



## **StorHouse/RM Concepts**

StorHouse/RM Release 3.4

Publication Number  
900132 Rev. H

September 17, 2008

---

The FileTek logo consists of the word "FileTek" in white, bold, sans-serif font, centered within a teal square.



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Copyright © 1998-2008 FileTek, Inc. As an Unpublished Licensed Work.  
Publication Number: 900132 Rev. H

#### 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 Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited.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. Sun, Microsystems, and Ultra Enterprise are registered trademarks or trademarks of Sun Microsystems, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. UNIX is a registered trademark of the Open Group. Oracle is a registered trademark and Oracle8i is a trademark of Oracle Corporation. IBM, DB2, Distributed Relational Database Architecture, and DataJoiner are registered trademarks or trademarks of International Business Machines Corporation. SequeLink is a registered trademark of DataDirect Technologies. 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>Products and applications</b>	<b>4</b>
<b>StorHouse databases</b>	<b>6</b>
<b>User tables</b>	<b>8</b>
<b>Large objects</b>	<b>10</b>
<b>Indexes</b>	<b>12</b>
<b>User tablespaces</b>	<b>14</b>
<b>Metadata</b>	<b>16</b>
<b>Storage management</b>	<b>18</b>
<b>Backup</b>	<b>20</b>
<b>Recovery</b>	<b>22</b>
<b>Software architecture</b>	<b>24</b>
<b>SQL</b>	<b>26</b>
<b>ESQL</b>	<b>28</b>
<b>Data loaders</b>	<b>32</b>
<b>Data unloader</b>	<b>34</b>
<b>ODBC interface</b>	<b>36</b>
<b>StorHouse/UDB Link</b>	<b>38</b>
<b>Queries</b>	<b>40</b>
<b>Concurrency</b>	<b>42</b>
<b>Database security</b>	<b>44</b>
<b>Administration</b>	<b>46</b>



## Welcome

This publication describes the key concepts of StorHouse/RM. It explains the structures that store relational data on StorHouse and the facilities that access and manage that data.

# Products and applications

## Detail data

Detail data is the most granular level of information that an enterprise can collect at a single point of service. A call detail record, ATM transaction, point-of-sale data, clickstream, photo, audio or video clip, and e-mail message are all examples of detail data.

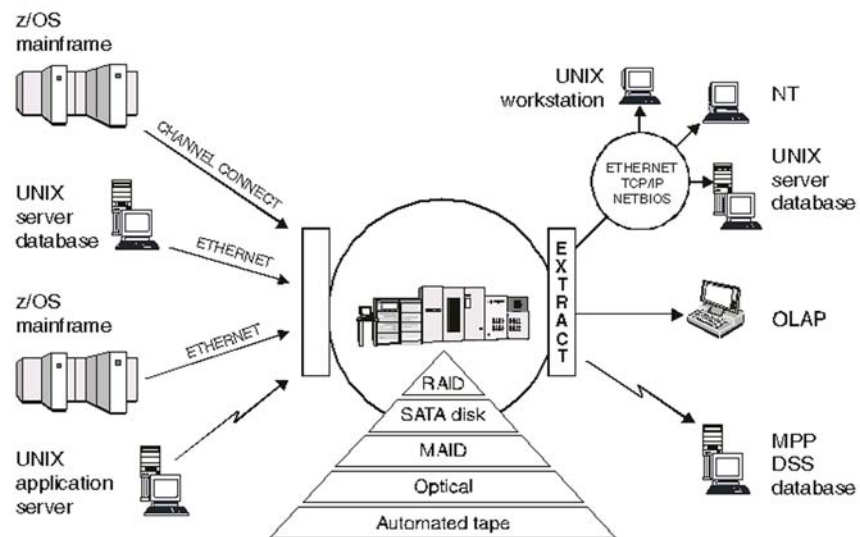
## StorHouse/SM

StorHouse/SM, the storage management component, controls a hierarchy of storage devices composed of cache, redundant array of independent disks (RAID), Serial Advanced Technology Attached (SATA) disks, massive array of idle disks (MAID), erasable and write-once-read-many (WORM) optical disk jukeboxes, and erasable and WORM automated tape libraries. StorHouse/SM is also responsible for critical system management tasks, like data migration, backup, and recovery. It provides system-managed storage that optimizes media usage, response time, and storage costs for each application. StorHouse/SM runs on Sun™ Microsystems™ Sun Fire™ Servers, on Hewlett-Packard PA-RISC systems, and on IBM® pSeries servers.

*StorHouse®* is the FileTek® enterprise-wide solution for managing the capture, storage, movement, and access of gigabytes (GB) to petabytes (PB) of relational and non-relational detail data. StorHouse technology combines industry-leading, scalable storage devices and Open System processors with specialized storage management and relational database management system (RDBMS) software components. Together, these components make StorHouse the ideal hub server, active archive, or database extension for data warehousing and high volume data-intensive applications such as Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), and e-business.

## StorHouse as a hub server

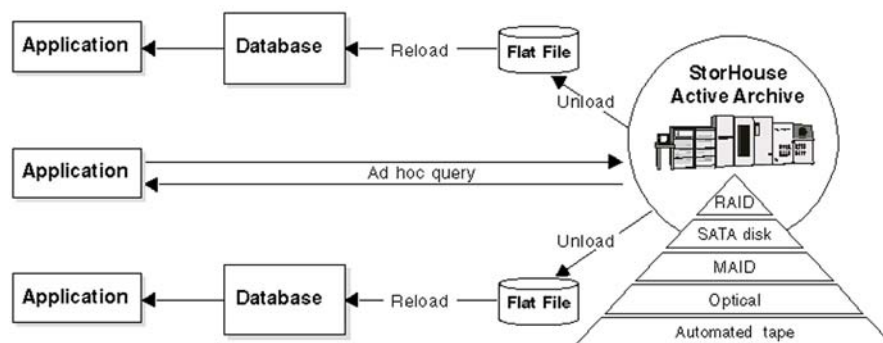
As a hub server, StorHouse is an enterprise-wide data warehouse for transaction-level data. Different operational systems can load detail information into the centralized StorHouse hub, and multiple dependent data marts can query, or mine, the hub on a regular or ad hoc basis.



The StorHouse hub server stores static operational data from multiple operational systems safely and securely. As a centralized, single storage repository, it provides timely, shared access to one data source, a consistent view of business activity across the enterprise, and information used to fuel data marts for decision support and other analytical applications.

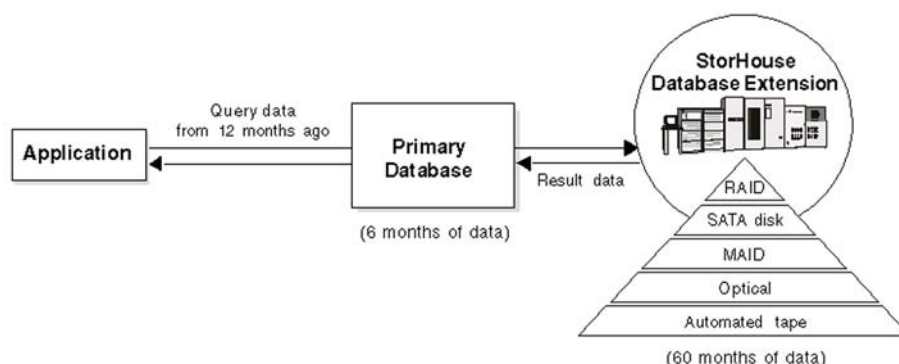
## StorHouse as an active archive

As an archive server, StorHouse is a high capacity back-end archive for one or more database systems. You can store operational and informational data in StorHouse and then unload/reload it into the database as required. Data is always available for ad hoc querying and can be restored to the database for decision support at any time. Some typical active archive applications include bill regeneration, subpoena compliance, and audits.



## StorHouse as a database extension

As a database extension, StorHouse is a high capacity back-end storage repository for a primary database. Users and applications need not know the location of the data. They simply submit a query through the primary database. A database extension is suitable for operational and historical data, as well as relational and non-relational data.



### StorHouse/RM

StorHouse/RM, the FileTek RDBMS component, works in conjunction with StorHouse/SM to specifically administer the storage, access, and movement of relational data. StorHouse/RM provides row-level SQL access to high volumes of detail data on any layer, including tape, in the StorHouse storage hierarchy. SQL access is available from different platforms through a variety of industry-standard protocols.

### StorHouse/Control Center

StorHouse/Control Center is the FileTek Windows®-based network computing system for providing administrative control of StorHouse. StorHouse/Control Center consists of one or more servers that communicate with clients over an IP network. The StorHouse/Control Center server provides network connectivity to StorHouse. The StorHouse/Control Center clients consist of one or more graphical user interface (GUI) modules for performing StorHouse system and database administration tasks, configuring and managing servers, and analyzing StorHouse activity and performance.

# StorHouse databases

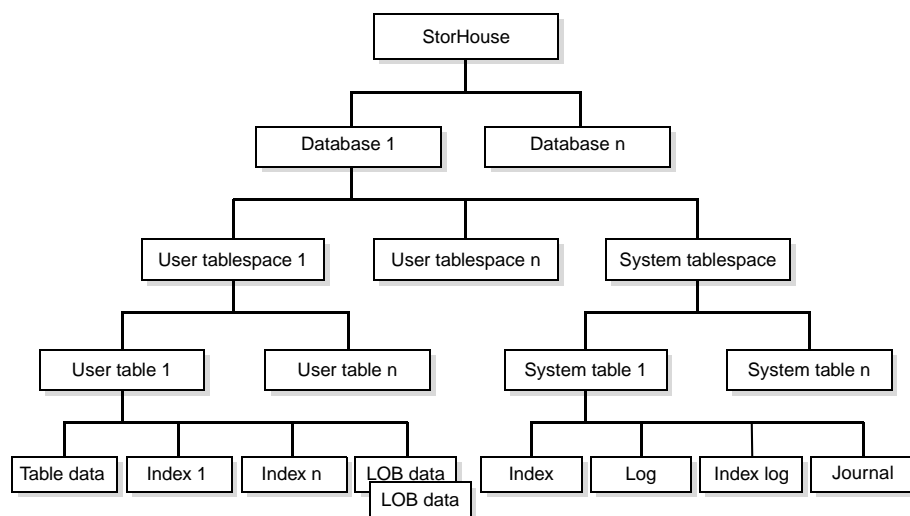
## Capabilities

- Maximum number of user tablespaces: no meaningful limit
- Maximum number of user and system tables in a database: no limit
- Maximum number of indexes in a database: no meaningful limit
- Maximum number of indexed columns for a table: 2400

A *StorHouse database* is a write-once-read-many database, ideal for storing high volumes of detail data. Each database contains:

- User table data that you store and access
- Optional indexes—value, hash, and range—that locate the table data
- Metadata that describes database components

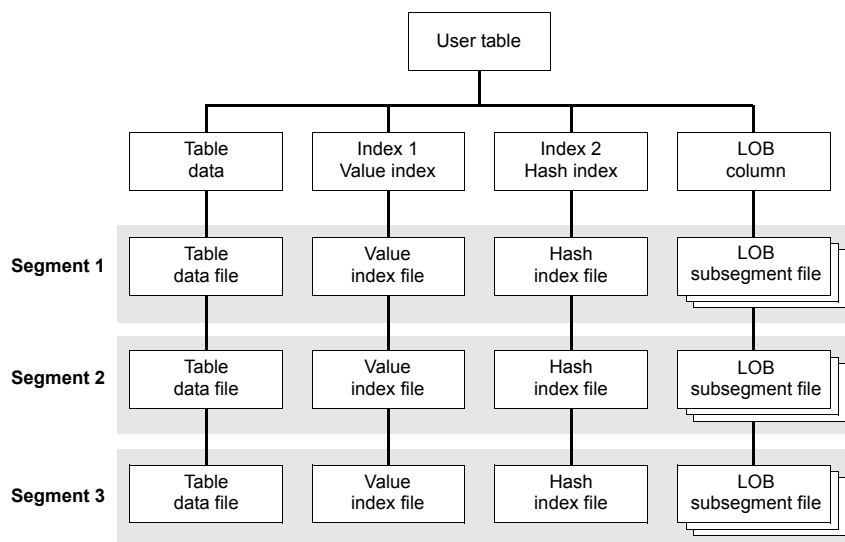
A StorHouse database has both a logical and a physical structure. Logically, user tables, indexes, and large object (LOB) data reside in user tablespaces and metadata resides in a system tablespace. Physically, these components reside in files. Table data, indexes, and LOB data can reside in the same user tablespace (as shown below) or different user tablespaces.



## StorHouse database user files

StorHouse database user files reside on the StorHouse storage hierarchy. Each user table consists of one or more *segments*. Each segment consists of a table data file, an index file for each index, and one or more LOB subsegment files. Each time you load data into a user table, StorHouse creates a segment with this set of *segment files*. You can load multiple segments at a time and replace existing segments. Replacing a segment does not remove it from StorHouse but rather invalidates it, making the files in the segment inaccessible. You can also merge (or coalesce) and delete segments later. Merging segments enhances performance because it reduces the number of file and extent opens and closes for a query.

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



Range indexes are stored in system tables instead of segments. Depending on size or by user request, LOB values may be stored in the table data file, or LOB values in different columns may be stored in the same LOB subsegment file, or LOB values in a single column may be stored in multiple LOB subsegment files.

## StorHouse database system files

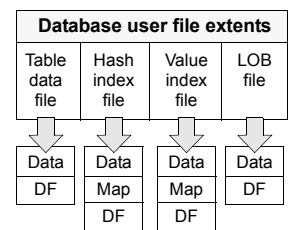
StorHouse database system files reside on and are managed by the UNIX® file system. These database system files contain the following *system components* or *metadata*:

- System tables
- System table indexes
- System table logs
- System table index logs
- (Optional) Journal files

Each system component is a separate UNIX file. The journal files must be on disk systems separate from the other files.

## Extents

Segments consist of StorHouse files that can reside on any storage device in the StorHouse storage hierarchy. Files are composed of different extents, or file components:



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

You can hold some or all extents in the performance buffer to enhance performance. For instance, you can hold the value index and hash index DF and map extents in the performance buffer longer than the table data extent to speed access to the data.

# User tables

## What's different about StorHouse user tables?

- User tables are read-only.
  - Ensures a permanent record of critical detail data
  - Maximizes concurrency of data to all users (minimal locking/no updating)
  - Eliminates need to allocate additional space for inserts and index updates
  - Eliminates need for continual backups
- All table data is bulk-loaded.
  - Loads faster
  - Optimizes data organization
- User tables are loaded in multiple segments.
  - Maximizes concurrent loading
  - Bypasses time-intensive index updates
- Dropped user tables may be undropped, providing a safety mechanism for errors.
- Segments can be invalidated and later validated or deleted.
  - Provides a way to recover from load errors or to replace data
  - Prohibits access to certain table data

A *user table* is the basic unit of data storage in a StorHouse database. User tables hold user-accessible data. Logically, StorHouse user tables are like most RDBMS user tables; they consist of columns and rows of data. Each row contains data values conforming to the constraints of the columns that make up the row.

*Columns represent specific data types and domains.*

	ACCOUNT	LAST_NAME	BILLING
<i>Rows represent specific database records or tuples.</i>	91256	ANDERSEN	12023.98
	91347	WHITE	39022903.89
	91486	MCGUIRE	3988229.55
	97865	CORNFELD	283.00
	99003	HAWKINS	109936.10

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

Physically, user tables are stored in files on the StorHouse storage hierarchy. The user tablespace defines the target storage device and the migration path through the hierarchy. For instance, you can store time critical data on magnetic disk and then migrate that data to tape or optical as the data ages. The StorHouse software automatically manages storage and migration based on your user tablespace parameters.

## Integrity constraints on tables

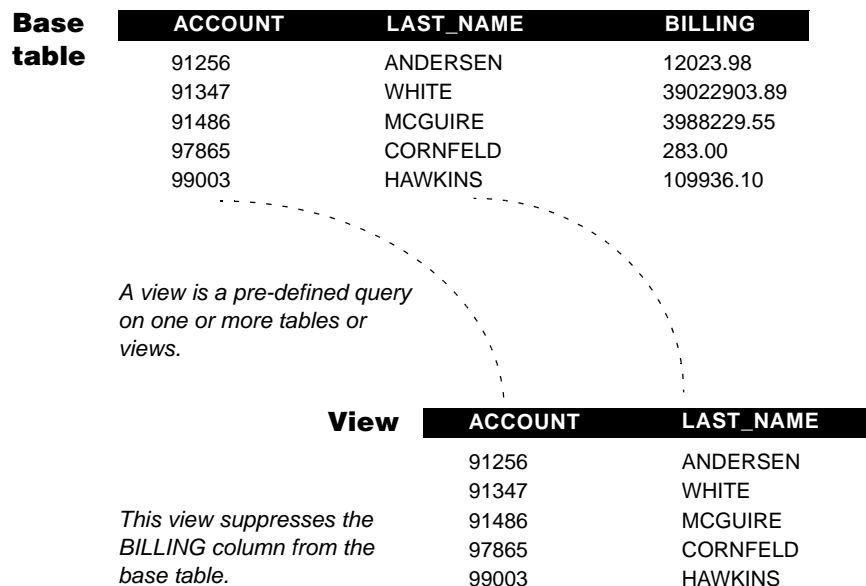
The *data type* of a column determines the maximum length of data values in the column and the kind of data that is valid for the column. A data type enforces an *integrity constraint* on a column. When you load data into a user table, the value that you load into a column must be consistent with the length and compatible with the data type of that column.

Prohibiting null values in a column is another integrity constraint. A *null value* stands for an unknown, not applicable, or missing value. If a column allows null values, then you don't have to load data values into that column. But remember that StorHouse user tables are read-only; so, if you define a column as null and don't load data into that column, then those data values will always be null.



## Views on tables

You can create a virtual table or *view* that appears and acts like a table but draws its content from one or more existing tables and/or views. A view does not actually contain data and it does not take up storage space like a table. Instead, a view is defined by a query that references base tables. When you query a view, you actually query the tables referenced by the view. Views are useful for tailoring or limiting user access to data. They provide convenience, security, or both by letting you determine which data in which tables is available to which users.



You can create views on user tables and system tables, but you can't insert, update, or delete rows in views based on user tables. StorHouse user tables are read-only.

## Column data types

**BIGINT** – signed big integer (64-bit) from -9223372036854775808 through 9223372036854775807

**BINARY** – array of bytes with a length from 1 to 256

**BLOB** – variable-length array of bytes with a length from 1 to 2147483638 bytes

**CHAR** – array of characters with a length from 1 to 256

**CLOB** – variable-length array of characters with a length from 1 to 2147483638 bytes

**DATE** – date value in month, day, year

**DOUBLE PRECISION** – double precision, floating-point number

**INTEGER** – signed integer from -2147483648 through 2147483647

**NUMERIC** or **DECIMAL** – decimal fixed-point number with precision and scale up to 31 digits

**REAL** – single precision, floating-point number

**SMALLINT** – signed small integer from -32768 through 32767

**TIME** – time value in hours, minutes, seconds, and optionally milliseconds

**TIMESTAMP** – date and time combination with optional milliseconds and microseconds

**VARBINARY** – variable-length array of bytes with a length from 1 to 32705 bytes

# Large objects

## LOB data types

The following data types define LOB columns and mechanisms for loading, unloading, and accessing LOB data.

### BLOB and CLOB.

- When creating a table, defines a column as a variable-length array of bytes (BLOB) or as a variable-length character string of character-based data (CLOB) up to 2 GB in size.
- When loading or unloading, defines LOB data in the input data file or the result file.

### BLOB\_FILE and CLOB\_FILE.

- When loading, specifies host, path, and user information to access separate LOB files containing BLOB or CLOB values.
- When unloading, specifies host, path, and user information to write LOB result files for BLOB or CLOB values.
- When transferring LOB data to a client file through ESQL, defines a file reference variable to represent the file.

**BLOB\_LOCATOR and CLOB\_LOCATOR.** Defines a locator variable to identify, access, and manipulate a BLOB or CLOB at the StorHouse server.

StorHouse/RM supports the storage and retrieval of large objects (LOBs), including binary large objects (BLOBs) and character large objects (CLOBs). LOBs may be stored with the rest of the table data (in-line) or in separate files (out-of-line). StorHouse/RM determines in-line and out-of-line storage on a row-by-row basis.

## In-line LOBs

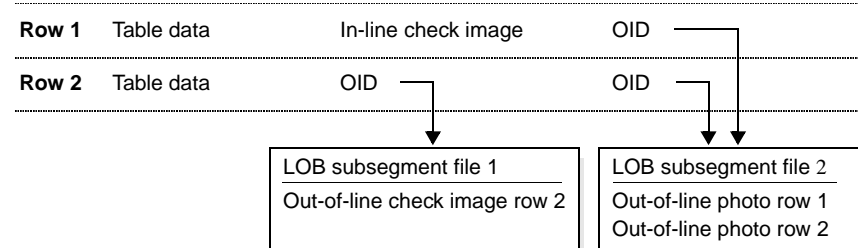
An *in-line LOB* is a LOB value that's stored with the table data. If space permits—a LOB value does not exceed any user-defined in-line limit and the row does not exceed 32 KB—you can store a LOB value in a table data file. For instance:

Row 1 - 31 KB	Table data 3 KB	In-line check image 28 KB
---------------	-----------------	---------------------------

StorHouse/RM treats in-line LOB values like any other variable-length, nullable field. In-line storage is the default, that is, StorHouse/RM attempts to store LOBs with the table data unless you specify otherwise.

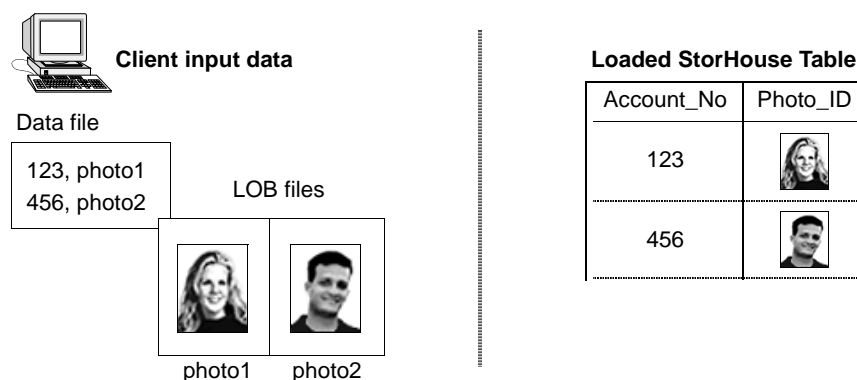
## Out-of-line LOBs

An *out-of-line LOB* is a LOB value that's stored in a separate file—called a *LOB subsegment file*—from the table data. When defining a LOB column, you can choose to: (1) always store LOB values out-of-line, (2) or store LOB values in-line when possible and out-of-line when necessary, (3) or even store values of different LOB columns in the same LOB subsegment file, for instance, check images and photos in the same LOB subsegment file. For out-of-line LOBs, StorHouse/RM inserts an *object identifier* (OID) in the table row to identify the LOB subsegment file containing the LOB value.



## LOB loading and unloading

With the FileTek FTP Data Loader, you can load LOB data from three locations: (1) in a client file with the other input data, (2) in separate LOB data files (one LOB value per file) on your client computer, (3) in separate LOB data files on a remote system. For instance, in the following example, account numbers reside in a client data file and photos reside in separate LOB files. The data file contains the file names containing the photos.



Likewise, with the FileTek FTP Data Unloader, you can unload LOB data to three locations: (1) in a client file with the other result data, (2) in separate LOB result files (one LOB value per file) on your client computer, (3) in separate LOB result files on a remote system.

## LOB access

When accessing a LOB value from an embedded SQL (ESQL) program, you can:

- Place the entire LOB value into a host variable that is large enough to hold the value. The entire LOB value moves from the StorHouse server to the client.
- Place a reference to a LOB value or part of a value into a *locator variable*. The locator variable moves to the client while the LOB value remains on the StorHouse server. A program can manipulate the value using the locator variable and fetch it in many parts back to the client.
- Place a file name into a *file reference variable* to identify a client file to which LOB data can be moved. For instance, you can transfer an XML document from StorHouse to a client file to be read by an XML reader.

## LOB functions

The following functions take LOB arguments:

- ASCII
- BIT\_LENGTH
- BLOB
- CHAR\_LENGTH
- CLOB
- CONCAT
- COUNT
- COUNT\_BIG
- INITCAP
- INSTR
- LENGTH
- LOWER
- LPAD
- LTRIM
- NVL
- OCTET\_LENGTH
- OVERLAY
- POSITION
- RPAD
- RTRIM
- SUBSTR
- TO\_CHAR
- TO\_DATE
- TO\_HEX
- TO\_NUMBER
- TO\_TIME
- TRANSLATE
- TRIM
- UPPER

## LOB statements

These SQL statements are exclusive for LOBs.

- VALUES INTO  
manipulates a LOB selected with a locator variable.
- FREE LOCATOR  
releases a locator variable before the end of a transaction.

## LOB operators

The following operators are valid for LOB operands: equal (=), unequal (<>), and string concatenation (||).

# Indexes

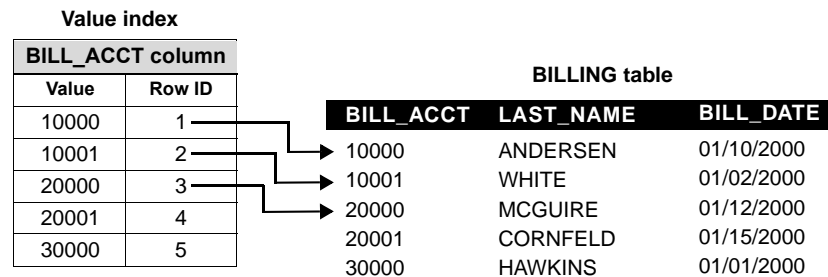
## Capabilities

- Maximum number of indexed columns for a table: 2400
- Maximum length of an indexed column (including VARBINARY and VARCHAR columns): 256 bytes
- Maximum number of columns in a compound index: 16
- Maximum size of a compound index: 2048 bytes

*Indexes* provide efficient access to table data. You can create an index on a column or combination of columns in a user table. An index based on one column is a *simple index*. An index based on multiple columns is a *compound index*. StorHouse supports three index types—value, hash, and range.

## Value index

Value indexes work best with queries that return multiple rows based on a range of values. A *value index* contains an ascending list of all the values in a column (or group of columns for a compound value index). For each column value, the index contains an index map to the table row containing that value. By searching the index rather than the table, then matching column values to row IDs, StorHouse/RM can more efficiently find requested table rows.



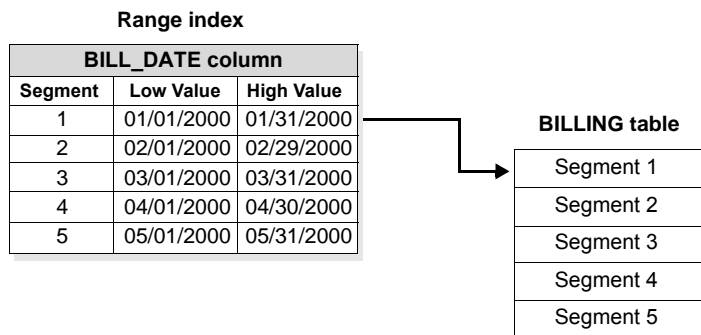
## Hash index

Hash indexes work best with queries that return specific rows based on a specific value. A *hash index* is a two-part index based on an index map extent and a hit list that uses a proprietary StorHouse algorithm to effectively locate individual table rows based on individual index values. For hash indexes (unlike for value indexes), StorHouse/RM must access the data row to determine that the row's column content meets the selection criteria. StorHouse/RM, for example, might use a hash index for the following SQL statement:

```
SELECT ACCOUNT
FROM CUSTOMER
WHERE ACCOUNT = 99003
```

## Range index

Range indexes are useful for user tables with multiple segments. A range index contains the lowest and highest column data values for each segment in a user table. Instead of searching through multiple segments, StorHouse/RM first looks at the range index to find the specific segment with the requested data values. Then, StorHouse/RM might use any hash or value indexes to find a specific data value or range of values in the segment. For instance, in the following example, if you requested data for 01/15/2000, StorHouse/RM would check the range index and determine that segment 1 contains the requested value.



Each time you load new data into a table, StorHouse/RM enters the low and high data values into the range index. If there's one segment, there's one low and high value pair in the range index. If there are 100 segments, there are 100 low and high value pairs in the range index.

Range indexes are stored in a set of system tables. There are one or two system tables for each column data type.

## Index basics

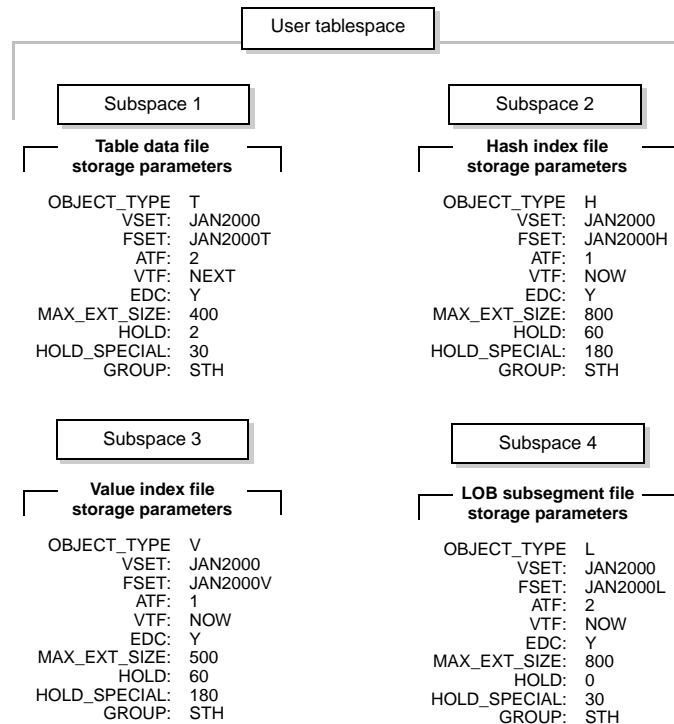
- You can create any index type for any non-LOB column.
- You can create different types of indexes for the same column. For instance, you can create a range and a value index for the same column. StorHouse/RM decides which index to use to satisfy a specific query.
- You can create indexes before or after you load the table data. A deferred index is an index created after a table is loaded.
- Value and hash indexes are stored in index files on the StorHouse storage hierarchy.
- Range indexes are stored in system tables on the UNIX file system.
- Index files can reside on different storage devices from table data files and LOB subsegment files.
- The StorHouse software automatically manages the migration and backup of index files based on the user tablespace definition.
- All index values are stored in ascending order.
- Invalidating a segment invalidates the segment's index files as well as the table data file and any LOB subsegment files.
- Deleting a segment deletes all its segment files, including index files.

# User tablespaces

## User tablespace basics

- A database has one or more user tablespaces.
- When you create a user table, you assign it to a user tablespace.
- You can assign LOB columns to the same user tablespace as the table or to different user tablespaces.
- When you create an index for a user table, you can assign it to the same user tablespace as the table or to a different user tablespace.
- You can assign different indexes of a table to different user tablespaces.
- Table data files of a user table can be stored in multiple VSETs and FSETs.
- Index files and LOB subsegment files can be stored in multiple VSETs and FSETs and different VSETs and FSETs from their associated table data file.
- You can change storage parameters later to store subsequent segments differently.

A *user tablespace* defines where segment files are stored on the StorHouse storage hierarchy. It also sets attributes that influence storage management, like backup and migration. Each user tablespace consists of one or more *subspaces* that define different storage parameters for different components.



**Assigning storage.** The VSET (volume set) and FSET (file set) subspace parameters assign storage on StorHouse. A *VSET* is one or more physical volumes that are treated as a logical unit of storage. A *volume* is a unit of media, such as an optical disk cartridge or a tape cartridge, on which data can be recorded and read. You use VSETs to control the physical grouping of files. An *FSET* is an area of storage within a volume set. Files are stored in FSETs.

Whether you use multiple subspaces and different VSETs and FSETs in a user tablespace depends on your data and your access and performance requirements. For example, to manage the storage of table data, indexes, and LOB data in different ways but remove them from StorHouse at the same time, you can create multiple subspaces and assign different FSETs for the same VSET. Or to manage the storage of all components the same way, you can create one subspace and assign one FSET in one VSET.

**Migrating performance copies.** The ATF (*Access Time Factor*) subspace parameter works with the StorHouse migrate function to keep data that is most likely to be accessed in the StorHouse performance buffer while maintaining a supply of free space. Additionally, the HOLD (for data extents) and HOLD\_SPECIAL (for DF and map extents) subspace parameters define the number of days to hold these extents in the performance buffer.

**Creating performance and primary file copies.** The VTF (*Vulnerability Time Factor*) subspace parameter controls the creation of performance and primary copies of segment files. Performance copies reside in the StorHouse performance buffer. Primary copies reside in resident file sets on the designated StorHouse media. During a load, StorHouse can:

- Bypass the performance buffer and directly write the files to their resident file sets. StorHouse, however, always writes DF and map extents to both the performance buffer and resident file sets.
- Write extents to the performance buffer first and copy them to their file sets second. StorHouse copies each extent to the resident file set after creating the extent on the performance buffer. So for a table file, StorHouse creates the data extent on the performance buffer and then copies the data extent to its resident file set. Then StorHouse creates the DF extent on the performance buffer and then copies it to the file set.
- Write extents to the performance buffer then copy them to their file sets during the next StorHouse write-back operation.

**Using error detection checking.** The EDC (Error Detection Code) subspace parameter determines whether to use the StorHouse error detection feature for files. StorHouse generates EDCs during data loads and uses EDCs to detect errors during data movement in StorHouse.

**Setting data extent sizes.** The MAX\_EXT\_SIZE subspace parameter specifies the maximum data extent size (in megabytes) that StorHouse/RM writes to StorHouse during a load operation. StorHouse checkpoints each data extent when it reaches the maximum size and then creates a new one. And StorHouse can recover data extents to the last successful checkpoint.

## Subspace basics

- A user tablespace contains one or more subspaces.
- A subspace defines the storage parameters for a specific component—table data only, hash indexes only, value indexes only, or LOB data only—or for all components.
- The OBJECT\_TYPE parameter determines the type of component allowed in a subspace: T for table data, H for hash index, V for value index, L for LOB data, or "" or " " for all types.
- You can create multiple subspaces for the same component type, for instance, one subspace for January table data, another subspace for February table data, and so on.
- When loading data, loading deferred indexes, and merging segments you can use default subspaces, select specific subspaces, or rotate among subspaces for component types.

# Metadata

## Metadata basics

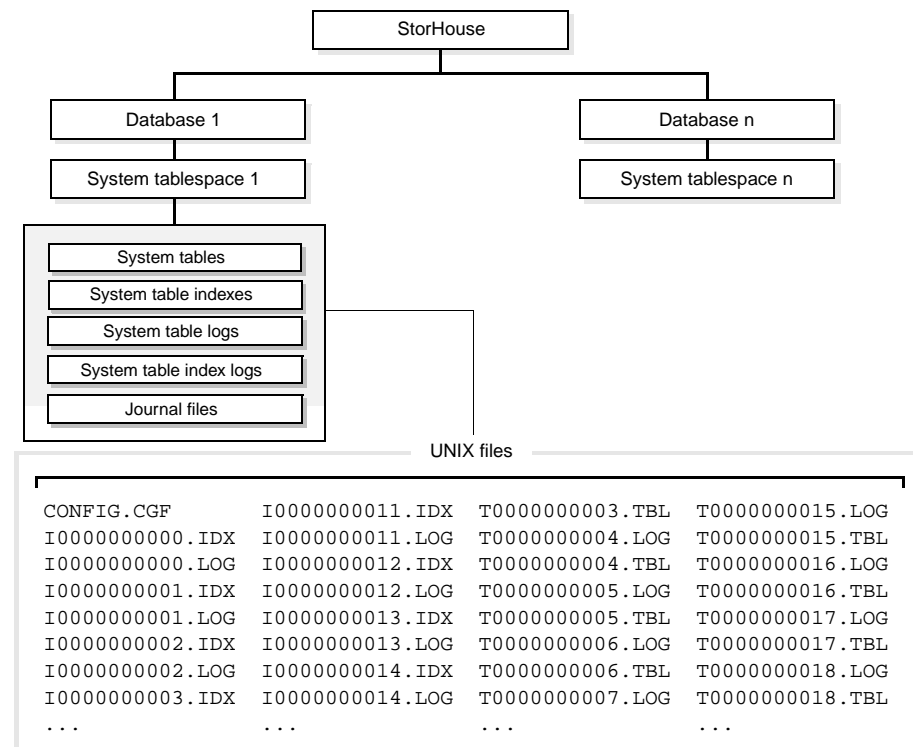
- StorHouse/RM creates a system tablespace and metadata for each new database.
- Each system component is a separate UNIX large file.
- All system tables have corresponding logs.
- Some, not all, system tables have indexes.
- Range indexes are stored in system tables.
- The StorHouse metadata backup utility backs up system components and other files in the system tablespace.
- The StorHouse metadata restore utility recovers system components with the latest metadata backup files.
- The redo journaling utilities recover changes to system components since a metadata backup.
- System tables, indexes, and logs are on storage devices separate from journal files.

*Metadata* are system components that StorHouse/RM creates and uses to manage a database. These components, stored on the UNIX file system within a system tablespace, are as follows:

- System tables
- System table indexes
- System table logs
- System table index logs
- Journal files

## System tablespaces

For each database, StorHouse/RM creates a separate database directory on the StorHouse server. This database directory, also called *system tablespace*, contains all of the system components for a specific database. Physically, the database system components are UNIX files.





## System tables

StorHouse/RM creates a set of system tables for each database in the database system tablespace. *System tables* contain information about a database. StorHouse/RM updates system tables when you create database components, reads system tables to verify that database components exist and that accounts are authorized to access them, and updates system tables after confirmed loads. Authorized users can query system tables by submitting a SELECT statement. For instance, you can query the SYSTBLSPACES system table to list the user tablespaces in a database or the SYSINDEXES system table to determine the types of indexes defined for a user table. The following table describes each system table.

SYSTEM TABLE	DESCRIPTION
SYSCOLAUTH	Contains column update privileges
SYSCOLUMNS	Describes columns of user tables
SYSDBAUTH	Lists account database privileges
SYSDROP_PEND	Contains information about dropped user tables and indexes
SYSINDEXES	Defines each indexed column
SYSPACKAGE	Defines each package in a database
SYSPACKSTMT	Contains each statement in a package
SYSRANGES_datatype	Contain range index entries
SYSSMUSERS	Contains default user tablespaces
SYSSTAT_COL	Contains statistics for each indexed column of a segment
SYSSTAT_HIST	Contains the spread and frequency of each histogram bucket
SYSSTAT_IDX	Contains row averages for queries that use value indexes
SYSSTAT_SMATRIX	Contains matrixes for calculating spreads of non-numeric values
SYSSTHFILES	Contains StorHouse file and group names for segment files
SYSSTHSEGMENTS	Contains information about segments
SYSSTHSPACES	Describes subspaces in user tablespaces
SYSSYNONYMS	Defines synonyms in a database
SYSTABAUTH	Contains account table privileges
SYSTABLES	Defines each user table, system table, and view
SYSTBLSPACES	Defines each user tablespace in a database
SYSVIEWS	Describes each view in a database

## System table indexes

StorHouse/RM creates system table indexes for specific system tables when a database is created. System table indexes are stored as UNIX files in the same directory as the system table files. Their operation is transparent to you.

## System table logs

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

## System table index logs

Each system table index has a log that can recover changes to system table indexes. The operation of the index log is similar to the system table log.

# Storage management

## File migration factor

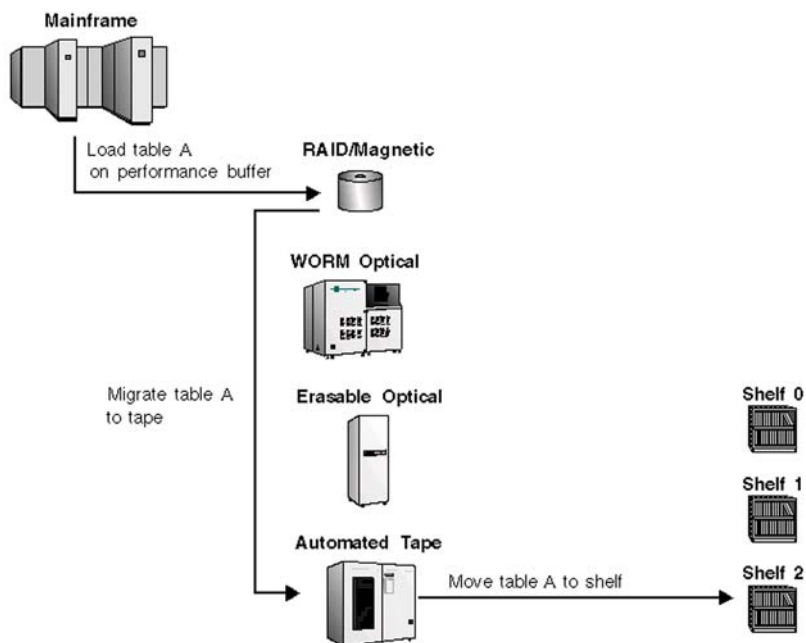
The purpose of file migration is to keep highly accessed extents in the performance buffer while maintaining a supply of free space in the buffer for new high-access files. To accomplish this, StorHouse maintains a migration factor for each extent. User-controllable parameters determine *when* to start a migration. The migration factor determines *which* extents to migrate. This migration factor is derived from the file ATF attribute, the size, and the access history. Typically:

- Extents with smaller migration factor values are migrated off the performance buffer first.
- Extents with many accesses tend to have larger migration factors than extents with few accesses.
- Extents with larger sizes tend to have smaller migration factors than extents with smaller file sizes.
- Extents with older accesses tend to have smaller migration factors than extents with more recent accesses.

StorHouse/SM complements its relational counterpart by providing system-managed storage features for table data files, index files, LOB subsegment files, metadata backup files, and archived journal files located anywhere in the StorHouse storage hierarchy.

## File management

With StorHouse/SM, you can design and tune data availability strategies and migration paths through the storage hierarchy. For instance, you can initially load table data on the performance buffer for fast write time and fast access, or you can load table data directly to optical or tape. StorHouse/SM can automatically duplicate file copies in different libraries to increase data availability and to improve access performance. As access requirements diminish, StorHouse/SM can migrate files to a lower-cost-per-megabyte media or move files located on selected volumes to shelf storage automatically based on usage, access requirements, available space, and user-defined parameters. The following diagram shows an example of how data can move through the storage hierarchy.



## Volume management

A *physical volume* is a unit of media on which data can be recorded and read. Optical disk and tape cartridges are examples of physical volumes supported by StorHouse. A *logical volume* is one side or surface of a physical volume. Some physical volumes, such as magnetic tape, have only one usable side and contain only one logical volume. Others, such as most types of optical disks, have two usable sides and contain two logical volumes. Some of the features for managing volumes are described here.

**Migration.** StorHouse automatically migrates volumes between libraries and shelf devices. It selects volumes that aren't in use, have the oldest access time, or aren't flagged for retention—or *volume holding*—in the library device. You can also manually move specific volumes between library devices or between a library device and shelf storage at any time.

**Retirement.** This feature protects against read errors due to erasable media degradation. StorHouse retires a volume by moving file extents from erasable volumes (such as tape) to one or more other volumes in the same volume set. StorHouse selects volumes for retirement based on media mount limits and volume mount counts.

- The *media mount limit* (recommended by the media manufacturer and initially set at installation) indicates the number of mounts that a volume of the media can undergo before the risk of unrecoverable errors due to media degradation exceeds a safe threshold.
- The *volume mount count* indicates the number of times a volume has been used (mounted and dismounted).

You can also manually retire a volume regardless of the mount count and mount limit values.

**Erasure.** For erasable optical and tape media, you can erase entire volumes and volume sets to remove unused data. Space on erased volumes is available for reallocation in the volume set.

### Storage performance features

- Volume holding lets you keep frequently accessed volumes in a library device. Volumes that you hold migrate to shelf storage after other volumes.
- Pre-emptive priority processing ensures that all transaction-oriented, time-sensitive requests have priority over sequential or batch requests.
- Look-ahead queuing minimizes response time by servicing all current requests for a mounted cartridge before dismounting it.
- Platter cycling ensures that a greater percentage of chronological data is mounted and available for retrieval without the need to flip cartridges.
- Duplexing simplifies file recovery, increases the availability of data to near 100 percent, and improves overall response time. StorHouse automatically determines when to access the duplex copy instead of the primary copy for better overall system performance.

# Backup

## File copies

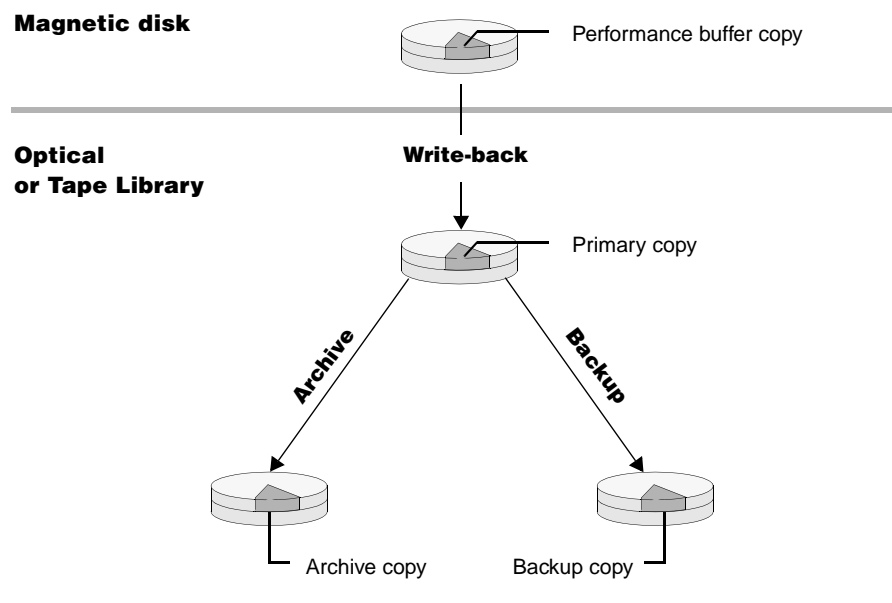
- Performance buffer copy – active file extents stored on high-speed magnetic disk for optimal access.
- Primary copy – a file copy available for normal access.
- Backup copy – a duplicate copy of a primary file. You can use a backup copy to recover a primary file that has been corrupted or destroyed.
- Archive copy – a duplicate copy of a primary file. You can use an archive copy to recover a primary file that has been corrupted or destroyed.

StorHouse backup facilities simplify data recovery should the need arise.

## Backup operations for segment files

StorHouse provides backup operations for creating secondary copies of table data files, index files, and LOB subsegment files.

- *Write-back* copies new file extents from the performance buffer to their primary file sets. The VTF attribute in each subspace of a user tablespace determines when write-back occurs for segment files.
- *Backup* creates a backup copy of a primary file, leaving the source file (primary) intact. You can schedule a backup to run automatically at specified intervals.
- *Archive* creates an archive copy of a primary file, leaving the source file intact. You can schedule an archive to run automatically at specified intervals.
- *Replicate* creates a second copy of a file on another StorHouse system.

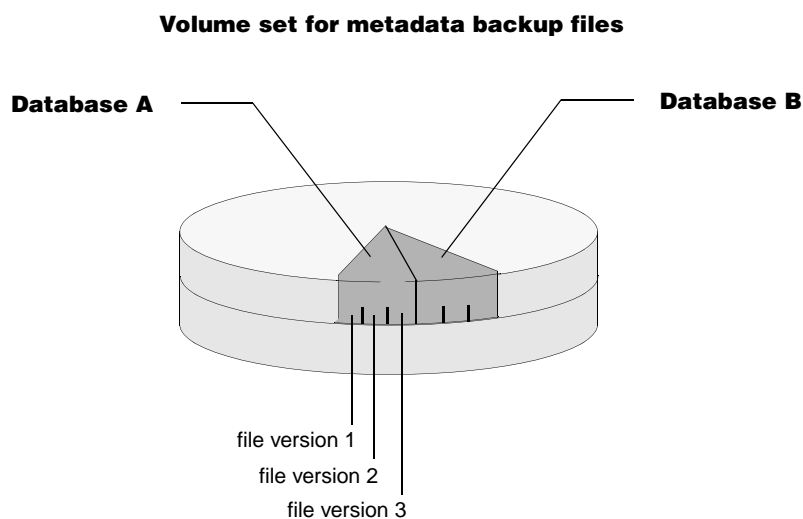


## Backup utility for metadata

The *metadata backup utility* copies all files from a database directory on UNIX to one primary file copy on the StorHouse storage hierarchy. These files include:

- System tables, including range indexes
- System table indexes
- System table logs
- System table index logs

This utility creates metadata backup files in the file set and volume set that you name using StorHouse system parameters. All metadata backup files for all databases are grouped in one file set and volume set. Each time you run a backup, the utility creates a new backup file version for the specified database.



You can create backup and archive copies of metadata backup files, then move those copies to shelf and store them off-site for disaster recovery purposes.

### Metadata backup utility features

- You can back up metadata for one database or multiple databases at the same time.
- You can schedule a metadata backup to run at specific frequencies (like hourly, daily) and times.
- The metadata backup utility stops when it detects any active loads.
- You can set a limit for the number of backup file versions to keep. The maximum is 99 for each database.
- StorHouse automatically deletes old file versions (based on your limit) to make room for new ones.
- The most recent metadata backup file is the default version used to recover metadata. However, you can specify an older version to restore metadata through older backups.
- You can enable journaling for an existing StorHouse database and reset the journaling environment with metadata backup command options.
- Only an authorized account with the StorHouse OPERATOR, SERVICE, or SYSTEM privilege can run the metadata backup utility.

# Recovery

## Disaster recovery, general recovery

Disaster recovery is the process of re-creating a destroyed production system after a devastating event or natural disaster (for example, a fire, flood, tornado, or earthquake). Such events can prevent users from writing and accessing critical business data because of damaged or destroyed hardware, software, and media. StorHouse provides software features that expedite disaster recovery. For more information about these features, refer to the *FileTek Recovery Strategies* manual, publication number 900117.

General recovery is the process of re-creating misplaced files and broken cartridges or recovering data because of events like StorHouse/RM failures, power losses, or operating system crashes. For instance, should heavily accessed tape cartridges begin to degrade, general StorHouse user file recovery can re-create the data on new volumes and retire the old ones. Or if your site loses power while StorHouse/RM is updating the metadata, general metadata recovery can protect your database and restore it to a consistent state.

StorHouse attempts to recover from error conditions automatically, but there are some conditions that require assistance. In the event of a failure or error, StorHouse provides recovery tools for metadata and segment files. This topic focuses on general recovery features.

## Metadata recovery

The purpose of metadata recovery is to protect databases left in an inconsistent state due to a failure.

- An *inconsistent database* occurs when a single atomic operation that updates one or more system tables fails before completing and committing all updates.
- An *unprotected database* allows access (“dirty reads”) and further updates to inconsistent metadata.

Structures used to recover metadata are metadata backup files, redo journals, and undo records in system table logs and system table index logs. Processes used to recover metadata are the automated metadata recovery process, the metadata restore utility, and redo journaling utilities. Metadata recovery scenarios are described below.

**SQL engine failure.** If an engine (connection instance) terminates abnormally during DDL or metadata UPDATE, INSERT, or DELETE processing, the *automated metadata recovery process* invokes a recovery process to roll back incomplete updates and non-committed metadata. Any locks in place at the time of the failure are held and then released after the roll back.

**Power loss or system crash.** When the system initializes after a power loss or operating system crash, the automated metadata recovery process inspects the database directories for an inconsistent state and invokes a recovery process for each inconsistent database. The recovery process locks the metadata to prohibit dirty reads and then releases the locks when recovery completes.

**Hardware failure.** In the unlikely event that the magnetic disks containing the database directories should fail, you can run the *metadata restore utility* to recover the metadata of one or more databases with metadata backup files. And, you can use *redo journaling utilities* to recover committed transactions since the last metadata backup or to a point in time (specified in days).

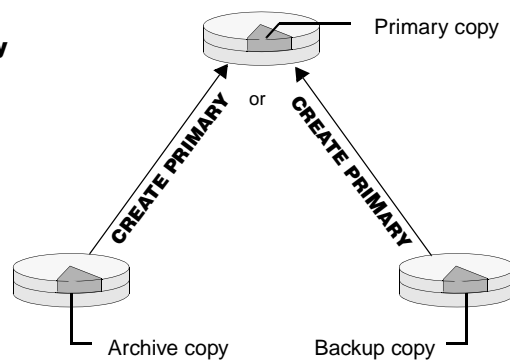
## Segment file recovery

The purpose of segment file recovery is to ensure access to table, index, and LOB data. Structures used to recover segment files are backup and archive file copies. Recovery scenarios are described below.

**Broken or misplaced volume.** If a volume containing segment files breaks or becomes misplaced or unreadable, you can create a replacement volume as long as you have backup or archive copies of the unreadable files. To recover a volume, you first disable it and then issue the StorHouse RECOVER VOLUME command.

**Unreadable files.** If a primary copy of a segment file is unreadable for any reason, you can create a new primary copy from the backup or archive copy. To recover unreadable segment files, first delete and remove the unreadable primary files and then re-create the primary files with the StorHouse CREATE PRIMARY command.

### Optical or Tape Library



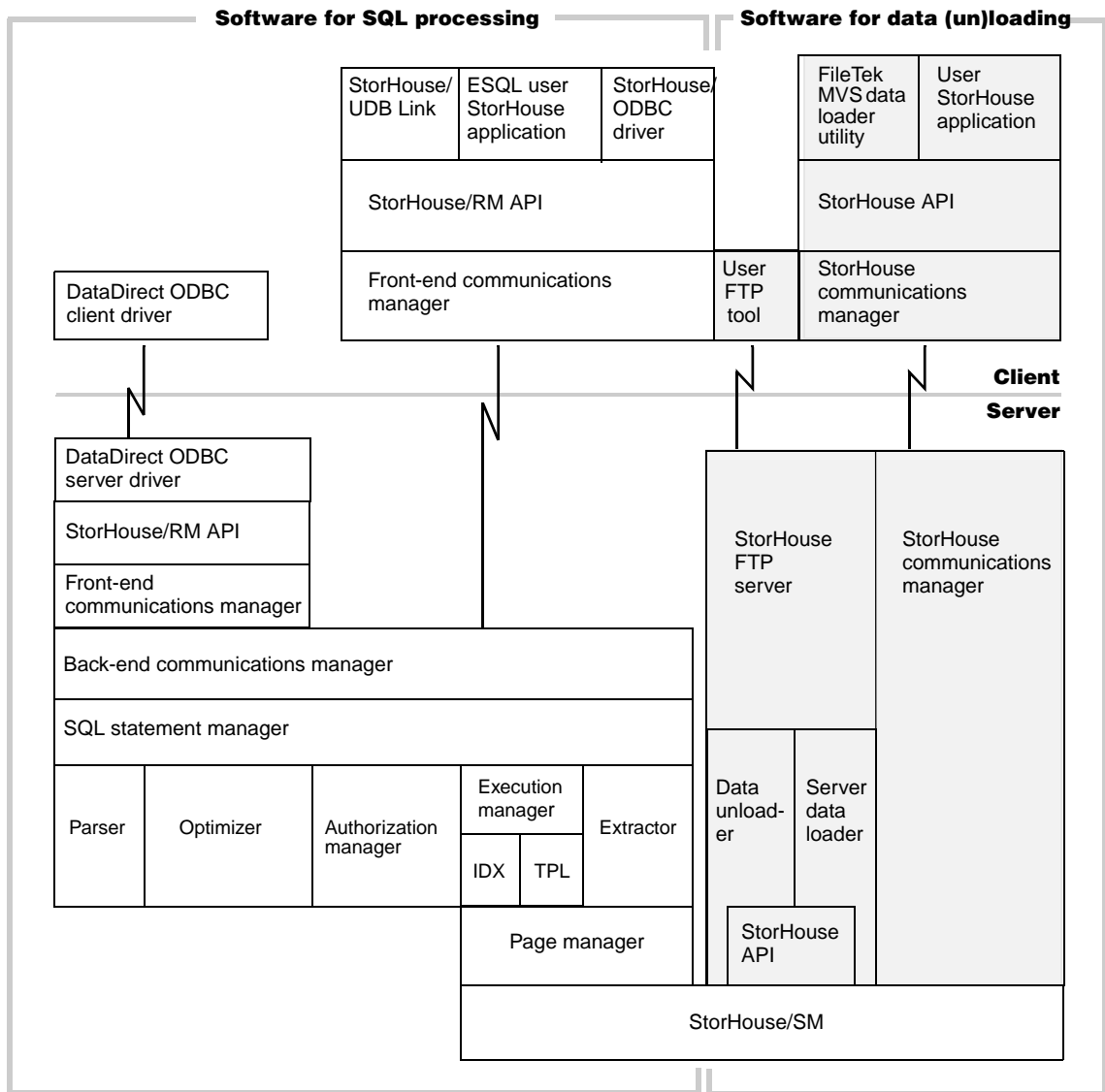
**Load errors.** If a user table contains incorrect data, you can invalidate segments and later delete and remove them from StorHouse. If a load fails at any point or if StorHouse is shut down during a load, the FileTek data loaders provide a restart capability that continues a load from the last checkpoint (data extent) and an abort capability that automatically deletes and removes any partially written segments. A failure during an in-progress load does not require metadata recovery because metadata updates occur only when a load completes successfully and is confirmed.

### Segment file recovery tools

- Create a replacement volume in the same volume set to be recovered by copying extents from primary (if readable), backup, or archive copies
- Create new primary copies from backup or archive copies
- Move selected backup or archive volumes to shelf storage and store them off-site for disaster recovery purposes
- Disable lost, destroyed, or otherwise unusable volumes
- Preview the backup or archive volumes that are needed to recover one or more disabled primary volumes
- Retire tape volumes that have degraded with use by copying extents from one volume in a volume set to one or more new volumes in a volume set
- Validate that all extents on a volume can be read
- Implement duplex support to access backup or archive copies when the primary copy is on a disabled volume or in an offline device

# Software architecture

StorHouse/RM is composed of the following software modules.\*



\* The following modules are customer-supplied: DataDirect ODBC client driver, User StorHouse application, and User FTP tool.



## Client software

DataDirect ODBC client driver	Provides access to StorHouse databases from ODBC-enabled client, Web, and server applications
StorHouse/UDB Link	Enables federation through IBM® DB2® Universal Database (UDB)
ESQL	Enables C and C++ applications to access StorHouse databases
User StorHouse application	Contains API calls that enable user applications to access StorHouse
StorHouse/ODBC driver	Provides access to StorHouse databases from ODBC-enabled client, Web, and server applications
StorHouse/RM API	Passes prepared SQL statements to the front-end communications manager
Front-end communications manager	Arranges (marshals) prepared SQL in communication packets and unmarshals result set data
User FTP tool	Transfers input data being loaded and receives result data being unloaded
FileTek MVS data loader utility	Prepares input data for loading and transfers the load stream to StorHouse
StorHouse API	Lets user applications access StorHouse, perform file functions, and transfer data
StorHouse communications manager	Manages socket-level communications between StorHouse client and server systems

## Server software

DataDirect ODBC server driver	Lets StorHouse/RM communicate with ODBC-enabled client, web, and server applications
Back-end communications manager	Unmarshals SQL statements and marshals result set data
SQL statement manager	Tracks SQL statements through all processing steps
Parser	Checks SQL syntax, validates database components, and builds a preliminary execution tree
Optimizer	Builds a final execution tree
Authorization manager	Checks granted privileges on database components
Execution manager	Runs the constructed execution tree built by the optimizer
Extractor	Processes qualifying queries that result in full segment scans
IDX routine	Fetches index data through the page manager
TPL routine	Fetches tuple (row) data through the page manager
Page manager	Translates retrieval requests into calls to StorHouse/SM data retrieval services
StorHouse FTP server	Formats input or result data into a stream and transfers it to StorHouse or a user FTP tool
Server data loader	Receives load streams, loads data into StorHouse user tables, and builds index entries
Data unloader	Executes unload requests
StorHouse API	Enables the server data loader and data unloader to perform StorHouse file functions
StorHouse communications manager	Manages communications between the client and server, including ESCON channel support
StorHouse/SM	Reads, writes, and manages data on media in the StorHouse storage hierarchy

# SQL

## SQL types

StorHouse SQL can be static or dynamic.

- Static SQL statements are embedded in a program.
- Dynamic SQL statements are prepared and executed by a program at runtime.
- The StorHouse Embedded SQL (ESQL) interface lets you code static SQL in C and C++ programs.
- All StorHouse SQL statements can be embedded in a program.

## SQL categories

Four categories of StorHouse SQL are as follows:

- Data definition language (DDL) statements maintain database components and grant and revoke privileges.
- Data manipulation language (DML) statements query and manipulate data.
- Transaction control statements manage database changes.
- ESQL statements, declarative and executable, can be included in program.

StorHouse provides industry-standard Structured Query Language (SQL).

## Statements

StorHouse supports a subset of ANSI-standard SQL plus extensions defined by FileTek to support additional capabilities. StorHouse SQL statements are:

STATEMENT	TYPE	CATEGORY
ALTER TABLESPACE	static and dynamic	DDL
BEGIN DECLARE SECTION	static	ESQL, declarative
CLOSE	static	ESQL, executable
COMMIT WORK	static	transaction control
CONNECT	static	ESQL, executable
CREATE INDEX	static and dynamic	DDL
CREATE SYNONYM	static and dynamic	DDL
CREATE EXPLAIN TABLES	dynamic	DDL
CREATE TABLE	static and dynamic	DDL
CREATE TABLE SPACE	static and dynamic	DDL
CREATE VIEW	static and dynamic	DDL
DECLARE	static	ESQL, declarative
DELETE	static and dynamic	DML
DESCRIBE	static	ESQL, executable
DISCONNECT	static	ESQL, executable
DROP EXPLAIN TABLES	dynamic	DDL
DROP INDEX	static and dynamic	DDL
DROP SYNONYM	static and dynamic	DDL
DROP TABLE	static and dynamic	DDL
DROP TABLE SPACE	static and dynamic	DDL
DROP VIEW	static and dynamic	DDL
END DECLARE SECTION	static	ESQL, declarative
EXECUTE	static	ESQL, executable
EXECUTE IMMEDIATE	static	ESQL, executable
EXPLAIN PLAN	dynamic	DML
FETCH	static	ESQL, executable
FREE LOCATOR	static	ESQL, executable
GRANT	static and dynamic	DDL
INSERT	static and dynamic	DML
OPEN	static	ESQL, executable
PREPARE	static	ESQL, executable
PURGE TABLE	static and dynamic	DDL
RENAME	static and dynamic	DDL
REVOKE	static and dynamic	DDL
ROLLBACK WORK	static	transaction control
SELECT	static and dynamic	DML
SET CONNECTION	static	ESQL, executable
UPDATE	static and dynamic	DML
VALUES INTO	static and dynamic	DML
WHENEVER	static	ESQL, declarative

## Predicates

A *predicate* reduces the number of rows returned by a query. With predicates, you can compare values by using operators or keywords. StorHouse supports these predicates:

Basic	EXISTS	NULL
BETWEEN	IN	Quantified
Complex	LIKE	

## Functions

A *function* is a named operation in an SQL statement, followed by one or more expressions. StorHouse supports these aggregate and scalar functions:

ABS	GREATEST	hour
ADD_MONTHS	hour	POSITION
ASCII	INITCAP	QUARTER
AVG	INSTR	RIGHT
BIT_LENGTH	LAST_DAY	RPAD
BLOB	LEAST	RTRIM
CHAR_LENGTH	LENGTH	SECOND
CHR	LOWER	SUBSTR
CLOB	LPAD	SUBSTR_UBD
CONCAT	LTRIM	SUM
COUNT	MAX	TO_CHAR
COUNT_BIG	MIN	TO_DATE
DAYOFMONTH	MINUTE	TO_HEX
DAYOFWEEK	MONTH	TO_NUMBER
DAYOFYEAR	MONTHS_BETWEEN	TO_TIME
DAYS	NEXT_DAY	TRANSLATE
DAYOFMONTH	NVL	TRIM
DECODE	OCTET_LENGTH	UPPER
		WEEK

## Operators

Basic predicates compare values with a relational operator, while complex predicates combine basic predicates using logical operators.

- Relational operators are:  
=, >, <, >=, <=, <>
- Logical operators are:  
NOT, AND, and OR

Quantified predicates use the keywords ANY or SOME.

## Aggregate and scalar functions

Aggregate functions summarize information about groups of rows in a table. The result contains a single row per group that summarizes all selected rows. StorHouse aggregate functions are:

- AVG
- COUNT
- COUNT\_BIG
- MAX
- MIN
- SUM

Scalar functions produce a single value from another value. In the functions table, all functions except AVG, COUNT, COUNT\_BIG, MAX, MIN, and SUM are scalar functions.

# ESQL

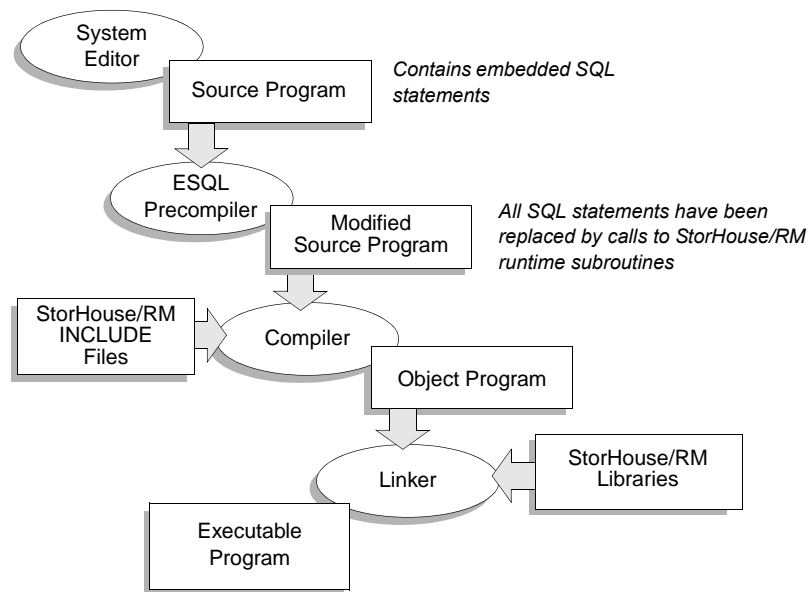
## Static versus dynamic SQL

Static SQL statements are hardcoded in a program. These statements are compiled when the rest of the program is compiled. You use static SQL when you know—at compile time—which SQL statements you're going to issue and the names of the tables and columns you plan to select. Only the values of host variables in your search condition may change from one execution to the next. An example of a static query is an airline reservation system that checks for available seats. You would use host variables to tailor the flight number and date, but the table and columns in the SELECT statement remain static. Unlike static SQL statements, which are embedded in a program, dynamic SQL statements are built at runtime and placed in a string host variable.

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

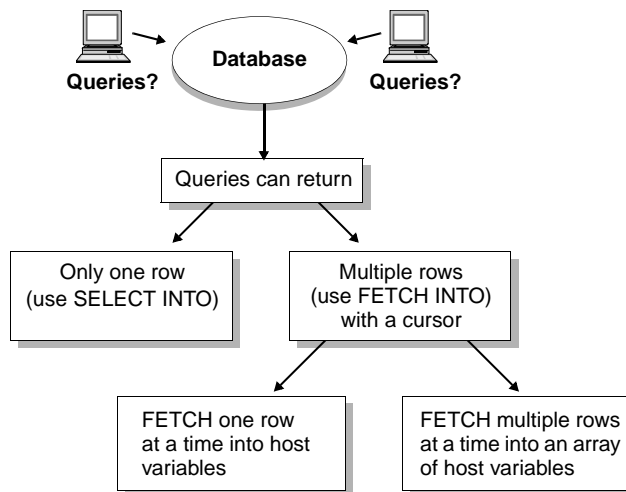
## Compiling an ESQL program

Because an ESQL program contains a mix of SQL and host language statements, you cannot submit it directly to a host language compiler. You must first submit it to the *StorHouse ESQL precompiler*, which scans your source program and translates the embedded SQL into host language statements that include StorHouse/RM runtime subroutines. The output of this translation is a pure C or C++ program, which you can compile, link, and execute. The ESQL precompiler also accepts C or C++ object files and passes them to the C or C++ linker. The following diagram illustrates the path from source code to executable for an ESQL program.



## Submitting queries with ESQL

You submit queries in an ESQL program with the SELECT statement. StorHouse ESQL supports the following SELECT statement clauses: FROM, GROUP BY, HAVING, INTO, ORDER BY, and WHERE. For queries that return only one row, you use the SELECT statement INTO clause. For queries that return more than one row, you use a *cursor* to retrieve—or fetch—one row at a time or an array of rows into output host variables.



## Checking the status of SQL

ESQL programs require a data structure called the SQL Communications Area (SQLCA) to hold information about the status of your most recently executed SQL statement. StorHouse updates the SQLCA after every executable SQL statement. You can use the SQLCA to check return code information, number of rows fetched, and warning flags. C programs implement the SQLCA as a global structure that the ESQL precompiler automatically declares and defines.

### Host variables

You can use host variables to tailor an SQL statement.

- Input host variables pass data to StorHouse/RM. They are typically used in WHERE clauses.
- Output host variables pass data and status information to your program. They are typically used in the INTO clause of a SELECT or FETCH statement or in the VALUES INTO statement.

For LOBs, you can define and use the following host variables:

- A locator variable is used to identify and manipulate a LOB value at the server or to access parts of a LOB value.
- A file reference variable is used to transfer a LOB value (or a part of it) to or from a client file.

You can associate a host variable with an optional indicator variable to detect NULL or truncated values.

## Excerpt from an ESQL program

This sample ESQL program illustrates some of the static SQL statements and techniques used to code an ESQL program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int  static_select( void );
static int  usage
(
    char *prog
);

static int  usage
(
    char *prog
)
{
    fprintf( stderr, "Usage: %s <dbname>\n\n", prog );
    return ( 0 );
}

main
(
    int argc,
    char *argv[]
)
{
    int rc = 0;
```

This is a Declare Section—a required ESQL program component that contains your host variables, indicator variables, and new type declarations. The ESQL precompiler generates the corresponding host language declarations for these variables and types so that you can use them at your convenience in SQL and C. ESQL does not recognize variables or types defined in C language statements coded outside a Declare Section.

```
EXEC SQL BEGIN DECLARE SECTION;
    char dbname[64];
EXEC SQL END DECLARE SECTION;

if ( argc != 2 )
    return ( usage( argv[0] ) );

strcpy( dbname, argv[1] );

EXEC SQL
    WHENEVER SQLERROR GOTO err;

EXEC SQL
    CONNECT TO :dbname AS 'conn1';

rc = static_select();

EXEC SQL
    DISCONNECT 'conn1';

EXEC SQL
    WHENEVER SQLERROR CONTINUE;

return ( rc );
```

← Your program must connect to a StorHouse database before it can submit queries. A single program can connect to up to 10 databases at a time. The CONNECT and DISCONNECT statements manage connectivity.

```

err:
printf( "SQL Error (%ld) %s\n", sqlca.sqlcode, sqlca.sqlerrm );

return ( -1 );
}

static int  static_select()
{
EXEC SQL BEGIN DECLARE SECTION;
    char tretval[33];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    WHENEVER SQLERROR GOTO err;

EXEC SQL
    DECLARE stcur CURSOR FOR
        SELECT tbl
        FROM sysadm.systables
        WHERE tbl NOT LIKE 'SYS%';

EXEC SQL
    WHENEVER NOT FOUND GOTO over;

EXEC SQL
    OPEN stcur;

for ( ; ; )
{
    tretval[0] = '\0';
    EXEC SQL
        FETCH stcur INTO :tretval;
    printf( "%s\n", tretval );
}

over:
EXEC SQL
    CLOSE stcur;

EXEC SQL
    COMMIT WORK;

printf( "Static select statement executed successfully\n" );

return ( 0 );

err:
fprintf( stderr, "SQL Error: %d %s\n", sqlca.sqlcode,
sqlca.sqlerrm );

EXEC SQL
    WHENEVER SQLERROR CONTINUE;
EXEC SQL
    ROLLBACK WORK;

return ( -1 );
}

```

WHENEVER automates condition checking and error handling. This statement checks the SQLCA for errors, warnings, or successful execution and tests all executable SQL statements that physically follow it in the source file. It stays in effect until it is superseded by another WHENEVER statement that checks for the same condition or until the end of the source file.

StorHouse uses a cursor to process the rows that satisfy your queries. DECLARE names a cursor and associates the cursor with the query that follows. You must explicitly define a cursor for queries that return more than one row.

OPEN executes the associated SELECT statement with the current program variables and identifies the result set.

FETCH reads the rows of the result set and returns the values into host variables.

CLOSE terminates cursor processing. Once a cursor is closed, you cannot perform FETCH operations on the cursor.

COMMIT WORK ends a transaction. It releases locks and makes any changes to the database during that transaction permanent.

ROLLBACK WORK cancels the current transaction and rolls back any database changes performed during the transaction.

# Data loaders

## Loading features

**Concurrency.** You can load the same or different user tables in parallel as well as query a user table while it's being loaded.

**Conversion.** Data fields with different but compatible data types are automatically converted to the format of the columns in the tables.

**Segment management.** You can load one or multiple segments at a time, invalidate or merge existing segments, and name segments in the event they need to be replaced.

**Subspace selection.** You can use default subspaces, select specific subspaces, or rotate among subspaces for each component type.

**Data generation.** You can load columns with generated values, such as the current date, a constant value, a sequence of values, or a record number.

**Restart capability.** You can restart loads from the point of failure or abort a load and start at the beginning.

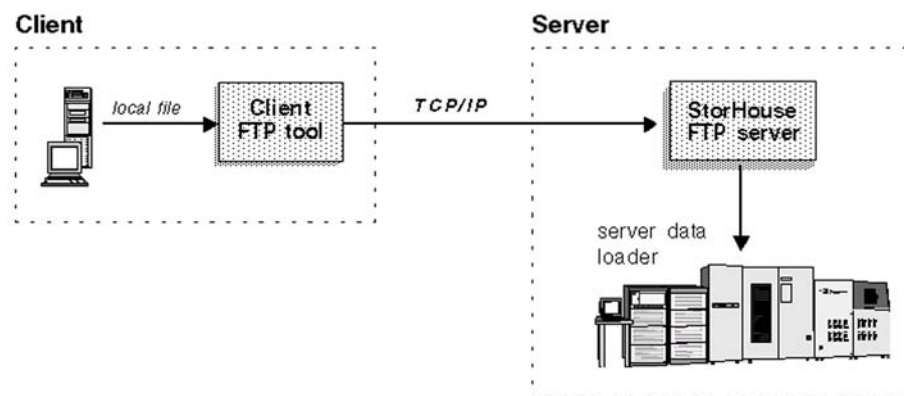
**SQL tool.** In addition to loading data, you can use a FileTek data loader to submit StorHouse SQL statements. The data loader commits each statement when it completes.

**Deferred index load.** You can use a data loader to create index entries for existing segments.

Comprehensive data loader programs developed by FileTek transfer large amounts of relational data from host environments to StorHouse. You load data from a UNIX, Windows, Linux, VAX, or other File Transfer Protocol (FTP) enabled host with the FileTek FTP Data Loader. You load data from an IBM MVS environment with the FileTek MVS Data Loader utility. If your MVS machine is FTP-enabled, you can also run the FileTek FTP Data Loader from MVS.

## Loading with FTP

With the *FileTek FTP Data Loader*, you use your standard client FTP software (or tool) to communicate with the StorHouse FTP server. These two programs communicate over a TCP/IP connection to transfer files from your local file system on your host to a remote file system on StorHouse. The server data loader then loads your data into StorHouse user tables and builds and stores any indexes



Some of the source code for the StorHouse FTP server was derived from the source code used in the BSD (University of California at Berkeley) FTP tool. You use standard FTP commands including a standard put command with customized parameters to transfer data.

If your client supports it, FTPS can be used for secure, encrypted communication with the StorHouse server.

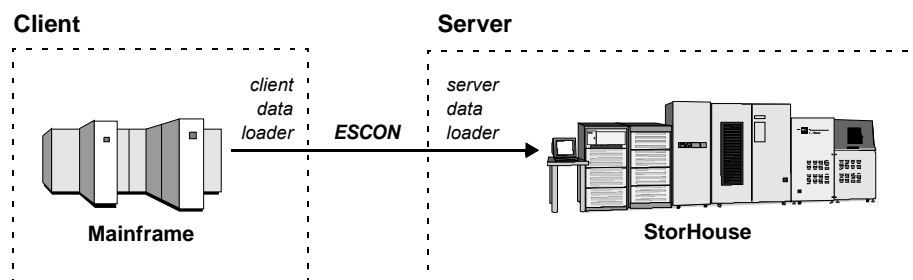


## Loading from MVS

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

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

The FileTek MVS Data Loader utility uses the channel connection to achieve maximum data transfer rates.



## LOAD DATA statement

The SQL-like LOAD DATA statement describes load characteristics and input data. This statement is similar and compatible with the control information supplied for Oracle® and DB2 load utilities. A FileTek data loader accepts clauses that are not part of the StorHouse syntax but ignores those that do not apply to StorHouse. The LOAD DATA statement clauses are as follows:

- CHARACTERSET
- CONCATENTATE
- CONSTANT
- CONTINUEIF
- DEFAULTIF
- DIFFERENT SEGMENT
- DISCARDFILE
- DISCARDS
- ESCAPED BY
- FIELDS
- INFILE
- INTO TABLE
- LOAD
- NULLIF
- POSITION
- PRESERVE BLANKS
- RECNUM
- REPLACE SEGMENT
- SAME SEGMENT
- SEGMENT
- SEQUENCE
- SUBSPACE number
- SUBSPACE ROTATE
- SYSDATE
- TRAILING NULLCOLLS
- WHEN

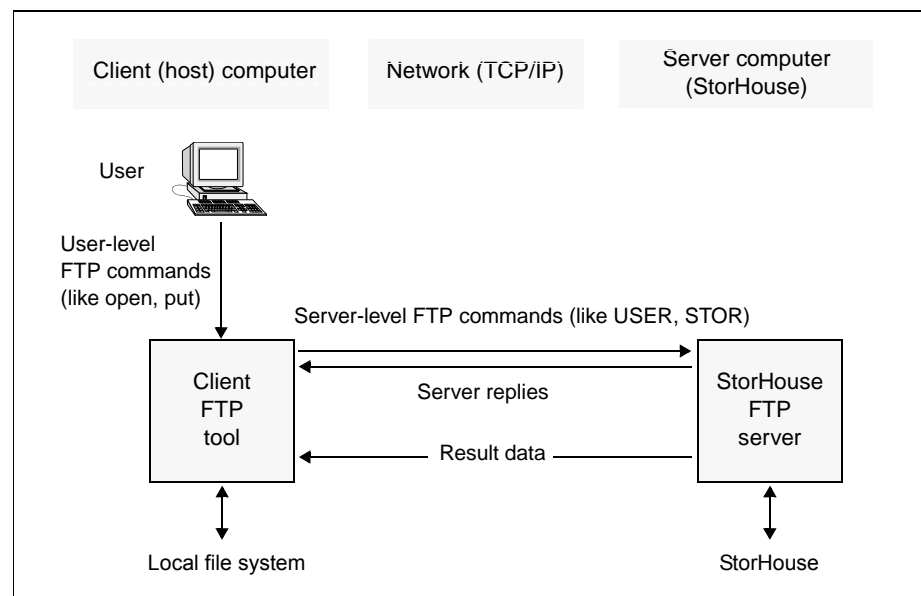
# Data unloader

## Unload features

- Run multiple unload and load operations during an FTP session
- Pipe the output to another program, such as a data loading utility
- Unload an entire table, specific columns or rows of a table, or multiple tables (join)
- Format result data records in any of three record formats: text, fixed-length, or variable-length
- Create result data fields with the following data types:
  - BIGINT
  - BINARY
  - BINARY EXTERNAL
  - BLOB
  - CHARACTER
  - CLOB
  - DATE EXTERNAL
  - DECIMAL
  - DECIMAL EXTERNAL
  - DOUBLE
  - FLOAT
  - FLOAT EXTERNAL
  - INTEGER
  - INTEGER EXTERNAL
  - SMALLINT
  - TIME EXTERNAL
  - TIMESTAMP EXTERNAL
  - VARBINARY
  - VARCHAR
- Place each BLOB or CLOB value in a file on the local host or a remote host

The *FileTek FTP Data Unloader* is a tool for copying data from StorHouse user tables to your host. This utility executes a SELECT statement, then it formats and transfers the result data to a sequential (or flat) file on your host or to a VRAM file on StorHouse. You can receive LOB data with the other result data (space permitting) or in separate files on a client or remote computer.

With the FileTek FTP Data Unloader, you use your standard client FTP tool to communicate with the StorHouse FTP server. These two programs communicate over a TCP/IP connection to transfer files between the local and remote file systems.



If your client supports it, FTPS can be used for secure, encrypted communication with the StorHouse server.

Your *client FTP tool* interacts with you and your local file system. It sends server-level FTP commands and your control file to the StorHouse FTP server. The *StorHouse FTP server* interacts with the remote file system—StorHouse database tables. It replies to your client FTP commands, invokes a StorHouse process to execute the unload query, and transfers the result data.

# The unload process

To unload data from StorHouse user tables:

## 1 Prepare the input

At your computer, prepare a control file containing an UNLOAD statement.

## 2 Transfer the control file

With your client FTP tool:

- Start FTP and log into the StorHouse FTP server with the ftp or open command.
- Set the transfer type to ASCII or BINARY, if needed, with the type command.
- Transfer the control file with the put command.

The StorHouse FTP server parses the FTP commands, then a StorHouse engine or extractor reads the UNLOAD statement and prepares your query.

## 3 Receive the result data

With your client FTP tool, retrieve the result data with the get command. This step is not needed when unloading data to a StorHouse VRAM file.

## UNLOAD statement

The SQL-like UNLOAD statement is the input to an unload operation. This statement describes how to format the result data and contains the query that selects the StorHouse data to unload. Optional UNLOAD clauses enable you to:

- Specify the character set of result data (CHARACTERSET clause)
- Define delimiters for character result data (FIELDS clause)
- Format all data fields as CHARACTER data type (FIELDS clause)
- Specify a character to append to the end of result records (RECORDS clause)
- Specify an escape character to insert into the result data (ESCAPED BY clause)
- Describe each data field in result records (USING clause), including data type, position, and null handling
- Insert constant text into result records (CONSTANT clause)
- Specify a VRAM file name to unload data to a file on StorHouse (OUTFILE clause)

# ODBC interface

## Setting up the Oracle–StorHouse environment

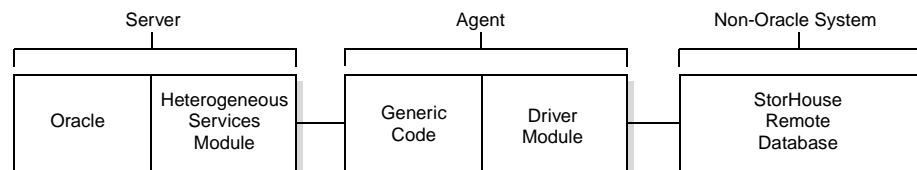
In order to incorporate Oracle Heterogeneous Services in an Oracle–StorHouse environment, you first install one of the FileTek ODBC driver products (Database Interconnect or the StorHouse/ODBC driver) and set up a data source that points to the StorHouse database you want to access. Then you set up Oracle Heterogeneous Services for StorHouse as follows:

- Install the Oracle Heterogeneous Services data dictionary
- Set up your Oracle environment to access Heterogeneous Services agents
- Configure Generic Connectivity agents
- Create and test a database link to your StorHouse database

StorHouse supports the Microsoft Open Database Connectivity (ODBC) interface, an open software architecture that enable applications to use SQL to access relational data stored on a variety of RDBMSs. For instance, Microsoft SQL Server™ 7.0 and Oracle8i™ Heterogeneous Services can access StorHouse through ODBC.

## Oracle access to StorHouse

Oracle users can access StorHouse data transparently through the Oracle8i Heterogeneous Services and ODBC. To users, StorHouse appears as part of the local Oracle database. The Oracle Heterogeneous Services architecture consists of three components: server, agent, and non-Oracle system.

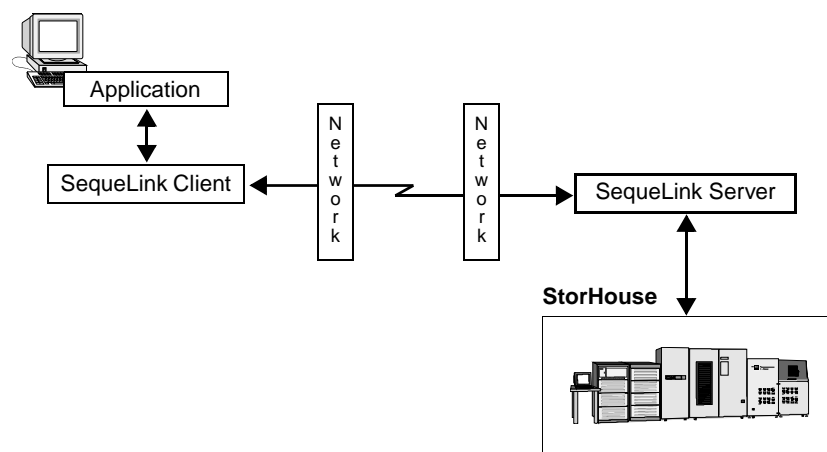


The agent code, generic to all Heterogeneous Services-based products, provides database communication and multithreading support. Oracle has Generic Connectivity agents for ODBC and OLE DB. These agents enable Oracle applications to use ODBC and OLE DB drivers in conjunction with FileTek-supplied ODBC drivers to access StorHouse databases.

## Supported ODBC drivers

FileTek provides an ODBC client driver that communicates with StorHouse/RM through the StorHouse/RM API. The StorHouse/RFS product, for instance, uses this driver. FileTek also supports the DataDirect SequeLink system, which provides ODBC-enabled client, Web, and server applications access to StorHouse. These applications may be running on a variety of platforms, including Windows, Solaris, Digital UNIX, SGI IRIX, OS/390, and any JVM-enabled platform.

The DataDirect SequeLink system consists of client and server components. The *SequeLink server* provides data access services between ODBC client applications and databases like StorHouse. The *SequeLink client* sends ODBC calls across your network to the SequeLink server. The server then passes the request to a StorHouse engine, which processes the request and passes the result to the SequeLink server, which passes it to the SequeLink client and application.



## Microsoft SQL Server 7.0 access to StorHouse

Through an ODBC connection with Microsoft SQL Server 7.0, Windows users can connect to and access StorHouse data through a single, familiar interface. For instance, with the SQL Server 7.0 import feature, users can replicate StorHouse user tables. The SQL Server 7.0 import utility:

- Retrieves the table definitions from StorHouse
- Creates the tables and indexes on the SQL Server
- Unloads the data from StorHouse
- Loads the data into the SQL Server tables

# StorHouse/UDB Link

## Federated system

A federated system is an RDBMS that supports applications and users who submit SQL statements that reference two or more RDBMSs or databases in a single statement (for example, a join between tables in different databases).

## Federator

A federator is the software component in a federated system that coordinates SQL processing. Websphere Information Integrator is the federator for DB2 UDB.

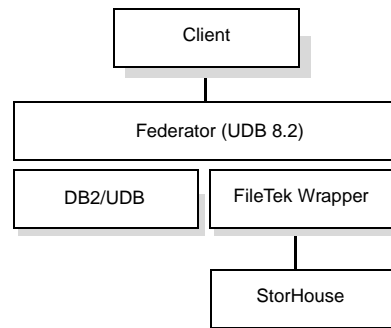
## Wrapper

A wrapper is software that implements a connection in a federated system. The StorHouse/UDB Link is the FileTek-supplied wrapper software to implement the connection between DB2 UDB and StorHouse. You install the software and then define the wrapper with the DB2 CREATE WRAPPER statement.

## Data source

A data source is a server or a database in a federated system. Each StorHouse database is a separate data source. You define data sources with the DB2 CREATE SERVER statement.

The *StorHouse/UDB Link* gives DB2 Universal Database (UDB) users near-transparent access to the terabytes to petabytes of data supported by StorHouse/RM. This software, also called the FileTek wrapper, implements the connection between DB2 UDB release 8.2 and StorHouse databases. The components of a federated DB2 system with the StorHouse/UDB Link are:



## How it works

In a federated system, a client communicates with a federator. The federator appears to the client as a database (even though it may not be one). The client need not know that the data sources exist or participate in a query. The federator:

- Accepts and parses a query from the client
- Breaks a query into smaller queries
- Submits these queries to one or more data sources
- Receives the results from the data sources
- Assembles the results into a single answer set
- Returns the answer set to the client

The federator maintains a local data dictionary. This dictionary contains descriptions of the data sources, also called *servers*, and their available data, defined by *nicknames*. The federator uses this information to decide what portions of the original query will be processed by each data source. The dictionary also contains *user mappings*, which associate DB2 authorization IDs with StorHouse account IDs for specific data sources.

## Partitioning data

In a federated system, you must determine where you partition, or distribute, data between DB2 and StorHouse. Criteria for partitioning data include frequency of access, volume of data, and volatility of data. Partitioning models that work well with StorHouse are as follows.

### Horizontal partitioning

Horizontal partitioning divides rows into sets. Within each set, the rows remain intact. Partitioning criteria are usually range-based, like date. For instance, rows with the current month could be stored in DB2 and rows with past months could be stored in StorHouse. Or rows with pending orders could be stored in DB2 and rows with filled orders could be stored in StorHouse.

### Vertical partitioning

Vertical partitioning splits rows into two or more sets of columns and bases the data placement decision on the use of columns rather than rows. Columns that are frequently accessed, smaller, or more volatile could be stored in DB2. Those columns that are infrequently accessed, larger, or more stable could be stored in StorHouse.

### Summary-Detail partitioning

A third data model places summary data in DB2 and detailed data in StorHouse. This can be effective if you can divide users into groups based on data access patterns. Frequently, summary data is sufficient for most users, while only a small number of users need access to the full detail.

### Duplication

In some cases, it may be beneficial to duplicate data between DB2 and StorHouse. This can be full duplication of all data or duplication of a subset. Full duplication turns StorHouse into a disaster recovery facility.

### Nickname

A nickname is the representation in a DB2 catalog of a remote table or a view controlled by a particular data source. You define nicknames with the DB2 CREATE NICKNAME statement.

### User mapping

User mapping is the correspondence between a user's DB2 identity, or authorization ID, and the identity used when communicating with a data source, for instance, a StorHouse account ID. You define user mappings with the DB2 CREATE USER MAPPING statement.

### Passthru

Passthru is the capability of a client to issue queries and other SQL statements directly to a data source. You authorize passthru for specific data sources with the DB2 GRANT PASSTHRU statement.

### Pushdown

Pushdown is the act of moving SQL processing into a particular data source.

# Queries

## SELECT features

The StorHouse SELECT statement lets you:

- Retrieve data from one or multiple tables or views in one query
- Specify restrictions or search conditions to return only those rows that satisfy the criteria
- Group the rows of a result set
- Apply one or more qualifying conditions to groups of rows
- Sort retrieved data in ascending or descending order on one or multiple columns
- Eliminate duplicate rows from a result set
- Perform arithmetic computations on column data
- Combine SELECT statements using UNION and UNION ALL set operators

## Explain facility

You use the explain facility by submitting a set of SQL statements—CREATE, EXPLAIN TABLES and EXPLAIN PLAN—and querying the result tables. The explain facility can help you determine whether StorHouse/RM uses an index and which index for a query. Or you can determine the chosen join method and join predicate.

You access StorHouse relational data by submitting a *query* with a StorHouse SELECT statement. StorHouse supports these types of queries: selection, join, extraction, and subquery. You can analyze the execution plan for a specific query by using the StorHouse *explain facility*.

## Selection

A *selection* returns specified columns from one or more rows in one table. For instance, when you access all the information in a user table for a specific account number, then you are performing a selection.

## Join

A *join* creates a result set from data in multiple tables or views. StorHouse/RM supports these types of joins:

- An *inner-join* combines the matched rows of the tables. The unmatched rows are omitted from the result set.
- A *left outer-join* combines the matched rows of tables, and for unmatched rows, combines the values of the left table with null values for the right table.
- An *equi-join* joins a column from one table with a column from another table by using predicates that specify equalities.
- A *self- or auto-join* joins a table with itself. For example, you can select all customer names and numbers that are from the same city as another customer in the table.
- A *cartesian product* joins all rows of two or more tables.

StorHouse/RM supports nested loop and hybrid IN join operations. A *hybrid IN* is a type of merge join (single pass through the tables). The optimizer chooses the most efficient join operation for the query but may consider one type over the other when certain conditions are met. For instance, the optimizer may use a hybrid IN join operation when the query is an equi-join and the inner table has a value index on the join column.

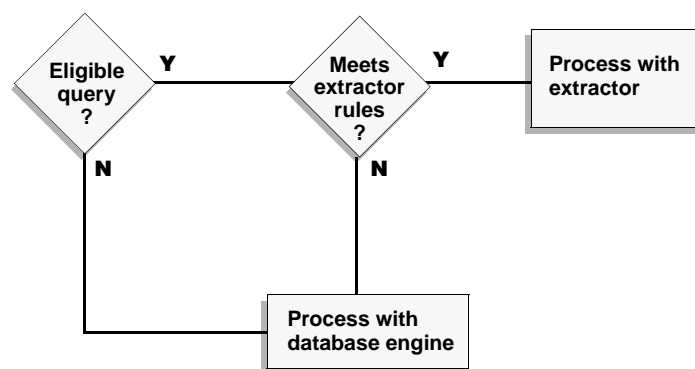


## Extraction

An *extraction* returns one or more columns for all rows of a user table or a range of segments. Two types of queries are eligible for extraction:

- A *simple query* results in a *full table scan*, which reads every row in a table without the use of an index.
- A *full segment select query* returns data from one or more entire segments with the use of a range index.

The *StorHouse extractor* software processes full table scans and full segment selects quicker and more efficiently than a StorHouse engine. Furthermore, when a table resides on both StorHouse optical and tape media, the extractor always uses the tape copy when available to benefit from faster sequential I/O. Simple and full segment select queries must meet query requirements and additional extractor requirements to qualify for extractor processing.



## Subquery

A *subquery* is a SELECT statement nested in another SQL statement. The statement containing the subquery is the *parent* statement. StorHouse supports simple and correlated subqueries. A *simple subquery* executes once for the entire parent statement, while a *correlated subquery* executes once for each row produced by the parent statement. The main uses of subqueries in StorHouse are to define the set of rows to be included in views and to answer multiple-part questions in queries.

### Simple query requirements

- The account issuing the query must have the SCAN database privilege.
- The query must not contain WHERE, ORDER BY, GROUP BY, or DISTINCT clauses.

### Full segment select query requirements

- All predicates in the WHERE condition must be based on columns in range indexes.
- The predicates must select all rows from one or more segments.

### Extractor requirements

- The query must refer to one table and must not contain any subqueries.
- The host and StorHouse systems must have the same native values key so that no byte-reordering of INTEGER and SMALLINT columns is necessary.
- The application issuing the query must not have changed any values in SQLDA fields set by DESCRIBE.

# Concurrency

## Controlling concurrency

Three tunable system parameters help manage the number of concurrent operations for optimal performance.

### SQL\_LDR\_MAXLOAD.

Specifies the maximum number of LOAD statements that can be processed at a time. One engine is required per LOAD statement. Requests beyond the limit are queued. When the queue length exceeds twice the maximum, then new requests are rejected.

### SQL\_LDR\_MAXINTO.

Specifies the maximum number of INTO TABLE clauses in any one LOAD statement. A load fails when the maximum number of INTO TABLE clauses is exceeded.

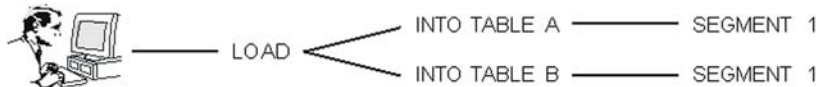
**SQL\_SESSIONS.** Specifies the maximum number of StorHouse engines that can run concurrently for all users. This sets the maximum number of connections allowed system-wide. The maximum number of connections includes the number of loads (one engine per LOAD statement) plus the number of queries. Requests beyond the limit are rejected.

StorHouse supports the *serializable* ANSI/ISO transaction isolation level, which guarantees the highest read consistency and data integrity in a database. StorHouse concurrency software facilitates maximum simultaneous access to data.

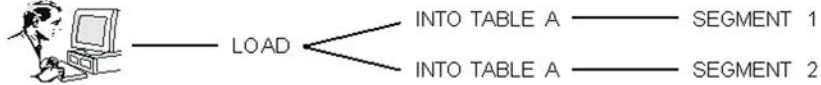
## Parallelism

Users can load the same or different user tables concurrently. Users can query a user table while it's being loaded, and they can access the new segments after the load completes. One StorHouse engine is required for each load and to handle each query. System parameters control concurrency. Some of the parallel operations are shown below.

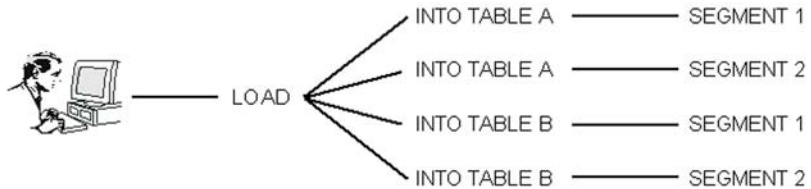
### Load different user tables in one load



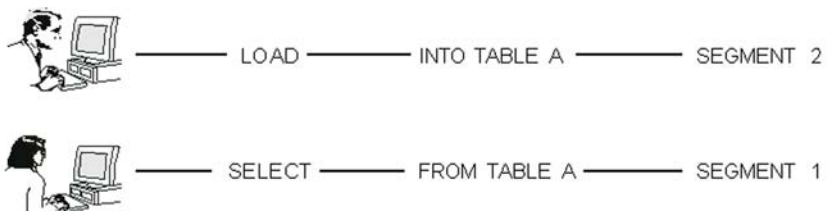
### Load multiple segments of the same user table in one load



### Load multiple segments of multiple user tables in one load



### Query a user table while it's being loaded



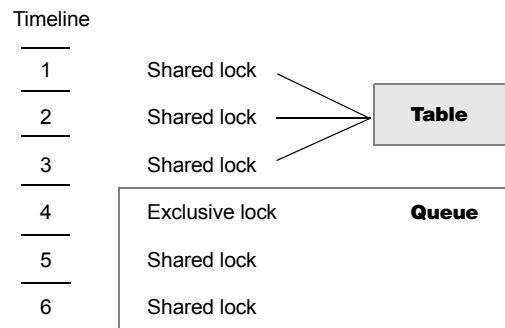
# Locking

StorHouse locking is fully automatic and requires no user action. StorHouse implements table-level locks. Tables include user tables, system tables, and views. Two types of table locks are:

- A *shared* (or read) *lock* reserves a table for reading only. This lock prevents a table from being dropped. Multiple engines can have a shared lock on the same table.
- An *exclusive* (or write) *lock* reserves a table for updating only. This lock prohibits a table from being shared. One engine can have an exclusive lock on a table. All other lock requests (shared and exclusive) for the table are queued.

An engine holds on to locks—both shared and exclusive—against user tables throughout a transaction. It releases shared locks against system tables as soon as it's done processing the system tables, and it releases exclusive locks against system tables when the transaction ends. Note that for DDL statements, the operation is atomic, so the transaction boundaries match the statement boundaries.

When an engine places a shared lock on a table, subsequent shared lock requests can access the same table, but an exclusive lock request starts a queue. When queueing begins, subsequent requests—both shared locks and exclusive locks—queue up behind the exclusive lock.



When an engine releases an exclusive lock, the next entry in the queue gets the lock. In the above example, a shared lock (entry 5) would be granted, followed by the next shared lock (entry 6) in the queue.

## When locks occur

**Metadata backup.** A metadata backup uses a database-level lock that has the effect of read-locking every system table. This allows queries to continue and prevents DDL statements from changing the metadata. DDL statements are queued until after the lock is released.

**Metadata recovery.** Any locks in place at the time of a transaction failure are held and then released after roll back. A recovery process places any necessary locks on applicable metadata during system initialization and releases those locks after recovery. Journaling uses locks during physical I/O to the current journal file and during journal cycling.

**Loads.** A load acquires a shared lock on the user table during commit processing. No locks are held during the middle of a load. A load acquires and releases an exclusive lock on SYSTABLES at load start and on SYSSTHSEGMENTS at load end.

**DDL processing.** All DDL statements acquire an exclusive lock on applicable system tables. Additionally, CREATE, GRANT, and REVOKE statements acquire a shared lock on applicable user tables; and DROP statements acquire an exclusive lock on applicable user tables.

**Queries.** A query requires a shared lock on the user table.

# Database security

## Administrative accounts

Each StorHouse/RM system comes with two administrative accounts.

**SYSADM.** This administrator account has all privileges in all StorHouse databases. SYSADM can perform all StorHouse database and system administration tasks. SYSADM owns the system tables for each StorHouse database.

**PUBLIC.** This special-purpose account simplifies the process of granting and revoking database component privileges. Any StorHouse account with SQLEXECUTE has PUBLIC access to all StorHouse databases. PUBLIC privileges, however, vary from database to database.

StorHouse security controls access to StorHouse databases, the administrative tasks a user can perform, and the tables a user can query or load. This multilevel security consists of account and privilege facilities.

## StorHouse accounts

Only users with valid StorHouse accounts can access a StorHouse database. Users need a StorHouse account with certain StorHouse privileges to:

- Perform StorHouse system and database administration tasks
- Submit StorHouse SQL statements
- Load data into StorHouse user tables
- Unload data from StorHouse user tables
- Access StorHouse data from a host language application (using ESQL) or a local database application (such as a DB2 application)

Account passwords provide additional validation. An account must always specify a password when loading data or connecting to a StorHouse database.

## Security validation

StorHouse/SM validates all connects to StorHouse databases. Only valid StorHouse accounts with passwords can connect to StorHouse databases. StorHouse/SM also validates account access and command privileges. StorHouse/RM validates account database and database component privileges.

StorHouse tracks all connects and disconnects as well as requests denied due to invalid account IDs or passwords. It also logs SQL statements submitted and transaction statistics for each completed transaction. You can then use Control Center to analyze possible security violations and account activity.

# StorHouse privileges

Privileges control access to user table data and determine the administration tasks a StorHouse account can perform in all databases, in specific databases, or for specific database components.

TYPE	PRIVILEGE	AUTHORIZATION
Access	SQLADMIN	Have DBA privilege in all databases
Command	SQLEXECUTE	Submit SQL statements in all databases
	SQLCOMMAND	Run a FileTek data loader in all databases
Database	DBA	Perform the following in a specific database: <ul style="list-style-type: none"><li>■ Create user tables, indexes, views, and synonyms for other accounts</li><li>■ Insert, update, and delete data in system tables</li><li>■ Grant and revoke database and database component privileges</li><li>■ Access any table, view, or synonym</li></ul>
	RESOURCE	Perform the following in a specific database: <ul style="list-style-type: none"><li>■ Create user tables, indexes, views, and synonyms for own account</li><li>■ Access those database components</li><li>■ Grant other accounts SELECT and INDEX privileges on those components</li></ul>
	SCAN	Read all rows in any user table on which the account has SELECT privilege
Database component	ALL	Have all database component privileges for a specified component
	DELETE	Delete rows from a system table or system view
	INDEX	Create an index for a user table
	INSERT	Load data into a user table or insert rows into a system table or system view
	SELECT	Access a user table or system table or view
	UPDATE	Update columns in a system table or system view

## Privilege types

There are four types of StorHouse privileges.

### Access privilege.

Provides the broadest level of security. It enables an account to perform database administration tasks in all StorHouse databases.

### Command privileges.

Let accounts submit SQL statements and certain StorHouse commands in all StorHouse databases. At a minimum, an account must have SQLEXECUTE to access a StorHouse database.

### Database privileges.

Control the functions an account can perform in a specific database.

**Database component privileges.** Determine account access to specific components—like tables and views or columns within tables and views—in a specific database. An account authorized to load data must have INSERT privilege on the user tables it can load. Otherwise, DELETE, INSERT, and UPDATE apply to system tables only because StorHouse does not allow updates of user tables.

# Administration

## StorHouse/ Performance Monitor

With StorHouse/  
Performance Monitor you  
can:

- Display near real-time activity (such as mounts, megabytes read and written, shelf requests, file opens, CPU average utilization, and so on)
- Display historical activity to determine trends
- Graphically depict system performance measures in pre-defined reports
- Create custom reports

## CCAdmin

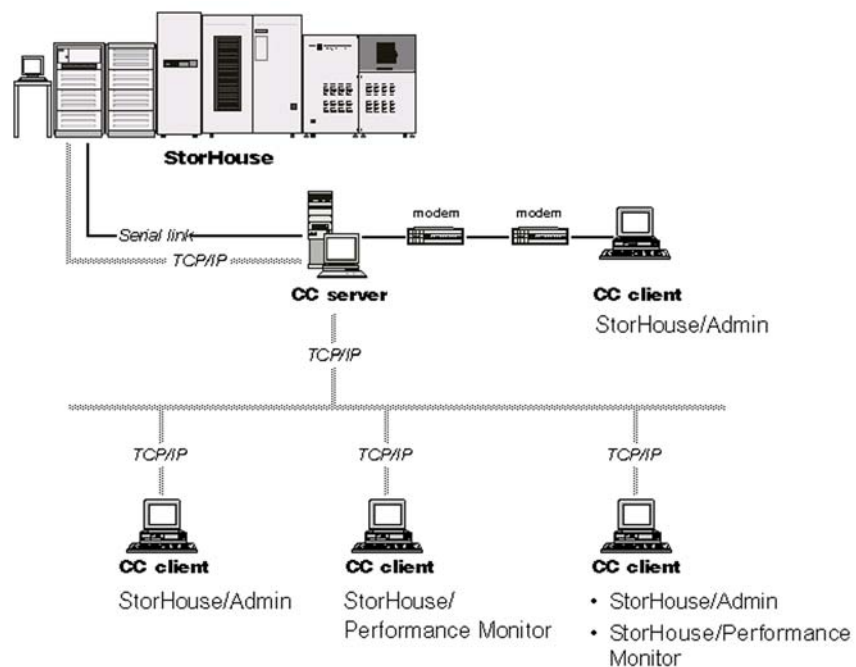
With CCAdmin you can:

- Make StorHouse systems available or unavailable to a StorHouse/Control Center server
- Establish or terminate connections between StorHouse/Control Center servers and specific StorHouse systems
- Configure label printers to print labels for blank volumes
- Collect diagnostic information, performance statistics, and activity information about StorHouse/Control Center servers

StorHouse/*Control Center* (CC) is the FileTek network computing system used for StorHouse administration. StorHouse/Control Center consists of server and client components. A *StorHouse/Control Center server*, which runs on the Microsoft Windows NT, XP Pro, or 2000 platform, is a program that enables StorHouse/Control Center clients to communicate with StorHouse systems through a TCP/IP network and optional serial ports. A *StorHouse/Control Center client*, which runs on Windows 95, 98, 2000, XP Pro, or NT platforms, consists of three graphical user interface (GUI) software modules:

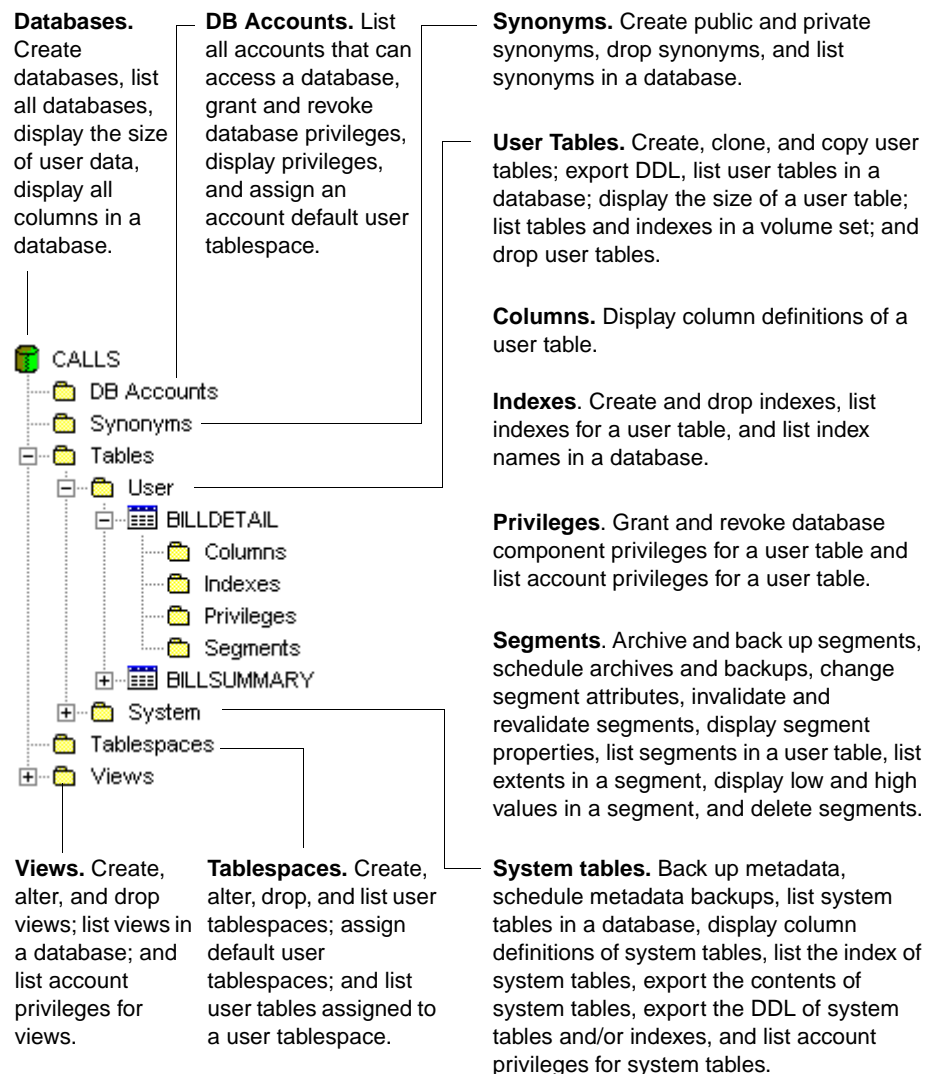
- StorHouse/Admin (system and database administration)
- StorHouse/Performance Monitor (system performance analysis)
- CCAdmin (StorHouse/Control Center server administration)

You can install all of the client modules or some combination on any client machine, as indicated in the following sample configuration.



# StorHouse/Admin

StorHouse/Admin combines StorHouse/SM and StorHouse/RM administration in one user interface, simplifying the tasks of storage management, database administration, system operation, and security control. By navigating with a folder list, you can perform the following database administration functions.



## The GUI ISQL tool

StorHouse/Admin provides an Interactive SQL (ISQL) tool for submitting SQL statements to StorHouse. The following list describes what you can do with the ISQL tool.

SQL-entry features:

- Submit a new SQL statement, the previous one, or the next one
- Save one or more SQL statements as a script and later load the script and run it

Result set features:

- Save a result set as a text file on your computer or network
- Print a result set
- Save a result set as a report and then load that report as needed

General ISQL features:

- Log an ISQL session to a text file
- Limit result sets to 100 rows or allow larger results
- Set options for scripts, such as stop a script when an SQL statement fails or when no rows are returned
- Access online help for StorHouse SQL

