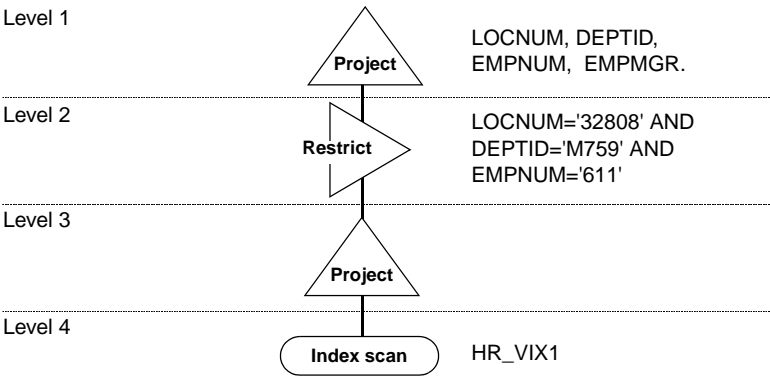
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. Note that level 4 is an INDEX SCAN and the index used is HR_VIX1.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	----	-----	---	--	-----
1	33		1	L	PROJECT
1	15	33	2	L	RESTRICT
1	30	15	3	L	PROJECT
1	31	30	4	L	INDEX SCAN OF USER3.HR INDEX HR_VIX1

The tree representation of this execution plan looks like this:



Step 4: Display the operators of the index scan node. Query the STH_EXPLAIN_OPR table to verify that both columns are used in the index predicate. In this example, the NODE number of the index scan node is 31, so the ASSOC_NODE used in the query is 31.

```
SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=1 AND ASSOC_NODE=31
ORDER BY ENTRY;
```

The result table is as follows. The E in the TBL (or TB) column means that for both rows the LEFT_SIDE and RIGHT_SIDE column represents entries in the STH_EXPLAIN_EXPR table.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TB	LEFT_SIDE	RIGHT_SIDE
1	31	1	38	EQ	E	1	2
1	31	2	41	EQ	E	1	2

Step 5: Display the expressions of the index scan node. Query the STH_EXPLAIN_EXPR table to display the details of the expressions. The NODE

numbers are 38 and 41. For example, the following query displays the details for ENTRY 1, NODE 38.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=38
ORDER BY ENTRY;
```

The result table is as follows. These rows represent the first component of the compound predicate `LOCNUM= '32808'`.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
--	-----	-----	----	-----	-----
1	38	1	59	col	HR.LOCNUM
1	38	2	40	const	'32808'

The following query displays the details for ENTRY 2, NODE 41 for the index scan node.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=41
ORDER BY ENTRY;
```

The result table is as follows. These rows represent the second component of the compound predicate `DEPTID= 'M759'`.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
--	-----	-----	----	-----	-----
1	41	1	60	col	HR.DEPTID
1	41	2	43	const	'M759'

Step 6: Display the operators for the restrict node. Query the `STH_EXPLAIN_OPR` table to display the details of the restrict node. The `NODE` number is 15.

```
SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=1 AND ASSOC_NODE=15
ORDER BY ENTRY;
```

The result table is as follows. The **E** in the **TBL** (or **TB**) column indicates the **LEFT_SIDE** and **RIGHT_SIDE** columns represent entries in the **STH_EXPLAIN_EXPR** table.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TB	LEFT_SIDE	RIGHT_SIDE
---	-----	-----	----	-----	---	-----	-----
1	15	1	13	EQ	E	1	2

Step 7: Display the expressions for the restrict node. Query the **STH_EXPLAIN_EXPR** table to display the details of the expressions. The **NODE** number is 13.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=13
ORDER BY ENTRY;
```

The result table is as follows. These rows represent the third component of the compound predicate **EMPNUM= '611'**.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
---	-----	-----	----	-----	-----
1	13	1	2	col	HR.EMPNUM
1	13	2	12	const	'611'

Example 7: Join with a complex predicate

This example analyzes a join with a complex predicate. The user tables are **EMPLOYEES** and **LOCATION**, both owned by **USER5**. The **EMPLOYEES** table columns are **BADGEID**, **DEPTID**, **EMPLOYEEENO**, and **EMPLOYEEENAME**. The **LOCATION** table columns are **BADGEID**, **DEPTID**, **EMPLOYEEENO**, and **MAILSTOP**. A value index is defined for the **EMPLOYEES** table on columns

BADGEID, DEPTID, EMPLOYEEENO. Assume you, USER5, execute the SQL statements. The query is as follows:

```
SELECT * FROM EMPLOYEES JOIN LOCATION
ON EMPLOYEES.BADGEID=LOCATION.BADGEID
AND EMPLOYEES.DEPTID=LOCATION.DEPTID
AND EMPLOYEES.EMPLOYEEENO=LOCATION.EMPLOYEEENO;
```

Step 1: Run the explain facility. Create the explain tables and then submit the EXPLAIN PLAN statement.

```
CREATE EXPLAIN TABLES;
```

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE7'
FOR SELECT * FROM EMPLOYEES JOIN LOCATION
ON EMPLOYEES.BADGEID=LOCATION.BADGEID
AND EMPLOYEES.DEPTID=LOCATION.DEPTID
AND EMPLOYEES.EMPLOYEEENO=LOCATION.EMPLOYEEENO;
```

Step 2: Obtain the execution plan ID. Query the STH_EXPLAIN_ID table to obtain the execution plan ID generated by StorHouse/RM.

```
SELECT * FROM STH_EXPLAIN_ID;
```

The result table is as follows. In this example, the execution plan ID is 1.

STMT_ID	STATEMENT_TIMESTAMP	ID
-----	-----	--
EXAMPLE7	10/03/2003 16:32:36.000000	1

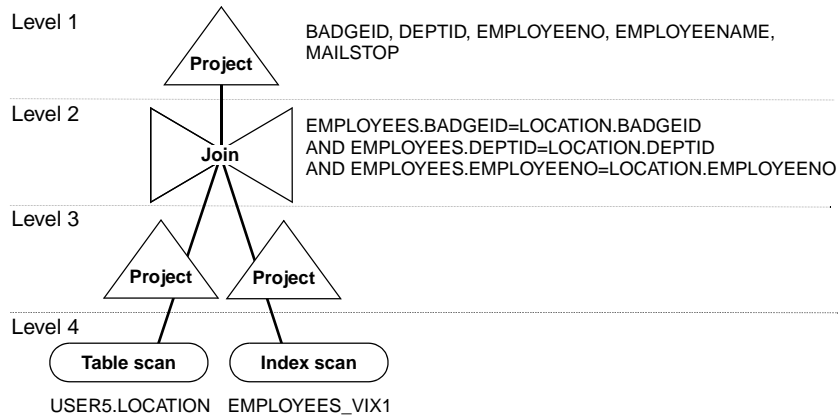
Step 3: Display the execution plan. Query the STH_EXPLAIN_PLAN table to examine the execution plan.

```
SELECT * FROM STH_EXPLAIN_PLAN
WHERE ID=1 ORDER BY LVL;
```

The result table is as follows. The join node indicates the optimizer chose a merge join. Note that there are no restrict nodes.

ID	NODE	PAR_NODE	LVL	LR	EXPLAIN_PLAN
--	----	-----	----	--	-----
1	53		1	L	PROJECT
1	51	53	2	L	JOIN[MERGE-JOIN]
1	42	51	3	R	PROJECT
1	48	51	3	L	PROJECT
1	49	42	4	L	INDEX SCAN OF USER5.EMPLOYEES INDEX EMPLOYEES_VIX1
1	50	48	4	L	TABLE SCAN OF USER5.LOCATION

The tree representation of this execution plan looks like this:



Step 4: Display the operators in the index scan node. Query the STH_EXPLAIN_OPR table to examine the index predicate for this index scan. The NODE number is 49.

```

SELECT * FROM STH_EXPLAIN_OPR
WHERE ID=1 AND ASSOC_NODE=49
ORDER BY ENTRY;

```

The result table is as follows. The result is slightly different from the previous example because the index scan for a join is constructed in a slightly different way.

ID	ASSOC_NO	ENTRY	NODE	PREDICATE	TB	LEFT_SIDE	RIGHT_SIDE
---	-----	-----	---	-----	---	-----	-----
1	49	1	86	EQ	E	1-3	4-6

In example 6, the two components of the index predicate are separate entries. The values of the constants are known during optimization.

```
component1=const (LOCNUM= '32808')
component2=const (DEPTID='M759')
```

In this case, the index predicate is one entry and the values of the constants aren't known until execution time.

```
(component1, component2, component3) = (const, const, const)
```

Step 5: Display the expressions of the index scan node. Query the STH_EXPLAIN_EXPR table to display the details of the expressions for ENTRY 1. The NODE number is 86.

```
SELECT * FROM STH_EXPLAIN_EXPR
WHERE ID=1 AND ASSOC_NODE=86
ORDER BY ENTRY;
```

The result table is as follows. ENTRY 1, 2, and 3 (LEFT_SIDE) contain the column references. ENTRY 4, 5, and 6 contain the word `constant`. At execution time, StorHouse/RM replaces the constants on the right side with values from the LOCATION table.

ID	ASSOC_NO	ENTRY	NODE	PROJ_T	PROJECT_COLUMN
---	-----	-----	----	-----	-----
1	86	1	87	col	EMPLOYEES.BADGEID
1	86	2	88	col	EMPLOYEES.DEPTID
1	86	3	89	col	EMPLOYEES.EMPLOYEEENO
1	86	4	83	const	constant
1	86	5	84	const	constant
1	86	6	85	const	constant

Accessing the StorHouse/Admin ISQL window

You can submit the explain statements and query the explain tables with the ISQL facility in StorHouse/Admin. This procedure explains how to access the ISQL window, which is the location where you submit SQL statements.

1. Log in to StorHouse/Admin.
2. In the folder list, expand the Databases folder.
3. Expand the database.
4. On the System menu, click **ISQL**.

Creating explain tables

You must create the explain tables before running the explain facility for a query. You create explain tables by submitting the CREATE EXPLAIN TABLES statement. The tables remain empty until you submit the EXPLAIN PLAN statement.

The format of the CREATE EXPLAIN TABLES statement is:

CREATE EXPLAIN TABLES [UID identifier]

When you create explain tables, you can optionally specify an owner (UID identifier) for the tables. If you omit the owner, then the account ID you use to log in to the StorHouse database is the owner of the explain tables. You must have DBA database privilege to specify an owner other than your account ID. Only the owner can submit the EXPLAIN PLAN statement for a set of explain tables.

The identifier is a character string with a maximum of 32 alphanumeric characters. It can be unquoted or enclosed in double quotes ("). If you use quotes, spaces are valid, for example, TKA is different from " TKA". StorHouse/RM stores the UID value in uppercase and you must subsequently use uppercase when referring to that identifier in SQL statements.

▼ To create explain tables

Privileges: SQLEXECUTE and either DBA or RESOURCE

1. If necessary, access the StorHouse/Admin Interactive SQL working window. "See Accessing the StorHouse/Admin ISQL window" on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the CREATE EXPLAIN TABLES statement. Be sure to include the semi-colon at the end of the SQL statement. Some examples follow.

- To specify an owner:

```
CREATE EXPLAIN TABLES UID TKA;
```

- To omit the owner to use the account ID used to log in to the StorHouse database as the owner:

```
CREATE EXPLAIN TABLES;
```

3. Click .

Running the explain facility

You run the explain facility by submitting the EXPLAIN PLAN statement. You must be the owner of the explain tables to submit this statement. The format of the EXPLAIN PLAN statement is:

```
EXPLAIN PLAN [SET STATEMENT_ID='statement_id'] FOR query;
```

The `statement_id` (up to 32 characters) identifies the execution plan and must be unique. It can be any string (including a reserved word and spaces) and any combination of upper and lower case. If you omit the statement ID, the default is `STH_EXPLAIN_DEFAULT`. An error occurs if you have already executed an EXPLAIN PLAN statement with the default statement ID.

The query must follow the format of the SELECT statement described in the *StorHouse SQL Reference Manual*.

▼ To run the explain facility

Privileges: SQLEXECUTE and owner of the explain tables

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the EXPLAIN PLAN statement. Be sure to include the semi-colon at the end of the SQL statement. Some examples follow.

- To run the explain facility when you know the statement ID:

```
EXPLAIN PLAN SET STATEMENT_ID='statement_id'  
FOR query;
```

For example:

```
EXPLAIN PLAN SET STATEMENT_ID='EXAMPLE2'  
FOR SELECT * FROM HR WHERE TAGNUM='32808';
```

- To run the explain facility when you don't know the statement ID:

```
EXPLAIN PLAN FOR query;
```

For example:

```
EXPLAIN PLAN FOR  
SELECT * FROM HR WHERE TAGNUM='32808';
```

3. Click .

Obtaining an execution plan ID

After you run the explain facility, StorHouse/RM generates a numeric ID for that execution plan. You can query the `STH_EXPLAIN_ID` table to obtain the execution plan ID and then use that plan ID to query the other explain tables.

▼ To obtain an execution plan ID

Privileges: `SQLEXECUTE` and owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter a `SELECT` statement for the `STH_EXPLAIN_ID` table. Some examples follow.
 - To display all execution plan IDs when you don’t know the statement ID:

```
SELECT * FROM owner.STH_EXPLAIN_ID;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_ID;
```

- To display an execution plan ID when you know the statement ID:

```
SELECT * FROM owner.STH_EXPLAIN_ID  
WHERE STMT_ID ='statement_id';
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_ID  
WHERE STMT_ID ='EXAMPLE2';
```

3. Click .

Displaying a query in an execution plan

You can display the query that is explained in an execution plan. You do this by querying the STH_EXPLAIN_STMT table.

▼ To display a query in an execution plan

Privileges: SQLEXECUTE and owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter a SELECT statement for the STH_EXPLAIN_ID table. Some examples follow.
 - To display an explained query when you don’t know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_STMT;
```

For example, in the following statement, the owner is omitted; therefore, the explain table owner must be the same as the account ID used to log in to the StorHouse database.

```
SELECT * FROM STH_EXPLAIN_STMT;
```

- To display an explained query when you know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_STMT  
WHERE ID =execution_id;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_STMT
WHERE ID=1;
```

3. Click .

Displaying an execution plan

After you run the explain facility, you can display the execution plan which identifies the nodes. You do this by querying the STH_EXPLAIN_PLAN table.

▼ To display an execution plan

Privileges: SQLEXECUTE and owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the SELECT statement for the STH_EXPLAIN_PLAN table. Some examples follow.
 - To display an execution plan when you don’t know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_PLAN;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_PLAN;
```

- To display an execution plan when you know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_PLAN  
WHERE ID =execution_id;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_PLAN  
WHERE ID =1;
```

3. Click .

Displaying expressions in an execution plan

You can display the expressions—aggregate functions, scalar functions, column references, constants, and parameters—associated with a node in an execution plan. You do this by querying the STH_EXPLAIN_EXPR table.

▼ To display expressions in an execution plan

Privileges: SQLEXECUTE and owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the SELECT statement for the STH_EXPLAIN_EXPR table. Some examples follow.

- To display all expressions in an execution plan when you don't know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_EXPR;
```

For example, in the following statement, the owner is omitted; therefore, the explain table owner must be the same as the account ID used to log in to the StorHouse database.

```
SELECT * FROM STH_EXPLAIN_EXPR;
```

- To display the expression for a specific node when you know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_EXPR  
WHERE ID =execution_id AND ASSOC_NODE=node_number  
ORDER BY ENTRY;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_EXPR  
WHERE ID=2 AND ASSOC_NODE=6  
ORDER BY ENTRY;
```

3. Click .

Displaying operators in an execution plan

You can display the operators—logical, relational, and arithmetic—associated with a node in an execution plan. You do this by querying the STH_EXPLAIN_OPR table.

▼ To display operators in an execution plan

Privileges: SQLEXECUTE and owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the SELECT statement for the STH_EXPLAIN_OPR table. Some examples follow.
 - To display all operators in an execution plan when you don’t know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_OPR;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_OPR;
```

- To display the operator for a specific node when you know the execution plan ID:

```
SELECT * FROM owner.STH_EXPLAIN_OPR  
WHERE ID =execution_id AND ASSOC_NODE=node_number;
```

For example:

```
SELECT * FROM TKA.STH_EXPLAIN_OPR  
WHERE ID=2 AND ASSOC_NODE=7;
```

3. Click .

Dropping explain tables

You can drop a set of explain tables by submitting the DROP EXPLAIN TABLES statement. You must be the owner of the tables to drop them or have DBA database privilege to drop tables for another account. You can omit the owner name only if the account ID you use to log in to the StorHouse database is the owner.

The format of the DROP EXPLAIN TABLES statement is:

DROP EXPLAIN TABLES [UID identifier];

▼ To drop explain tables

Privileges: SQLEXECUTE and either owner of the explain tables or DBA

1. If necessary, access the StorHouse/Admin Interactive SQL working window. “See Accessing the StorHouse/Admin ISQL window” on page 13-32 for instructions.
2. In the **Enter ISQL statements** area, enter the DROP EXPLAIN TABLES statement. Some examples follow.

- To drop your own explain tables (you must be logged in with the owner account ID):

DROP EXPLAIN TABLES;

- To drop explain tables for another user, for instance, TKA (you must have DBA privilege):

DROP EXPLAIN TABLES UID TKA;

Segment delete

This chapter describes the segment delete utility and explains how to run and schedule it.

About segment delete

The *segment delete utility* (sthseg_delete) identifies invalidated segments, removes the segment files (table data file, index files, LOB subsegments), and removes the associated entries in system tables. Use this facility to mass-remove invalidated segments and their associated objects from a StorHouse database.

You can also delete segments with StorHouse/Admin. The difference between deleting segments with StorHouse/Admin and the segment delete utility is as follows:

- With StorHouse/Admin, you select the invalidated segment(s) to delete or specify delete criteria for one user table at a time.
- With the segment delete utility, you identify the database, and the utility identifies the invalidated segments and deletes them. Utility options enable you to select or limit the scanned segments.

Before deleting segments

You must invalidate segments before deleting them. The segment delete utility removes only those segments that have been invalidated for a minimum of 14

days or a specified number of days (utility option). The interval between segment invalidation and deletion should not small, since a very long-running query might still be referencing the segment. Invalidating a segment that is in use does not create a problem, but deleting the associated StorHouse file might cause such a query to fail. You can invalidate segments in the following ways:

- StorHouse/Admin
- FileTek data loader REPLACE SEGMENT clause or a MERGE (COALESCE) operation
- PURGE TABLE (SQL) statement
- -p option on the sthseg_delete utility

Ways to run the segment delete utility

You can run the segment delete utility in the following ways:

- StorHouse Command Language RUN command
- StorHouse Command Language SCHEDULE command
- UNIX command line (StorHouse operating system prompt)
- UNIX cron command

This chapter describes how to run the utility with the StorHouse RUN and SCHEDULE commands.

Privileges required to run the utility

You must use the operator account and password to run the utility from the StorHouse operating system (UNIX) prompt. A StorHouse account with the OPERATOR, SERVICE, or SYSTEM command privilege can run the segment

delete utility with the StorHouse Command Language RUN and SCHEDULE commands.

Locking

Different locks occur during the segment delete process.

- The utility takes a read lock on system tables when scanning the StorHouse database to determine which segments to process.
- It takes a write lock when deleting table rows from system tables.
- It holds no locks when deleting segment files.

A utility option enables you to limit the number of seconds any system table write lock is held. The default is 2 seconds. If the database contains hundreds of thousands of segments to be scanned, specify a value of at least 10 seconds.

Running multiple delete processes

The utility operates on one StorHouse database at a time; however, you can run up to 10 segment delete processes at a time (for instance, to delete segments in different databases). FileTek recommends running only one instance that references the same table in the same database. All but one instance may fail in this case, although no database damage will occur and no StorHouse files will be orphaned.

Restarting an interrupted process

You can restart an interrupted process (for instance, should StorHouse shut down) by running the segment delete utility again. The utility creates a checkpoint record after deleting a segment table entry from the SYSTHFILES

system table or making any other associated table update to ensure restart and to prevent StorHouse files from being orphaned. The segment delete utility also creates a disk-resident table in the target database to track progress of the delete operation. The table, named `STH__SEG_DEL_UTIL`, provides current status information required to restart an interrupted operation.

After deleting segments

The segment delete utility moves the deleted segment files to the StorHouse deleted directory. Those files remain in that directory until you remove them with the StorHouse Command Language `REMOVE FILE` command or with `StorHouse/Admin`.

Running the segment delete utility

You should run the segment delete utility occasionally to remove invalidated segments. You can also schedule the utility to run at a specified time and frequency. The format of segment delete utility command is:

```
sthseg_delete [-options] database_name
```

Argument	Description
-options	(optional) Criteria to select or limit the segments that are scanned for deletion and to set locking and messaging preferences. The option characters are not case sensitive, for instance, you can specify <code>-d</code> or <code>-D</code> . All options have a default.
-d -D days	Minimum number of days a segment has been invalid. The default is 14 days.
-t -T tblid[,tblid]...	Table Id(s) to delete invalidated segments from specific user tables. The table ID is the ID value from the <code>SYSTABLES</code> entry for that table (that is, it is not the table name). The default is all user tables in the database.

Argument	Description
-b -B days	Minimum number of days the table must be dropped before the segments may be deleted. The default is the value of the SQL_DROP_HOLD system parameter.
-p -P	Option to purge all tables first. This is just like submitting the PURGE TABLE SQL statement.
-l -L lock_seconds	Limit on the number of seconds any system table write lock is held. The default is 2 seconds. Specify a larger value (for example, 10) if hundreds of thousands of segments are going to be scanned.
-v -V verbose_level	Level of message detail (echo activity to stdout). Values of verbose_level are 0 (minimal messaging), 1, 2, or 3 (most verbose). The default is 0.
database_name	(required) Name of the StorHouse database. Specify this parameter last on the command line (after any options). The database name is case sensitive. When running the segment delete utility with the RUN or SCHEDULE command, enclose a database name with lowercase characters in double quotes.

▼ To run the segment delete utility with the RUN command

Required privileges: OPERATOR, SERVICE, or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the RUN and sthseg_delete commands and press **Enter**. Some examples follow.
 - To delete all segments that have been invalidated for at least 10 days in database called Customer, type:

```
run sthseg_delete -d 10 "Customer"
```

- To delete all segments that have been invalidated for at least 14 days (default) in table 158 in the SALES database, type:

```
run sthseg_delete -t 158 SALES
```

- To purge all tables and then immediately delete all segments in the TEST database, type:

```
run sthseg_delete -p -d 0 -b 0 TEST
```

Scheduling the segment delete utility

You can schedule the segment delete utility to run at a specific time and frequency by using the StorHouse Command Language SCHEDULE command. It is not efficient to run the utility often, since it scans the entire segment table to determine which segments require deletion. Refer to the StorHouse *Command Language Reference Manual* for complete information about the SCHEDULE command. The RUN command must be the last item in the command statement

▼ To delete invalidated segments using the SCHEDULE command

Required privileges: SCHEDULE and OPERATOR or SERVICE or SYSTEM

1. If necessary, sign on to StorHouse.
2. At the command prompt (?), submit the SCHEDULE, RUN, and sthseg_delete commands and press **Enter**. Some examples follow.
 - To schedule a quarterly delete (every three months) of the Customer database, executed by account SYSTEM and password STEM, beginning at 2:00 a.m. on January 1, 2004, type:

```
SCHEDULE /START=01-JAN-2004:02:00:00 /  
SCHEDULE=MONTHLY:3 ACCOUNT=SYSTEM /PASSWORD=STEM  
!RUN STHSEG_DELETE "Customer"
```

- To schedule an end-of-month delete of the Customer database, executed by account SYSTEM and password STEM, beginning at 10:00 p.m. on October 1, 2003, type:

```
SCHEDULE /START=01-OCT-2003:22:00:00 /SCHEDULE=EOM  
ACCOUNT=SYSTEM /PASSWORD=STEM !RUN STHSEG_DELETE  
"Customer"
```


StorHouse system tables

This appendix contains reference information about StorHouse system tables, including:

- Where the system tables are located
- Who owns the system tables
- The purpose of each system table
- How to identify system tables
- Privileges for working with system tables
- Names and data types of the columns in each system table

About system tables

Each StorHouse database has its own collection of *system tables*. StorHouse/RM automatically creates these system tables for each new database. System tables contain information about a database. StorHouse/RM uses the system tables to record, verify, and conduct work. For example, StorHouse/RM updates various system tables when you create database components, and it reads system tables to verify that database components exist and that accounts are authorized to access them.

System table names

System table names, which you use to identify system tables in SQL statements, have the following format:

`SYSADM.SYSname`

where `SYSADM` is the owner of the system tables, `SYS` is a prefix for all system tables, and `name` is a descriptive title. For example, `SYSADM.SYSTABLES` is the name of the system table that describes all of the user tables, system tables, and views in a database.

Note: This publication refers to system tables in the *SYSname* format, assuming the `SYSADM` owner name.

System table names are *not* case sensitive, so you can use uppercase or lowercase letters when using system table names in SQL statements. For example, to select all information from `SYSTABLES`, you could type the `SELECT` statement in either of the following ways:

```
SELECT *  
FROM SYSADM.SYSTABLES
```

or

```
SELECT *  
FROM sysadm.systables
```

If you use the `SYSADM` account, you do not have to include the owner name as part of the system table name. For example:

```
SELECT *  
FROM SYSTABLES
```

System table privileges

The SYSADM account has ALL privilege for *all* system tables in *all* databases. SYSADM can use the following SQL statements for all system tables: SELECT, INSERT, UPDATE, and DELETE.

An account with DBA privilege has ALL privilege for *all* system tables in *a* specific database. This account can use the following SQL statements for system tables in that database: SELECT, INSERT, UPDATE, and DELETE.

PUBLIC has SELECT privilege for *all* system tables—except SYSRANGES—in *all* databases. StorHouse accounts with SQLEXECUTE privilege can query all system tables except SYSRANGES. You must use SYSADM or have DBA or SELECT privilege to access the SYSRANGES system tables.

Caution: System tables contain critical database information. With the exception of the SYSSMUSERS system table, users should *never* use any SQL statement other than SELECT on the system tables. If you do not want PUBLIC access to system tables, revoke the SELECT privilege from PUBLIC in all databases. Be aware, however, that if you update or revoke the PUBLIC privilege for a system table, the metadata conversion process ignores the change. You must re-do the change in the converted database.

List of system tables

The following table lists and briefly describes the StorHouse system tables.

System table	Contains	See page
SYSCOLAUTH	Column update privileges granted to accounts	A-6
SYSCOLUMNS	One row for each column of every table in a database	A-7
SYSDBAUTH	Database privileges for accounts	A-8
SYSDROP_PEND	One row for each dropped table	A-10
SYSINDEXES	One row for each indexed column	A-11
SYSPACKAGE	One row for each package in a database	A-13
SYSPACKSTMT	One or more rows for each statement in a package	A-15
SYSRANGES	Range indexes	A-17
SYSSMUSERS	One row for each account with a default tablespace, plus a default tablespace entry for PUBLIC	A-20
SYSSTAT_COL	Statistics for each indexed column of a segment	A-21
SYSSTAT_HIST	Spread and frequency of each histogram bucket	A-22
SYSSTAT_IDX	Average number of rows returned for queries that use value indexes	A-22
SYSSTAT_SMATRIX	Spread encoding matrixes used to calculate the spread of non-numeric values	A-24
SYSSTHFILES	StorHouse file and group names for segment files	A-25
SYSSTHSEGMENTS	Information about segments	A-26
SYSSTHSPACES	Storage parameters for user tablespaces	A-27

System table	Contains	See page
SYSSYNONYMS	One row for each synonym in a database	A-30
SYSTABAUTH	Table privileges held by accounts	A-31
SYSTABLES	One row for each user table, system table, and view in a database	A-33
SYSTBLSPACES	One row for each tablespace in a database	A-35
SYSVIEWS	One row for each view in a database	A-36

The following system tables are for future use and not described in this appendix:

- SYSDATATYPES
- SYSDBLINKS
- SYSIDXSTAT
- SYSTBLSTAT
- SYS_CHKCOL_USAGE
- SYS_CHK_CONSTRS
- SYS_KEYCOL_USAGE
- SYS_REF_CONSTRS
- SYS_TBL_CONSTRS

SYSCOLAUTH

The SYSCOLAUTH system table indicates the individual system table or system table view columns that accounts can update. StorHouse/RM updates this system table when you grant and revoke the UPDATE privilege for system table columns.

Column name	Data type	Description
GRANTOR	VARCHAR(32) NOT NULL	Account ID of the account who granted the column privileges.
GRANTEE	VARCHAR(32) NOT NULL	Account ID of the account who holds the column privileges.
TBLOWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the table or view on which the column privileges were granted.
TBL	VARCHAR(32) NOT NULL	Name of the table or view on which the column privileges were granted.
COL	VARCHAR(32) NOT NULL	Name of the column to which the UPDATE privilege applies.
UPD	VARCHAR(1) NOT NULL	Grantee's UPDATE privilege for the column in the table or view. Values are: y Can update this column in the system table [] (Blank) Indicates no UPDATE privilege
REF	VARCHAR(1) NOT NULL	For future use.



SYSCOLUMNS

The SYSCOLUMNS system table contains one row for each column in every table in a database. When you create a user table, StorHouse/RM inserts a row into this system table for each column in the table.

Column name	Data type	Description
ID	INTEGER NOT NULL	ID assigned to the column.
COL	VARCHAR(32) NOT NULL	Name of the column.
TBL	VARCHAR(32) NOT NULL	Name of the table that contains the column.
OWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the table.
COLTYPE	VARCHAR(10) NOT NULL	Data type of the column. Values are: BIGINT FLOAT BINARY INTEGER BLOB REAL CHARACTER SMALLINT CLOB TIME DATE TIMESTAMP DECIMAL/NUMERIC VARBINARY DOUBLE VARCHAR
WIDTH	INTEGER NOT NULL	Length of the column, or in the case of data type NUMERIC or DECIMAL, the precision of the data type.
SCALE	INTEGER NULL	Scale of data type NUMERIC or DECIMAL.
NULLFLAG	VARCHAR(1) NOT NULL	Null indicator for the column. Values are: Y Can be null N Cannot be null

A**StorHouse system tables****SYSDBAUTH**

Column name	Data type	Description
DFLT_VALUE	VARCHAR(250) NULL	Default value for the column, if defined in a CREATE TABLE statement.
LOB_TSID	INTEGER NOT NULL	ID of the user tablespace to which the LOB column is assigned. If the column is a non-LOB column, then LOB_TSID has a value of -1. If the LOB column is assigned to the same tablespace as the table, then LOB_TSID has the ID of the user tablespace for the table.
LOB_INLINE_MAX	INTEGER NULL	Maximum length of an in-line LOB for this column. A NULL indicates the column is a non-LOB data type or the LOB is always stored out-of-line. A 0 means the length was omitted on the INLINE clause; therefore, the default maximum in-line length is 32705.
LOB_STORE_WITH	INTEGER NULL	ID of the LOB column to share storage with this LOB column. The first column in a table has column ID 0. A NULL indicates one of the following: <ul style="list-style-type: none"> ■ Column is a non-LOB data type ■ LOB column does not share storage with another LOB column ■ LOB is always stored in-line

SYSDBAUTH

The SYSDBAUTH system table contains database privileges—DBA, RESOURCE, and SCAN—for accounts. StorHouse/RM maintains this system table when you grant and revoke database privileges.

Column name	Data type	Description
GRANTEE	VARCHAR(32) NOT NULL	Account ID of the account who holds the database privileges.
DBA_ACC	VARCHAR(1) NOT NULL	Grantee's DBA privilege for the database. Values are: y Can create and select database components [] (Blank) Cannot create and select database components
RES_ACC	VARCHAR(1) NOT NULL	Grantee's RESOURCE privilege for the database. Values are: y Can create database components [] (Blank) Cannot create database components
SCN_ACC	VARCHAR(1) NOT NULL	Grantee's SCAN privilege for the database. Values are: y Can scan user tables [] (Blank) Cannot scan user tables

SYSDROP_PEND

The SYSDROP_PEND system table contains information about dropped user tables and associated indexes. StorHouse/RM inserts a row into this system table when a user table is dropped and deletes rows when user tables are purged.

Column name	Data type	Description
OLD_OWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the dropped table or index.
OLD_NAME	VARCHAR(32) NOT NULL	Name of the dropped table or index.
NEW_OWNER	VARCHAR(32) NOT NULL	Same account ID as the OLD_OWNER.
NEW_NAME	VARCHAR(32) NOT NULL	Internal, unique name for the dropped table or index.
TBL_OR_IDX	CHAR(1) NOT NULL	Type of component. Values are: T User table I Index
TBLID	INTEGER NOT NULL	ID of the user table.
DROP_TIME	TIMESTAMP NOT NULL	Date and time the user table was dropped.
USERNAME	VARCHAR(32) NOT NULL	Account ID of the user who dropped the table.

SYSINDEXES

The SYSINDEXES system table contains one row for each indexed column of every system table and user table in a database. StorHouse/RM inserts a row (or multiple rows in the case of a compound index) into this system table when you create an index and deletes a row(s) when you drop an index.

For a compound index, StorHouse/RM inserts one row in this table for every column in the compound index. If the compound index is composed of two columns, then the SYSINDEXES system table contains two rows for that index.

Column name	Data type	Description
ID	INTEGER NOT NULL	ID assigned to the index.
TSID	INTEGER NOT NULL	ID of the tablespace to which the index is assigned. For a system table index, the value 0 indicates the system tablespace ID.
IDXNAME	VARCHAR(32) NOT NULL	Name of the index.
IDXOWNER	VARCHAR(32) NOT NULL	Name of the owner of the index.
TBL	VARCHAR(32) NOT NULL	Name of the table on which the index is defined.
TBLOWNER	VARCHAR(32) NOT NULL	Name of the owner of the table on which the index is defined.
COLNAME	VARCHAR(32) NOT NULL	Name of the column on which the index is defined.
IDXTYPE	VARCHAR(1) NOT NULL	Unique indicator. Values are: U Unique index (no duplicate column values) D Non-unique (duplicate) index

A

StorHouse system tables

SYSINDEXES

Column name	Data type	Description
IDXORDER	VARCHAR(1) NOT NULL	Order of values in the index. Values are: A Ascending values D Descending values
IDXCOMPRESS	VARCHAR(1) NOT NULL	Status of the index. Values are: N Normal (complete) C Deferred (incomplete)
IDXMETHOD	VARCHAR(1) NOT NULL	Type of index. Values are: B B-tree for system table index or value index for user table index H Hash index for user tables R Range index for user tables
IDXSEQ	INTEGER NOT NULL	Column sequence number within the index. Sequence numbers start with 0.
COMPRESSION	SMALLINT NOT NULL	For future use.

SYSPACKAGE

The SYSPACKAGE system table contains one row for each package in a database. The combination of RDBCOLID, PKGID, and VERSION_NAME uniquely identifies a StorHouse package.

Column name	Data type	Description
RDBNAM	CHAR(18) NOT NULL	Name of the StorHouse database.
RDBCOLID	CHAR(18) NOT NULL	Identifier of the package collection.
PKGID	CHAR(18) NOT NULL	Identifier of the specific StorHouse package.
PKGCNSTKN	CHAR(16) NOT NULL	Consistency token, which verifies synchronization of the requestor application and the database package. Packages with the same fully qualified package name cannot have the same consistency token.
CCSIDSBC	SMALLINT NULL	Coded character set identifier for single-byte characters. This column is not used in this release.
CCSIDDBC	SMALLINT NULL	Coded character set identifier for double-byte characters. This column is not used in this release.
CCSIDMBC	SMALLINT NULL	Coded character set identifier for mixed-byte characters. This column is not used in this release.
VERSION_NAME	VARCHAR(254) NULL	Version identifier associated with the package.
REPLACED_PKG_NAME	VARCHAR(254) NULL	Version identifier of the package that was replaced by the current package.
DECIMAL_PRECISION	SMALLINT NULL	Default precision used during decimal arithmetic processing.

Column name	Data type	Description
DEFAULT_ RDBCOLID	CHAR(18) NULL	Default collection identifier used (when necessary) to complete component names in SQL statements that are bound into this package.
AUTH_OPT	SMALLINT NULL	Package authorization option specifying whether the existing package authorizations are maintained or revoked when a package is replaced.
DEFAULT_CHAR_ SUBTYPE	SMALLINT NULL	Default SQL character subtype used if a character column is defined by an SQL CREATE TABLE or ALTER TABLE statement with no explicit subtype specification. Note that ALTER TABLE is not supported in this release.
ISOLATION_LEVEL	SMALLINT NULL	Isolation level used when SQL statements in the package are executed. This isolation level may be overridden by the target relational database run-time mechanism.
OWNER_ID	CHAR(8) NULL	Account ID of the package owner.
BLOCK_PROTOCOL	SMALLINT NULL	Controller of the query block protocol used when a query is opened. When specified in the BGNBND command, it controls the protocols used by all queries in the package unless overridden by the OPNQRY command.
RELEASE_OPTION	SMALLINT NULL	Option for releasing the package execution resources and the associated serialization or sharing of locks.
DATE_FORMAT	SMALLINT NULL	Default date format used in SQL statements.
DECIMAL_ DELIMITER	SMALLINT NULL	Default character string used as the decimal delimiter in SQL statements.

Column name	Data type	Description
STRING_DELIMITER	SMALLINT NULL	Characters that delimit strings and delimited SQL identifiers in SQL statements.
TIME_FORMAT	SMALLINT NULL	Default time format used in SQL statements.
TITLE	VARCHAR(255) NULL	Brief description of the package.

SYSPACKSTMT

The SYSPACKSTMT system table contains one or more rows for each statement in a package.

Column name	Data type	Description
RDBNAM	CHAR(18) NOT NULL	Name of the StorHouse database.
RDBCOLID	CHAR(18) NOT NULL	Identifier of the package collection.
PKGID	CHAR(18) NOT NULL	Identifier of the StorHouse package.
PKGCNSTKN	CHAR(16) NOT NULL	Consistency token, which verifies synchronization of the requestor application and the database package. Packages with the same fully qualified package name cannot have the same consistency token.
PKGSN	SMALLINT NOT NULL	StorHouse package section number.

StorHouse system tables

SYSPACKSTMT

Column name	Data type	Description
STMTSEQNO	SMALLINT NOT NULL	Statement sequence number. If the STATEMENT column contains text for the SQL statement that the row represents, then STMTSEQNO specifies the sequence number of that text within the entire SQL statement.
STATEMENT	VARCHAR(254) NOT NULL	All or a portion of the text for the SQL statement that the row represents.
STATEMENT_TYPE	SMALLINT NULL	Type of statement. Values are: 1 - DECLARE 7 - GRANT 2 - SELECT 8 - REVOKE 3 - INSERT 9 - DYNAMIC 4 - UPDATE 10 - COMMIT 5 - DELETE 11 - ROLLBACK 6 - CREATE
TYPDEFNAM	CHAR(9) NULL	Name of a data type definition.
CCSIDSBBC	SMALLINT NULL	Coded character set identifier for single-byte characters. This column is not used in this release.
CCSIDDBC	SMALLINT NULL	Coded character set identifier for double-byte characters. This column is not used in this release.
CCSIDMBC	SMALLINT NULL	Coded character set identifier for mixed-byte characters. This column is not used in this release.

SYSRANGES

The SYSRANGES system tables contain range indexes for each data type (except BLOB and CLOB). PUBLIC does not have SELECT privilege on these system tables. Only SYSADM or an account with DBA privilege or SELECT privilege can access these system tables. The convention for naming the range index system tables is:

SYSRANGES_datatype

where datatype is the name of the column data type. The following table maps column data types to range index system tables.

Data type	Length	System table name
BIGINT		SYSRANGES_BIGINT
BINARY	1-32	SYSRANGES_VARBINARY_SHORT
BINARY	33-256	SYSRANGES_VARBINARY_LONG
CHAR	1-32	SYSRANGES_VARCHAR_SHORT
CHAR	33-256	SYSRANGES_VARCHAR_LONG
DATE		SYSRANGES_DATE
DOUBLE PRECISION		SYSRANGES_DOUBLE
INTEGER		SYSRANGES_INTEGER
NUMERIC		SYSRANGES_DECIMAL
REAL		SYSRANGES_REAL
SMALLINT		SYSRANGES_SMALLINT
TIME		SYSRANGES_TIME
TIMESTAMP		SYSRANGES_TIMESTAMP
VARBINARY	1-32	SYSRANGES_VARBINARY_SHORT
VARBINARY	33-256	SYSRANGES_VARBINARY_LONG

Data type	Length	System table name
VARCHAR	1-32	SYSRANGES_VARCHAR_SHORT
VARCHAR	33-256	SYSRANGES_VARCHAR_LONG

The SYSRANGES system tables contain the following columns.

Column name	Data type	Description
IDXID	INTEGER NOT NULL	ID assigned to the index.
COLSEQ	INTEGER NOT NULL	Sequence number of the column in the index. Values are 0 to 15.
SEGID	INTEGER NOT NULL	ID of the segment.
NULLID	CHAR(1) NOT NULL	Flag indicating whether any value in the column is NULL. Values are: Y At least one value in the column is NULL. N None of the values in the column is NULL.
LOWVAL	Determined by the data type of the range index system table	Lowest data value in the segment. A NULL value is not considered as a LOWVAL unless all values in the column are NULL.
HIVAL	Determined by the data type of the range index system table	Highest data value in the segment. A NULL value is not considered as a HIVAL unless all values in the column are NULL.

The following table contains the data type of the LOWVAL and HIVAL columns for each SYSRANGES system table.

SYSRANGES system tables	LOWVAL and HIVAL data types
SYSRANGES_VARCHAR_SHORT	VARCHAR(32)
SYSRANGES_VARCHAR_LONG	VARCHAR(256)
SYSRANGES_VARBINARY_SHORT	VARBINARY(32)
SYSRANGES_VARBINARY_LONG	VARBINARY(256)
SYSRANGES_BIGINT	BIGINT
SYSRANGES_SMALLINT	SMALLINT
SYSRANGES_INTEGER	INTEGER
SYSRANGES_DECIMAL	DECIMAL(31,0)*
SYSRANGES_REAL	REAL
SYSRANGES_DOUBLE	DOUBLE
SYSRANGES_DATE	DATE
SYSRANGES_TIME	TIME
SYSRANGES_TIMESTAMP	TIMESTAMP

* The actual scale of the stored value is determined from the column definition in the SYSCOLUMNS system table.

SYSSMUSERS

The SYSSMUSERS system table contains an entry for each account with a default user tablespace. It also contains an entry for PUBLIC, which designates the default user tablespace for the database. Accounts that are excluded from this table have no account default user tablespace. You can maintain this system table.

Caution: When inserting, updating, or deleting a row in the SYSSMUSERS system table, remember to type the account ID in uppercase. Also, if you delimited the tablespace name when you created the user tablespace, and the name contains lowercase letters (for instance, CREATE TABLE SPACE “Mar2000”), be sure to type the name in the same case (Mar2000, not MAR2000 or mar2000). If you didn’t delimit the tablespace name, type it in uppercase letters.

Column name	Data type	Description
ACCOUNTID	CHAR(12) NOT NULL	Account ID of the StorHouse account or PUBLIC for the database default tablespace.
DEFAULT_TS	CHAR(32) NOT NULL	Name of the default user tablespace. StorHouse uses this tablespace when the account creates a user table and does not specify a tablespace name.

SYSSTAT_COL

The SYSSTAT_COL system table contains statistics for each indexed column of a segment.

Column name	Data type	Description
TBLID	INTEGER NOT NULL	ID of the user table.
COLID	INTEGER NOT NULL	ID of the indexed column.
SEGID	INTEGER NOT NULL	ID of the segment.
LOW_VAL	VARCHAR(40)* NULL	Lowest value in the column sample.
LOW_VAL2	VARCHAR(40)* NULL	Second lowest value in the column sample.
HI_VAL2	VARCHAR(40)* NULL	Second highest value in the column sample.
HI_VAL	VARCHAR(40)* NULL	Highest value in the column sample.
HISTID	INTEGER NOT NULL	ID of the histogram.
NUM_BUCKETS	SMALLINT NOT NULL	Total number of buckets in the histogram.
NULLVAL_SEL	REAL NOT NULL	Selectivity of NULL values in the column sample.
SMATID	INTEGER NULL	ID of the spread matrix used to calculate the spread for non-numeric values.
IO_FACTOR	REAL NOT NULL	Expected number of page I/O operations required to retrieve one row.

* The data stored in these VARCHAR columns may be of any data type. StorHouse/RM performs any necessary conversions and stores the values in literal form. BINARY or VARBINARY, however,

is stored without conversion and may contain unprintable characters. CHAR, VARCHAR, BINARY, and VARBINARY values longer than 40 bytes are truncated.

SYSSTAT_HIST

The SYSSTAT_HIST system table contains the spread and frequency of each histogram bucket.

Column name	Data type	Description
HISTID	INTEGER NOT NULL	ID of the histogram.
BUCKET_SEQ	SMALLINT NOT NULL	Sequence number of the histogram bucket.
NUM_DISTINCTVAL	SMALLINT NOT NULL	Number of distinct column values in the bucket, estimated from a reservoir sample.
LOW_VAL	DOUBLE PRECISION NOT NULL	Lowest value in the histogram bucket.
SPREAD	DOUBLE PRECISION NOT NULL	Spread (difference) between the highest and lowest values in the histogram bucket.
FREQUENCY	SMALLINT NOT NULL	Total frequency (number of records) in the histogram bucket.

SYSSTAT_IDX

The SYSSTAT_IDX system table contains the average number of rows returned for queries that use value indexes, that is, queries that contain an equals predicate for one or more columns.

Column name	Data type	Description
IDXID	INTEGER NOT NULL	ID of the value index.
SEGID	INTEGER NOT NULL	ID of the segment.
IN_SORT	CHAR(1) NULL	Flag indicating whether the table rows are in sorted order of the indexed column's key values. Valid values: Y In sorted order N Not in sorted order
COL1_CARD	DOUBLE PRECISION NOT NULL	Average number of result rows for a query with an equals predicate on column 1.
COL12_CARD	DOUBLE PRECISION NULL	Average number of result rows for a query with an equals predicate on columns 1 and 2.
COL123_CARD	DOUBLE PRECISION NULL	Average number of result rows for a query with an equals predicate on columns 1, 2, and 3.
COL1234_CARD	DOUBLE PRECISION NULL	Average number of result rows for a query with an equals predicate on columns 1, 2, 3, and 4.

SYSSTAT_SMATRIX

The SYSSTAT_SMATRIX system table stores spread encoding matrixes used to calculate the spread of non-numeric values.

Column name	Data type	Description
SMATID	INTEGER NOT NULL	ID of the spread encoding matrix.
SMAT_BASE	SMALLINT NOT NULL	Number of marker array entries that are set to 1.
SMAT_MATRIX	BINARY(32) NOT NULL	Spread encoding matrix.

SYSSTHFILES

The SYSSTHFILES system table contains the StorHouse file name and group name for each table data file, index file, and LOB subsegment file. A StorHouse engine updates this system table after a confirmed load, after an index load and a merge operation, and when you drop user tables and indexes. A StorHouse engine uses this system table to locate the data during query processing.

Column name	Data type	Description
ID	INTEGER NOT NULL	ID of the table or index.
OBJECT_TYPE	CHAR(1) NOT NULL	Type of database component. Values are: T Table data H Hash index V Value index L LOB data
SUBSEGID	INTEGER	ID of the LOB subsegment.
SEGID	INTEGER NOT NULL	ID of the segment.
FILENAME	VARCHAR(56) NOT NULL	Name of the StorHouse file.
GROUPID	VARCHAR(8) NOT NULL	Name of the StorHouse file access group.

SYSSTHSEGMENTS

The SYSSTHSEGMENTS system table contains information about segments. A StorHouse engine updates this system table after a data load, replace, or merge operation and when you drop tables.

Column name	Data type	Description
TBLID	INTEGER NOT NULL	ID of the user table.
SEGID	INTEGER NOT NULL	ID of the segment.
ROW_COUNT	DOUBLE PRECISION	Number of rows in the table data file. A NULL value indicates that the row count was not collected when the segment was loaded.
ROWS_PER_PAGE	REAL	Average row count per page in the table data file. A NULL value indicates that the average row count was not collected when the segment was loaded.
LOAD_CONF_TIME	TIMESTAMP	Date and time that the operation was confirmed. A NULL value indicates that the load confirmation time was not recorded when the segment was loaded.
INVALID_FLAG	CHAR(1) NOT NULL	Flag indicating that the segment has been invalidated. Values are: Y Segment is invalidated. N Segment is loaded and accessible.

Column name	Data type	Description
INVALID_TIME	TIMESTAMP	Date and time that the segment was invalidated. A NULL value indicates that the segment has never been invalidated.
SEGMENT_TAG	VARCHAR(40) NOT NULL	Name of the segment (either the segment tag provided on the SEGMENT clause of a LOAD statement or the load ID if the SEGMENT clause was omitted). For data loaded before StorHouse/RM release 2.2, there is no SEGMENT_TAG, that is, the value is 0 length.

SYSSTHSPACES

The SYSSTHSPACES system table contains one row for each subspace in a user tablespace. This system table identifies the storage specifications for each user tablespace. StorHouse/RM maintains this system table when you create, alter, and drop user tablespaces.

Column name	Data type	Description
TSID	INTEGER NOT NULL	ID of the user tablespace.
TSSUBSPACE_NUM	INTEGER NOT NULL	Number assigned to the subspace.
TSOBJECT_TYPE	CHAR(1) NOT NULL	Type of component allowed in this subspace. Values are: <div> " " or "" (" " is the default) All components (no restriction) T Table data only H Hash index only V Value index only L LOB data only </div>

Column name	Data type	Description
TSVSET	CHAR(8) NOT NULL	Name of the volume set that contains components in this subspace.
TSFSET	CHAR(8) NOT NULL	Name of the file set that contains components in this subspace.
TSVTF	CHAR(8) NOT NULL	<p>Vulnerability time factor (VTF) for components in this subspace. Values are:</p> <p>DEFAULT (default) Use the value of the StorHouse VTF system parameter.</p> <p>NOW Write the file to the performance buffer first and then copy it immediately to its file set.</p> <p>NEXT Write the file to the performance buffer and copy it to its file set during the next StorHouse write-back operation.</p> <p>DIRECT Bypass the performance buffer and write the file directly to the file set. (DF and map extents, however, are always written to the performance buffer as well as to the file set.)</p>
TSATF	SMALLINT NOT NULL	<p>Access time factor (ATF) for components in this subspace. Values are:</p> <p>0 (default) Use the value of the StorHouse ATF system parameter.</p> <p>1 Short access time is important.</p> <p>2 Short access time is moderately important.</p> <p>3 Short access time is minimally important.</p>

Column name	Data type	Description
TSEDC	CHAR(1) NOT NULL	Error Detection Code option for components in this subspace. Values are: D (default) Use the value of the StorHouse EDC system parameter. Y Use EDC. N Do not use EDC.
TSGROUP	CHAR(8) NOT NULL	Name of the StorHouse file access group for components in this subspace. The default file access group is STH.
TSMAX_EXT_SIZE	SMALLINT NOT NULL	Maximum size (in megabytes) of extents in this subspace. The size range is 1 to the maximum surface size for the VSET. The default 0 uses 100 MB for LOB subsegment files or the value of the applicable StorHouse system parameter: ■ SQL_MAX_EXT_DATA for table data files ■ SQL_MAX_EXT_HASH for hash index files ■ SQL_MAX_EXT_VAL for value index files
TSHOLD	SMALLINT NOT NULL	Number of days to keep data extents in the performance buffer. The range of days is 0 to 32767. For LOB subsegment files, the default 0 uses 0 days. For table data files and index files, the default 0 uses the value of the applicable StorHouse system parameter: ■ SQL_HOLD_DATA for table data files ■ SQL_HOLD_INDX for index files
TSHOLD_SPECIAL	SMALLINT NOT NULL	Number of days to keep DF and map extents in the performance buffer. The range of days is 0 to 32767. The default 0 uses the value of the SQL_HOLD_SPECIAL system parameter.

SYSSYNONYMS

The SYSSYNONYMS system table contains one entry for each synonym in a database. StorHouse/RM maintains this system table when you create and drop synonyms.

Column name	Data type	Description
SNAME	VARCHAR(32) NOT NULL	Name of the synonym.
SOWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the synonym.
SCREATOR	VARCHAR(32) NOT NULL	Account ID of the creator of the synonym.
STBL	VARCHAR(32) NOT NULL	Name of the table or view for which the synonym was created.
STBOWNER	VARCHAR(32) NOT NULL	Name of the owner of the table or view for which the synonym was created.
ISPUBLIC	SMALLINT NOT NULL	Type of synonym. Values are: 0 Private synonym 1 Public synonym
SREMDB	VARCHAR(32) NULL	Remote database name.

SYSTABAUTH

The SYSTABAUTH system table contains the table and view privileges for accounts. StorHouse/RM maintains this system table when you grant and revoke database component privileges.

Column name	Data type	Description
GRANTOR	VARCHAR(32) NOT NULL	Account ID of the account that granted the table privileges.
GRANTEE	VARCHAR(32) NOT NULL	Account ID of the account that holds the table privileges. The grantee can also be PUBLIC.
TBLOWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the table or view on which the table privileges were granted.
TBL	VARCHAR(32) NOT NULL	Name of the table or view on which the table privileges were granted.
INS	VARCHAR(1) NOT NULL	Grantee's INSERT privilege for loading data into the user table or inserting data into the system table or system table view. Values are: <ul style="list-style-type: none"> y Can insert rows into the system table or load data into the user table g Can insert or load rows, WITH GRANT OPTION [] (Blank) Cannot insert or load rows
DEL	VARCHAR(1) NOT NULL	Grantee's DELETE privilege for the system table or system table view. Values are: <ul style="list-style-type: none"> y Can delete rows from the system table g Can delete rows, WITH GRANT OPTION [] (Blank) Cannot delete rows from the system table

A

StorHouse system tables

SYSTABAUTH

Column name	Data type	Description
UPD	VARCHAR(1) NOT NULL	<p>Grantee's UPDATE privilege for the system table or system table view. Values are:</p> <p>y Can update rows in the system table</p> <p>g Can update rows, WITH GRANT OPTION</p> <p>[] (Blank) Cannot update rows in the system table</p> <p>* Privilege exists for some, but not all columns in the table</p>
SEL	VARCHAR(1) NOT NULL	<p>Grantee's SELECT privilege for the user table, user table view, system table, or system table view. Values are:</p> <p>y Can query the table or view</p> <p>g Can query the table or view, WITH GRANT OPTION</p> <p>[] (Blank) Cannot query the table or view</p>
NDX	VARCHAR(1) NOT NULL	<p>Grantee's INDEX privilege for the user table or system table. Values are:</p> <p>y Can create an index for the table</p> <p>g Can create an index, WITH GRANT OPTION</p> <p>[] (Blank) Cannot create an index for the table</p>
ALT	VARCHAR(1) NOT NULL	For future use.
REF	VARCHAR(1) NOT NULL	For future use.

SYSTABLES

The SYSTABLES system table contains one row for each user table, system table, and view in a database. StorHouse/RM inserts rows for system tables when you create a database and inserts rows for user tables and views when you create them. StorHouse/RM deletes rows when you drop user tables or views. In the following table, the word *table* refers to system tables, user tables, and views.

Note: StorHouse/RM prevents updates to the rows for system tables.

Column name	Data type	Description
ID	INTEGER NOT NULL	ID assigned to the table.
TBL	VARCHAR(32) NOT NULL	Name of the table.
CREATOR	VARCHAR(32) NOT NULL	Account ID of the creator of the table.
OWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the table.
TBLTYPE	VARCHAR(1) NOT NULL	Type of table. Values are: T Table V View
TBLPCTFREE	INTEGER NOT NULL	For future use.
TBLSPACEID	INTEGER NOT NULL	ID of the user tablespace to which the user table is assigned.
HAS_PCNSTRS	VARCHAR(1) NOT NULL	For future use.
HAS_FCNSTRS	VARCHAR(1) NOT NULL	For future use.
HAS_CCNSTRS	VARCHAR(1) NOT NULL	For future use.

A**StorHouse system tables****SYSTABLES**

Column name	Data type	Description
HAS_UCNSTRS	VARCHAR(1) NOT NULL	For future use.
TBL_STATUS	VARCHAR(1) NOT NULL	Not used.
RSSID	INTEGER NOT NULL	Identifier for the storage system. Values are: 1 StorHouse 0 System or temporary tablespace
SM_LOAD_SEGMENT	INTEGER NOT NULL	Next segment ID to use for loading.
COMPRESSION	SMALLINT NOT NULL	For future use.

SYSTBLSPACES

The SYSTBLSPACES system table contains one row for each tablespace in a database. StorHouse/RM inserts a row into this system table when you create a user tablespace and deletes a row when you drop a user tablespace.

Column name	Data type	Description
ID	INTEGER NOT NULL	ID of the tablespace.
TSNAME	VARCHAR(32) NOT NULL	Name of tablespace.
STORAGE_MANAGER	VARCHAR(32) NOT NULL	Identifier for the storage system. Values are: STH StorHouse DICT System tablespace TEMP Temporary tablespace
NUM_SUBSPACES	INTEGER NOT NULL	Number of subspaces in the user tablespace.

SYSVIEWS

The SYSVIEWS system table contains one row for each view in a database. StorHouse/RM maintains this system table when you create and drop views.

Column name	Data type	Description
VIEWNAME	VARCHAR(32) NOT NULL	Name of the view.
CREATOR	VARCHAR(32) NOT NULL	Account ID of the creator of the view.
OWNER	VARCHAR(32) NOT NULL	Account ID of the owner of the view.
SEQ	INTEGER NOT NULL	Sequence number used for packing and unpacking view text.
VIEWTEXT	VARCHAR(900) NOT NULL	SQL text of the view.

System limits

The following table summarizes the limits for StorHouse databases. “No meaningful limit” means that index, tablespace, and subspace numbers are limited only by internal structures with maximum values like $2^{*}31$ or $10^{*}9$.

Item	Limit
Length of a column name	32 characters
Length of a database name	32 characters
Length of a default value for a column	32 characters
Length of a file set name	8 characters
Length of an owner name	32 characters
Length of a table name	32 characters
Length of a StorHouse account ID	12 characters
Length of a StorHouse file name	56 characters
Length of a user tablespace name	32 characters
Length of a volume set name	8 characters
Length of an index name	32 characters
Number of columns in a compound index	16 columns
Number of columns in a table	1,024 columns
Number of databases	No limit
Number of active databases at a time	100 databases
Number of engines that can run concurrently	Value of SQL_SESSIONS

Item	Limit
Number of indexes in a database	No meaningful limit
Number of indexed columns for a user table	2400 columns
Number of table files, index files, and LOB subsegment files in all user tables	7,000,000 files
Number of tablespaces	No meaningful limit
Number of subspaces in a user tablespace	No meaningful limit
Number of user and system tables in a database	none
Number of read locks on a user or system table	100 locks
Number of user and system tables that can be accessed in a database at a time	100 tables
Number of range indexes that can be referenced in a single query block	32 range indexes
Number of concurrently open cursors	32 cursors
Size of a row in a table	32,705 bytes
Size of an in-line LOB	32,705 bytes
Size of an out-of-line LOB	2**31-9
Size of a column in an index	256 bytes
Size of a key in a compound index	4,096 bytes
Size of a table data file	100 gigabytes
Size of a StorHouse file extent	Configurable (but cannot exceed media restriction)
Size of an SQL statement	32,767 characters

The following table summarizes the limits for StorHouse database data types.

Item	Limit
BINARY	256 bytes
BLOB	2**31-9
CHAR	256 characters
CLOB	2**31-9
DATE	0001-01-01 to 9999-12-31
DOUBLE PRECISION	Sign bit, 11-bit exponent, 52-bit fraction
INTEGER	-2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31}-1$)
NUMERIC	1- 10^{31} to $10^{31}-1$
REAL	Sign bit, 8-bit exponent, 23-bit fraction
SMALLINT	-32,768 to 32,767
TIME	Hours from 00 to 24, Minutes from 00 to 59, Seconds from 00 to 62, Milliseconds from 000 to 999
TIMESTAMP	Year from 1 to 9999, Month from 1 to 12, Day from 1 to 28, 29, 30, 31, Hours from 00 to 24, Minutes from 00 to 59, Seconds from 00 to 62, Milliseconds from 000 to 999, Microseconds from 000000 to 999999
VARBINARY	32,705 bytes (256 for an indexed column)
VARCHAR	32,705 characters (256 for an indexed column)

B

System limits

System parameters for StorHouse/RM

This appendix describes the systems parameters that affect StorHouse/RM and explains how to:

- Display a system parameter value
- Set or change a system parameter value

Refer to the StorHouse *Command Language Reference Manual* for more information about all StorHouse system parameters.

About system parameters

System parameters are named data fields that StorHouse uses to manage resources and to provide default information for system operations. Each parameter has a name and a value. You typically choose values for many system parameters during system installation, but you can change them to configure and tune StorHouse for maximum performance at your site.

System parameter types

There are three types of system parameters:

- A *static* system parameter is one you cannot change while the StorHouse software is operating. Your FileTek customer support representative can assist you should you need to change a static parameter.

- A *dynamic* system parameter is one you can change while the StorHouse software is operating. Changes take effect immediately.
- A *deferred dynamic* system parameter is one you can change while the StorHouse software is operating, but changes do not take effect until after you restart the system.

System parameter access

Some system parameters you can display only, while others you can set and change. The system parameter descriptions starting on page C-4 indicate the type of user access available: SET, SHOW, or SET and SHOW.

- SET indicates you can change the parameter value with the SET SYSTEM command.
- SHOW indicates you can display the parameter value with the SHOW SYSTEM command.
- SET and SHOW indicate you can change and display the parameter value with the SET SYSTEM and SHOW SYSTEM commands.

System parameter list

The following table lists the system parameters that affect StorHouse/RM:

System parameter	Description	Page
LOG_SQL_STMT	User log control for SQL statements	C-4
LOG_SQL_TRANS	User log control for transaction statistics	C-4
SQL_BKUP_ACCOUNT	Metadata backup account	C-5
SQL_BKUP_FSET	Metadata backup file set	C-5
SQL_BKUP_GROUP	Metadata backup file access group	C-5
SQL_BKUP_LIMIT	Metadata backup file limit (maximum)	C-6
SQL_BKUP_VSET	Metadata backup volume set	C-6
SQL_BLD_INDX_MEM	Megabytes of memory for building indexes	C-6
SQL_DROP_HOLD	Minimum number of days a table must be dropped before it may be purged	C-7
SQL_HOLD_DATA	Days to hold data extents in buffer	C-7
SQL_HOLD_INDX	Days to hold index extents in buffer	C-8
SQL_HOLD_SPECIAL	Days to hold DF and map extents in buffer	C-8
SQL_INDX_TYPE	Default index type	C-9
SQL_LDR_ENGINES	Loader maximum engines	C-9
SQL_LDR_MAXINTO	Loader maximum INTO TABLE clauses	C-9
SQL_LDR_MAXLOAD	Loader maximum LOAD statements	C-10
SQL_MAX_EXT_DATA	Maximum size of a table data extent	C-10
SQL_MAX_EXT_HASH	Maximum size of a hash index data extent	C-11
SQL_MAX_EXT_VAL	Maximum size of a value index data extent	C-11
SQL_SESSIONS	Maximum number of connections	C-12

System parameter	Description	Page
STORHOUSE_REL	StorHouse/RM software release number	C-12
SYSTEM_ID	Identifier for a StorHouse system	C-12

System parameter descriptions

This section describes the subset of StorHouse system parameters that affect StorHouse/RM.

LOG_SQL_STMT

Enables or disables logging of SQL statements into the user log. TRUE enables logging. FALSE disables logging.

- Expanded Name: User log control for sql_statement records
- Type: Dynamic parameter
- Range: TRUE, FALSE
- Default: TRUE
- User Access: SET, SHOW

LOG_SQL_TRANS

Enables or disables logging of transaction statistics records into the user log. TRUE enables logging. FALSE disables logging.

- Expanded Name: User log control for sql_trans records
- Type: Dynamic parameter
- Range: TRUE, FALSE
- Default: TRUE
- User Access: SET, SHOW

SQL_BKUP_ACCOUNT

Specifies the account that the metadata backup utility uses to write metadata backup files.

- Expanded Name: Metadata backup account
- Type: Dynamic parameter
- Range: Any valid account identifier
- Default: SYSTEM
- User Access: SET, SHOW

SQL_BKUP_FSET

Specifies the file set where the metadata backup utility writes metadata backup files. SQL_BKUP_VSET specifies the volume set.

- Expanded Name: Metadata backup file set
- Type: Dynamic parameter
- Range: Any valid file set name
- Default: RMMDBKUP
- User Access: SET, SHOW

SQL_BKUP_GROUP

Specifies the file access group that the metadata backup utility uses to write metadata backup files.

- Expanded Name: Metadata backup group
- Type: Dynamic parameter
- Range: Any valid file access group name
- Default: RMMDBKUP
- User Access: SET, SHOW

SQL_BKUP_LIMIT

Indicates the maximum number of metadata backup file versions to keep for each database. Based on this value, StorHouse automatically deletes old backup file versions to accommodate new ones. Only the last metadata backup file version is used to recover metadata.

- Expanded Name: Metadata file limit (maximum)
- Type: Dynamic parameter
- Range: 1 through 99
- Default: 10
- User Access: SET, SHOW

SQL_BKUP_VSET

Specifies the volume set where the metadata backup utility writes metadata backup files. SQL_BKUP_FSET specifies the file set.

- Expanded Name: Metadata backup volume set
- Type: Dynamic parameter
- Range: Any valid volume set name
- Default: RMMDBKUP
- User Access: SET, SHOW

SQL_BLD_INDX_MEM

Specifies the number of megabytes of memory that can be used in building indexes during data loads. This memory is used to perform in-memory sorts of blocks of index entries. These blocks are written to disk and later merged to create the index. The larger the memory, the faster the sort but the fewer the loads that can be performed efficiently (without paging) in parallel. The product of this value and SQL_LDR_MAXLOAD should not exceed the virtual memory that can be allocated to data loading processes.

Sorts invoked during user queries will use the value of this parameter to determine the amount of memory used by the sort (the amount used = a minimum of 5 MB or the value of this parameter divided by 10).

- Expanded Name: SQL build index memory
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 10 to 200 MB
- Default: 100
- User Access: SET, SHOW

SQL_DROP_HOLD

Specifies the minimum number of days a table must be dropped before it may be purged with the PURGE TABLE statement or the -p option on the segment delete utility. A value of 0 indicates a purge may occur immediately after a delete operation.

- Expanded Name: SQL length of time to defer actual table drop actions
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 0 to 65000
- Default: 0
- User Access: SET, SHOW

SQL_HOLD_DATA

Specifies the number of days to hold table data extents in the performance buffer. A value for the HOLD parameter on the CREATE TABLE SPACE or ALTER TABLE SPACE statement overrides this system parameter for a subspace.

Note: Setting values for SQL_HOLD_DATA depends on your data usage patterns. Your FileTek customer support representative can help you set the appropriate value for your site.

- Expanded Name: SQL hold data extents
- Type: Dynamic parameter (affects all engines started after the change)

- Range: 0 to 32767
- Default: 0
- User Access: SET, SHOW

SQL_HOLD_INDX

Specifies the number of days to hold index data extents in the performance buffer. A value for the HOLD parameter on the CREATE TABLE SPACE or ALTER TABLE SPACE statement overrides this system parameter for a subspace.

Note: Setting values for SQL_HOLD_INDX depends on your data usage patterns. Your FileTek customer support representative can help you set the appropriate value for your site.

- Expanded Name: SQL hold index extents
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 0 to 32767
- Default: 0
- User Access: SET, SHOW

SQL_HOLD_SPECIAL

Specifies the number of days to hold DF and index map extents in the performance buffer. A value for the HOLD_SPECIAL parameter on the CREATE TABLE SPACE or ALTER TABLE SPACE statement overrides this system parameter for a subspace.

Note: FileTek recommends that you use very large retention periods for these extents, particularly if you plan to migrate your data to tape.

- Expanded Name: SQL hold special extents
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 0 to 32767
- Default: 0
- User Access: SET, SHOW

SQL_INDIX_TYPE

Specifies the default index type for CREATE INDEX statements.

- Expanded Name: SQL index type
- Type: Dynamic parameter (affects all engines started after the change)
- Range: HASH, VALUE, or RANGE
- Default: HASH
- User Access: SET, SHOW

SQL_LDR_ENGINES

Specifies the maximum number of concurrent StorHouse engines used for load operations. One engine is required per LOAD statement. Requests beyond the maximum limit are queued. The SQL_LDR_MAXLOAD system parameter has the same function as SQL_LDR_ENGINES. If the values differ, the smaller value is used.

- Expanded Name: SQL loader maximum engines
- Type: Deferred dynamic parameter (takes effect after system restart)
- Range: 1 through 99
- Default: 25
- User Access: SET, SHOW

SQL_LDR_MAXINTO

Specifies the maximum number of INTO TABLE clauses in any one LOAD statement. If the maximum number of INTO TABLE clauses is exceeded, the load operation fails.

- Expanded Name: SQL loader maximum INTO TABLE clauses
- Type: Deferred dynamic parameter (takes effect after system restart)
- Range: 1 through 99
- Default: 6
- User Access: SET, SHOW

SQL_LDR_MAXLOAD

Specifies the maximum number of LOAD statements that can be processed at a time. One engine is required per LOAD statement. Requests beyond the limit are queued. When the queue length exceeds twice the maximum, then new requests are rejected. The SQL_LDR_ENGINES system parameter has the same function as SQL_LDR_MAXLOAD. If the values differ, the smaller value is used.

- Expanded Name: SQL loader maximum LOAD statements
- Type: Deferred dynamic parameter (takes effect after system restart)
- Range: 1 through 99
- Default: 25
- User Access: SET, SHOW

SQL_MAX_EXT_DATA

For table data files, specifies the maximum data extent size (in megabytes) that StorHouse/RM writes to StorHouse. StorHouse/RM checkpoints each data extent when it reaches the maximum size and then creates a new one. StorHouse/RM can recover data extents to the last successful checkpoint. A value for the MAX_EXT_SIZE parameter on CREATE TABLE SPACE or ALTER TABLE SPACE overrides this system parameter for a subspace.

Note: Contact your FileTek customer support representative if you require a data extent size smaller than 100 MB. For a table data file to reach the maximum size (100 GB), set this value to the maximum size (800 MB).

- Expanded Name: SQL maximum extent data
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 100 through 800 MB
- Default: 400
- User Access: SET, SHOW

SQL_MAX_EXT_HASH

For hash index files, specifies the maximum data extent size (in megabytes) that StorHouse/RM writes to StorHouse. StorHouse/RM checkpoints each hash index data extent when it reaches the maximum size and then creates a new one. StorHouse/RM can recover index data extents to the last successful checkpoint. FileTek recommends that you try to contain a hash index in one data extent. Keep in mind, however, that the index must fit on the target device. A value for the MAX_EXT_SIZE parameter on CREATE TABLE SPACE or ALTER TABLE SPACE overrides this system parameter for a subspace.

- Expanded Name: SQL maximum extent hash
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 100 through 800 MB
- Default: 800
- User Access: SET, SHOW

SQL_MAX_EXT_VAL

For value index files, specifies the maximum data extent size (in megabytes) that StorHouse/RM writes to StorHouse. StorHouse/RM checkpoints each value index data extent when it reaches the maximum size and then creates a new one. StorHouse/RM can recover index data extents to the last successful checkpoint. FileTek recommends that you try to contain a value index in one data extent. Keep in mind, however, that the index must fit on the target device. A value for the MAX_EXT_SIZE parameter on CREATE TABLE SPACE or ALTER TABLE SPACE overrides this system parameter for a subspace.

- Expanded Name: SQL maximum extent value
- Type: Dynamic parameter (affects all engines started after the change)
- Range: 100 through 800 MB
- Default: 500
- User Access: SET, SHOW

SQL_SESSIONS

Specifies the maximum number of StorHouse engines that can run concurrently. This sets the maximum number of connections allowed system wide. Note that the maximum number of connections includes the number of loads (one engine per LOAD statement) plus the number of queries. Requests beyond the maximum limit are rejected (the error code is XENORES).

- Expanded Name: SQL maximum number of sessions
- Type: Deferred dynamic parameter (takes effect after system restart)
- Range: 1 through 99
- Default: 50
- User Access: SET, SHOW

STORHOUSE_REL

Specifies the release number of the current StorHouse/RM software in the form x.yz, where x is the version number, y is the release number, and z is a build number. If the value is a null string, StorHouse/RM software is not configured in the system.

- Expanded Name: StorHouse/RM software release number
- Type: Static parameter
- Range: Null string or StorHouse/RM release number
- Default: (Null string)
- User Access: SHOW

SYSTEM_ID

Indicates a unique system identifier for a StorHouse system. FileTek assigns the value. This identifier is part of a segment file name.

- Expanded Name: StorHouse system identification code
- Type: Static parameter
- Range: 6 decimal characters (value assigned by FileTek)

- Default: Value assigned by FileTek
- Access: SHOW

Displaying a system parameter value

You can display the current value of a StorHouse system parameter by using the StorHouse Command Language SHOW SYSTEM command.

▼ To display a system parameter value

Required privilege: SYSTEM or SHOW

1. Sign on to StorHouse.
2. At the StorHouse command prompt (?), type the following command to display a system parameter value and press Enter:

```
SHOW SYSTEM parameter_name
```

where parameter_name is the name of the system parameter. Some examples follow.

- To display the default index type (value, hash, or range), type:

```
SHOW SYSTEM SQL_INDX_TYPE
```

- To display the maximum number of LOAD statements that can be processed at a time, type:

```
SHOW SYSTEM SQL_LDR_MAXLOAD
```

Setting or changing a system parameter value

You can set or change the value of a StorHouse system parameter by using the StorHouse Command Language SET SYSTEM command. Depending on the type of system parameter (static, dynamic, dynamic deferred), the change takes effect immediately or after system restart.

▼ To set or change a system parameter value

Required privilege: SYSTEM

1. Sign on to StorHouse.
2. At the StorHouse command prompt (?), type the following command to change a system parameter value and press **Enter**.

```
SET SYSTEM parameter_name value
```

where:

- parameter_name is the name of the system parameter
- value specifies the new value for the system parameter

Some examples follow.

- To change the maximum number of concurrent StorHouse engines used by a FileTek data loader for load operations to 60, type:

```
SET SYSTEM SQL_LDR_ENGINES 60
```

- To set the number of days to hold special extents in the performance buffer to 365 days, type:

```
SET SYSTEM SQL_HOLD_SPECIAL 36
```

StorHouse explain tables

This appendix describes the columns in the explain facility tables. See Chapter 13, “Explain facility,” for more information about creating and querying explain tables.

About explain tables

Explain tables contain a description of the execution plan for a query. These tables are similar to system tables in that they are stored in a system tablespace and backed up by the metadata backup utility. They are different from system tables in that users create explain tables (whereas StorHouse/RM creates system tables) and different users own explain tables (whereas only SYSADM owns system tables).

The format of an explain table name is `owner.STH_EXPLAIN_name`, where:

- `owner` is the account that created the tables or the assigned owner (UID)
- `STH_EXPLAIN` is a prefix for all explain tables
- `name` is a descriptive title

For example, in a table called `TKA.STH_EXPLAIN_ID`:

- `TKA` is the account ID and the owner
- `STH_EXPLAIN` is the table prefix
- `ID` is the descriptive part of the table name

STH_EXPLAIN_ID

The STH_EXPLAIN_ID table contains the statement ID and corresponding execution plan ID. StorHouse/RM inserts a new row in this table when you submit an EXPLAIN PLAN statement.

Column name	Data type	Description
STMT_ID	CHAR(32)	Unique identifier specified on the EXPLAIN PLAN statement. This identifies the execution plan for a given query. If you omit the SET STATEMENT_ID clause on the EXPLAIN PLAN statement, the default is STH_EXPLAIN_DEFAULT.
STATEMENT_TIMESTAMP	TIMESTAMP	Date and time StorHouse/RM posted the explain data to the explain tables.
ID	INT	Execution plan ID for the STATEMENT_ID. You can use this execution plan ID to query the other explain tables in the set.

STH_EXPLAIN_PLAN

The STH_EXPLAIN_PLAN table lists the nodes in an execution plan.

Column name	Data type	Description
ID	INT	Execution plan ID. This ID matches the ID column in the STH_EXPLAIN_ID table.
NODE	SMALLINT	Node number of the execution node. StorHouse/RM assigns each execution node a unique node number. This number is used to correlate this row with expression and operator rows.
PAR_NODE	SMALLINT	Node number of the parent node. If the row is the root node, then the PAR_NODE column is NULL.
LVL	SMALLINT	Level number of the execution node in the execution plan. Level 1 is the root node.
LR	CHAR(1)	Type of node in relation to the parent node. Join nodes and union nodes have a left and right child node. All other nodes, with the exception of a leaf node, may have a left child node. Values are: L Left child of the parent R Right child of the parent – (hyphen) Root node



StorHouse explain tables

STH_EXPLAIN_PLAN

Column name	Data type	Description
EXPLAIN_PLAN	VARCHAR(150)	Description of the node operation. Valid values: PROJECT Project operation. This column also contains [GROUP BY (nn[-nn])] if the query contains a GROUP BY clause. The STH_EXPLAIN_EXPR table contains more information about the nn. RESTRICT Restrict operation. The STH_EXPLAIN_OPR table contains information about the restrict node operators. JOIN Inner join operation. This column also contains the join method: Nested loop join, Merge-Join, or Augmented-Nested loop join. OUTER JOIN Outer-join operation. This column also contains the join method: Nested loop join, Merge-Join, or Augmented-Nested loop join. TABLE SCAN Table scan operation. This column also contains the table name. INDEX SCAN Index scan operation. This column also contains the index name. SORT Sort operation. UNION Union operation.

STH_EXPLAIN_STMT

The STH_EXPLAIN_STMT system table contains the query, or SELECT statement, being explained.

Column name	Data type	Description
ID	INT	Execution plan ID. This ID matches the ID column in the STH_EXPLAIN_ID table.
ENTRY	SMALLINT	Row number. The first number is 0.
STMT	VARCHAR(100)	SELECT statement. Each STMT column can contain a maximum of 100 bytes. A SELECT statement can contain up to 32,767 bytes; therefore, a SELECT statement may be listed in multiple rows in the STH_EXPLAIN_STMT table. For example, a 1,000-byte SELECT statement is stored in 10 rows in this table. Excess white space in the statement may be eliminated when it is stored in this table.

STH_EXPLAIN_EXPR

The STH_EXPLAIN_EXPR table contains the following types of expressions that may appear in an execution plan: aggregate functions, scalar functions, column references, constants, and parameters.

Column name	Data type	Description
ID	INT	Execution plan ID. This ID matches the ID column in the STH_EXPLAIN_ID table.
ASSOC_NODE	SMALLINT	Node number that corresponds to the NODE column of the STH_EXPLAIN_PLAN table.

Column name	Data type	Description
ENTRY	SMALLINT	Unique value that identifies a specific row in this table for the ID and ASSOC_NODE. For instance, if the expression is a function that takes multiple arguments, then the ENTRY column is used to identify the arguments. For instance, ENTRY 1 identifies the first argument and ENTRY 2 identifies the second argument.
NODE	SMALLINT	Node number for this expression.
PROJ_TYPE	CHAR(5)	Type of projected column. Values are: <ul style="list-style-type: none"> ■ aggr – Aggregate function ■ sfunc – Scalar function ■ col – Column ■ const – Constant ■ param – Host variable parameter
PROJECT_COLUMN	VARCHAR(80)	Description of the projected column. <ul style="list-style-type: none"> ■ Name of the function (if this row represents a function) ■ table_name.column_name (if this row represents a column) ■ Value of the constant (if this row represents a constant) ■ ?? (if this row represents a parameter)

STH_EXPLAIN_OPR

The STH_EXPLAIN_OPR table contains more information about logical, relational, and arithmetic operators in an execution plan.

Column name	Data type	Description
ID	INT	Execution plan ID. This ID matches the ID column in the STH_EXPLAIN_ID table.
ASSOC_NODE	SMALLINT	Node number that corresponds to the NODE column of a restrict node or a join node in the STH_EXPLAIN_PLAN table.
ENTRY	SMALLINT	Unique value that identifies a specific row in this table for the ID and ASSOC_NODE.
NODE	SMALLINT	Node number of this operator.
PREDICATE	CHAR(14)	Type of operator. Values are: <ul style="list-style-type: none"> ■ EQ (=), NE (<>), LT (<), GT (>), LE (<=), GE (>=) – relational operators in restrict and join nodes ■ IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE – relational operators in restrict nodes only ■ EXISTS, NOT EXISTS – relational operators in join nodes ■ ANY and ALL operators ■ AND, OR, NOT – logical operators in restrict nodes only ■ : +, -, *, / – arithmetic operators
TBL	CHAR(1)	Type of table—expression or operator—that contains entries for the LEFT_SIDE and RIGHT_SIDE fields. Values are: <ul style="list-style-type: none"> ■ E – STH_EXPLAIN_EXPR table ■ O – STH_EXPLAIN_OPR table

D**StorHouse explain tables**

STH_EXPLAIN_OPR

Column name	Data type	Description
LEFT_SIDE	CHAR(11)	Character representation of the numeric value in the ENTRY column of the STH_EXPLAIN_EXPR table or the STH_EXPLAIN_OPR table for this ID and ASSOC_NODE.
RIGHT_SIDE	CHAR(11)	Character representation of the numeric value in the ENTRY column of the STH_EXPLAIN_EXPR table or the STH_EXPLAIN_OPR table for this ID and ASSOC_NODE.

Index

A

- access privilege 4-3
- access time factor (ATF) 5-7
- access to system parameters
 - SET C-2
 - SET and SHOW C-2
 - SHOW C-2
- accessing StorHouse from a
 - host database application 4-9
 - host language application 4-1, 4-10
 - terminal (Interactive Interface) 3-9
- account ID
 - definition 4-2
 - naming conventions 4-2
 - on INSERT statement 5-28
 - on UPDATE statement 5-29
- accounts (StorHouse)
 - creating 4-14
 - definition 4-1
 - disabling 4-15
 - general database user 4-8
 - loader 4-8
 - metadata backup 4-9, 10-3
 - password 4-2
 - PUBLIC 4-8
 - redo journaling 12-7
 - removing 4-27, 4-29
 - segment delete 14-2
 - SQL-enabled 4-8
 - SYSADM 4-7
 - unloader 4-9
- alias, database 2-21
- ALL privilege 4-6
- ALTER TABLE SPACE statement 5-32
- alternate database 11-5
- ANSI 1992 Level 2 SQL 1-2, 3-2
- application developer, role 3-2
- ARCHIVE command 3-4, 10-2
- assigning a default user tablespace 5-28
- ATF attribute 5-26
- auditing a redo journal 12-19

B

- backing up
 - metadata 10-1
 - segments 10-2
- BINARY data type 6-7
- BLOB data type 6-8

C

- changing

Index

D

- default user tablespaces 5-29
 - StorHouse access or command privileges 4-21
 - StorHouse account passwords 4-20
 - StorHouse system parameters C-14
 - volume and file sets in user tablespaces 5-32
- CHARACTER data type 6-8
- checkpoint, replay 12-6
- client data loader 1-5
- client/server environment feature 1-3
- CLOB data type 6-8
- column
- definitions 6-5
 - names in user tables 6-6
 - names in views 8-9
 - order in user tables 6-6
- column default values
- account ID 6-11
 - current date 6-11
 - current time 6-11
 - literal 6-11
 - NULL 6-11
- Command Language 3-4
- command privileges 4-4
- commands (StorHouse)
- ARCHIVE 10-2
 - CREATE ACCOUNT 4-14
 - CREATE BACKUP 10-2
 - CREATE FSET 5-23
 - CREATE VSET 5-23
 - EXTRACT DIRECTORY 10-5
 - REMOVE ACCOUNT 4-29
 - SCHEDULE 10-10, 14-6
 - SET ACCOUNT 4-20, 4-21, 4-22
 - SET SYSTEM 3-7, C-14
 - SHOW ACCOUNT 4-23
 - SHOW FILE 6-21, 7-16
 - SHOW FSET 6-21, 7-16
 - SHOW SYSTEM 3-7, 5-30, C-13
 - SHOW VSET 6-21, 7-16
- composite index 7-8
- compound index 7-8
- concurrent access feature of StorHouse/RM 1-2
- controlled access/database security feature 1-4
- conventions, SQL identifier 2-17
- CREATE ACCOUNT command 4-14
- CREATE BACKUP command 10-2
- CREATE FSET command 5-23
- CREATE INDEX statement 7-11
- CREATE VSET command 5-23
- creating
- accounts 4-14
 - file sets 5-23, 5-24
 - indexes for user tables 7-11
 - synonyms 9-3
 - user tables 6-15
 - user tablespaces 5-25
 - views from multiple tables 8-7
 - volume sets 5-23, 5-24
- creator
- user table 6-3
 - view 8-3
- ## D
- data extent 2-15

- data loader feature 1-3
- data types
 - compression 6-7
 - definition 6-6
 - limits B-2
 - types supported in StorHouse 6-7
- data value 6-2
- database
 - administrator 3-1
 - alias 2-21
 - component privileges 4-6
 - directory 2-5
 - integrity 1-4
 - local 1-3
 - names 2-17
 - naming conventions 2-17
 - privileges 4-4
 - remote 1-3
 - system components 2-5
 - user components 2-2
- database down utility 11-7
- database up utility 11-9
- DATE data type 6-8
- DB2 relational database system 1-2
- DBA privilege 4-5
- DDL statements 10-2
- default definition for a column 6-11
- default group 5-10
- default index type 7-9
- default storage specifications, displaying 5-30
- default user tablespace
 - assignment priorities 5-19
 - definition 5-19
 - for a database 5-28
 - for an account 5-28
 - for an index 5-20, 7-4, 7-5
- default values, user tablespace 5-26
- DEFAULT VTF value 5-9
- deferred dynamic system parameters C-2
- definitions
 - account 4-1
 - ATF 5-7
 - client data loader 1-5
 - Command Language 3-4
 - compound index 7-8
 - data extent 2-15
 - data type 6-6
 - data value 6-2
 - database alias 2-21
 - database directory 2-5
 - database system components 2-5
 - database user components 2-2
 - default user tablespace 5-19
 - deferred dynamic system parameter C-2
 - DF extent 2-15
 - dynamic system parameter C-2
 - EDC 5-9
 - extension (SQL) 3-2
 - extent 2-15
 - extraction 7-8
 - field 2-3
 - file access group 5-10
 - file set (FSET) 5-5
 - file set name 5-23
 - general database user account 4-5
 - hash index 7-4
 - hash index file 2-9
 - index 7-1
 - integrity constraint 6-7

Index**D**

- Interactive Interface 3-9
- local database 1-3
- map extent 2-15
- metadata 2-5
- metadata backup 10-1
- middleware 1-3
- not null 6-10
- null value 6-10
- object identifier 2-13
- object type 5-5
- out-of-line LOB 2-11
- primary journal file 12-2
- private synonym 2-5
- public synonym 2-5
- range index 7-5
- record 2-3
- redo journaling 12-1
- remote database 1-3
- secondary journal file 12-2
- segment 2-7
- segment files 2-7
- segment tag 2-20
- server data loader 1-5
- simple index 7-8
- static system parameter C-1
- StorHouse database 2-1
- subspace 5-2
- synonym 2-5, 9-1
- system parameter 3-7, C-1
- system table 2-6, A-1
- system table index 2-7
- system tablespace 2-5
- table data file 2-9
- tuple 2-3
- undo record 2-7
- user log 3-7
- user table 2-3, 6-1
- user table index 2-4
- user tablespace 2-3, 5-1
- user tablespace ID 5-21
- value index 7-2
- value index file 2-9
- view 2-4, 8-1
- volume set (VSET) 5-4
- VTF 5-8
- definitions (DF) extent 2-15
- DELETE privilege 4-6
- delimited SQL identifiers 2-18
- delta time 12-6
- devices, storage 1-2
- DIRECT VTF value 5-8
- DISABLED parameter modifier 4-15
- disabling an account 4-15
- displaying
 - column definitions 6-19
 - file set information 6-22, 7-17
 - information about indexes 7-13
 - information about synonyms 9-4, 9-5
 - information about user tables 6-18
 - information about user tablespaces 5-34
 - information about views 8-10
 - list of metadata backup files 10-16
 - segment information 6-23, 7-17
 - storage specifications for user tablespaces 5-34
 - StorHouse system parameters C-13
 - system default storage specifications 5-30
 - volume set information 6-21, 7-16
- DOUBLE PRECISION data type 6-8
- DROP INDEX statement 7-20
- DROP SYNONYM statement 9-6
- dropping
 - accounts 4-27

- indexes for user tables 7-20
- private synonyms 9-6
- public synonyms 9-7
- user tables 6-25
- user tablespaces 5-35
- view 8-12

dynamic system parameters C-2

E

EDC_COPY system parameter 5-10

EDC_INTERNAL system parameter 5-10

Embedded SQL Interface (ESQL) 1-3

engine 1-7

error detection code (EDC) 5-9

explain tables

- name format D-1
- privileges 13-9
- STH_EXPLAIN_EXPR D-5
- STH_EXPLAIN_ID D-2
- STH_EXPLAIN_OPR D-7
- STH_EXPLAIN_PLAN D-3
- STH_EXPLAIN_STMT D-5

exporting metadata backup files 10-5

extensions to StorHouse SQL 3-2

extents

- data 2-15
- definition 2-15
- DF 2-15
- map 2-15
- maximum size 5-10
- maximum size defaults 5-11
- retention defaults 5-12

- retention in performance buffer 5-11
- special 5-11

EXTRACT DIRECTORY command 10-5

extraction 7-8

extractor 1-3, 7-8

F

features of StorHouse/RM 1-2

field 2-3

file access group 5-10

file names

- journal 12-3
- segment 2-20
- UNIX 2-26

file naming conventions 2-20

file set name 5-23

File Transfer Protocol (FTP) 1-6

FileTek customer support representative, role 3-2

FileTek FTP Data Loader 1-6

FileTek MVS Data Loader utility 1-5

FSET (file set) 5-5

FTP server, StorHouse 1-6

G

gateway 1-3

general database user account 4-5

Index

H

GRANT statement 4-11, 4-17

granting

- database component privileges to accounts 4-17
- database privileges to accounts 4-16
- privileges to PUBLIC 4-12

group 5-10

H

hash index file 2-9

hash indexes

- and join processing 7-5
- default user tablespace 7-5
- definition 7-4
- how they are stored 7-5
- queries that use 7-4

HOLD tablespace parameter 5-26

HOLD_SPECIAL tablespace parameter 5-26

hybrid IN join operation 7-3

I

IDs

- account 4-2
- segment 2-19
- subsegment 2-19
- system 2-21
- system table 2-26
- system table index 2-26
- system tablespace 2-26
- temporary tablespace 2-26
- user table 2-19
- user table index 2-19

user tablespace 2-19

index file 2-9

index IDs

- conventions 7-10
- listing 7-14

index names

- conventions 7-10
- listing 7-14

INDEX privilege 4-6

indexes

- compound 7-8
- creating 7-11
- definition 2-4, 7-1
- displaying information 7-13
- dropping 7-20
- general rules 7-1
- hash 7-4
- listing IDs 7-14
- listing names 7-14
- owners 7-10
- range 7-5
- simple 7-8
- value 7-2

indexing feature 1-3

in-line LOBs 2-11

- setting the maximum length 6-12
- user tablespace used 6-14

INSERT privilege 4-6

INTEGER data type 6-8

integrity constraint 6-7

Interactive Interface 3-9

issuing

- SQL statements 3-9

StorHouse commands 3-9

J

journal chain 12-4

journal info file 12-4

L

large objects (LOBs)

BLOB data type 6-8

CLOB data type 6-8

INLINE clause 6-12

in-line LOBs 2-11

NOT INLINE clause 6-12

out-of-line LOBs 2-13

setting the maximum length for in-line data 6-12

sharing storage with another LOB column 6-14

storage options 6-12

STORE WITH clause 6-12

storing LOBs in a separate subsegment file 6-13

storing values in a different user tablespace 6-14

subsegment file 2-7

subsegment ID 2-19

TABLESPACE clause 6-12

limits for StorHouse/RM B-1

listing

accounts and their default tablespaces 5-29

database component privileges 4-26

database privileges 4-25

index names and IDs 7-14

metadata backup files 10-16

privileges assigned to PUBLIC 4-24

StorHouse account IDs 4-23

synonym names 9-5

table names and IDs 6-19

user tablespace names and IDs 5-33

view names 8-11

literal column default value 6-11

local database 1-3

LOG_FILE system parameter 3-8

LOG_SQL_STMT system parameter 3-8, C-4

LOG_SQL_TRANS system parameter 3-8

logs

system table 2-7

system table index 2-7

user 3-7

M

map extent 2-15

MAX_EXT_SIZE tablespace parameter 5-26

metadata 2-5, 10-1

metadata backup

backup file naming conventions 10-6

description 10-4

displaying a list of backup files 10-16

exporting backup files 10-5

file retention 10-7

files backed up 10-2

messages 10-12

scheduling 10-10

system parameters 10-3

what to do after 10-5

when to run 10-2

where files are stored 10-5

metadata restore utility

before using it 11-2

Index

N

- description 11-1
- error messages 11-11
- format 11-4
- output 11-3

middleware 1-3

N

naming a segment 2-20

naming columns in views 8-9

naming conventions

- for account IDs 4-2
- for account passwords 4-2
- for database user components 2-17, 2-18
- for databases 2-17
- for indexes 7-10
- for SQL identifiers 2-17
- for subspace numbers 5-21
- for synonyms 9-2
- for system tables A-2
- for user tables 6-4
- for user tablespaces 5-20
- for views 8-4
- in StorHouse 2-16

NEXT VTF value 5-9

not null, definition 6-10

NOW VTF value 5-8

NULL column default value 6-11

null value, definition 6-10

number, subspace 5-21

NUMERIC data type 6-9

O

object identifiers 2-13

object type 5-5

OBJECT_TYPE tablespace parameter 5-26

OIDs 2-13

operator account (UNIX) 3-10

OPERATOR command privilege 10-3

optimizing SQL 1-7

ordering columns in a user table 6-6

out-of-line LOBs 2-13

owner

- explain table 13-33
- index 7-10
- synonym 9-2
- user table 6-3
- view 8-3

P

package, database A-13

parsing SQL 1-7

password, account 4-2

performance buffer 5-8, 5-11

private synonym

- creating 9-3
- definition 2-5
- dropping 9-7

privileges in StorHouse

- access (SQLADMIN) 4-3

- command
 - SQLCOMMAND 4-4
 - SQLEXECUTE 4-4
 - database
 - DBA 4-5
 - RESOURCE 4-5
 - SCAN 4-6
 - database component
 - ALL 4-3, 4-6
 - DELETE 4-3, 4-6
 - INDEX 4-6
 - INSERT 4-6
 - SELECT 4-6
 - UPDATE 4-6
 - PUBLIC
 - access to system tables A-3
 - considerations 4-12
 - default user tablespace 5-28
 - definition 4-8
 - public synonym
 - creating 9-4
 - definition 2-5
 - dropping 9-7
 - purging user tables 6-28
- ## Q
- qualified table name 6-3
- ## R
- range indexes
 - and extractor processing 7-8
 - definition 7-5
 - how they are stored 7-8
 - queries that use 7-6
 - storing A-17
 - system tables A-17
 - REAL data type 6-9
 - record 2-3
 - redo journaling
 - applying a journal file 12-15
 - cycling a journal file 12-11
 - description 12-1
 - enabling journaling for a new database 12-9
 - enabling journaling for an unjournalized database 12-10
 - file location 12-8
 - file names 12-3
 - journal chain 12-4
 - journal info file 12-4
 - list of utilities 12-4
 - locking 12-9
 - privileges 12-7
 - replay checkpointing 12-6
 - remote database 1-3
 - REMOVE ACCOUNT command 4-29
 - removing
 - accounts 4-27
 - indexes 7-20
 - private synonyms 9-7
 - public synonyms 9-7
 - StorHouse access or command privileges 4-22
 - StorHouse accounts 4-27, 4-29
 - user tables 6-25
 - user tablespaces 5-35
 - views 8-12
 - Renaming a user table, view, or synonym 6-25, 8-12
 - replay checkpointing 12-6
 - RESOURCE privilege 4-5

Index

S

- restrictions on views 8-4
- retrieving data from StorHouse 1-7
- revoking
 - database component privileges 4-19
 - database privileges 4-18
- RMMDBKUP volume/file set 10-5
- roles for managing StorHouse
 - application developer 3-2
 - chief database administrator 3-1
 - departmental DBA 3-1
 - FileTek customer support representative 3-2
 - system administrator 3-1
- row 2-3
- rules for
 - SQL identifiers 2-18
 - StorHouse databases 2-27
- RUN command
 - general format 3-10
- S**
- SCAN privilege 4-6, 4-9
- SCHEDULE command 10-10, 14-6
- segment delete utility
 - description 14-1
 - locking 14-3
 - privileges 14-2
 - restarting 14-3
 - running 14-4
 - scheduling 14-6
 - ways to run the utility 14-2
- segment file 2-7
- segment ID 2-19
- segment tag 2-20
- segments
 - backing up 2-16
 - definition 2-7
 - file names 2-20
 - index file 2-9
 - invalidation 2-16
 - maximum size 2-8, B-2
 - naming 2-20
- SELECT privilege 4-6, 4-9
- SELECT statement
 - displaying a column definition 6-20
 - displaying information about an index 7-13
 - displaying information on tables 6-18
 - listing index IDs 7-15
 - listing index names 7-15
 - listing synonym information 9-4, 9-6
 - listing table IDs 6-19
 - listing table names 6-19
 - unloading data 1-6
- server data loader 1-5
- SERVICE command privilege 10-3
- SET access to system parameters C-2
- SET ACCOUNT command 4-20, 4-21, 4-22
- SET and SHOW access to system parameters C-2
- SET SYSTEM command 3-7, C-14
- setting storage parameters for a user tablespace 5-25
- setting up a new user tablespace 5-22
- SHOW ACCESS to system parameters C-2
- SHOW ACCOUNT command 4-23
- SHOW FILE /EXTENT command 6-24, 7-19

- SHOW FILE command 6-21, 7-16
- SHOW FSET command 6-21, 7-16
- SHOW SYSTEM command 3-7, 5-30, C-13
- simple index 7-8
- SMALLINT data type 6-9
- software components of StorHouse/RM 1-4
- specifying column defaults 6-11
- SQL identifiers 2-17, 2-18
- SQL statements, general
 - description 1-1
 - features of 1-2
 - how to submit 3-9
 - list for database administration 3-3
- SQL statements, specific
 - ALTER TABLE SPACE 5-32
 - CREATE INDEX 7-11
 - CREATE TABLE 6-16, 6-17
 - CREATE TABLE FROM DROPPED 6-26
 - CREATE TABLE SPACE 5-25, 5-26
 - CREATE VIEW 8-6
 - DELETE 6-5
 - DROP INDEX 7-20
 - DROP SYNONYM 9-6
 - DROP TABLE 6-15, 6-25
 - DROP TABLE SPACE 5-35
 - DROP VIEW 8-12
 - GRANT 4-11
 - INSERT 6-5
 - PURGE TABLE 6-28
 - RENAME 6-25, 8-12
 - REVOKE 4-18, 4-19
 - SELECT 4-25, 6-19, 7-13, 9-4, 9-6
 - UPDATE 5-29, 6-5
- SQL status codes 3-9
- SQL_BKUP_ACCOUNT system parameter 10-3, C-5
- SQL_BKUP_FSET system parameter 10-3, 10-5, C-5
- SQL_BKUP_GROUP system parameter 10-3, C-5
- SQL_BKUP_LIMIT system parameter 10-3, 10-7, C-6
- SQL_BKUP_VSET system parameter 10-3, 10-5, C-6
- SQL_BLD_INDX_MEM system parameter C-6
- SQL_DEFER_DROP system parameter 6-28
- SQL_DROP_HOLD system parameter C-7
- SQL_HOLD_DATA system parameter 5-12, C-7
- SQL_HOLD_INDX system parameter C-8
- SQL_HOLD_SPECIAL system parameter C-8
- SQL_INDX_TYPE system parameter 7-9, C-9
- SQL_LDR_ENGINES system parameter C-9
- SQL_LDR_MAXINTO system parameter C-9
- SQL_LDR_MAXLOAD system parameter C-10
- SQL_MAX_EXT_DATA system parameter 5-11, C-10
- SQL_MAX_EXT_HASH system parameter 5-11, C-11
- SQL_MAX_EXT_VAL system parameter 5-11, C-11
- SQL_SESSIONS system parameter C-12
- SQLADMIN privilege 4-3
- SQLCOMMAND privilege 4-4, 4-9
- SQLEXECUTE command privilege 4-4

Index

S

- SQLEXECUTE privilege 4-4, 4-9
- static system parameters C-1
- status codes for SQL statements 3-9
- STH default group ID 5-10
- STH_EXPLAIN_EXPR table D-5
- STH_EXPLAIN_ID table D-2
- STH_EXPLAIN_OPR table D-7
- STH_EXPLAIN_PLAN table D-3
- STH_EXPLAIN_STMT table D-5
- sthjou_archive utility 12-12
- sthjou_cycle utility 12-11
- sthjou_replay utility 12-15
- storage control attributes
 - ATF 5-7
 - EDC 5-9
 - VTF 5-8
- storage devices 1-2
- storage management feature 1-4
- storage specifications 5-4
- StorHouse
 - accounts 4-1, 4-14
 - Command Language 3-4
 - databases 2-1
 - file names 2-20
 - file versions 10-7
 - FTP server 1-6
 - Interactive Interface 3-9
 - system administrator 3-1
 - system parameters 3-7
 - user log 3-7
- StorHouse utilities
 - database up (sthdb_up) 11-9
 - journal archive (sthjou_archive) 12-12
 - journal cycle (sthjou_cycle) 12-11
 - journal replay (sthjou_replay) 12-15
 - list of 3-5
 - metadata backup (sthdb_backup) 10-1
 - metadata restore (sthdb_restore) 11-1
- StorHouse/Control Center, description xx
- StorHouse/RM
 - engine 1-7
 - features 1-2
 - position in network 1-2
 - software components 1-4
- StorHouse/SM 2-1
- STORHOUSE_REL system parameter C-12
- Structured Query Language (SQL) 1-1
- submitting
 - SQL statements 3-9
 - StorHouse commands 3-9
- subsegment files 2-7
- subsegment ID 2-19
- subspaces
 - definition 5-2
 - naming 5-21
 - selection during loads 5-18
- synonyms
 - creating 9-3
 - definition 2-5, 9-1
 - displaying information 9-4, 9-5
 - dropping 9-6
 - owners 9-2
 - private 9-2
 - public 9-2
 - removing 9-6

- renaming 9-6
- SYSADM account 3-11, 4-7
- SYSCOLAUTH system table 4-26, A-6
- SYSCOLUMNS system table 6-20, A-7
- SYSDATE column default value 6-11
- SYSDBAUTH system table 4-25, A-8
- SYSDROP_PEND system table A-10
- SYSINDEXES system table 7-14, A-11
- SYSPACKAGE system table A-13
- SYSPACKSTMT system table A-15
- SYSRANGES system tables A-17
- SYSSMUSERS system table 5-29, A-20
- SYSSTAT_COL system table A-21
- SYSSTAT_HIST system table A-22
- SYSSTAT_IDX system table A-22
- SYSSTAT_SMATRIX system table A-24
- SYSSTHFILES system table A-25
- SYSSTHSEGMENTS system table A-26
- SYSSTHSPACES system table A-27
- SYSSYNONYMS system table A-30
- SYSTABAUTH system table 4-26, A-31
- SYSTABLES system table 6-18, 6-19, A-33
- SYSTBLSPACES system table A-35
- system administrator, role 3-1
- SYSTEM command privilege 10-3
- system components, database 2-1
- system ID 2-21
- system limits B-1
- system parameters (StorHouse), general
 - changing C-14
 - deferred dynamic C-2
 - definition C-1
 - displaying C-13
 - dynamic C-2
 - SET access C-2
 - SET and SHOW access C-2
 - SHOW access C-2
 - static C-1
- system parameters (StorHouse), specific
 - EDC_COPY 5-10
 - EDC_INTERNAL 5-10
 - LOG_FILE 3-8
 - LOG_SQL_STMT 3-8, C-4
 - LOG_SQL_TRANS 3-8, C-4
 - SQL_BKUP_ACCOUNT 10-3, C-5
 - SQL_BKUP_FSET 10-3, C-5
 - SQL_BKUP_GROUP 10-3, C-5
 - SQL_BKUP_LIMIT 10-3, C-6
 - SQL_BKUP_VSET 10-3, C-6
 - SQL_BLD_IDX_MEM C-6
 - SQL_DROP_HOLD C-7
 - SQL_HOLD_DATA 5-12, C-7
 - SQL_HOLD_IDX 5-12, C-8
 - SQL_HOLD_SPECIAL 5-12, C-8
 - SQL_IDX_TYPE C-9
 - SQL_LDR_ENGINES C-9
 - SQL_LDR_MAXINTO C-9
 - SQL_LDR_MAXLOAD C-10
 - SQL_MAX_EXT_DATA 5-11, C-10
 - SQL_MAX_EXT_HASH 5-11, C-11
 - SQL_MAX_EXT_VAL 5-11, C-11
 - SQL_SESSIONS C-12
 - STORHOUSE_REL C-12
 - SYSTEM_ID C-12

Index**T**

system table index 2-7

system table index log 2-7

system table log 2-7

system tables, general

- definition 2-6, A-1
- list A-4
- name format A-2
- privileges A-3
- UNIX file name 2-26

system tables, specific

- SYSCOLAUTH 4-26, A-6
- SYSCOLUMNS A-7
- SYSDBAUTH 4-25, A-8
- SYSDROP_PEND A-10
- SYSINDEXES 7-13, A-11
- SYSPACKAGE A-13
- SYSPACKSTMT A-15
- SYSRANGES A-17
- SYSSMUSERS 5-29, A-20
- SYSSTAT_COL A-21
- SYSSTAT_HIST A-22
- SYSSTAT_IDX A-22
- SYSSTAT_SMATRIX A-24
- SYSSTHFILES A-25
- SYSSTHSEGMENTS A-26
- SYSSTHSPACES A-27
- SYSSYNONYMS A-30
- SYSTABAUTH 4-26, A-31
- SYSTABLES 6-19, A-33
- SYSTBLSPACES A-35
- SYSVIEWS 8-10, 8-11, A-36

system tablespace 2-5

SYSTEM_ID system parameter C-12

SYSTIME column default value 6-11

SYSVIEWS system table 8-10, 8-11, A-36

T

table data file 2-9

table IDs 6-4

table names 6-3

tables

- explain D-1
- system A-1
- user 6-1

tablespace IDs 5-21

tag, segment 2-20

TIME data type 6-9

TIMESTAMP data type 6-9

transferring data to StorHouse 1-7

troubleshooting

- metadata backup 10-12
- metadata restore 11-11

tuple 2-3

U

UID 13-33

undo record 2-7

undropping user tables 6-26

UNIX file names 2-26

unloader 1-3, 1-6

UPDATE privilege 4-6

UPDATE statement 5-29

updating default user tablespaces 5-29

USER column default value 6-11

user log 3-7

user tables

column data types 6-6

column definitions 6-5

column order 6-6

creating indexes 7-11

creator 6-3

definition 2-3, 6-1

dropping 6-25

dropping indexes 7-20

example 2-3

owners 6-3

purging 6-28

renaming 6-25, 8-12

undropping 6-26

what's different about StorHouse user tables 6-5

user tablespaces

altering 5-32

creating 5-25

default priorities 5-19, 5-20

default values 5-26

definition 2-3, 5-1

dropping 5-35

examples 5-12

naming conventions 5-20

storage specifications in 5-4

subspaces in 5-2

UTC 12-6

utilities, StorHouse

database down (sthdb_down) 11-7

database up (sthdb_up) 11-9

journal archive (sthjou_archive) 12-12

journal cycle (sthjou_cycle) 12-11

list of 3-5

metadata backup (sthdb_backup) 10-1

metadata restore (sthdb_restore) 11-1

sthjou_replay 12-15

V

value index file 2-9

value indexes

and join processing 7-3

and segments 7-4

default user tablespace 7-4

how they are stored 7-4

queries that use 7-3

VARBINARY data type 6-9

VARCHAR data type 6-9

versions, StorHouse file 10-7

views

creating 8-6

creating from multiple tables 8-7

creator 8-3

definition 2-4, 8-1

how they work 8-1

naming columns 8-9

owners 8-3

renaming 8-12

restrictions 8-4

volume set name 5-23

VSET (volume set) 5-4

vulnerability time factor (VTF) 5-8



W

WITH GRANT OPTION 4-11