



# StorHouse®

## Release Notes for StorHouse/RM 3.2

### StorHouse/RM Release 3.2

Publication Number  
900126 Rev. M

April 17, 2002

---

**FileTek**





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.

This publication Copyright © 1996-2002 by FileTek, Inc., Rockville, MD  
Publication Number: 900126 Rev. M

**NOTE: U.S. GOVERNMENT USERS**

**Restricted Rights Legend**

Use, duplication or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or the Commercial Computer Software - Restricted Rights clause at 48 CFR 52.227-19, as applicable. Unpublished-rights reserved under the copyright laws of the United States. 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

<b>Welcome .....</b>	<b>vii</b>
Intended audience .....	vii
Contents of document .....	vii
Related documentation .....	viii
 <b>Chapter 1: Changes and enhancements .....</b>	 <b>1-1</b>
System requirements .....	1-1
Metadata conversion .....	1-1
Re-definition of segment .....	1-2
Definition of in-line and out-of-line LOBs .....	1-3
SQL updates .....	1-3
New SQL statements .....	1-3
Changed SQL statements .....	1-4
Defining a subspace for LOB data .....	1-4
Creating a table with a BLOB or CLOB column .....	1-4
Creating an index on a loaded table .....	1-5
New functions .....	1-6
Functions that now allow LOB arguments .....	1-7
Functions that now allow binary arguments .....	1-8
Additional format for SUBSTR .....	1-8
Result type changes for LENGTH, TO_CHAR, SUM, and AVG functions ....	1-8
Results of DECIMAL expressions .....	1-9



Data type updates .....	1-10
New data types .....	1-10
Guaranteed-size data types for C types int and long .....	1-11
Changed default lengths of loader and unloader data types .....	1-12
New default TIME [EXTERNAL] format mask .....	1-13
Changed ESQL data types and C structures .....	1-13
ESQL changes .....	1-14
New host variable types for LOB access .....	1-15
Locator variables .....	1-15
File reference variables .....	1-15
SQLCA changes .....	1-16
SQLDA changes .....	1-17
SQLDA structure .....	1-17
SQLDA functions .....	1-18
New default compile and link commands .....	1-20
Sun Solaris .....	1-20
HP .....	1-20
Data loader changes .....	1-20
Loading LOB data .....	1-21
Selecting or rotating among subspaces for LOB data .....	1-21
Generating field specifications for all data fields .....	1-21
Describing NULL flags in input data .....	1-22
Loading a NULL or default value for empty or blank data fields .....	1-22
Checking for data file or load control file errors .....	1-22
Loading a deferred index .....	1-23
Merging segments of a table .....	1-24
Loading data from multiple VRAM files .....	1-26
Loading in FTP passive mode .....	1-26
Data unloader changes .....	1-26
Unloading LOB data .....	1-27
Inserting NULL flags in result data .....	1-27
Clarification about delimiters in result data .....	1-27
Join enhancements .....	1-28
Join syntax .....	1-28
Join guidelines .....	1-30



Optimizer enhancements .....	1-30
Extended file naming convention for segment files .....	1-31
New field in the SQL transaction record .....	1-31
IP version 6 addressing support .....	1-32
Thread safe front-end .....	1-32
System limit change .....	1-32
Metadata enhancements .....	1-32
Automatic index rebuild .....	1-32
Improved system table indexing performance .....	1-33
System table documentation changes .....	1-33
SYSINDEXES system table .....	1-33
SYSCOLUMNS system table .....	1-33
SQL codes .....	1-34
New reserved words .....	1-35

## **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
ODBC .....	2-4
ESQL .....	2-4



**Contents**

---

SQL .....2-4

Use of SYS in table names .....2-4

Use of control characters as delimiters .....2-4



# Welcome

*Release Notes for StorHouse/RM 3.2* identifies changes, enhancements, and special considerations for StorHouse/RM since release 2.3.

## 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 since release 2.3.
- 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 since release 2.3.

## System requirements

StorHouse/RM is now available on Hewlett-Packard HP-UX systems. StorHouse/RM release 3.2 requires the following:

- StorHouse/SM release 5.2 or 5.3 (deliveries 124, 149, and 157 are required and delivery 183 is highly recommended)
- Sun<sup>TM</sup> Solaris<sup>TM</sup> 2.6, Solaris 8, or HP-UX 11.0

StorHouse/RM also supports StorHouse release 5.4 on both Solaris and HP-UX systems. A separate build of StorHouse/RM 3.2 is required for StorHouse/SM 5.4.

## Metadata conversion

Metadata may be stored in UNIX large files. Each metadata file can now be larger than 2 gigabytes (GB). If you require this feature, you must run the metadata conversion utility, using the copyout and copyin module provided with release 3.2. If you don't require this feature, you can run StorHouse/RM release 3.2 with



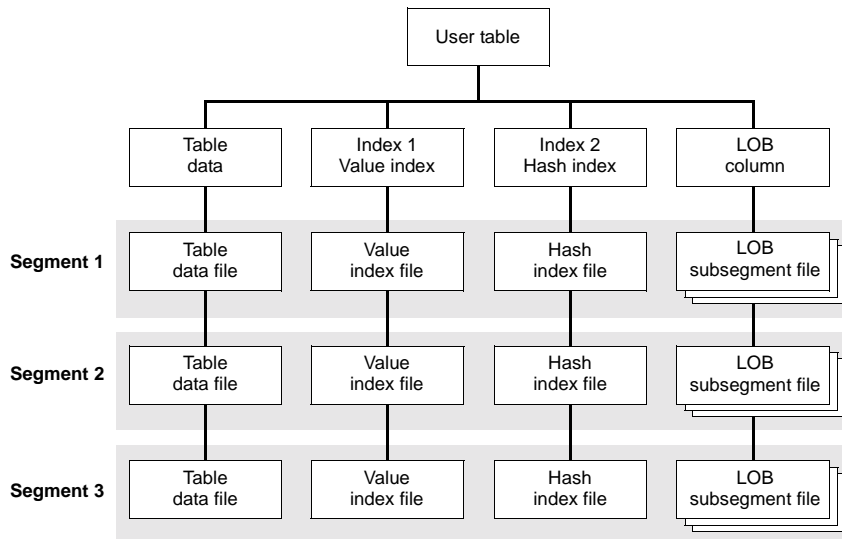
earlier metadata. Note, however, that a database created with the StorHouse/RM release 3.2 syscreate utility cannot be used with earlier releases of StorHouse/RM.

## Re-definition of segment

The definition of segment has changed. Previously, a segment was defined as a StorHouse file. User table data was stored in table segments, and hash indexes and value indexes were stored in index segments. Now, the StorHouse/RM User Document Set contains the following definition:

*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.*

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





## Definition of in-line and out-of-line LOBs

StorHouse/RM supports the loading, storage, and unloading of large objects (LOBs). A *LOB* is a binary large object (BLOB) or a character large object (CLOB). These types allow storage of up to 2G - 9 bytes in a single instance of a single column. An *in-line LOB* is a LOB value stored in the table data file with the rest of the table data. The maximum size of a row in a table data file is 32,705 bytes. An *out-of-line LOB* is a LOB value stored in a separate StorHouse file called a *LOB subsegment file*. Depending on the actual size and by user request (on the CREATE TABLE statement), 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 a single column may be stored in multiple LOB subsegment files. See “Creating a table with a BLOB or CLOB column” on page 1-4 for LOB storage options.

## SQL updates

Some SQL statements and functions are new and others have changed. Additionally, the result precision and scale for arithmetic operations on DECIMAL values have changed.

## New SQL statements

Two new SQL statements are exclusive to LOBs:

- **VALUES INTO** – Manipulate LOB values and expressions previously selected using locator variables. See “Locator variables” on page 1-15 for more information about locator variables.
- **FREE LOCATOR** – Release one or more locator variables before the end of a transaction, freeing the server storage used by the locator variable.



## Changed SQL statements

The formats of the CREATE TABLESPACE, ALTER TABLESPACE, and CREATE TABLE statements have changed to support LOBs. The CREATE INDEX statement has a new DEFERRED keyword.

### Defining a subspace for LOB data

When creating or altering a user tablespace, you can now define a subspace (set of storage parameters) for LOB data. With LOB subspaces, you can manage the storage of LOB data differently from the rest of the table data and the index data. The OBJECT\_TYPE subspace parameter allows the value L for LOBs. For example:

```
CREATE TABLE SPACE STATEMENTS
(SUBSPACE 1 VSET JAN2001L FSET JAN2001L OBJECT_TYPE L
ATF 2 VTF NOW MAX_EXT_SIZE 800 HOLD 30 HOLD_SPECIAL 60)
```

### Creating a table with a BLOB or CLOB column

When creating a user table, you can now define BLOB and CLOB columns to hold about 2 gigabytes ( $2^{31}-9$ ) of data. The following new CREATE TABLE clauses define LOB storage options:

- **INLINE [(length [K])]** – Set the maximum length for LOB values stored (in-line) with the table data.
- **NOT INLINE** – Store LOB values in a different file (out-of-line) from the table data, even if the values fit in a row.
- **STORE WITH column\_name** – Share storage (a LOB subsegment file) with another LOB column in the table.
- **TABLE SPACE tablespace\_name** – Assign the LOB column to a user tablespace different from the tablespace of table.



For example:

```
CREATE TABLE LOB_EXAMPLE
(CHAR_COL CHAR(6),
FIRST_CLOB CLOB(1M) STORE WITH SECOND_CLOB,
SECOND_CLOB CLOB TABLESPACE XYZ,
FIRST_BLOB BLOB(1G) NOT INLINE,
SECOND_BLOB INLINE(2K))
TABLE SPACE ABC
```

Refer to the *StorHouse Database Administration Guide* for more information about defining LOB subspaces and creating user tables with LOB column definitions.

## Creating an index on a loaded table

You can now create indexes after a table has been loaded and then load the indexes for the existing segments. This index is called a *deferred index*. You can create deferred indexes for all index types—value, hash, and range.

To create a deferred index, use the new DEFERRED keyword on the CREATE INDEX statement. Place this keyword after the column list and before any TABLE SPACE clause. An error occurs if you omit the DEFERRED keyword and the table contains data.

The new CREATE INDEX format is:

```
CREATE [VALUE | HASH | RANGE] INDEX index_name
ON [owner.]table_name (column_name [,column_name]... )
[DEFERRED]
[TABLE SPACE tablespace_name ]
```



Here's an example statement with the DEFERRED keyword:

```
CREATE VALUE INDEX ORDERS2000
ON ORDERS (ORDER_NO, CUSTOMER_NAME)
DEFERRED
TABLE SPACE ORDERS2000
```

## New functions

The following functions are new in StorHouse/RM.

### New StorHouse functions

Function	Description
BIT_LENGTH	Returns the length (in bits) of a BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR expression.
BLOB	Returns a BLOB representation of a BINARY, CHAR, VARBINARY, or VARCHAR expression.
CHAR_LENGTH	Returns the length (in characters) of a BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR expression.
CLOB	Returns a CLOB representation of a CHAR or VARCHAR expression.
OCTET_LENGTH	Returns the length (in bytes) of a BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR expression.
OVERLAY	Replaces a substring from the first expression (BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR data type) with the second expression.
POSITION	Determines the starting position at which the first expression (BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR data type) is found in the second expression.
TRIM	Removes leading values, trailing values, or both from a BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR expression

---



## Functions that now allow LOB arguments

You can now use BLOB and/or CLOB arguments with these existing functions:

### Functions that take LOB arguments

Function	BLOB argument	CLOB argument
ASCII		X
CONCAT	X	X
COUNT	X	X
INITCAP	X	X
INSTR	X	X
LENGTH	X	X
LOWER		X
LPAD		X
LTRIM	X	X
NVL	X	X
RPAD		X
RTRIM	X	X
SUBSTR	X	X
TO_CHAR	X	X
TO_DATE		X
TO_HEX	X	
TO_NUMBER		X
TO_TIME		X
TRANSLATE		X
UPPER		X



## Functions that now allow binary arguments

Several functions that previously accepted only character arguments now also allow binary arguments.

**Functions that now take binary arguments**

Function	Previous type(s)	Current types
CONCAT	CHAR or VARCHAR	BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR
INSTR	CHAR	BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR
LENGTH	CHAR	BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR
LTRIM	CHAR or VARCHAR	BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR
RTRIM	CHAR or VARCHAR	BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR

## Additional format for SUBSTR

The SUBSTR function now accepts the SQL-99 format in addition to the current StorHouse format.

- Current format: SUBSTR ( expression, start\_position [, length] )
- SQL-99 format: SUBSTR ( expression FROM start\_position [FOR length] )

## Result type changes for LENGTH, TO\_CHAR, SUM, and AVG functions

The result data type of the LENGTH function is now INTEGER (previously SMALLINT).



The result data type of the TO\_CHAR function is VARCHAR if the expression is CLOB or CHAR if the expression is any other data type (except BLOB). TO\_CHAR does not support a BLOB expression. StorHouse/RM silently truncates any CLOB larger than 32705.

The result data type of the SUM and AVG functions for integer or decimal input types is as follows:

**Result types for SUM and AVG functions**

Input type	SUM result	AVG result
REAL	DOUBLE	DOUBLE
DOUBLE	DOUBLE	DOUBLE
SMALLINT	DECIMAL(15)	DECIMAL(13,8)
INTEGER	DECIMAL(20)	DECIMAL(18,8)
DECIMAL(p1,s1)	$p = \text{MIN} ( 31, p1 + 10 )$ $s = s1$	$s = \text{MIN} ( 31 - p1, 8 ) + s1$ $p = p1 - s1 + s$

## Results of DECIMAL expressions

The result precision and scale for arithmetic operations involving DECIMAL values are as now follows.

**Result precision and scale for operations on DECIMAL values**

Operation	Result
+ or -	$m = \text{MAX} (m1, m2) + 1$ $s = \text{MAX} (s1, s2)$
*	$m = m1 + m2$ $s = s1 + s2$
/	$m = m1 + s2$ $s = \text{MAX} ( \text{MIN} (m2, 6) + s1 + 1, 8 )$

Note that  $m = p - s$  and that  $m$  and  $s$  are limited to 31. The  $p$  (or  $s$  if division) is further reduced if needed to limit  $p$  to 31.



## Data type updates

There are new data types and changes to existing data types.

### New data types

StorHouse/RM now supports the following new data types.

#### New StorHouse data types

Function	Data type
Defining columns	BLOB [(length [K M G])]
	BINARY LARGE OBJECT [(length [K M G])]
	CLOB [(length [K M G])]
	CHARACTER LARGE OBJECT [(length [K M G])]
Loading data	BLOB [(max_length [K M G])]
	BLOB_FILE [(length)] [delimiter_spec] [HOST hostname] [PATH path_spec] [USER username/password]
	CLOB_FILE [(length)] [CHARSET ccsid] [delimiter_spec] [HOST hostname] [PATH path_spec] [USER username/password]
	CLOB [(max_length [K M G])] [CHARSET ccsid]
Unloading data	BLOB [(max_length [K M G])]
	BLOB_FILE [(length)] [delimiter_spec] [HOST hostname] [PATH path_spec] FILENAME filename_spec [OVERWRITE] [USER username/password]
	CLOB [(max_length [K M G])] [CHARSET ccsid]
	CLOB_FILE [(length)] [CHARSET ccsid] [delimiter_spec] [HOST hostname] [PATH path_spec] FILENAME filename_spec [OVERWRITE] [USER username/password]
Declaring variables	BLOB
	BLOB_FILE



**New StorHouse data types (continued)**

Function	Data type
	BLOB_LOCATOR
	CLOB
	CLOB_FILE
	CLOB_LOCATOR

## Guaranteed-size data types for C types int and long

Depending on the platform, the C type int may be 16 or 32 bits and the C type long may be 32 or 64 bits. You can now declare host variables with the following guaranteed-size types in ESQL applications:

**Other types for int and long**

Data type	Description
int64_t	64-bit signed integer
uint64_t	64-bit unsigned integer
int32_t	32-bit signed integer
uint32_t	32-bit unsigned integer
int16_t	16-bit signed integer
uint16_t	16-bit unsigned integer
int8_t	8-bit signed integer
uint8_t	8-bit unsigned integer



## Changed default lengths of loader and unloader data types

The default lengths of some loader and unloader data types have changed.

- CHARACTER – In release 2.3, the default length for CHAR was as follows:

- 256 (with delimiter\_spec)
- or CREATE TABLE length if the source data was BINARY or CHAR
- otherwise 1

Now, the default length for CHAR is as follows:

- MIN(CREATE TABLE length, 32705) if the source data is BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR
- or else 256 (with delimiter\_spec)
- else 1

- DATE EXTERNAL – The default length for the unloader DATE EXTERNAL data type has changed.

- In release 2.3, the default length was 256 (with delimiter\_spec) or 75 (without a delimiter\_spec).
- Now, the default length is 256 (with delimiter\_spec) or else 75 (with mask) or else 10.

- TIME EXTERNAL – In release 2.3, the default length for a TIME EXTERNAL data field was:

- Unloader: 75 (without delimiter\_spec) or 256 (with delimiter\_spec)



- Loader: 8 (without delimiter\_spec) or 256 (with delimiter\_spec)

Now, the default length is as follows:

- Unloader: 256 (with delimiter\_spec) or else 75 (with mask) or else 12
  - Loader: 12 (without delimiter\_spec) or 256 (with delimiter\_spec)
- **TIMESTAMP EXTERNAL** – The default length for the unloader **TIMESTAMP EXTERNAL** data type has changed.
    - In release 2.3, the default length was 75 (without delimiter\_spec) or 256 (with delimiter\_spec).
    - Now, the default length is 256 (with delimiter\_spec) or else 75 (with mask) or else 26.

## **New default TIME [EXTERNAL] format mask**

The current unloader default mask for a **TIME EXTERNAL** data field does not include a milliseconds (MLS) field. A milliseconds value, however, is necessary in order to unload and reload a column of type **TIME** in all cases. Now, the default unloader **TIME** mask is **H24:MI:SS.MLS**. A FileTek data loader accepts a **TIME** data field with no **MLS** field. The FileTek unloader produces the **MLS** field only if the milliseconds part of the **TIME** value being unloaded is nonzero. If the length is specified and is less than 12, the default mask does not contain **MLS**.

## **Changed ESQL data types and C structures**

Changes to ESQL data types and C structures are as follows:

- The **TPE\_DT\_SMALLFLOAT** data type has been eliminated.
- The **TPE\_DT\_FLOAT** data type has been changed to **TPE\_DT\_DOUBLE**.



- The pointer fetch return area for VARCHAR and VARBINARY data types no longer contains the four-byte "total length" field. This length information is available in the new SQLDA.
- The `tpe_time_t` C structure definition for the TIME data type is extended with internal padding to a total length of 6 bytes (to maintain alignment). The new structure definition for `tpe_time_t` is:

```
typedef struct {
    uint8_t    hours;
    uint8_t    mins;
    uint8_t    secs;
    uint8_t    reserved;
    uint16_t   msecs;
} tpe_time_t;
```

- The `tpe_timestamp_t` C structure for the TIMESTAMP data type is renamed (previously `tpe_tstime_t`) and redesigned, eliminating the separate date and time sections. The new structure definition for `tpe_timestamp_t` is:

```
typedef struct {
    uint16_t   year;
    uint8_t    month;
    uint8_t    day;
    uint8_t    hours;
    uint8_t    mins;
    uint8_t    secs;
    uint8_t    reserved;
    uint32_t   usecs;
} tpe_timestamp_t ;
```

## ESQL changes

StorHouse/RM now provides two types of host variables for accessing LOB values. Additionally, the SQLCA, the SQLDA, and the default compile and link commands have changed.



## New host variable types for LOB access

StorHouse/RM supports locator variables and file reference variables for accessing and retrieving LOB values from ESQL applications.

### Locator variables

With a *locator variable*, an application can:

- Refer to a LOB value at the StorHouse server without retrieving it into a client buffer
- Manipulate a LOB value with functions such as SUBSTRING, CONCAT, and INSTR
- Fetch a part of a LOB value to the client

You define locator variables with the BLOB\_LOCATOR and CLOB\_LOCATOR data types. For example, the following Declare Section contains two locator variables for CLOB values.

```
EXEC SQL BEGIN DECLARE SECTION;  
    int32_t hv_start_descr;  
    int32_t hv_end_descr;  
    int32_t hv_bonus_start;  
    CLOB_LOCATOR hv_prod_locator;  
    CLOB_LOCATOR hv_prod_desc_locator;  
    CLOB(2M)hv_product;  
EXEC SQL END DECLARE SECTION;
```

### File reference variables

A *file reference variable* identifies a client file to which a LOB value may be transferred. You define file reference variables with the BLOB\_FILE and CLOB\_FILE. For instance, the following example defines a file reference variable



named ProdFile:

```
EXEC SQL BEGIN DECLARE SECTION;  
    CLOB_FILE ProdFile;  
EXEC SQL END DECLARE SECTION;
```

The file reference variable expands into a type definition with four parts. For instance, the structure definition for CLOB\_FILE (tpe\_clob\_file\_t) is:

```
typedef struct {  
    uint32_t    name_length;  
    uint32_t    data_length;  
    uint32_t    file_options;  
    char        name[255];  
} tpe_clob_file_t;
```

## SQLCA changes

The SQLCA (SQL communications area) provides diagnostic information about the execution of an SQL request. Changes to the SQLCA are as follows:

- Various components are defined with the guaranteed-size types for int and long.
- A new component, sqlstate, will contain (in a future release) completion state information.
- The number of warning flags in sqlwarn increased from 8 to 9 to maintain alignment.



The new structure definition for the SQLCA is as follows:

```
struct tpe_sqlca {  
    char sqlcaid[8];      /* Eye-catcher, "TPESQLCA" */  
    int32_t sqlcode;      /* Result of execution */  
    int16_t sqlcabc;      /* Length of tpe_sqlca */  
    uint16_t sqlerrml;    /* Length of message */  
    uint8_t sqlerrm[74]; /* Null-terminated message */  
    char sqlerrp[8];      /* (reserved) */  
    int32_t sqlerrd[8];   /* Diagnostic information */  
    char sqlwarn[9];      /* Warning flags */  
    char sqlstate[5];     /* Completion state */  
};
```

## SQLDA changes

The structure and functions of the SQLDA (SQL descriptor area) have been updated.

### SQLDA structure

In release 2.3, the SQLDA contained pointers to arrays of values. Each entry in an array represented a value for one of the variables (columns, expressions, parameters, and so on) described by the SQLDA. There was an array for the types of variables, an array for the pointers to the buffers for the variables, and several other arrays.

Now, one array replaces these separate arrays. The structures of this array contain values for items like the data type of the variable and the buffer pointer. The SQLDA is renamed `tpe_sqlda` to distinguish this new version from older versions.



The new structure definition for the SQLDA is as follows:

```
struct tpe_sqlda {
    char sqldaaid[8];      /* Eye-catcher 'TPE_SQLD' */
    uint8_t sqldvrsn;      /* SQLDA version... must be 0 */
    uint8_t sqldfmod;      /* Flag: fetch mode */
    int16_t sqldsize;      /* Number of entries allocated */
    int16_t sqldnvar;      /* Number of entries in use */
    int16_t sqldrsv1;      /* (reserved) */
    int16_t sqldnrow;      /* Number of rows to be fetched */
    int16_t sqldvnl;      /* Maximum varname length */
    tpe_sqlvar* sqldvar; /* tpe_sqlvar elements */
};
```

The structure `tpe_sqlvar` contains an entry for each variable in the SQLDA.

```
struct tpe_sqlvar {
    int32_t sqlvln32;      /* Variable (maximum) length */
    int32_t* sqlvlenp;     /* Pointer to actual length */
    int32_t sqlvbl32;      /* Buffer length (pointer mode) */
    void* sqlvdata;        /* Pointer to data */
    int16_t* sqlvind;      /* Pointer to indicator variable */
    int16_t sqlvtype;      /* Variable type (TPE_DT_XXX) */
    uint8_t sqlvprec;      /* Precision */
    uint8_t sqlvscal;      /* Scale */
    uint8_t sqlvisnl;      /* Nullable flag */
    uint8_t sqlvrsv1[3];   /* (reserved) */
    char* sqlvname;        /* Variable name */
    int32_t sqlvrsv2[2];   /* (reserved) */
};
```

## SQLDA functions

SQLDA functions manipulate an SQLDA structure. General changes are as follows:

- All functions now return a status value.
- The `sqld_` prefix in function names has been replaced with `tpe_da_`.



The following new or renamed functions manipulate an SQLDA structure.

#### New or renamed SQLDA functions

SQLDA function	Description
tpe_da_alloc	Allocates, initializes, and returns a pointer to an SQLDA.
tpe_da_alloc_varnames	Allocates space for variable (column) names, and sets the appropriate fields in the SQLDA.
tpe_da_free	Deletes an SQLDA that was allocated by tpe_da_alloc.
tpe_da_getbsize	Calculates and returns the length of the buffer required to hold the data described in the provided SQLDA. This includes space for indicator variables.
tpe_da_getnbytes	Calculates the size, in bytes, of an SQLDA.
tpe_da_getvnbytes	Calculates the size, in bytes, of the space required to store variable name (sqlvname) data.
tpe_da_setentry	Sets values in an SQLDA variable entry. Specifically, this function sets the data type, length, indicator variable pointer and data pointer to the values provided.
tpe_da_setptrs	Initializes the buffer pointers (sqlvdata, sqlvind) in the provided SQLDA. This function was formerly named tpe_setup_sqlda.
tpe_da_setup	Initializes storage as an SQLDA.

The following functions are used in obsolete programs and have been eliminated in StorHouse/RM:

- tpe\_char\_sqlda
- tpe\_host\_sqlda
- tpe\_print\_sqlda
- tpe\_set\_ptrs
- tpe\_set\_sqlda



## New default compile and link commands

The new default compile and link commands differ based on operating system. You can still override the defaults with the ESQL\_CC and ESQL\_LINK environment variables.

### Sun Solaris

On Sun Solaris, the default compile and link commands are:

Compiling:

```
cc -c -I<sth_dir>/include/ <app_program.c>
```

Linking:

```
CC -o <app_program> <app_program.o> /<sth_dir>/lib/libsthfe.a -lm -lnsl -lsocket
```

### HP

On HP, the default compile and link commands are:

Compiling:

```
cc -Ae -c -I<sth_dir>/include/ <app_program>.c
```

Linking:

```
aCC -AA -o <app_program> <app_program>.o /<sth_dir>/lib/libsthfe.a -lm -lnsl
```

## Data loader changes

You can now perform the following data loading functions:

- Load LOB data
- Select subspaces or rotate among subspaces for LOB data
- Generate field specifications
- Describe NULL flags in input data



- Load NULL or default values for empty or blank data fields
- Check for data file and load control file errors before loading
- Load a deferred index
- Merge segments in a table
- Load data from multiple VRAM files
- Load in FTP passive mode

## **Loading LOB data**

You can now load LOB data into StorHouse user tables. With the FileTek FTP Data Loader, you can load LOB input data in one data file with the rest of the table data or in separate files—one LOB value per file—on your client or a remote host. You use the following loader data types to describe LOB data fields: BLOB, BLOB\_FILE, CLOB, and CLOB\_FILE.

With the FileTek MVS Data Loader utility, you can load LOB input data in one data file with the rest of the table data. You cannot load LOB values from separate files. You use the BLOB and CLOB data types to describe LOB data fields.

## **Selecting or rotating among subspaces for LOB data**

You can select LOB subspaces during a load and rotate among applicable subspaces for LOB values. LOB subspaces (OBJECT\_TYPE is L in a tablespace) are valid for out-of-line LOBs only.

## **Generating field specifications for all data fields**

The FileTek data loaders can generate all field specifications as the CHARACTER loader data type when you include the CHAR keyword with the FIELDS clause.



The data loaders determine the lengths of each field with the default-length rules for CHARACTER loader data type.

## **Describing NULL flags in input data**

If your input data contains NULL flags (T for NULL or F for NOT NULL), you can describe them with the new NULLFLAGS keyword. The FileTek data loader appends a NULLIF clause to each generated field specification so that when a NULL flag is T, it loads a NULL into the corresponding table column.

## **Loading a NULL or default value for empty or blank data fields**

You can use NULLIF and DEFAULTIF clauses in a field specification to describe a condition for loading NULL and default values. Now you can also specify NULLIF and/or DEFAULTIF clauses with the FIELDS clause to load a NULL value or a column's default value for any data field that is empty (two adjacent delimiters) or contains blanks. You use the keywords EMPTY or BLANK with the NULLIF and DEFAULTIF clauses in a FIELDS clause. The NULLIF and DEFAULTIF clauses in a FIELDS clause apply to any data fields that do not have NULLIF and/or DEFAULTIF clauses in their field specifications.

## **Checking for data file or load control file errors**

Before starting an FTP load, you can now verify whether your data file and load control file are okay by using validate for the load\_type keyword. If there's a syntax error, you can correct the error and then start the load. The current alternative is to start the load and abort it if you need to change a data file or load control file. This feature requires StorHouse/SM 5.2 delivery 157.



## Loading a deferred index

After creating a deferred index, you can load index entries for some or all of the segments of the table. You use the new LOAD INDEX statement in a load control file and a FileTek data loader to perform an *index load operation*. The default is to load all segments, after which each loaded index is marked as complete or no longer deferred.

A LOAD INDEX statement in a load control file identifies the names of the indexes to load and optionally specific segments. Only one LOAD INDEX statement is allowed in a control file.

```
LOAD INDEX[ES] index_name [ ,index_name ]... [subspace_clause]
[SEGMENTS segment_list]
```

Argument	Format
subspace_clause	SUBSPACE ROTATE   [ VALUE   HASH ] SUBSPACE number
segment_list	segment_list_item [ , segment_list_item ]...
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment

Here are some example LOAD INDEX statements:

- To load index files for the ORDERS2000 index for all segments of the table:

```
LOAD INDEX ORDERS2000
```

The data loader uses the lowest-numbered subspace for the index type because the SUBSPACE num and SUBSPACE ROTATE clauses are omitted.

- To load an index file for the ORDERS2000 index for the first segment of the table, using subspace 2:



LOAD INDEX ORDERS2000 SEGMENTS 0 SUBSPACE 2

- To load index files for the ORDERS2000 index for a range of segments, rotating among subspaces that are valid for the index type:

LOAD INDEX ORDERS2000 SEGMENTS 0-5 SUBSPACE ROTATE

- To load index files for the ORDERS2000 and ORDERSDETAIL indexes for all segments of the table:

LOAD INDEX ORDERS2000, ORDERSDETAIL

## Merging segments of a table

You can now consolidate segments in a table. A *merge*, or *coalesce operation* merges a group of segments into one segment (possibly more) and invalidates the input segments. You have full control over which segments are grouped, for instance, by specific segment IDs or segment tags or by size criteria. LOB subsegment files, however, are not merged. Merging segments enhances performance because it reduces the number of file and extent opens and closes for a query.

You use the new MERGE (or COALESCE) statement with a FileTek data loader to merge segments. The MERGE statement in a load control file identifies the segment or merge criteria.



```
{MERGE | COALESCE} INTO TABLE table_name [subspace_clause]
[SEGMENT segment_tag] [SEGMENTS segment_list]
[EXCLUDE segment_list] [MAXINSIZE n] [MINOUTSIZE n]
```

Argument	Format
subclause_clause	SUBSPACE ROTATE   [ VALUE   HASH   TABLE ] SUBSPACE number
segment_list	IDS segment_list_item [ , segment_list_item ]...   TAGS segment_tag [ , segment_tag ]...
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment

Here are some example MERGE statements.

- To merge all segments of the CALLSDETAIL table into one segment:

```
MERGE INTO TABLE CALLSDETAIL
```

- To merge the first five segments of the CALLSDETAIL table into one segment:

```
MERGE INTO TABLE CALLSDETAIL SEGMENTS IDS 0-4
```

- To merge all segments of the CALLSDETAIL table except segment ID 0:

```
COALESCE INTO TABLE CALLSDETAIL EXCLUDE IDS 0
```

- To merge all small segments that are less than 100MB of the CALLSDETAIL table into segments of at least 1GB but ignore segments with the segment tag late\_entries:

```
MERGE INTO TABLE CALLSDETAIL MAXINSIZE 100M
MINOUTSIZE 1G EXCLUDE TAGS "late_entries"
```



## Loading data from multiple VRAM files

On the INFILE clause, you can now specify a list of VRAM files to load data from multiple files into a table. You can place the NOENVIRON keyword before the file list if it applies to all files. For example:

```
LOAD INFILE NOENVIRON (FILE1/ATM, FILE2/ATM, FILE3/ATM)
```

Otherwise, you can place the NOENVIRON keyword after any specific file in the list if it applies to that file only.

```
LOAD INFILE (FILE1/ATM NOENVIRON, FILE2/ATM, FILE3/ATM)
```

## Loading in FTP passive mode

The FileTek FTP Data Loader now supports FTP passive mode. This change accommodates firewalls and FTP applications (like Web browsers) that are designed to use passive mode.

## Data unloader changes

You can now perform the following data unloading functions:

- Unload LOB data
- Insert NULL flags in result data

This section also contains information about delimiters in result data.



## Unloading LOB data

You can unload LOB data from StorHouse user tables with the FileTek FTP Data Unloader. You can place the LOB data in the same result file with any other result data, or you can place each LOB value in a different result file on your client or a remote host. You use the following unloader data types to describe LOB data fields: BLOB, BLOB\_FILE, CLOB, and CLOB\_FILE.

## Inserting NULL flags in result data

You can insert NULL flags (T for NULL or F for NOT NULL) at the beginning of each result record for each data field. For example, in the following result rows, the third data field in row 1 is NULL and the second data field in row 2 is NULL.

```
FFTplug,12, ;  
FTFcable, ,out;
```

You do this by including the new NULLFLAGS keyword with the FIELDS clause.

## Clarification about delimiters in result data

The following information about delimiters affects the *FileTek FTP Data Unloader Manual*.

- A delimiter\_spec in a FIELDS clause or in a field\_spec in a USING clause does not apply to CONSTANT field\_specs and IFNULL strings. This means you must include any delimiters in the CONSTANT string or IFNULL string if you want them.
- If you use a FIELDS clause to specify a terminator delimiter and there's no RECORDS clause, the FileTek FTP Data Unloader uses the FIELDS delimiter as the record terminator. For example, assume a FIELDS clause specifies ! as the terminator delimiter and there's no RECORDS clause.



FIELDS TERMINATED BY '!'

The unloader adds the ! delimiter after the last result data field like this:

```
2839!McGuire!Jack!  
2388!Cornflake!Sue!
```

- For data with terminator-type delimiters, the unloader no longer inserts a space when the value is NULL.

## Join enhancements

StorHouse/RM now supports the ANSI SQL join syntax. This change enables you to write queries with explicit join operations in the FROM clause of a SELECT statement, using an ON clause to specify the join condition. Previously, you specified a join operation by naming the tables on the FROM clause and by specifying the columns with their respective join conditions in the WHERE clause.

This release supports inner-join and left outer-join operations. Full outer-join and right outer-join operations are not yet implemented.

### Join syntax

The new format of the FROM clause is as follows:

```
FROM table_spec [, table_spec ]...
```

where table\_spec is:

```
table_reference [ correlation ] | joined_table
```



Argument	Description
table_reference	The name of a table participating in a join.
correlation	A correlation name for a table participating in a join. The format is: [AS] correlation_name
joined_table	The join specification. The format is: (joined_table)   table_spec { [ INNER ]   LEFT [ OUTER ] } JOIN table_spec ON join_condition
(joined_table)	Anything that qualifies as a joined_table can be enclosed in parentheses and considered as a joined_table itself. For example:  FROM pilot JOIN (service JOIN plane ON plane.serial_num = service.serial_num) ON plane.serial_num = pilot.serial_num
INNER JOIN	A join operator that specifies an inner-join operation. When the join condition is true, the matched rows of the tables are combined. The unmatched rows are omitted from the result table. The following operators are valid for inner-join: <ul style="list-style-type: none"> <li>■ JOIN</li> <li>■ INNER JOIN</li> </ul>
LEFT OUTER JOIN	A join operator that specifies a left outer-join operation. When the join condition is true, the matched rows of the tables are combined (like an inner-join) and the unmatched rows of the table to the left of the join operator are preserved, combined with NULL values for the columns in the table to the right of the join operator. The following operators are valid for outer-join: <ul style="list-style-type: none"> <li>■ LEFT JOIN</li> <li>■ LEFT OUTER JOIN</li> </ul>
join_condition	A search condition, or predicate, that evaluates to true, false, or unknown for a given row. You can specify multiple predicates with logical operators AND and OR. The join condition cannot contain a subquery.



## Join guidelines

Basic guidelines for writing queries with the new join syntax are as follows:

- You can perform multiple join operations in the same query, for instance, multiple inner-join operations, multiple outer-join operations, or a combination of the two.
- Any column referenced in a join condition must be a column in one of the tables of the associated join operation.
- Table order is significant for outer-join operations, insignificant for inner-join operations.
- For left outer-joins, the table referenced to the left of the join operator is the preserved table.
- You can use parentheses to specify the sequence to perform join operations.
- The WHERE clause has a different effect on query results from the ON clause.

## Optimizer enhancements

The optimizer has been enhanced as follows:

- Previously, the optimizer used nested loop processing for outer-joins. Now, outer-joins may qualify for hybrid IN processing.
- The optimizer now assigns default selectivity values when host variables are used in query predicates to avoid bypassing the use of indexes to resolve those predicates.
- The range index selection is deferred until the values for all host variables are available to minimize the number of segments that the query must search.

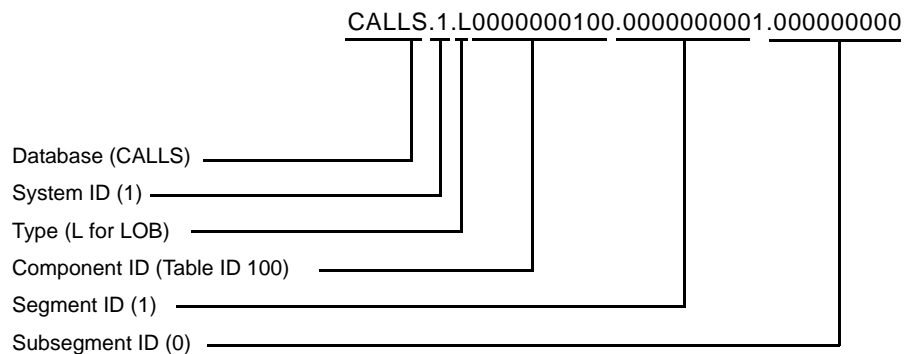


## Extended file naming convention for segment files

The file naming convention for segment files has been extended to incorporate a LOB subsegment ID. Segment files loaded prior to release 3.2 retain the existing file name. The file name format is as follows:

database\_name.system\_id.typecomponent\_id.segment\_id.lob\_subsegment\_id

For example, the following StorHouse file name identifies the first LOB subsegment file in the second segment (segment ID1) of a user table (table ID 100) in the CALLS database.



## New field in the SQL transaction record

StorHouse/RM now tracks the number of bytes retrieved in the SQL transaction record in the user log. Refer to the StorHouse *User Log Format* manual, publication number 900028, for more information about the user log and the types of events recorded.



## IP version 6 addressing support

All functions in StorHouse/RM that are dependent on the Internet Address Family version have been updated to operate correctly with either IP version 4 or 6.

## Thread safe front-end

The StorHouse/RM client interface (the API) has been made thread-safe. Connections can now be used from multiple threads.

## System limit change

There is now no limit to the number of StorHouse databases that you can create; however, at most 100 databases can be active at a time. An *active database* has one or more sessions currently connected.

## Metadata enhancements

Improvements have been made to the metadata recovery process and to the use of system table indexes.

### Automatic index rebuild

If a database recovery fails because changes to the index cannot be correctly rolled back, StorHouse/RM automatically rebuilds an index on a metadata table.



## Improved system table indexing performance

The I/O routines supporting the UNIX (flat-file) storage system have been changed to improve indexing performance. UNIX read and write system calls are used instead of fread and fwrite, index fan-out has been increased to reduce tree depth, and the row size of index entries has been reduced to increase the number of index nodes per block.

## System table documentation changes

Documentation changes to two system tables are as follows.

### **SYSINDEXES system table**

The IDXCOMPRESS column in the SYSINDEXES system table indicates whether an index is deferred or complete. The values are:

- N for normal (complete)
- C for deferred (incomplete)

### **SYSCOLUMNS system table**

The description of the values for the LOB\_TSID column in Appendix A of the *StorHouse Database Administrator's Guide* has changed. The *LOB\_TSID* is the 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.



## SQL codes

The text of SQL code -20223 has changed:

- Old text: Multiple local connections
- New text: Multiple default (DB\_NAME used) connections

The new SQL codes are as follows:

-20145 Invalid expression for No-table SELECT (or VALUES INTO)

-20146 LOB\_FILE not allowed with array fetch

-20147 LOB datatypes are not allowed on the DISTINCT, ORDER BY or GROUP BY clauses

-20248 Unique qualifier not allowed on user table indexes

-30012 Communication packet overflow failure

-30013 Output SQLDA changed between FETCH requests

-60026 Loader: FIELDS CHAR not allowed with fields\_spec list

-60027 Loader: FIELDS NULLFLAGS not allowed with field\_spec list

-60028 Loader: LOB type fields must be last in fields\_spec list

--60029 Loader: LOB type fields must be at POSITION(\*)

-60030 Loader: LOB type field can only be used with a LOB column

-60031 Loader: Last LOB type field did not end at end of a record

-60032 Loader: LOB field data cannot be used in a comparison



-60206 Unloader: No FILENAME specified for LOB\_FILE

-60207 Unloader: FIELDS NULLFLAGS not allowed with USING clause

-60208 Unloader: RECORDS terminator not allowed with LOB fields

## **New reserved words**

The following words are now reserved:

- BLOB
- CLOB
- LARGE
- OBJECT



**1**

**Changes and enhancements**

---

New reserved words



## 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 3.2 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 ODBC and ESQL and using LOBs in SQL are as follows.

**ODBC.** ODBC does not support the use of LOB data types.

**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.