



Release Notes for StorHouse/RM 3.3

StorHouse/RM Release 3.3

Publication Number
900126 Rev. N

August 9, 2004

The FileTek logo consists of the word "FileTek" in white, bold, sans-serif font, set against a solid teal square background.



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Copyright © 1996-2004 FileTek, Inc. As an Unpublished Licensed Work.
Publication Number: 900126 Rev. N

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 Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited.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/manufacture 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	vii
Intended audience	vii
Contents of document	vii
Related documentation	viii
 Chapter 1: Changes and enhancements	 1-1
System requirements	1-1
Utility updates	1-2
Renamed utilities	1-2
Changed utilities	1-2
Redo journaling	1-3
Journal file	1-4
Primary and secondary journal files	1-4
Journal file status	1-5
Journal file names	1-5
Redo journaling utilities	1-6
Ways to run redo journaling utilities	1-6
Privileges for running redo journaling utilities	1-7
Location of the redo journal	1-7
Locking	1-8
Enabling journaling for a new database	1-9
Enabling journaling for an unjournaled database	1-9
Cycling journal files	1-10
Archiving and purging journal files	1-11

Replaying a journal file	1-13
Segment delete utility	1-15
Before deleting segments	1-16
Ways to run the segment delete utility	1-16
Privileges required to run the segment delete utility	1-17
Locking	1-17
Running the segment delete utility	1-18
Running multiple delete processes	1-19
Restarting an interrupted process	1-19
After deleting segments	1-20
Explain facility	1-20
Execution tree	1-20
Nodes	1-21
Node operations	1-21
Project	1-23
Restrict.	1-23
Join	1-23
Sort.	1-23
Union.	1-23
Table scan	1-23
Index scan	1-24
Expression tree	1-24
Explain facility SQL statements	1-25
Explain facility result tables	1-26
Explain privileges	1-26
Explain example	1-27
Step 1: Run the explain facility	1-27
Step 2: Obtain the execution plan ID	1-27
Step 3: Display the execution plan	1-28
Step 4: Display the expression for the project node	1-29
Data type changes	1-29
ODBC changes	1-30
ESQL compilers	1-31
Solaris	1-31

HP-UX	1-31
AIX	1-31
Data loader changes	1-32
New ESCAPED BY clause	1-32
Format of ESCAPED BY clause	1-33
Example ESCAPED BY clause	1-34
Enhanced pending load check	1-34
Data unloader changes	1-35
New ESCAPED BY clause	1-35
New RECORDS NOT TERMINATED clause	1-35
Removed reserved words	1-35
Limit changes	1-36
New SQL codes	1-36

Chapter 2: Special considerations 2-1

SQL code -301031	2-1
SQL code -30033	2-1
DESCRIBE BIND restrictions	2-2
Design advisory for join operations	2-2
ISQL product status	2-2
DDL processing in general	2-3
Host variables as BINARY, VARBINARY, and VARCHAR data types	2-3
Immediate restart after a load failure	2-3
LOB restrictions	2-4
ESQL	2-4
SQL	2-4
Use of SYS in table names	2-4
Use of control characters as delimiters	2-4



Contents

Welcome

The *Release Notes for StorHouse/RM 3.3* identifies changes, enhancements, and special considerations for StorHouse/RM release 3.3.

Intended audience

This document is intended for StorHouse/RM users who are familiar with the StorHouse/RM software and for new users who want a summary of changes.

Contents of document

This publication contains the following chapters:

- Chapter 1, “Changes and enhancements,” summarizes updates and new features in StorHouse/RM release 3.3.
- Chapter 2, “Special considerations,” describes issues that may, in certain environments or fields of use, require careful review during assessment of an application’s use of StorHouse/RM at this time.

Related documentation

Refer to the following documents for information about StorHouse/RM.

- The *StorHouse SQL Reference Manual*, publication number 900111, describes the SQL statements, predicates, and functions supported by StorHouse®.
- The *StorHouse SQL Quick Reference*, publication number 900122, provides a summary of the material in the *StorHouse SQL Reference Manual*.
- The *StorHouse Database Administration Guide*, publication number 900108, describes StorHouse database concepts and explains how to create user tables and indexes, manage accounts and privileges, set up user tablespaces, and perform other StorHouse database administration tasks.
- The *StorHouse ESQL Manual*, publication number 900121, explains how to use StorHouse SQL in application programs.
- 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, explains how to load data into StorHouse user tables from UNIX®, VAX, or other hosts using your standard File Transfer Protocol (FTP) client software.
- The *FileTek FTP Data Unloader Manual*, publication number 900137, explains how to unload data from StorHouse databases using FTP. It describes the UNLOAD control statement you prepare to format result data, the SELECT statement you prepare to select the data to unload, and the subset of FTP commands you use to transfer control information and to receive result data.
- The *StorHouse/RM Metadata Conversion Manual*, publication number 900142, explains how to convert metadata from one StorHouse/RM release to another.



- The *StorHouse/RM Glossary*, publication number 900112, defines the terminology used in the StorHouse/RM User Document Set.



Welcome

Related documentation

Changes and enhancements

This chapter describes the changes and enhancements to StorHouse/RM for release 3.3. Significant new features are as follows:

- Redo journaling, which supplements the metadata backup and restore utilities in capturing and restoring transactions that affect metadata
- Segment delete utility, which removes invalidated segments and associated objects (for example, index files and entries in system tables) from StorHouse
- Explain facility, which enables you to examine the strategy, or execution plan, created by the optimizer for a given query

System requirements

StorHouse/RM release 3.3 requires the following:

- StorHouse/SM release 5.4 (delivery 51) or a later StorHouse/SM release
- UNIX operating system environment/platform:
 - Sun™ Solaris™ 2.6, 8, or 9 on Sparc systems
 - HP-UX 11.x on PA-RISC systems
 - IBM AIX 5.2 on pSeries servers

See “ESQL compilers” on page 1-31 for information about supported ESQL compilers.

Utility updates

Some existing utilities were renamed and modified in StorHouse/RM 3.3.

Renamed utilities

The commands of the following utilities were renamed to use a consistent naming scheme.

Utility command changes

Utility	Old command	New command
Database down	dbdown	sthdb_down
Database up	dbup	sthdb_up
Metadata backup	meta_bkup	sthdb_backup
Metadata restore	meta_rstr	sthdb_restore

Changed utilities

The -d option letter is no longer required on the database down (sthdb_down) and database up (sthdb_up) utilities. StorHouse/RM, however, accepts -d for compatibility with previous releases.

The -n option is available on the database down (sthdb_down) and database up (sthdb_up) utilities. Include this option to run the utility in non-interactive mode (no prompting).

Additional options are now available on the database creation (syscreate) and metadata backup (sthdb_backup) utilities to support redo journaling.

Command options for existing utilities

Utility	Options	Description
syscreate	-j or -J	Enable journaling for a new database
sthdb_backup	-f or -F	Ignore the data loader checkpoint status
	-h or -H	List command usage information
	-j or -J	Enable journaling for an existing database
	-n or -N	Run in non-interactive mode
	-v or -V	Receive detailed status messages
	-s or -S	Set values for VSET, FSET, and VTF. If you omit this option, the utility uses StorHouse system parameters to obtain the values. The format of {sm_options} is:
	{sm_options}	VSET=name,FSET=name,VTF=NOW DIRECT NEXT For example: -s VSET=MYVSET,FSET=MYFSET,VTF=NOW

Redo journaling

StorHouse/RM 3.3 supports *redo journaling*—a set of utilities for capturing, storing, and restoring transactions that affect metadata. Redo journaling is used in conjunction with the metadata backup and restore utilities to restore transactions that affect the metadata since the last metadata backup.

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 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	1-9
Enable journaling for an unjournaled database	sthdb_backup	1-9
Close the current journal file and start a new one	sthjou_cycle	1-10
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	1-11
Apply all journal files to finish restoring a database	sthjou_replay	1-13

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 SCHEDULE 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. 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. The `syscreate` format is:

```
syscreate [-j | -J] database_name
```

For example, to enable journaling for a new StorHouse database called `CUSTOMERDB`:

```
$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.

For example, to back up a database called `CUSTOMERDB` and to enable journaling:

```
run sthdb_backup -j CUSTOMERDB
```

In this example, the metadata backup utility terminates if any loads are active (there's no `-f` option on the command statement). In this case, no backup is created and journaling is not started for the database.

Cycling journal files

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 becomes large or at a preset interval (cron job). 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.
-h -H	Option to display a description of the <code>sthjou_cycle</code> utility and its command syntax.
database_name	(required) Name of the StorHouse database with journaling enabled. When running the utility with the StorHouse RUN command, enclose a database name with lowercase characters in double quotes.

For example, to cycle journal files for the Calls database using the -v option:

```
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 in addition to archiving journal files. Only journal files that have been successfully written to StorHouse and verified as safe on StorHouse are purged. N.nnnn is the age criteria for the disk journal file, where N (required) is days and .nnnnn (optional) is a fraction of a day. For example, -p 1 indicates to purge disk journal files that were archived at least one day ago.</p>
-o -O	<p>Option to only purge disk journal files that have been previously written to StorHouse. No archiving occurs.</p>
-c N -C N	<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, 2, or 3. 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 a file exists on StorHouse but it's not important where, for example, only in the performance buffer.</p>

Argument	Description
database_name	(required) Name of the StorHouse database with journaling enabled. When running the utility with the StorHouse RUN command, enclose a database name with lowercase characters in double quotes.

Some examples follow.

- To archive journal files to a VSET called v2003 and an FSET called f2003 for a database named CUSTOMERDB:

```
run sthjou_archive -s vset=v2003,fset=f2003 CUSTOMERDB
```

- To archive journal files to the default VSET and FSET (omit the -s option) and to purge journal files that were archived at least seven days ago and have one copy on StorHouse:

```
run sthjou_archive -p 7 -c 1 CUSTOMERDB
```

- To purge all archived disk journal files:

```
run sthjou_archive -O CUSTOMERDB
```

Replaying a journal file

The journal replay utility (sthjou_replay) applies all journal files created since the last metadata backup. 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.

Note: Contact FileTek customer support before restoring a database and replaying a redo journal.

1

Changes and enhancementsRedo journaling

The format of the `sthjou_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.
-h -H	Option to display a description of the <code>sthjou_replay</code> utility and its command syntax.
database_name	(required) Name of the StorHouse database with journaling enabled.

For example, to replay the redo journal 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\  
sthjou_replay has completed successfully
```

Segment delete utility

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 (SYSSTHFILES, SYSSTHSEGMENTS, range index tables, and so on). Use this facility to mass-remove invalidated segments and their associated objects from a StorHouse database. An *invalidated segment* is one that has been replaced or made obsolete through a FileTek data loader merge, or coalesce, operation.

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 be 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 with StorHouse/Admin and with a FileTek data loader (REPLACE SEGMENT clause or a MERGE (COALESCE) operation).

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

Privileges required to run the segment delete 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 the segment delete utility

You can 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 -d or -D. All options have a default.
-d or -D days	Minimum number of days a segment has been invalid. The default is 14 days.
-t or -T tblid[,tblid]...	Table Id(s) to delete invalidated segments from specific user tables. The table ID is the ID value from the SYSTABLES entry for that table (that is, it is not the table name). The default is all user tables in the database.
-l or -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 or -V verbose_level	Level of message detail (echo activity to stdout). Values of verbose_level are 0 (minimal messaging), 1, or 2 (most verbose). The default is 0.
database_name	(required) Name of the StorHouse database. When running the utility with the StorHouse RUN command, enclose a database name with lowercase characters in double quotes.

Some examples follow.

- To delete all segments that have been invalidated for at least 10 days in database called Customer:

```
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:

```
run sthseg_delete -t 158 SALES
```

Running multiple delete processes

The segment delete 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 updates 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.

Explain facility

With the new StorHouse/RM *explain facility*, you can examine the strategy, or *execution plan*, used by the optimizer for a specific 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

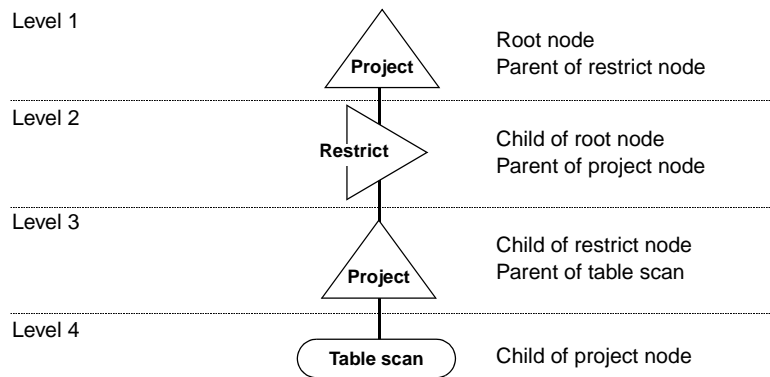
There are six types of nodes, one for each of the six primitive data operations:

- Project
- Restrict
- Join
- Sort
- Union
- Table scan
- Index scan

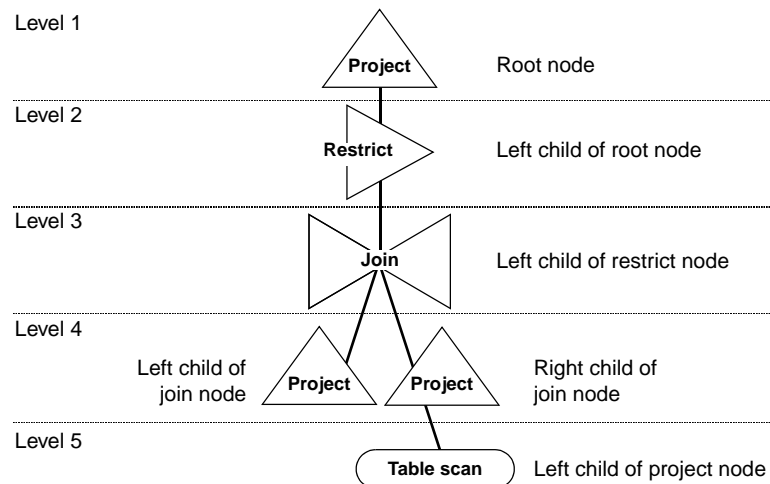
Changes and enhancements

Explain facility

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

Expressions and operators in an expression tree (continued)

Type	Description
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. You can submit the explain facility SQL statements using any method supported by StorHouse. The SQL statements for the explain facility are as follows.

Explain statements

Statement	Description
CREATE EXPLAIN TABLES	Create a set of empty explain tables to hold result data
DROP EXPLAIN TABLES	Remove a set of explain tables
EXPLAIN PLAN	Specify the query to be explained and run the explain facility to insert the results into the explain tables
SELECT	Query the explain tables

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.

Explain 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 EXPLAIN TABLES and DROP EXPLAIN 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 example

This example analyzes a query without a predicate. The user table is named 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;
```

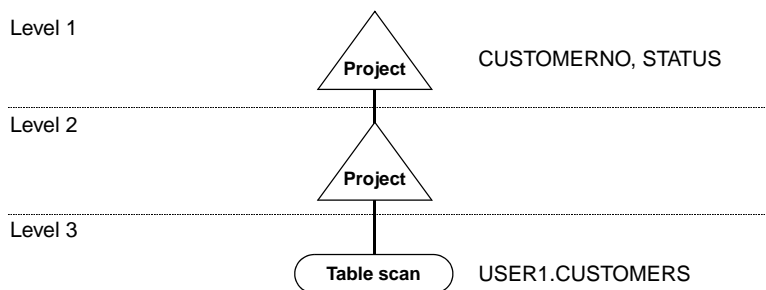
The result table is as follows. This example execution plan consists of three nodes. The table scan node indicates an index is not used to execute the query.

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

Column descriptions are as follows.

- 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;
```

Data type changes

Data type changes are as follows:

- The conversion of CHAR to/from BINARY/VARBINARY literals and columns is no longer supported.
- The TO_CHAR function supports the BLOB data type as the input expression.
- For any arithmetic operation or comparison, if any operand is of type REAL or DOUBLE, the result (or comparison) is REAL or DOUBLE.

- The SUBSTR function supports BINARY and VARBINARY data types for its first argument. The result data type is the same as the first argument. This change was implemented in release 3.2 build 02 but is documented in the StorHouse/RM 3.3 user documentation.
- Conversions from CHAR data types to TIMESTAMP data types have been extended to allow strings formatted as TIME and DATE values. Support has also been added for conversion of TIME data types and TIME literal values to the TIMESTAMP data type.

ODBC changes

StorHouse/RM uses the StorHouse/ODBC driver, which is a separate product with its own release cycle. The newest release of the StorHouse/ODBC driver, release 2.15, includes the following changes:

- LOB data types are supported and appear as ODBC LONG data types.
- SQL statements can be longer than 10,000 characters, now up to 32,767 characters.
- A driver specification is now supported in the ODBC connect string. In Windows, the DRIVER attribute tells the Windows Driver Manager to use the StorHouse/ODBC driver. In UNIX, this attribute can be used for compatibility with the Windows specification and to indicate that the ODBC.ini file should not be used.

You can use the release 2.14 ODBC driver with either StorHouse/RM 3.2 or 3.3, however, that driver doesn't support LOB data types or statements longer than 10,000 characters.

ESQL compilers

The following changes were made to the ESQL compilers.

Solaris

For Solaris, application programs can now be compiled and linked with a C compiler, specifically with SUNWspro cc (version 4.2, 5.0, or later) or gcc (current version). Previously the link step of ESQL program preparation required the SUNWspro 5.0 CC compiler. Note that for the C compilers, you must explicitly add the library `/usr/lib/libCrun.so.1` to the link command. For the g++ compiler, the `LD_LIBRARY_PATH` must include the path to the g++ libraries.

HP-UX

For HP-UX, the aCC compiler (with the `-AA` option) is supported for compile or link. The cc compiler, with the `-Ae` option, is supported but only for the compile step. The aCC compiler with the `-AA` option must always be used for linking. Use of other compilers or use of the aCC compiler without `-AA` is not supported.

AIX

For AIX, to compile and link an ESQL program containing C++ syntax, the ESQL source file should have a file extension of `.pcx`. In this case, ESQL generates an intermediate file with a file type `.CPP`, which is accepted by the AIX x1C C++ compiler as a source file containing C++ syntax. (The `.pcx` file extension is also accepted on non-AIX platforms, although it is not necessary.)

You can use the C++ compiler, x1C, to compile both C and C++ programs. The compiler determines whether the source file is written in C or C++ based on the file extension. A program containing C++ syntax with a source file extension of `.c` (the file extension for intermediate files generated by the ESQLC preprocessor) causes the compiler to issue errors and warnings about the C++ syntax.

The following example sets the default value for the environment variable and invokes the ESQL compiler for an ESQL program on AIX.

```
setenv ESQL_CC xlc -g -c  
esqlc mysource.pcx -o myexecutable
```

Data loader changes

Two enhancements were made to the The FileTek FTP Data Loader and the FileTek MVS Data Loader.

New ESCAPED BY clause

If data fields are delimited and the data itself contains the delimiter character, an *escape character* informs the FileTek data loader to interpret the delimiter character as data. The escape character typically precedes the data value. You use the ESCAPED BY clause to specify the escape character to use for loading data into StorHouse.

Format of ESCAPED BY clause

ESCAPED BY [DELIMITER | 'char' | NONE]

Argument	Description
DELIMITER	<p>(optional) Keyword to use the enclosure delimiter (for example, specified on the FIELDS clause) as the escape character. The FileTek data loader interprets a doubled (two consecutive occurrences without whitespace between them) delimiter as a single data character that is one instance of that delimiter. The FileTek data unloader doubles an enclosure delimiter found in the data stream. This is the default if the data is enclosed by delimiters and you omit an ESCAPED BY option.</p> <ul style="list-style-type: none">■ If a field starts with a doubled delimiter, the first one is still considered the start delimiter (even if OPTIONALLY ENCLOSED).■ If the start and end enclosure delimiters are different, a doubled start delimiter (after the delimiter that starts the field) is also changed to a single delimiter. It is not necessary to escape such an instance of the start delimiter since any start delimiter encountered after the start of field is always considered a data character and remains in the data.■ If a field is OPTIONALLY ENCLOSED BY '"', a single '"' in data that doesn't start with a '"' is interpreted as data. A doubled delimiter in this case is not converted to a single '"'
'char'	<p>(optional) Value of the terminator or enclosure delimiter to use as the escape character, for instance, '\</p> <ul style="list-style-type: none">■ The 'char' value cannot be a whitespace character.■ If a field is TERMINATED BY WHITESPACE, an escaped whitespace character is not interpreted as part of the delimiter, that is, it is skipped or trimmed.
NONE	<p>(optional) Keyword to not specify an escape character. An error occurs if there are any delimiters in the data. This is the default if the data is terminated by delimiters and you omit an ESCAPED BY option.</p>

Note the following:

- The DELIMITER option is the default for enclosure delimiters in ENCLOSED data and NONE is the default for terminators.
- You cannot use the ESCAPED BY clause to escape a newline. In this case, the data loader returns a short record error.
- When you use the ESCAPED BY NONE option, the data loader interprets four consecutive delimiters as two empty fields, rather than one field containing a single data character that is the delimiter character. For example, """" (with " the enclosing delimiter) is two data fields, each of them empty. Without BY NONE, the data loader interprets this data pattern (""""") as one field that contains a single quote (").

The C-style escape sequences `\r`, `\n`, and `\t` are now allowed in the control file (not the data) for convenience in specifying the characters CR, LF, and TAB as delimiters. For example, you could specify TERMINATED BY `'\t'` in a CONSTANT string. The C-style escape `'\0'` is not supported.

Example ESCAPED BY clause

To load a data file that was unloaded from Informix (using the default terminator):

```
LOAD ESCAPED BY '\ ' FIELDS TERMINATED BY '|' INTO TABLE ...
```

Enhanced pending load check

The method for checking for pending data loads was enhanced. Previously, the metadata backup test for pending loads excluded any active load operations. Now it doesn't.

Data unloader changes

The FileTek FTP Data Unloader provides two new clauses.

New ESCAPED BY clause

The ESCAPED BY clause enables you to specify the escape character to use for unloading data from StorHouse. If a delimiter is found in the data that is being unloaded, the data unloader adds the escape character before it outputs the data character. The format of the ESCAPED BY clause is the same as described at “Data loader changes” on page 1-32. Note that when ESCAPED BY DELIMITER is used, the data unloader ensures that consecutive zero-length fields do not result in adjacent (doubled) terminators by inserting a blank in output data.

New RECORDS NOT TERMINATED clause

The RECORDS NOT TERMINATED clause enables you to suppress the terminator for the last field in each record, leaving only the newline at the end of the record. The format of the clause is:

RECORDS NOT TERMINATED

Removed reserved words

The following words are no longer reserved in SQL statement syntax:

- CONNECT
- LINK
- NUMBER

Limit changes

The limits on the number of columns in a table and the length of an SQL statement changed as follows:

- The maximum number of columns in a table is now 1,024 (previously 500).
- The maximum length of an SQL statement is now 32,767 characters (previously 10,000).

If you want to submit longer SQL statements, you must use the StorHouse/RM release 3.3 of ISQL and the 2.15 release of ODBC. You may also have to rebuild any ESQLC applications.

New SQL codes

The following SQL codes are new.

- 10306 Illegal attempt to acquire duplicate lock.
- 20151 Illegal zero length SQL statement.
- 20152 Invalid EXPLAIN STATEMENT_ID.
- 20153 Illegal EXPLAIN SQL statement.
- 20154 Illegal explain user.
- 20155 EXPLAIN STATEMENT_ID already in use.
- 20156 Error dropping explain tables.
- 20157 Explain tables do not exist.
- 20158 EXPLAIN UID already in use
- 20159 Invalid EXPLAIN UID.
- 85001 Error opening Redo Journal file.
- 85002 Journal CRC function failure. CRC status undefined.
- 85003 Error creating Redo Journal lock.
- 85004 Internal Error: error enabling journaling. Backup Database!
- 85005 Read beyond the Journal EOF. Journal incomplete or corrupt.
- 85006 Error reading Journal.

- 85007 Fatal error while Journaling. Write operations disabled.
- 85008 Error creating Redo Journal.
- 85009 Journal CRC failure, indicates corruption in journal file.
- 85010 Journal error, Error performing SM operation.
- 85011 Journal error, unable to locate SM file in storage machine.
- 85012 Journal error, missing journal file.
- 85013 Journal error, unprocessed data found in journal file.
- 85014 Journal error, incorrect or inaccessible sthdb environment.
- 85015 Journal error, error producing runtime statistics.
- 85016 Journal error, unknown record type encountered in journal.
- 85017 Journal error, tmp directory not found.
- 85018 Journal error, error purging journal file.
- 85019 Journal error, error processing replay cache file.
- 85020 Journal error, replay table operation failure.
- 85021 Journal error, general replay failure - see alog.
- 85022 Journal error, uninitialized journal object.
- 85023 Journal error, journaling not enabled.
- 85024 Journal error, replay required before database can be accessed.
- 85025 Journal error, illegal replay requested.
- 85100 Warning: one of more Redo Journal files may be corrupt.

1

Changes and enhancements

New SQL codes

Special considerations

This chapter highlights known issues that may, in certain environments or fields of use, require careful review during assessment of an application's use of StorHouse/RM at this time. If applicable to your environment, please discuss these issues with your FileTek systems engineer to explore possible design alternatives.

SQL code -301031

For some StorHouse/RM processing errors, SQL code -301031 (transaction aborted) may be mistakenly returned instead of the correct error code. Usually, this is caused by an out-of-space condition in a volume set or a miscellaneous hardware failure. If you receive this code, you may need to call FileTek Customer Support for help determining necessary corrective action.

SQL code -30033

SLQ code -30033 is a generic return code that applies to any unrecoverable error caused by a relational engine's exit. The code may appear in conjunction with several error situations. Whether these errors are program-induced or user-induced, the code always indicates that a serious error has occurred and requires assistance from FileTek Customer Support.

DESCRIBE BIND restrictions

DESCRIBE BIND VARIABLES does not correctly process an SQL statement that contains both scalar functions and host variable markers. For example, the following SQL statement, which contains the scalar function TO_HEX and host variable markers, does not work properly with DESCRIBE BIND VARIABLES:

```
SELECT * FROM table WHERE ( TO_HEX (bin_column) LIKE :var )
```

The following SQL statement works correctly with DESCRIBE BIND VARIABLES because it contains no scalar function:

```
SELECT * FROM table WHERE bin_column > :var
```

Refer to the *StorHouse SQL Reference Manual* for complete documentation about DESCRIBE BIND VARIABLES.

Design advisory for join operations

Queries that use extensive join operations may not be good candidates for StorHouse/RM execution, especially at the high data volumes that you generally expect in large database environments. When you require such queries, consult your FileTek systems engineer for performance analysis and modeling assistance.

ISQL product status

The ISQL tool allows interactive processing of SQL statements. Use this tool only as a general-purpose development tool. You should not incorporate it into production software or operations procedures. If you do use it, you must use the new version included in the release 3.3 client (host.sol2) tar file. Older versions of ISQL cannot be used.

DDL processing in general

In StorHouse/RM, DDL statement execution is atomic and permanent. The software performs an implicit COMMIT before and after every DDL statement. Transaction logic (user-specific bundling of statements) may lead to undesirable table-level locks that restrict the entire database from use. To protect against this, StorHouse/RM adds to user-specified transaction (BEGIN-END groups) logic an implicit COMMIT before and after every DDL statement.

Host variables as BINARY, VARBINARY, and VARCHAR data types

In an ESQL program, you cannot declare host variables as BINARY, VARBINARY, and VARCHAR data types in a Declare Section. To define these types of variables, set up an SQLDA and use the DESCRIBE statement as documented in the *StorHouse ESQL Manual*.

Immediate restart after a load failure

If a data load fails and you restart it immediately, the restart may fail if the load cleanup hasn't completed. You receive the following errors when this happens:

```
500- %L-I-XLDINFO, \sqlcode=<-60021> Loader: Unrecoverable segment file\
```

```
500- %L-I-XLDINFO, \Unrecoverable segment, table=RESL_TBL1\
```

You can avoid this situation by waiting a few seconds before restarting a load. If you receive these errors, try the restart again after the brief delay.

LOB restrictions

Restrictions on accessing LOBs via ESQL and using LOBs in SQL are as follows.

ESQL. You must use a single-row fetch with the BLOB_FILE and CLOB_FILE data types. You cannot use array or pointer-fetch with these data types. Additionally, StorHouse/RM does not support file reference variables as input, that is, to transfer a LOB value from a client file to StorHouse. StorHouse/RM supports output file reference variables to transfer a LOB value from StorHouse to a client file.

SQL. LOB data types are not allowed with the following SELECT statement clauses: DISTINCT, ORDER BY or GROUP BY. Additionally, LOB data types are not allowed with the following functions: MIN, MAX, and COUNT(DISTINCT lob_expr).

Use of SYS in table names

Table names may not start with SYS, for instance, SYS_STARTUP or SYSSERVICE. The SYS prefix is reserved for system tables.

Use of control characters as delimiters

For data loading, use of control characters as delimiters, especially whitespace characters, is discouraged in input data. A terminator declared explicitly that is also a whitespace character may result in a "terminator not found" error, particularly when the prior field is enclosed. If you are using one of these characters as a delimiter, specify WHITESPACE rather than the character.

The whitespace characters are SP (space), CR, FF, LF, VT, and HT (tab). In any ASCII code page these have hex values 20, 0D, 0C, 0A, 0B, and 09. In any EBCDIC code page the hex values are 40, 0D, 0C, 25, 0B, and 05.