



FileTek MVS Data Loader Utility Manual

StorHouse/RM Release 3.3

Publication Number
900109 Rev. L

July 20, 2004

The FileTek logo consists of the word "FileTek" in white, bold, sans-serif font, centered within a solid 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Copyright © 1996-2004 FileTek, Inc. As an Unpublished Licensed Work.
Publication Number: 900109 Rev. L

NOTICE: U.S. GOVERNMENT USERS

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 Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited.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

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

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

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

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

Contents

Welcome	xiii
StorHouse family of products	xiii
StorHouse/SM	xiii
StorHouse/RM	xiv
StorHouse/Control Center	xiv
Purpose of this document	xiv
Intended audience	xv
Contents	xv
Conventions	xvi
For more information	xvi
For quick reference	xvii
 Chapter 1: Introduction	 1-1
Loading features	1-1
Client and server data loaders	1-3
The data load process	1-4
What you need to load data and indexes	1-6
Before loading	1-7

Loads and segments	1-8
Segment size	1-10
Segment replacement	1-11
Segment merge	1-11
Loads and subspaces	1-11
Default selection of subspaces	1-12
When there is one subspace for all component types	1-13
When there is one subspace for each component type	1-13
When there are multiple subspaces for each component type	1-13
When indexes or LOB columns are assigned to multiple user tablespaces ..	1-15
Explicit selection of subspaces	1-16
When loading one segment	1-16
When loading multiple segments	1-17
When there are multiple indexes of the same type	1-19
When indexes are assigned to different user tablespaces	1-21
Rotation among subspaces	1-24
When there are multiple subspaces for each component type	1-25
When component types share a subspace	1-26
When indexes or LOB columns are assigned to different user tablespaces ..	1-28
Loads and indexes	1-31
Load parallelism	1-31
Loading different tables in one load	1-32
Loading multiple segments of a table in one load	1-33
Loading multiple segments of multiple tables in one load	1-33
Loading different tables in multiple loads	1-33
Loading the same table in multiple loads	1-34
Loading multiple segments of multiple tables in multiple loads	1-34
Querying a table while it's being loaded	1-35
Locking during loads	1-35
If an operation fails	1-35
Checkpoints	1-36
Restart	1-37
Abort	1-37

System table updates	1-38
Metadata updates for a data load operation	1-38
Metadata updates for a replace operation	1-38
Metadata updates for an index load operation	1-39
Metadata updates for a merge operation	1-39
Temporary VRAM file names	1-39

Chapter 2: Installation2-1

Installation overview	2-1
Software function identifier	2-1
System requirements	2-2
Files on the distribution tape	2-2
Installation procedure	2-3
Step 1: Load the SAMPLES dataset	2-3
Step 2: Allocate required datasets	2-4
Step 3: Customize SMP/E JCL procedure	2-5
Step 4: Initialize SMP/E CSI	2-5
Step 5: Execute SMP/E RECEIVE	2-6
Step 6: Execute SMP/E APPLY	2-6
Step 7: Execute SMP/E ACCEPT	2-6
Step 8: Build the checkpoint dataset	2-6

Chapter 3: Input data3-1

What's an input dataset?	3-1
What are input data records and data fields?	3-1
How should you create a host input dataset?	3-1
What record formats can you use?	3-2
Are there any considerations for using the host input dataset?	3-2
Where do you specify which input dataset you're using?	3-2

What's the difference between a column and a field in an input data record?	3-3
What's the difference between a logical record and a physical record?	3-4
Are there any considerations for VAR-type data?	3-5
How do you load LOB data?	3-5
What's delimited data?	3-6
Terminated data	3-7
Enclosed data	3-7
Are blank characters loaded?	3-7

Chapter 4: SYSSQL dataset4-1

About the SYSSQL dataset	4-1
Character set of SYSSQL	4-1
SYSSQL guidelines	4-2
Statement formats	4-3
Format conventions	4-3
SQL identifiers	4-4
LOAD DATA	4-5
LOAD INDEX	4-8
MERGE	4-9
Loading data already on StorHouse	4-9
Format of INFILE clause	4-10
Example INFILE clauses	4-11
To load data from a previous load operation	4-11
To load data from any other VRAM file	4-12
To load data from multiple VRAM files	4-13
To load discarded records	4-13
To load data from a host input dataset and collect discarded records	4-14
Collecting discarded records in a discard file	4-14
Format of DISCARDFILE clause	4-16
Example DISCARDFILE clause	4-16

Limiting the number of discarded records	4-17
Format of DISCARDS clause	4-17
Example DISCARDS clause	4-17
Specifying the character set of the input data	4-18
Format of CHARACTERSET clause	4-19
Example CHARACTERSET clause	4-19
Concatenating a fixed number of physical records into a logical record	4-19
Format of CONCATENATE clause	4-20
Example CONCATENATE clause	4-20
Combining a varied number of physical records into a logical record	4-21
Format of CONTINUEIF clause	4-21
Example CONTINUEIF clauses	4-23
To combine the current physical record with the next one	4-23
To combine the next physical record with the previous one	4-24
To use the last non-blank data column as the comparison value	4-25
To specify the starting column number of a continuation field	4-26
To specify starting and ending column numbers of a continuation field ...	4-27
To use a character string as a comparison value	4-28
To use a hex string as a comparison value	4-28
To use blank characters as a comparison value	4-29
To use a not equal comparison operator	4-30
Preserving blanks	4-30
Format of PRESERVE BLANKS clause	4-31
Example PRESERVE BLANKS clause	4-31
Rotating among subspaces	4-32
Format of SUBSPACE ROTATE clause	4-34
Example SUBSPACE ROTATE clauses	4-34
To rotate among subspaces in a user tablespace	4-34
To rotate among subspaces in multiple user tablespaces	4-35
Identifying the user table to load	4-38
Format of INTO TABLE clause	4-39
Example INTO TABLE clauses	4-39
To use the fully qualified table name	4-39
To omit the owner name	4-40

To use a symbolic variable to substitute an owner name, table name, or both ..	4-40
Choosing which rows to load	4-42
Format of WHEN clause	4-43
Example WHEN clauses	4-45
To specify the starting column number of the selection criteria	4-45
To specify starting and ending column numbers of the selection criteria ...	4-45
To use a column name to identify the selection criteria	4-46
To use a field name to identify the selection criteria	4-47
To use a character string as selection criteria	4-48
To use a hexadecimal string as selection criteria	4-48
To test blanks	4-49
To test multiple values (using AND)	4-49
To test one value or another (using OR)	4-49
To test one value or another and multiple values (using OR and AND) ...	4-50
Generating field_specs, identifying NULL flags, specifying default delimiters and other defaults	4-50
Guidelines for specifying a default delimiter	4-52
Format of FIELDS clause	4-54
Example FIELDS clauses	4-55
To describe data fields terminated with a character	4-56
To describe data fields terminated by a blank	4-56
To describe data fields enclosed by the same delimiter	4-57
To describe data fields enclosed by different delimiters	4-57
To describe data fields that are both terminated and enclosed	4-57
To generate CHAR field_specs	4-58
To identify NULL flags in input data records	4-58
To load NULL values for empty data fields	4-59
Identifying an escape character	4-59
Format of ESCAPED BY clause	4-60
Example ESCAPED BY clause	4-61
Loading missing data fields with null values	4-61
Format of TRAILING NULLCOLS clause	4-62
Example TRAILING NULLCOLS clause	4-63

Loading one or more segments	4-63
Format of SAME and DIFFERENT SEGMENT clauses	4-64
Example SAME and DIFFERENT SEGMENT clauses	4-64
To load multiple segments of the same user table	4-65
To load multiple segments of different user tables	4-65
Naming a segment	4-66
Format of SEGMENT clause	4-67
Example SEGMENT clauses	4-67
To use the load ID as the segment tag	4-67
To assign different segment tags for multiple segments of the same user table ..	4-68
Replacing a segment	4-68
Format of REPLACE SEGMENT clause	4-70
Example REPLACE SEGMENT clause	4-70
Selecting subspaces	4-70
Format of SUBSPACE number clause	4-72
Example SUBSPACE number clauses	4-72
To select subspaces when loading one segment	4-72
To select subspaces when loading multiple segments	4-74
To select subspaces in multiple user tablespaces	4-75
Describing data fields	4-77
Format of field_spec	4-78
Providing a field name	4-80
Providing a column name	4-80
Loading a record number into a column	4-81
Generating a sequence of values	4-81
Loading the current date into a column	4-82
Loading a constant value into a column	4-82
Specifying the position of a data field	4-83
Specifying the data type	4-86
Converting data types	4-101
Calculating the length of a data field	4-102
Specifying a character set for an individual data field	4-105
Specifying a delimiter for an individual data field	4-106
Specifying a BLOB or CLOB data type	4-107

Loading a column with a null value	4-108
Setting a column to the default value	4-109
Using multiple into_table_specs	4-110
Creating multiple logical records from one physical record	4-110
Using the same input dataset to load multiple user tables	4-111
Example LOAD DATA statements	4-112
Example 1: Loading all records into one user table	4-114
Example 2: Combining a fixed number of records and loading some of them into one user table	4-115
Example 3: Loading delimited data into multiple user tables	4-116
Example 4: Combining a variable number of records and loading null values	4-117
Example 5: Loading SMALLINT, DECIMAL, and VARCHAR data	4-119
Example 6: Using relative positioning to load delimited data into multiple user tables 4-121	4-126
Example 7: Using multiple selection criteria	4-128
Example 8: Replacing segments without loading	4-129
Example 9: Including SQL statements in the SYSSQL dataset	4-129
Example 10: Selecting subspaces for each component type	4-129
Example 11: Loading LOB data fields using a default field list and NULLFLAGS . 4-132	4-132
Loading a deferred index	4-134
Format of LOAD INDEX statement	4-135
Example LOAD INDEX statements	4-137
Merging segments of a table	4-137
Format of MERGE statement	4-138
Example MERGE statements	4-140

Chapter 5: Control statements5-1

About loader control statements	5-1
Statement components	5-2
Command verb	5-2
Keyword/Value pairs	5-2
General control statement syntax rules	5-4



LOAD	5-5
SMDEF	5-10

Chapter 6: Runtime 6-1

Preparing to run the utility	6-1
Types of operations	6-2
EXEC statement	6-3
DD statements	6-4
Return codes	6-5
Submitting an operation	6-6
Restarting an operation	6-8
Aborting an operation	6-10

Appendix A: Messages A-1

Index



Contents

Welcome

The *FileTek® MVS Data Loader utility* is a tool for loading database data from an MVS host into user tables on StorHouse®.

StorHouse family of products

StorHouse is the FileTek enterprise-wide solution for managing the capture, storage, movement, and access of gigabytes to petabytes of relational and non-relational detail data. StorHouse technology combines industry-leading, scalable storage devices and Open System processors with specialized storage management and relational database management system (RDBMS) software components.

StorHouse/SM

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

Welcome

Purpose of this document

StorHouse/RM

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

StorHouse/Control Center

StorHouse/Control Center (CC) is the FileTek Windows®-based network computing system for providing administrative control of the *StorHouse* family of products. *StorHouse/Control Center* works with *StorHouse/SM* release 4.2 and higher and consists of one or more *StorHouse/Control Center* servers that communicate with *StorHouse/Control Center* clients over a TCP/IP network. The *StorHouse/Control Center server*, which runs on Windows NT, XP Pro, and 2000 platforms, provides network connectivity to *StorHouse*. The *StorHouse/Control Center clients*, which run on Windows 95, 98, 2000, XP Pro, and NT platforms, consist of one or more graphical user interface (GUI) modules for performing *StorHouse* system and database administration tasks, configuring and managing *StorHouse/Control Center* servers, and analyzing and monitoring *StorHouse* activity and performance.

Purpose of this document

The *FileTek MVS Data Loader Utility Manual* describes how to install and run the FileTek MVS Data Loader utility. It also explains how to set up the LOAD DATA statement, SYSIN control statements, and execution Job Control Language (JCL) necessary to load data into *StorHouse* user tables.

Intended audience

The *FileTek MVS Data Loader Utility Manual* is intended for whomever is responsible for managing, scheduling, and/or running the FileTek MVS Data Loader utility at your organization. This manual assumes that you understand Structured Query Language (SQL), JCL, StorHouse fundamentals, and MVS concepts and facilities. It also assumes that you know how to install software with the IBM® System Modification Program Extended (SMP/E).

Contents

This document is organized as follows:

- Chapter 1, “Introduction,” describes the features and functions of the FileTek MVS Data Loader utility.
- Chapter 2, “Installation,” explains how to install the FileTek MVS Data Loader utility with the IBM System Modification Program Extended (SMP/E).
- Chapter 3, “Input data,” answers questions about input datasets and input data records.
- Chapter 4, “SYSSQL dataset,” describes the SQL-like statement that identifies the user table to be loaded and describes the data type and structure of the input data.
- Chapter 5, “Control statements,” describes the SYSIN control statements that supply runtime information to the FileTek MVS Data Loader utility.
- Chapter 6, “Runtime,” defines how to set up execution JCL for different types of loading operations.

- Appendix A, “Messages,” lists the FileTek MVS Data Loader utility error, warning, and informational messages.

Conventions

This book uses the following notational conventions:

Convention	Meaning
Courier font	JCL and control statements
<i>Italics</i>	New terms, emphasized text, variables, and titles
Helvetica font	LOAD DATA formats and examples
lowercase Helvetica font	Message text variables
▼	Procedures

See page 4-3 for format conventions of the LOAD DATA statement.

For more information

The following publications contain information related to the FileTek MVS Data Loader utility.

- To learn the basics of StorHouse/RM, refer to *StorHouse/RM Concepts*, publication number 900132.
- To perform StorHouse database administration tasks like creating user tables and indexes, managing accounts and privileges, and defining user tablespaces, refer to the *StorHouse Database Administration Guide*, publication number 900108.

- For explanations of StorHouse return codes you may receive in message listings, refer to the *StorHouse Messages and Codes Manual*, publication number 900032.
- To learn the concepts, structures, and functions of StorHouse, refer to the *StorHouse Concepts and Facilities Manual*, publication number 900026.
- To learn how to unload data from StorHouse user tables by using FTP, refer to the *FileTek FTP Data Unloader Manual*, publication number 900137.

For quick reference

This manual contains comprehensive descriptions of LOAD DATA, LOAD INDEX, and MERGE statements. Refer to the *StorHouse/RM SQL and Utility Quick Reference*, publication number 900122, for just the syntax of the LOAD DATA statement.



Welcome

For quick reference

Introduction

This chapter introduces the FileTek MVS Data Loader utility. It describes:

- Loading features
- Client and server data loaders
- The data load process
- What you need to load data and indexes
- What to do before loading
- Loads and segments
- Loads and subspaces
- Loads and indexes
- Load parallelism
- Locking during loads
- Checkpoints, restart, abort
- System table updates
- VRAM file names

Loading features

The FileTek MVS Data Loader utility is a bulk data loading system designed to load large volumes of data from your host computer (the client) into StorHouse user tables on StorHouse (the server). Those user tables can be empty or they can contain data from a previous load operation. You also use the FileTek MVS Data Loader utility to load deferred indexes and to merge existing segments. Key features of the FileTek MVS Data Loader utility are as follows.

Parallelism. StorHouse segmentation technology and read-only databases enable you to load the same table in parallel. You can also query and load the same table at the same time. See “Loads and segments” on page 1-8 for information about segmentation. See “Load parallelism” on page 1-31 for ways to load in parallel.

Multiple record formats. The FileTek MVS Data Loader utility accepts input data in fixed-length, variable-length, and large object (LOB) type formats. See Chapter 3 for considerations about input data.

Data type conversion. The FileTek MVS Data Loader utility supports a wide range of input data types. Compatible input data types are automatically converted to the data types of the user table. For example, you can load input data defined as INTEGER EXTERNAL into a column of a user table defined as CHARACTER, INTEGER, SMALLINT, or VARCHAR. See “Converting data types” on page 4-101 for the supported conversions.

Compatibility. The load information that you supply is similar and compatible with the load information accepted by DB2™ and Oracle® load utilities. The FileTek MVS Data Loader utility accepts DB2 and Oracle and clauses that are not part of the StorHouse syntax but ignores those that do not apply to StorHouse.

Exception processing. You can collect discarded records that don’t meet your load criteria and then load those discarded records as required. See “Collecting discarded records in a discard file” on page 4-14 for more information about exception processing.

Error reporting. Messages are captured in runtime and error listings to help you monitor the progress and status of load operations. Appendix A contains a list and explanation of messages.

Restart capability. You can restart an operation from the point of failure. You can also abort a load and then start from the beginning. See “If an operation fails” on page 1-35 for more information about restart options.

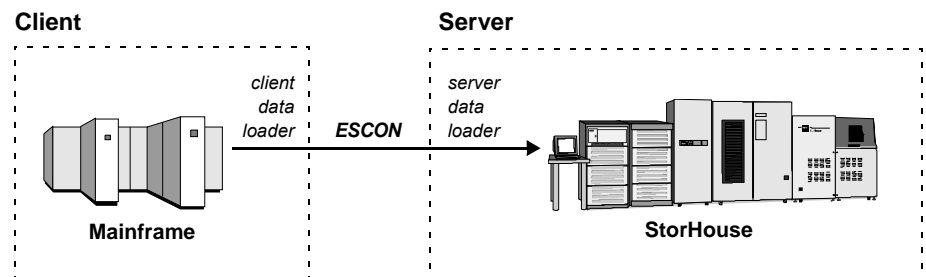
SQL tool. You can use the FileTek MVS Data Loader utility to submit StorHouse SQL statements. For example, before you load data you can create the user table and indexes by issuing CREATE TABLE and CREATE INDEX statements. Each SQL statement is committed when it completes. The SELECT statement is the only SQL statement you cannot submit with the FileTek MVS Data Loader utility. See page 4-129 for an example.

Client and server data loaders

Loading StorHouse user table data is a two-phase process that requires two FileTek programs:

- The *client data loader*, which you install and run on your host computer, prepares your data for loading and transfers it to StorHouse. This phase of a load operation is called the *copy phase*. You supply runtime parameters to the client data loader.
- The *server data loader*, which runs on StorHouse, performs any data conversions, loads your data into StorHouse user tables, and builds and stores any indexes for those user tables. This phase of a load operation is called the *load phase*.

The client and server data loaders use the channel connection to achieve maximum data transfer rates.



The *FileTek MVS Data Loader utility* is the FileTek-supplied client data loader. It is an MVS batch program that performs the following functions:

- Reads input datasets and builds a data stream consisting of control information and your user data
- Writes the data stream to a temporary Virtual Record Access Manager (VRAM™) file on StorHouse
- Issues StorHouse Command Language commands that instruct the server data loader to begin loading data from the temporary VRAM file to your StorHouse user tables and to update the metadata when the load completes
- Returns operation status information and messages
- Provides a restart capability at both the copy and load phases of a load operation

The data load process

The process of loading data with the FileTek MVS Data Loader utility is as follows:

1 Prepare the input and submit the job

At your host, prepare the following:

- A SYSREC sequential dataset with the input data you're loading. You can specify multiple input datasets by concatenating them to the SYSREC DD statement.
- A SYSSQL dataset with a LOAD DATA statement. This statement identifies the user table and describes the data you're loading.
- A SYSIN (or instream) dataset with a LOAD and SMDEF control statement. These control statements supply runtime parameters.

2 Build the data stream

After you submit the job, the client data loader builds a *data stream*, which is a sequence of four variable-length record types:

- Environment definition (built by the client data loader)
- SQL and LOAD DATA statements (from the SYSSQL dataset)
- Data delimiter (provided by the client data loader)
- Your data (from the SYSREC dataset)

Data records begin immediately after the delimiter record and end at physical end-of-file. The *environment definition* specifies defaults for all character coding so that the server data loader can correctly interpret the remainder of the data stream.

3 Copy the data stream to a StorHouse VRAM file

The client data loader then copies the data stream to a temporary VRAM file on StorHouse. You can keep the VRAM file after the load completes or request the client data loader to delete it. If a load fails, the VRAM file is always kept to allow a subsequent restart.

4 Invoke the server data loader

After writing the temporary VRAM file to StorHouse, the client data loader activates the server data loader by constructing and submitting a StorHouse Command Language EXECUTE STH_LOAD command. It builds the command based on internally generated parameters as well as those that you supply on the JCL EXEC PARM and SYSIN.

5 Execute the load

The server data loader parses and reads the LOAD DATA statement, and a StorHouse engine checks your StorHouse account privileges and obtains the metadata to store the table data and indexes. The server data loader then loads the StorHouse user table(s) and builds and stores the indexes. When the load completes successfully, a StorHouse engine updates the metadata.

What you need to load data and indexes

In addition to the FileTek MVS Data Loader utility client software, you need a *StorHouse account ID* and *password* to access StorHouse. You provide the account ID and password on the LOAD control statement in the SYSIN dataset. The StorHouse account ID must have the following StorHouse privileges to load data, load indexes, and merge segments.

- Database component privilege:
 - INSERT privilege for the user table(s) you're loading
- Access privileges:
 - SQLCOMMAND
 - SQLEXECUTE
- Command privileges:
 - ATF
 - DELETE
 - GET
 - PUT
 - RECORD
 - SETGROUP
 - VTF

If you submit StorHouse SQL statements with the FileTek MVS Data Loader utility, the account ID must have the appropriate privilege for the statement. For example, to submit CREATE TABLE and CREATE INDEX statements, an account ID must have the DBA or RESOURCE database privilege.

Refer to the *StorHouse Database Administration Guide* for more information about database component and access privileges. Refer to the *StorHouse Command Language Reference Manual* for more information about command privileges.

Before loading

The StorHouse system or database administrator must complete specific tasks before loading. The following table lists these tasks and who is typically responsible for performing them.

StorHouse task	Command/Statement	Person responsible
Create the StorHouse signon account used by the FileTek MVS Data Loader utility. This account must have the access and command privileges listed on page 1-6.	CREATE ACCOUNT	StorHouse system administrator
Create the volume sets and file sets for the VRAM files, discard files, user tables, and indexes, if needed.	CREATE VSET CREATE FSET	StorHouse system administrator
Create the user tablespace associated with the user table to be loaded.	CREATE TABLE SPACE	StorHouse database administrator
Create the user table to be loaded.	CREATE TABLE	StorHouse database administrator
Create indexes for the user table. An administrator can create deferred indexes after the table is loaded.	CREATE INDEX	StorHouse database administrator
Grant INSERT privilege to the StorHouse signon account ID for the user table to be loaded.	GRANT	StorHouse database administrator
Create discard files to collect discarded records that do not meet selection criteria. This file is a VRAM file.	CREATE FILE	StorHouse system administrator

Loads and segments

When the server data loader “loads” a user table and “stores” an index, it actually writes the table data and index entries to files on the StorHouse storage hierarchy. This set of files is called a *segment*. When you load data into a table for the first time, the server data loader creates a segment on StorHouse. When you load more data into that user table, the server data loader creates another segment. Each load into a table creates a segment.

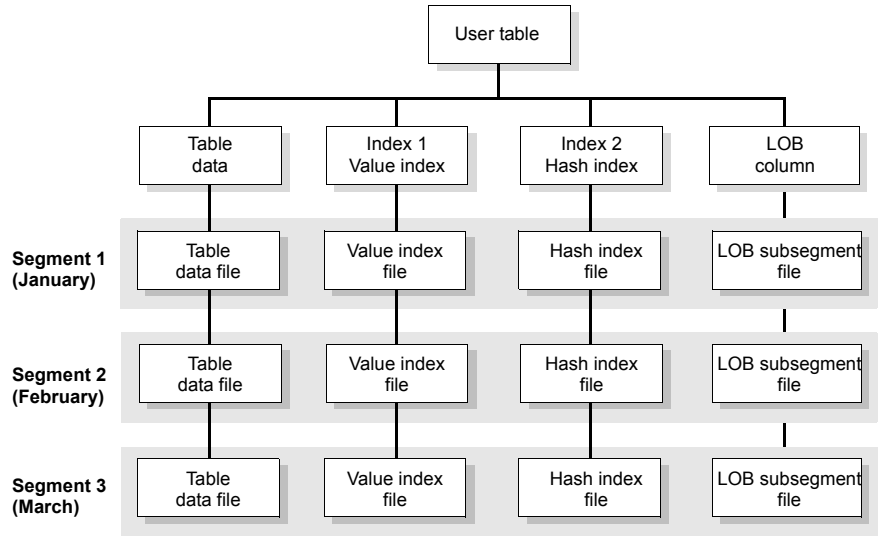
Each segment consists of:

- One table data file
- One index file for each value index
- One index file for each hash index
- One or more LOB subsegment files for LOB columns

A range index applies to all segments of the user table. Depending on the actual size or by user request, a LOB value may be stored in the table data file, or multiple LOB values in different columns may be stored in the same LOB subsegment file, or LOB values for the same column may be stored in multiple LOB subsegment files. An *in-line LOB* is a LOB value stored in the table data file. An *out-of-line LOB* is a LOB value stored in a LOB subsegment file. Refer to the *StorHouse Database Administration Guide* for more information about LOB storage options.

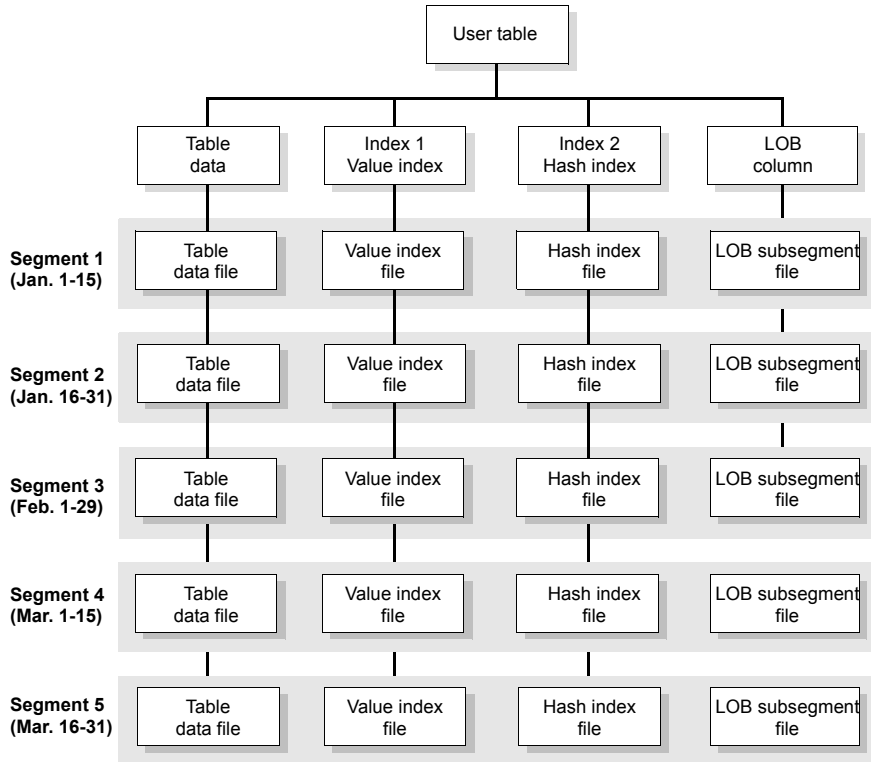
For example, suppose a user table consists of one value index, one hash index, and one LOB column. Assume you loaded data into the table in January, February, and March. After the March load, the table consists of three segments.

Each segment contains one table data file, one value index file, one hash index file, and one LOB subsegment file.



By default, the server data loader creates *one* segment for each load. You also have the option of creating *multiple* segments during one load. For example, in

January you could create two segments, in February you could create one segment, and in March you could create two segments.



Segment size

The maximum size of a table data file is approximately 100 GB. If your input data exceeds 100 GB, you must do one of the following:

- Load the data into multiple segments, ensuring that each segment does not exceed the maximum size. See page 4-62 for more information about loading data into multiple segments during one load.

- Split the data into multiple data files and run multiple loads. Each load writes to a different segment.

Segment replacement

You can replace an existing segment. Replacing a segment invalidates the segment's files, making them inaccessible. A *replaced segment* continues to reside on StorHouse and the segment files are system-managed according to the user tablespace parameters. You can delete and remove a replaced segment or re-validate it later if users need to access it. The range index entries for a replaced segment remain in the system tables.

Segment merge

You can merge existing segments of a table with the FileTek MVS Data Loader utility. A *merge*, or *coalesce*, *operation* consolidates selected segments into one segment or more segments depending on your merge criteria. You perform a merge operation separately from a load operation. See “Merging segments of a table” on page 4-137 for more information about performing a merge operation.

Loads and subspaces

A user tablespace consists of one or more *subspaces* that specify where and how segments are stored on StorHouse. When creating a user table, you assign it to a user tablespace. If the user table contains LOB columns, you can assign those LOB columns to the same user tablespace as the table or to different user tablespaces. And when creating 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 even assign different indexes of a table to different user tablespaces.

When you load data, the server data loader determines the user tablespace(s) and can store each component in a default subspace. The data loader uses the value of

the `OBJECT_TYPE` parameter (specified on a `CREATE TABLE SPACE` or `ALTER TABLE SPACE` statement) to determine the type of component allowed in a subspace. The `OBJECT_TYPE` values are:

- Blank for all components
- T for table data
- H for hash indexes
- V for value indexes
- L for LOB data

You can also explicitly select subspaces or rotate among subspaces that are valid for a component type. This section describes default selection, explicit selection, and rotation of subspaces. Refer to the *StorHouse Database Administration Guide* for more information about user tablespaces and subspaces.

Note: LOB subspaces and tablespaces are not used for in-line LOBs. The server data loader uses the table data subspace and the user table tablespace for LOB values that fit within a row.

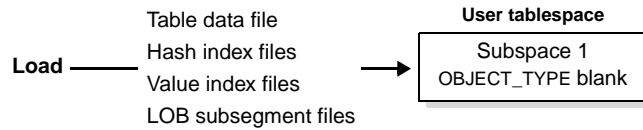
Default selection of subspaces

A *default subspace* is the lowest-numbered subspace that allows a component type. If you do not explicitly select subspaces or request rotation during a load, the data loader selects the default. This section describes the default selection of subspaces when:

- There is one subspace for all component types
- There is one subspace for each component type
- There are multiple subspaces for each component type
- Indexes or LOB columns are assigned to multiple user tablespaces

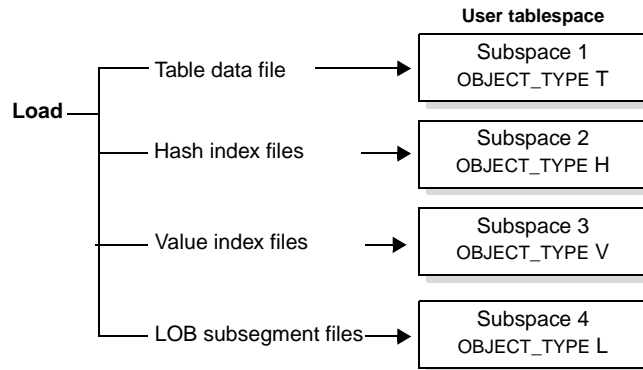
When there is one subspace for all component types

When a user tablespace contains one subspace for all component types, and the user table, indexes and LOB columns are assigned to the same user tablespace, the data loader automatically selects that subspace during a load.



When there is one subspace for each component type

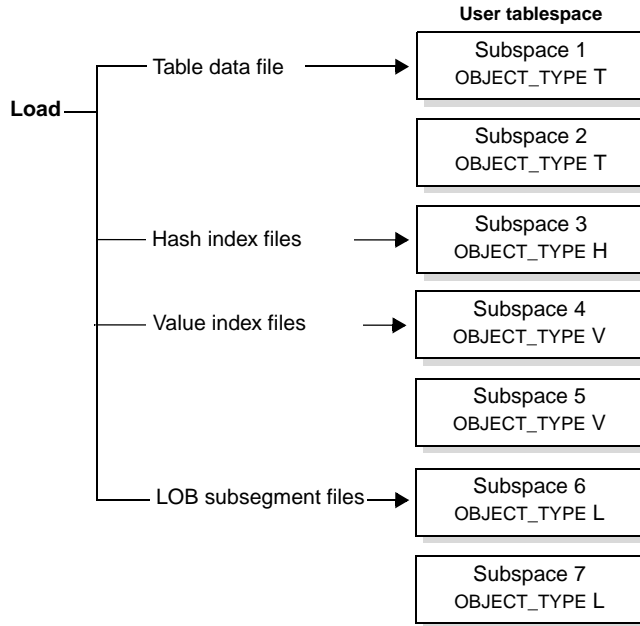
When a user tablespace contains one subspace for each component type, and the user table, indexes, and LOB columns are assigned to the same user tablespace, the data loader automatically selects the appropriate subspace for each component type.



When there are multiple subspaces for each component type

When a user tablespace contains multiple subspaces for the same component type or multiple subspaces with no component type restrictions (OBJECT_TYPE is blank), and the user table, indexes, and LOB columns are assigned to the same

user tablespace, the data loader automatically selects the lowest-numbered subspace for the component type.



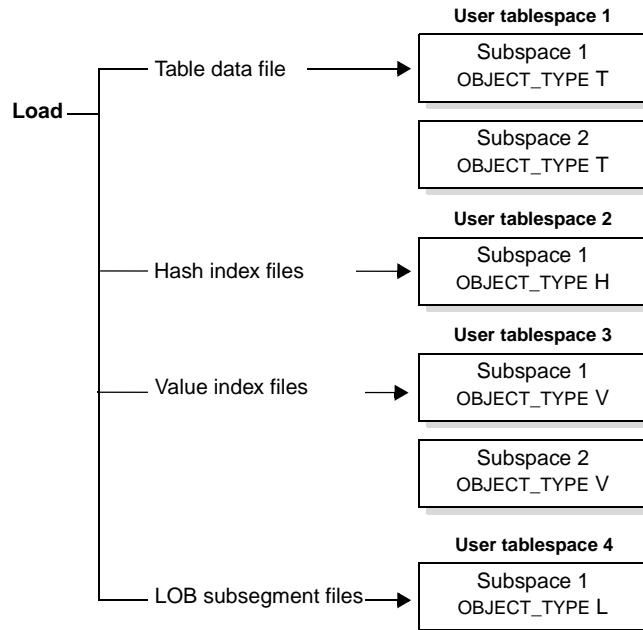
In this example, the data loader selects:

- Subspace 1 for the table data file (because it's the lowest-numbered subspace that allows table data)
- Subspace 3 for hash index files (because it's the only subspace that allows hash indexes)
- Subspace 4 for value index files (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 for LOB subsegment files (because it's the lowest-numbered subspace that allows LOB data)

The data loader does not use subspaces 2, 5, and 7 unless you explicitly select them.

When indexes or LOB columns are assigned to multiple user tablespaces

When any indexes or LOB columns for a user table are assigned to different user tablespaces, the data loader automatically selects the lowest-numbered subspace in the appropriate user tablespace for the component type. For instance, assume a user table has one hash index, one value index, and one LOB column. The user table is assigned to user tablespace 1, the hash index is assigned to user tablespace 2, the value index is assigned to user tablespace 3, and the LOB column is assigned to user tablespace 4. In the following example, the data loader selects subspace 1 as the default for each component type in each user tablespace.



Explicit selection of subspaces

If a user tablespace contains multiple subspaces for the same component type and you want to use a subspace other than the default, you can explicitly select that subspace during a data load, index load, or merge operation. You can select a subspace for each component type. This section describes explicit selection when:

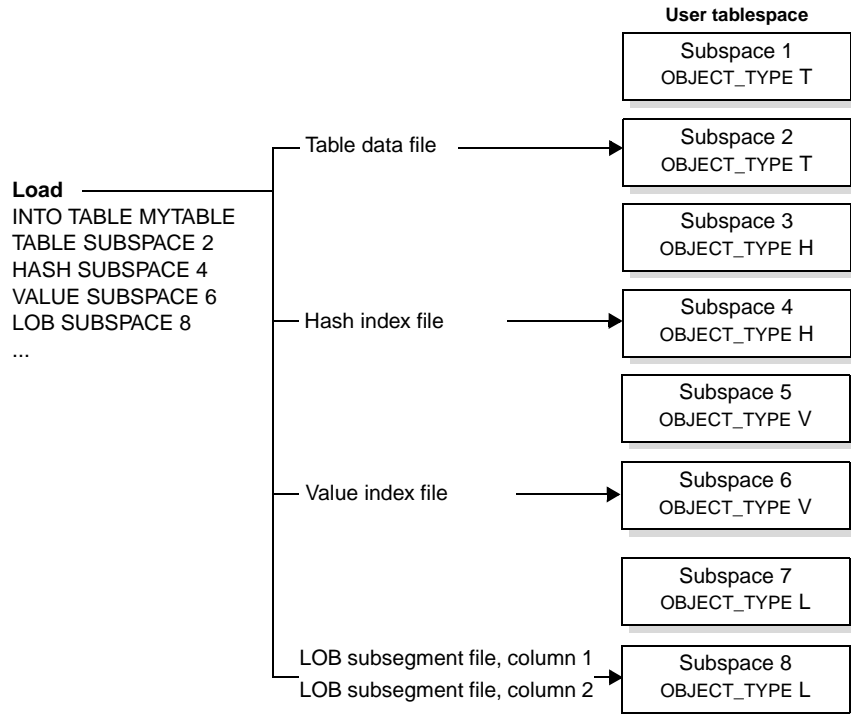
- Loading one segment
- Loading multiple segments
- There are multiple indexes of the same type
- Indexes are assigned to different user tablespaces

You include a SUBSPACE number clause on a LOAD DATA, LOAD INDEX, or MERGE statement to select subspaces. See “Selecting subspaces” on page 4-70 for the format, examples, and more information about this clause. This section also contains examples.

When loading one segment

When loading one segment into a user tablespace with multiple subspaces for component types, you can select a subspace for each component type. For example, assume the user table has one hash index, one value index, and two LOB columns assigned to the same user tablespace as the table. You’re loading one segment into a user tablespace that contains two subspaces for each component type. Remember that if you don’t explicitly select subspaces, the data loader automatically selects the subspace with the lowest subspace number for the applicable component type.

In the following example, to use subspace 2 for the table data, subspace 4 for the hash index, subspace 6 for the value index, and subspace 8 for the LOB data, you must explicitly select those subspaces for each component type.

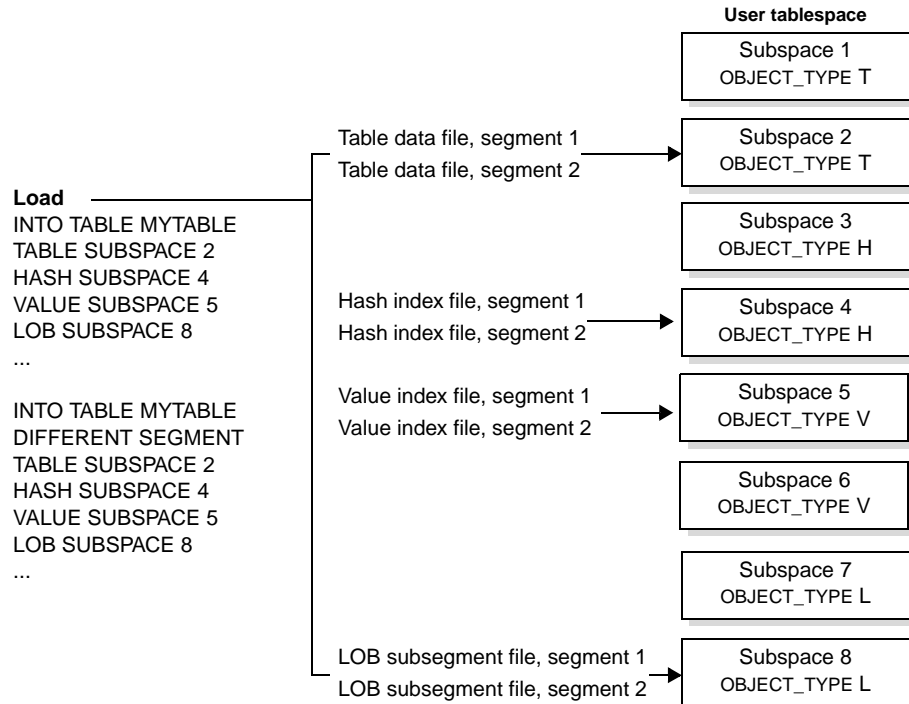


When loading multiple segments

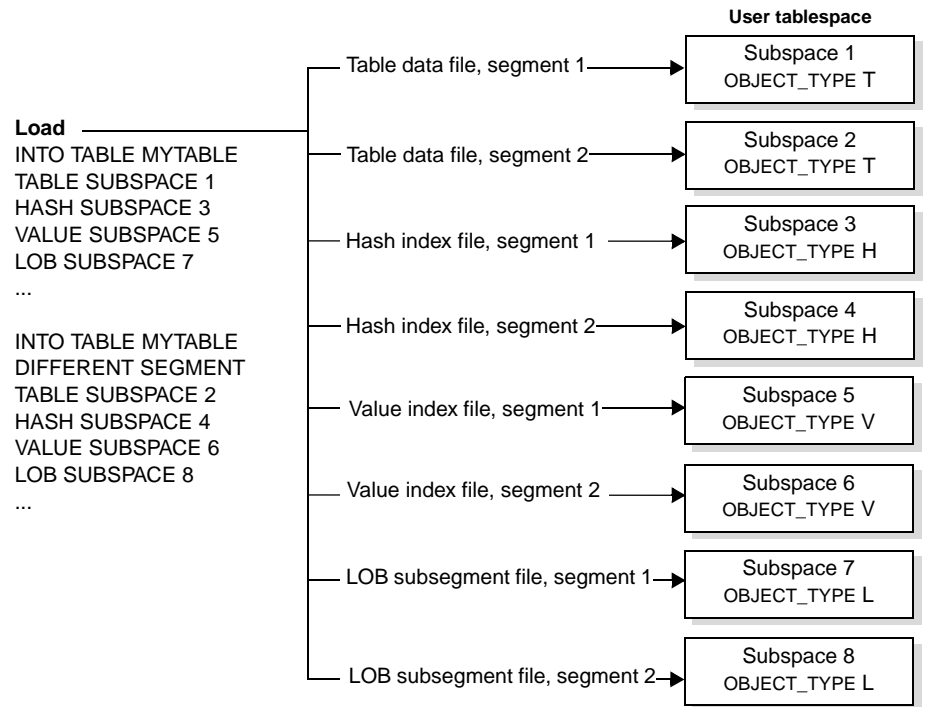
When loading multiple segments into a user tablespace with multiple subspaces for component types, you can select the same subspaces for all segments or different subspaces for each segment.

For example, assume a user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. You're loading two segments into a user tablespace that contains two subspaces for each component type. You can select the same subspace for each segment and component type, for instance, subspace 2 for both table data files, subspace 4 for both hash index files,

subspace 5 for both value index files, and subspace 8 for both LOB subsegment files.



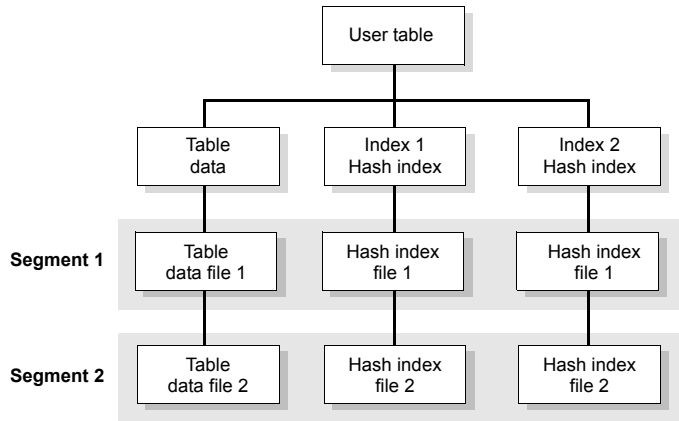
Or you can select different subspaces for each segment and component type.



When there are multiple indexes of the same type

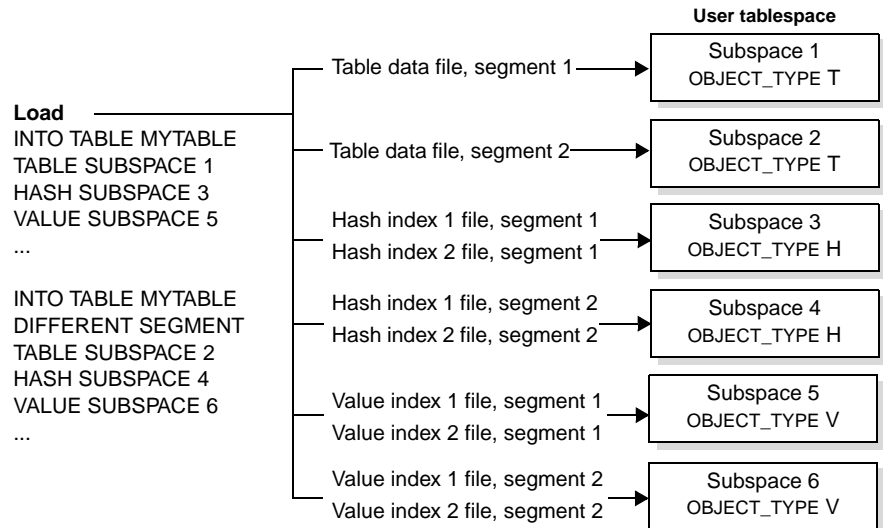
Remember that for each hash index and each value index on a user table, there's one hash index file and one value index file for each table data file. So if you're

loading two segments, and the user table has two hash indexes, then a load creates four hash index files (two files in each segment).



When you select subspaces for indexes, all same-type (hash or value) index files that correspond to the same segment use the same subspace. For example, assume you're loading two segments. The user table has two hash indexes and two value indexes assigned to the same user tablespace as the table. For the first segment, you could select subspace 3 for both hash index files and subspace 5 for both

value index files. For the second segment, you could select subspace 4 for both hash index files and subspace 6 for both value index files.

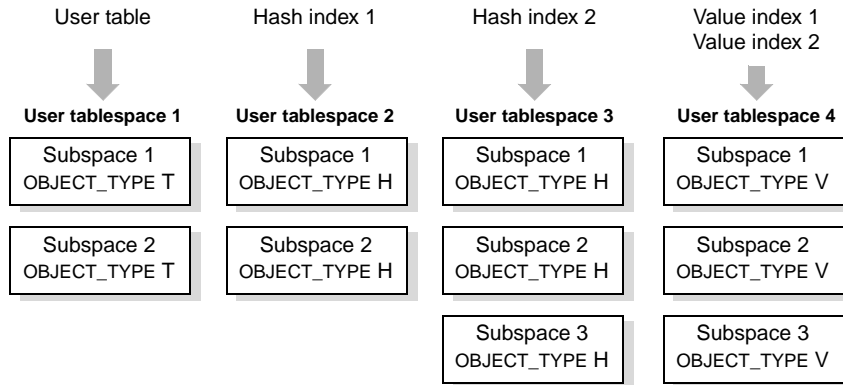


You don't select subspaces for certain indexes (like hash index 1 or value index 2). You select subspaces for component types (like hash indexes or value indexes). All hash index files that correspond to the same segment use the same subspace. All value index files that correspond to the same segment use the same subspace.

When indexes are assigned to different user tablespaces

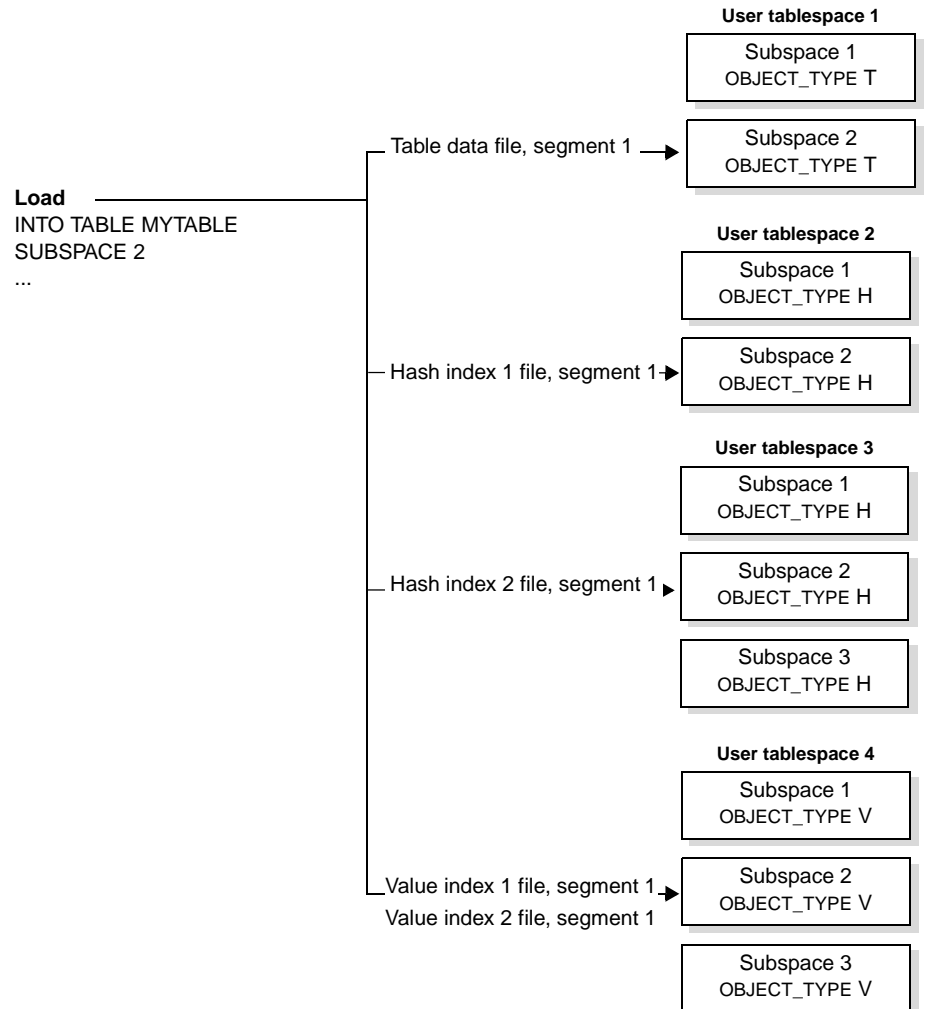
You can select a specific subspace for different component types, even when same-type indexes are assigned to different user tablespaces. The subspace number, however, must exist in all applicable user tablespaces and each subspace must allow that component type. For example, assume you're loading a user table with two hash indexes and two value indexes. The user table is assigned to user tablespace 1, the first hash index is assigned to user tablespace 2, the second hash

index is assigned to user tablespace 3, and both value indexes are assigned to user tablespace 4.



Notice that user tablespace 2 contains two subspaces for hash indexes and user tablespace 3 contains three subspaces for hash indexes. In this case, you cannot select subspace 3 for hash indexes because user tablespace 2 does not contain a subspace 3.

You can also select a specific subspace for all component types. For instance, you can simply select subspace 2. The data loader then uses subspace 2 in each user tablespace.



Rotation among subspaces

If you're loading multiple segments into a user tablespace with multiple subspaces for component types, you can rotate among the applicable subspaces. You can also rotate among subspaces for index load and merge operations. Rotation occurs independently for each component type. That is:

- For the table data file in the first segment, the data loader starts with the lowest-numbered subspace that allows table data and then uses the next subspace that allows table data for the next segment.
- For hash index files in the first segment, the data loader starts with the lowest-numbered subspace that allows hash indexes and then uses the next subspace that allows hash indexes for the next segment.
- For value index files and LOB subsegment files, the same rotation occurs.

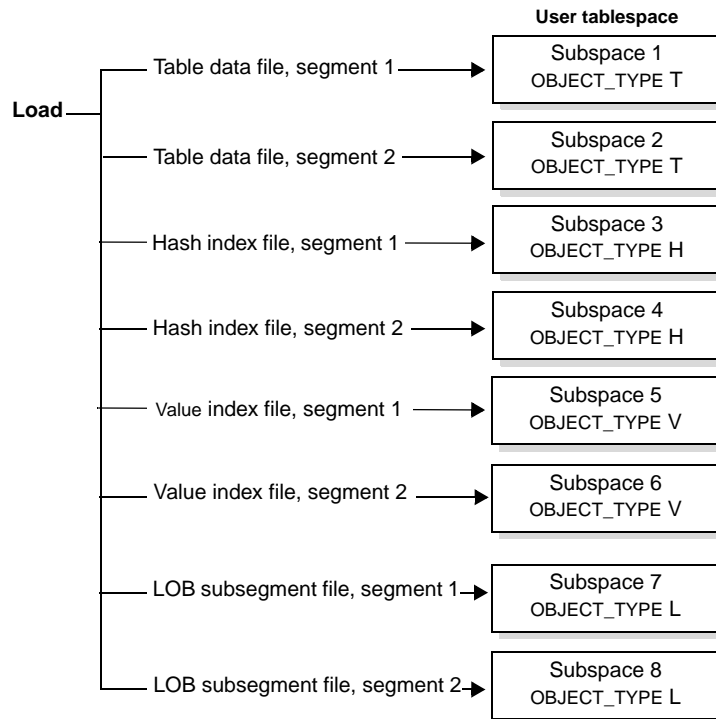
This section describes rotation when:

- There are multiple subspaces for each component type
- Component types share a subspace
- Indexes or LOB columns are assigned to different user tablespaces

You include a SUBSPACE ROTATE clause in a LOAD DATA, LOAD INDEX, or MERGE statement to rotate among subspaces. See “Rotating among subspaces” on page 4-33 for more information and additional examples.

When there are multiple subspaces for each component type

A user tablespace must contain multiple subspaces for rotation to occur. For example, assume a user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. The user tablespace contains two subspaces for each component type. You're loading two segments and request to rotate among subspaces. The data loader rotates among subspaces as follows:



In this example, the data loader uses:

- Subspace 1 for the table data file in segment 1 (because it's the lowest-numbered subspace that allows table data)
- Subspace 2 for the table data file in segment 2 (because it's the next subspace that allows table data)
- Subspace 3 for the hash index file in segment 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 4 for the hash index file in segment 2 (because it's the next subspace that allows hash indexes)
- Subspace 5 for the value index file in segment 1 (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 for the value index file in segment 2 (because it's the next subspace that allows value indexes)
- Subspace 7 for the LOB subsegment file in segment 1 (because it's the lowest-numbered subspace that allows LOB data)
- Subspace 8 for the LOB subsegment file in segment 2 (because it's the next subspace that allows LOB data)

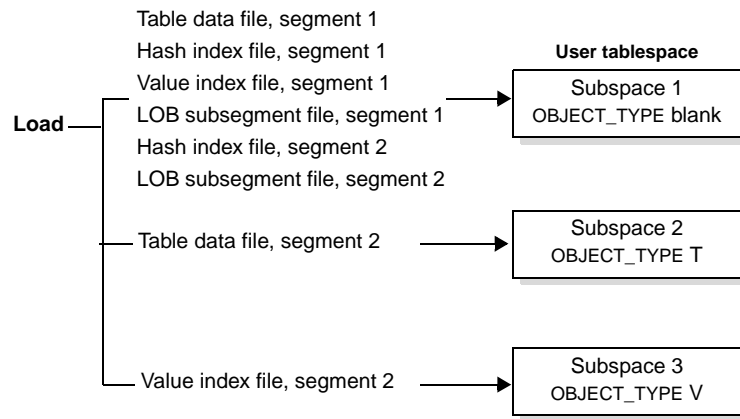
If you were loading a third segment, the data loader would use subspace 1 for the table data file, subspace 3 for the hash index file, subspace 5 for the value index file, and subspace 7 for the LOB subsegment file.

When component types share a subspace

A user tablespace does not have to contain a subspace for each component type. You can also define subspaces that allow all components types (OBJECT_TYPE is blank). For instance, assume a user tablespace contains three subspaces: one that allows all component types, a second for table data files only, and a third for value

index files only. In this example, subspace 1 is the only subspace that allows hash index files and LOB subsegment files, but it also allows table data files and value index files.

Assume you're loading two segments and request to rotate among the applicable subspaces for each component type. The user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. The data loader rotates among subspaces as follows:



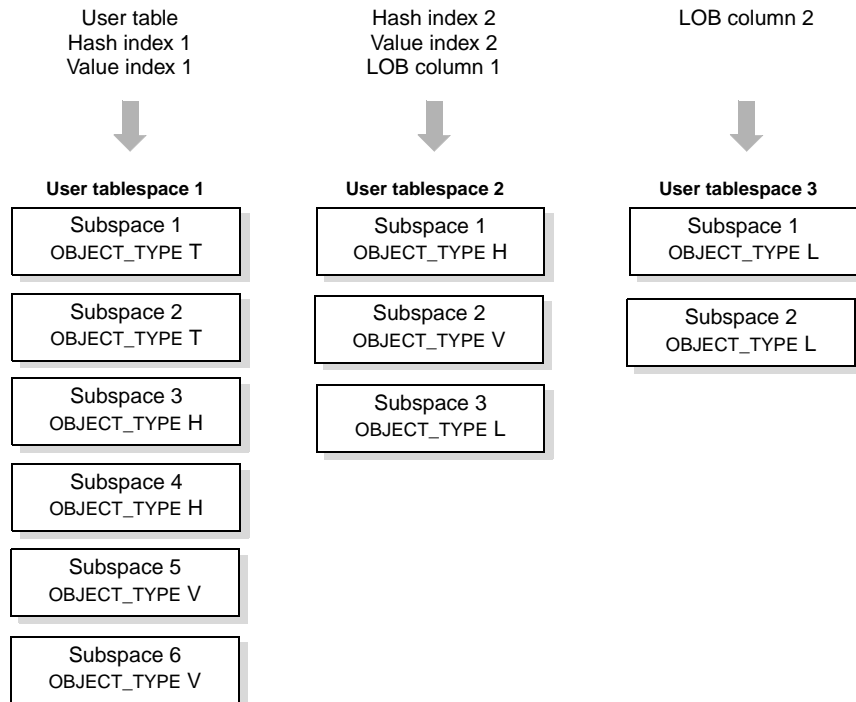
In this example, the data loader uses:

- Subspace 1 for the table data file, hash index file, value index file, and LOB subsegment file for segment 1 (because it's the lowest-numbered subspace that's valid for all component types)
- Subspace 2 for the table data file in segment 2 (because it's the next subspace that allows table data)
- Subspace 3 for the value index file in segment 2 (because it's the next subspace that allows value indexes)

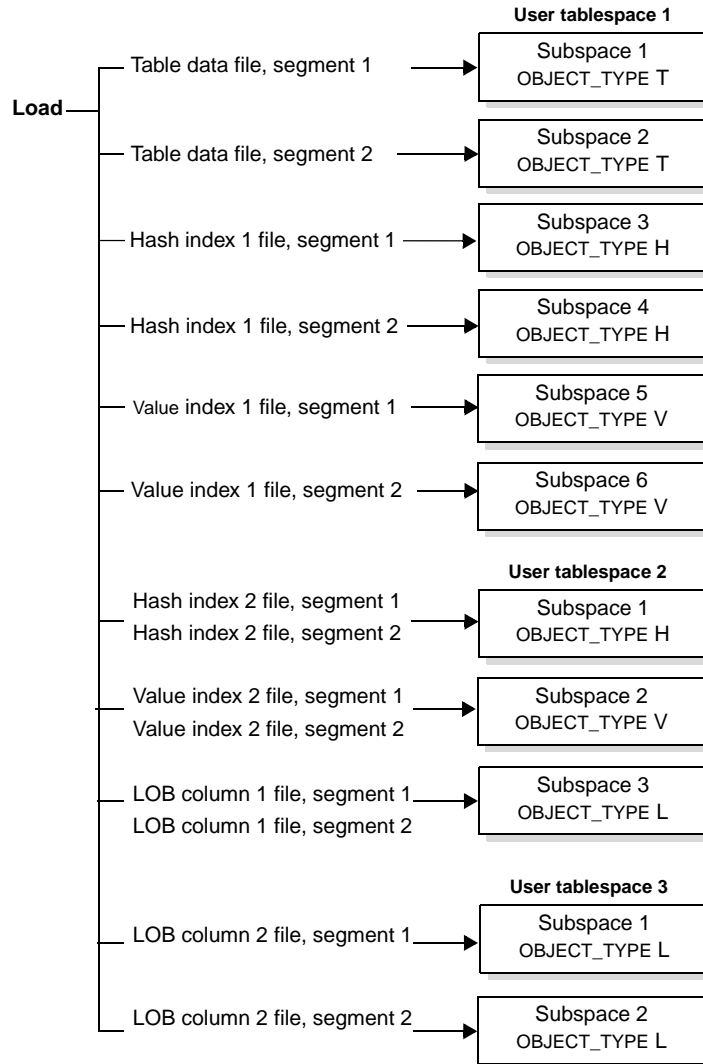
- Subspace 1 for the hash index file and LOB subsegment file in segment 2 (because it's the only subspace that allows hash indexes and LOB subsegment files)

When indexes or LOB columns are assigned to different user tablespaces

When any indexes or LOB columns for a user table are assigned to different user tablespaces, the data loader rotates among applicable subspaces in the applicable user tablespaces. For example, assume the user table has two hash indexes, two value indexes, and two LOB columns. The user table, hash index 1, and value index 1 are assigned to user tablespace 1; hash index 2, value index 2, and LOB column 1 are assigned to user tablespace 2; and LOB column 2 is assigned to user tablespace 3.



In the following example, the data loader rotates among subspaces in the user tablespaces as follows:



In this example, the data loader uses the following subspaces for table data files:

- Subspace 1 in user tablespace 1 for the table data file in segment 1 (because it's the lowest-numbered subspace that allows table data)
- Subspace 2 in user tablespace 1 for the table data file in segment 2 (because it's the next subspace that allows table data)

The data loader uses the following subspaces for hash index files:

- Subspace 3 in user tablespace 1 for the hash index file of hash index 1 in segment 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 4 in user tablespace 1 for the hash index file of hash index 1 in segment 2 (because it's the next subspace that allows hash indexes)
- Subspace 1 in user tablespace 2 for both hash index files of hash index 2 in both segments (because it's the only subspace that allows hash indexes)

The data loader uses the following subspaces for value index files:

- Subspace 5 in user tablespace 1 for the value index file of value index 1 in segment 1 (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 in user tablespace 1 for the value index file of value index 1 in segment 2 (because it's the next subspace that allows value indexes)
- Subspace 2 in user tablespace 2 for both value index files of value index 2 in both segments (because it's the only subspace that allows value indexes)

The data loader uses the following subspaces for LOB subsegment files:

- Subspace 3 in user tablespace 2 for LOB column 1 subsegment files for both segments (because it's the only subspace that allows LOB data)

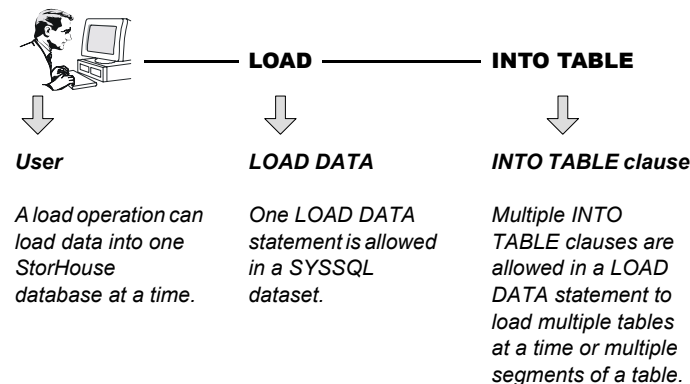
- Subspace 1 in user tablespace 3 for the LOB column 2 subsegment file for segment 1 (because it's the lowest-numbered subspace that allows LOB data)
- Subspace 2 in user tablespace 3 for the LOB column 2 subsegment file for segment 2 (because it's the next subspace that allows LOB data)

Loads and indexes

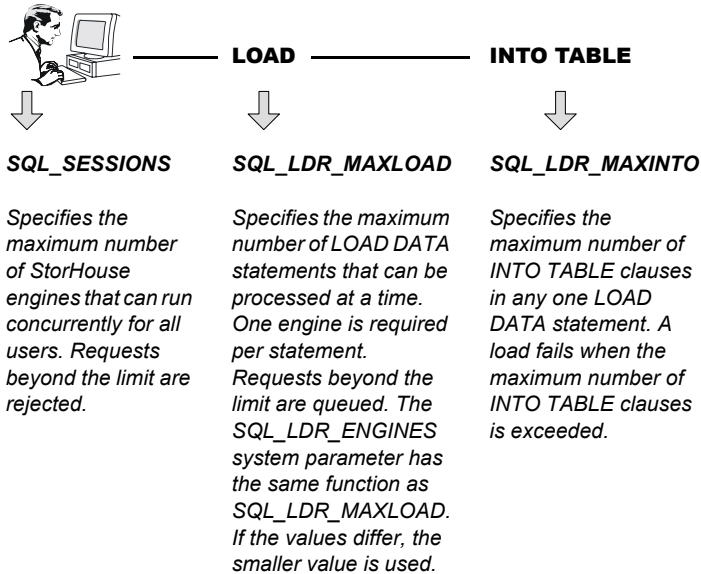
You can create indexes before or after a table has been loaded. A *deferred index* is an index created after a table has been loaded. You must perform an *index load operation* to load a deferred index. This operation is similar to the load process, except the SYSSQL dataset contains a LOAD INDEX statement that identifies the indexes and optionally the segments to be loaded. See “Loading a deferred index” on page 4-134 for more information about performing an index load operation.

Load parallelism

This section illustrates the different ways you can load in parallel. The graphics on the following pages contain these components:

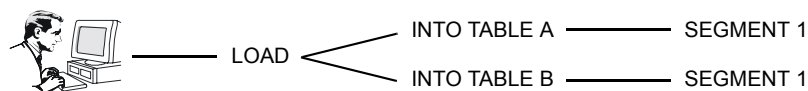


Three tunable StorHouse *system parameters* help control parallelism:



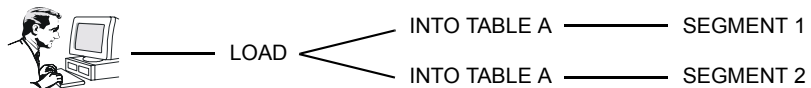
Loading different tables in one load

A user can load different user tables in one load. By default, a load writes to one segment.



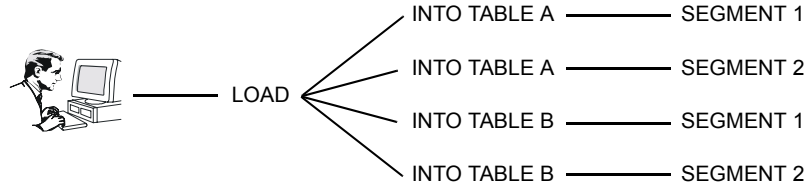
Loading multiple segments of a table in one load

A user can load one or more segments of the same user table in one load.



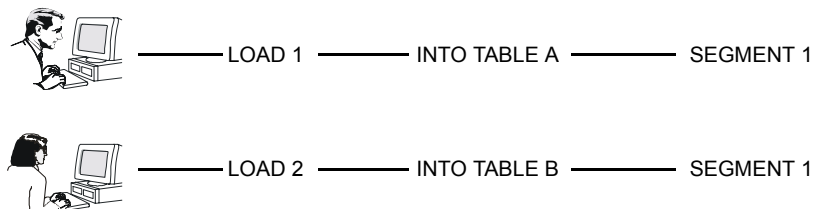
Loading multiple segments of multiple tables in one load

A user can load one or more segments of multiple user tables in one load.



Loading different tables in multiple loads

Multiple users can load different user tables concurrently. By default, a load writes to one segment.



Loading the same table in multiple loads

Multiple users can load the same user table concurrently. Each load writes to a different segment of that user table.



————— LOAD 1 ————— INTO TABLE A ————— SEGMENT 1



————— LOAD 2 ————— INTO TABLE A ————— SEGMENT 2

Loading multiple segments of multiple tables in multiple loads

Multiple users can load one or more segments of multiple user tables. Each load writes to a different segment of a table.



————— LOAD 1 —————
 / / INTO TABLE A ————— SEGMENT 1
 \ \ INTO TABLE B ————— SEGMENT 1



————— LOAD 2 —————
 / / INTO TABLE A ————— SEGMENT 2
 \ \ INTO TABLE B ————— SEGMENT 2

Querying a table while it's being loaded

A user can query existing segments while another user loads new data into that table. New segments can be accessed *after* the load completes.



————— LOAD ————— INTO TABLE A ————— SEGMENT 2



————— SELECT ————— FROM TABLE A ————— SEGMENT 1

Locking during loads

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

If an operation fails

If a data load, index load, or merge operation fails during the copy or load phase, you can restart it or abort it. Restart is possible through *checkpointing*. When loading LOB data, however, restart is not supported. You must abort the load and start over.

If an operation fails *before* the copy phase, there's no checkpoint, so it's not possible to restart the operation. More than likely your control statements (LOAD and SMDEF in the SYSIN dataset) contain errors. After fixing the control statement errors, simply submit the operation again.

Checkpoints

During the copy phase, the client data loader takes a checkpoint after writing a certain number of megabytes to the temporary VRAM file. You specify this number of megabytes on the CHECKPOINT keyword of the LOAD control statement (see page 5-6). Note the following:

- The client data loader maintains a checkpoint record in a host checkpoint dataset.
- You initialize this checkpoint dataset on your host during installation (see page 2-6).
- If you run multiple loads in parallel, you must initialize and use different checkpoint datasets for each load.
- When a load completes successfully or when you abort a load, the client data loader resets the checkpoint record in the checkpoint dataset. You can re-use checkpoint datasets for subsequent loads.

During the load phase, the server data loader takes a checkpoint for each data extent when it reaches the maximum size. You can specify the maximum data extent size with the MAX_EXT_SIZE parameter in a subspace. If you omit that parameter, the server data loader uses a default value of 100 megabytes for LOB subsegment files and the values of the following StorHouse system parameters:

- SQL_MAX_EXT_DATA for table data files
- SQL_MAX_EXT_HASH for hash index files
- SQL_MAX_EXT_VAL for value index files

The server data loader maintains checkpoints in checkpoint files on StorHouse. When a load completes successfully or when you abort a load, the server data loader deletes the checkpoint files.

Restart

When you *restart* a data load, your load continues from the last checkpoint. In other words, a restart skips completed work and continues with remaining work.

If a load fails during the copy phase, the client data loader begins at the last checkpoint in the host checkpoint dataset. For instance, if CHECKPOINT=100 and the copy phase failed after transferring 100 megabytes but before transferring 200 megabytes, then the client data loader does not re-copy the first 100 megabytes.

If a load fails during the load phase, the server data loader begins at the last data extent. For instance, if the server data loader had completed writing two data extents for a table data file and the load failed while writing the third data extent, then a restart begins at the third data extent of the table data file.

When you restart an index load or a segment merge operation, the server data loader aborts the operation and then starts over, restarting after the last completed segment.

Abort

When you *abort* a data load, the client and server data loaders remove all checkpoints for that load. The server data loader also deletes and removes any partially written segment files on StorHouse. If you want to load the table again, all work starts at the beginning, just like a new load.

When you abort an index load, the server data loader deletes and removes the last in-progress segment. Any completed segments have already been committed.

When you abort a segment merge, the server data loader deletes and removes any partially created result segment. You cannot abort a merge operation that completed successfully because the server data loader automatically commits each result segment after creating it.

System table updates

A StorHouse engine updates metadata during loading operations. The updates differ depending on the type of operation.

Metadata updates for a data load operation

During a data load, a StorHouse engine:

- Obtains and increments the segment ID in the SYSTABLES system table
- Inserts index entries into applicable range index system tables
- Inserts rows into the SYSSTHFILES system table for each segment file
- Inserts the following into the SYSSTHSEGMENTS system table:
 - Table ID of the user table
 - Segment ID of the segment
 - Number of logical records in a table data file
 - Average number of logical records per page in a table data file
 - Date and time the load was committed
 - Segment tag

Metadata updates for a replace operation

For replaced, or invalidated, segments, a StorHouse engine inserts the following into the SYSSTHSEGMENTS system table:

- Flag indicating that the segment was replaced
- Date and time the segment was replaced

Metadata updates for an index load operation

For an index load operation, a StorHouse engine:

- Inserts rows into the SYSSTHFILES system table for each hash index file and value index file
- Inserts index entries into applicable range index system tables
- Updates the IDXCOMPRESS column in the SYSINDEXES system table to N for normal (index complete) only if all index entries for all segments of the table are created

Metadata updates for a merge operation

For coalesced segments, a StorHouse engine:

- Updates the segment ID and subsegment ID in the SYSSTHFILES system table for all LOB subsegments
- Performs the same updates as a data load for the result segment
- Performs the same updates as a replace operation for the input segments

Temporary VRAM file names

A temporary VRAM file is used during data load, index load, and merge operations. The client data loader uses the following format to name the file:

prefix.Ldddddd.Kdddddd.Nnnnnnn

where:

- *prefix* is the file name prefix as specified in the FNPREFIX keyword on the LOAD control statement (see page 5-7)
- *L*, *K*, and *N* are constants
- *dddddddddd* is the load ID generated by the client data loader, padded with leading zeros if necessary
- *nnnnn* is 00000 for the first or only LOAD control statement in a job step, and one larger for each subsequent LOAD

For example, STHLDR.TEMPF.L003434.K45081.N00000

- STHLDR.TEMPF is the prefix
- 00343445081 is the load ID (padded with leading zeros)
- 00000 indicates there was only one LOAD control statement in the job step

Installation

This chapter explains the requirements, procedures, and job steps to install the FileTek MVS Data Loader utility, also called the LDLSLDR utility.

Installation overview

You install the FileTek MVS Data Loader utility with the IBM System Modification Program Extended (SMP/E).

Note: Because the installation process is designed to use a separate Consolidated Software Index (CSI), you should not install this product in an SMP/E target or distribution zone that was used for installing other products.

Software function identifier

The FileTek MVS Data Loader utility software is distributed as an SMP/E function with a function management ID (FMID) in the format *LDLSvr_f* where:

- *L* is the user-definable first character SMP/E FMID name.
- *DLS* is a constant.
- *v* is the software version number.
- *r* is the software release number.
- *f* is the function identifier. Currently only the base function 0 is defined.

The current version is 1 and the release is 1, so the current FMID is LDLS110.

System requirements

The following are required to run the FileTek MVS Data Loader utility:

- SMP/E
- Available disk space equal to approximately thirteen 3380 cylinders
- FileTek host software base (FMID LSM1700)

Files on the distribution tape

The distribution tape for LDLS110 is a 3480 cartridge, standard label, with a volume serial number of LDL110. The following table lists the files on the tape.

File	Dataset name	Contents
1	SMPMCS	SMP/E Modification Control Statements
2	LDLS110.F1	LDLS110 SMP/E JCLIN
3	LDLS110.F2	LDLS110 load library
4	SAMPLES	Installation assist and sample JCL

Files 1 through 3 are used by the SMP/E installation procedures. File 4 contains members to assist in the installation process and sample JCL for FileTek MVS Data Loader utility production execution.

Installation procedure

Installing the FileTek MVS Data Loader utility is an 8-step procedure:

1. Load the SAMPLES dataset.
2. Allocate required datasets.
3. Customize the SMP/E JCL procedure.
4. Initialize SMP/E CSI.
5. Execute SMP/E RECEIVE.
6. Execute SMP/E APPLY.
7. Execute SMP/E ACCEPT.
8. Build the checkpoint dataset.

Step 1: Load the SAMPLES dataset

Use the sample JCL that follows to load file 4, SAMPLES. Supply a valid JOB card and substitute appropriate values for DSN, VOL, and UNIT. The <=== symbol identifies the JCL lines that require customization.

```
//SAMPLES      JOB ...                                <===
//LOAD         EXEC PGM=IEBCOPY
//SYSPRINT     DD  SYSOUT=*
//SYSUT1       DD  DSN=SAMPLES,VOL=SER=LDL110,UNIT=unit, <===
//              LABEL=4,DISP=OLD
//SYSUT2       DD  DSN=ldlindex.SAMPLES,                <===
//              UNIT=unit,VOL=SER=volser,                <===
//              SPACE=(TRK,(2,1,6)),DISP=(,CATLG,DELETE)
//SYSUT3       DD  UNIT=SYSDA,SPACE=(TRK,(15))
//SYSUT4       DD  UNIT=SYSDA,SPACE=(TRK,(15))
//SYSIN        DD  DUMMY
```

All subsequent installation steps refer to members in SAMPLES. These members are listed in the following table. You will use the members marked *Install* during

installation. You will use the members marked *Operations* for normal operations and for administration of the FileTek MVS Data Loader utility after installation.

Member name	When used	Description
INITCKPT	Install	Build and initialize the checkpoint dataset
LDRSMPE	Install	SMP JCL procedure
RUNLOAD	Operations	Sample execution JCL
SMPACCEPT	Install	SMP processing, ACCEPT step
SMPALLOC	Install	Dataset allocation
SMPAPPLY	Install	SMP processing, APPLY step
SMPDDDEF	Install	SMP processing, ADD DDDEF statements
SMPRECV	Install	SMP processing, RECEIVE step
SMPUCLIN	Install	SMP UCLIN procedure
SQLSAMP	Operations	Sample LOAD DATA statement

Step 2: Allocate required datasets

Edit SAMPLES member SMPALLOC to specify the appropriate dataset high-level indexes and the dataset units and volumes. The following table shows the datasets that are allocated and their respective sizes. All datasets except SMPCSI are allocated by blocks in the amounts shown in the table. The LDLS110 SMP/E CSI requests an allocation of 4 cylinders of 3380 (or equivalent) disk space.

Dataset	Description	Blocks required	Block size
ALDLLOAD	Loader distribution zone load library	315	6144
LDLLOAD	Loader target zone load library	315	6144
SMPMTS	Required SMP/E dataset	7	6160

Dataset	Description	Blocks required	Block size
SMPSTS	Required SMP/E dataset	7	6160
SMPPTS	Required SMP/E dataset	91	6160
SMPSCDS	Required SMP/E dataset	56	6160
SMPCSI	Required SMP/E dataset	4 cylinders of 3380 (or equivalent) disk space	

Be sure that the requested space is available on the volumes specified in the SMPALLOC job. Submit this job after completing the updates to SMPALLOC. This job must finish successfully before you can proceed with Step 4 to initialize the SMP/E CSI.

Step 3: Customize SMP/E JCL procedure

SAMPLES member LDRSMPE is a JCL procedure used for the LDLS110 SMP/E install steps. Customize this procedure by replacing substitution parameters with the values used in SMPALLOC. Then store the procedure in a system PROCLIB dataset or insert it in the RECEIVE, APPLY, and ACCEPT jobstreams as an instream PROC.

Step 4: Initialize SMP/E CSI

SAMPLES member SMPUCLIN contains the JCL and UCLIN necessary to initialize the SMP/E CSI. Edit SMPUCLIN to specify the correct CSI CLUSTER and ZONE names, TLIB prefix, and dataset high-level index. Submit the job after completing all updates.

SAMPLES member SMPDDDEF contains SMP/E DDDEF statements for all zones in the CSI. Edit this member to specify the CSI CLUSTER and ZONE names, the TLIB unit and volume, and the correct high-level dataset name prefix. This job facilitates use of the ISPF SMP/E dialogs and is optional. You can use ISPF SMP/E

dialogs instead of the LDRSMPE JCL procedure in batch for the LDLS110 installation.

Step 5: Execute SMP/E RECEIVE

Edit SAMPLES member SMPRECV and submit the updated jobstream to perform the SMP/E RECEIVE process. This job must complete with a return code of zero (0).

Step 6: Execute SMP/E APPLY

Edit SAMPLES member SMPAPPLY and submit the updated jobstream to perform the SMP/E APPLY process. This job must complete with a return code of zero (0).

Step 7: Execute SMP/E ACCEPT

Edit member SMPACCEPT and submit the updated jobstream to perform the SMP/E ACCEPT process. This job must complete with a return code of zero (0).

Step 8: Build the checkpoint dataset

The FileTek MVS Data Loader utility recovers from an aborted/failed run by recovering the operation state information from a *checkpoint dataset*. Each time you run the utility, it reads the checkpoint dataset and verifies that the state of the *checkpoint record* is consistent with the operation requested. That is, if you restart a load, the checkpoint record must indicate a prior failed or in-progress run. Similarly, you cannot submit a normal load operation when the checkpoint record indicates a prior failure. Therefore, normal production use of the utility requires a checkpoint dataset that contains a valid checkpoint record.

You can use the sample job INITCKPT to create a valid checkpoint dataset. This sample job allocates the dataset (one track) and then runs the FileTek MVS Data Loader utility with PARM='INIT'. The INIT job produces a return code of 4, which indicates that no data loading has taken place. This is to prevent use of this parameter in production jobs, because an INIT could destroy a valid checkpoint record and force a complete rerun rather than a restart.

Note: The sample JCL allocates the checkpoint dataset with DISP=OLD. You can use only one checkpoint dataset for one load job at a time. If you plan to run several loads concurrently, each load should have its own checkpoint dataset.

2

Installation

Installation procedure

Input data

This section answers questions about input datasets and input data records used for loading data into StorHouse user tables.

What's an input dataset?

An *input dataset* is a file that contains the input data records you're loading. This input dataset can reside on your host (disk or tape) or in a VRAM file already on StorHouse. Input datasets that reside on your host are called *host input datasets*.

What are input data records and data fields?

Typically, an *input data record* corresponds to a row in a user table, but it's possible that multiple input data records make up one row in a user table. Each input data record is composed of *data fields* that often correspond to values in columns of a user table. The maximum size of an input data record is 32,767 bytes.

How should you create a host input dataset?

How you create a host input dataset is site-specific. For example, one of your application programs may create it, or you might use a database extractor product. To StorHouse, *how* you create a host input dataset doesn't matter. What

does matter is that the host input dataset that you use to load data into StorHouse user tables must be a sequential dataset. A host input dataset can be a member of a partitioned dataset.

What record formats can you use?

Input data records can be in any fixed-length or variable-length format. There's one exception: Do not define input data records as Variable-Blocked-Spanned (VBS). The FileTek MVS Data Loader utility does not support the VBS record format.

Are there any considerations for using the host input dataset?

Consider using FREE=CLOSE for host input datasets that reside on tape. Because the load phase can be long, releasing resources (such as a tape drive) is helpful. If you do code FREE=CLOSE, then the tape drive(s) are released at the end of the copy phase.

Where do you specify which input dataset you're using?

The SYSREC DD statement in the execution JCL contains the name of the host input dataset for a load operation. You can specify multiple host input datasets by concatenating them to the SYSREC DD statement. You can use a different DDname (other than SYSREC) and then specify that DDname in the FROMDD keyword of a LOAD control statement, as described in Chapter 5, "Control statements."

If the input dataset is a VRAM file from a previous load or created by another program, then the SYREC DD statement is not required, but the FROMDD keyword with a null value on the LOAD control statement is required. You also specify the name of the input dataset you're using in the INFILE clause of the LOAD DATA statement, as described on page 4-38.

What's the difference between a column and a field in an input data record?

A data field in an input data record can be a column or a field.

- *Column* – contiguous data bytes that you load into a column in the target user table.
- *Field* – contiguous data bytes that you don't load into a user table. You use a field to assign a name to a portion of a record to be used in a condition, for instance, to indicate which records to load. You distinguish field names from column names by preceding them with a colon in the field_spec.

For example, assume you're loading data into a user table that you created with the following CREATE TABLE statement:

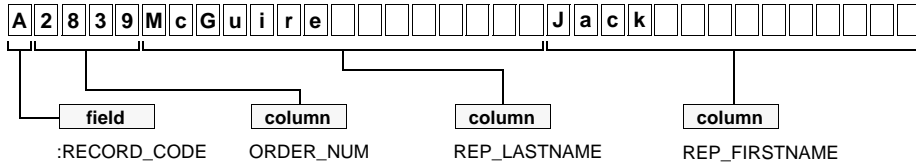
```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS95
```


3

Input data

What's the difference between a logical record and a physical record?

Now, assume you're loading this input data record into that user table:



The first data field—A (`:RECORD_CODE`)—is a field. You could use this field to select and load records that begin with a certain record code, like A. Fields are not loaded into user tables.

What's the difference between a logical record and a physical record?

One input data record in an input dataset is referred to as a *physical record*. A *logical record* is assembled from one or more physical records. Logical records can contain both fields and columns. In the following example, each physical record corresponds to one logical record:

Physical records			Logical records		
A	2839	McGuire	Jack	A	2839 McGuire Jack
B	2388	Cornflake	Sue	B	2388 Cornflake Sue

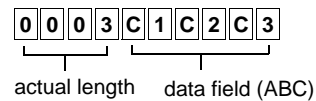
Now in the following example, two physical records actually make up one logical record:

Physical records				Logical records			
A	2839	McGuire	Jack	C	A	2839 McGuire Jack	120.00
B	2388	Cornflake	Sue	C	B	2388 Cornflake Sue	528.45

You can combine a fixed number of physical records into one logical record, or you can combine a varied number when physical records contain a *continuation field* with a *comparison value*. In the previous example, the C is the comparison value. In most cases, continuation fields are removed from physical records when the logical record is assembled. LOB records are physical records that are automatically included in the logical record.

Are there any considerations for VAR-type data?

When loading VARCHAR or VARBINARY data, each variable-length data field must be preceded by a two-byte SMALLINT field indicating the actual length of the data field. For instance, the following data field (hex, EBCDIC) has an actual length of 3:



How do you load LOB data?

With the FileTek MVS Data Loader utility, you can load LOB values two ways:

- When a LOB value fits in an input data record (the LOB value or the input data record does not exceed 32 KB), you can define that LOB value as a loader data type (for instance, VARCHAR or VARBINARY) and include it with the other input data in the data file. In this case, the following loader data types (source) are valid for BLOB and CLOB column (target) data types: BINARY, BINARY EXTERNAL, CHARACTER, VARBINARY, and VARCHAR.
- When a LOB value fits within or exceeds the record length, you can define it as a BLOB or CLOB data type. The LOB value then can span multiple records, but each record cannot exceed the maximum record length (32KB-1). A *LOB*

record is a BLOB or CLOB data field that consists of one or more physical records in the input data file. LOB records are automatically considered part of the logical record.

Note the following guidelines for including LOB data fields in the input dataset with other non-LOB data:

- Each LOB data field must start with a 64-bit length field followed by the data. The native values key affects the interpretation of the length.
- A LOB data field does not have to start at the beginning of a record. It can start in the same record as the non-LOB data.
- The last LOB data field must end at the end of a record.
- All other non-LOB data fields must still fit into a single record.
- LOB data fields must be placed after all non-LOB fields in the record.

See “Specifying a BLOB or CLOB data type” on page 4-107 for additional considerations about including LOB data in the input dataset.

What's delimited data?

One way to separate data fields in an input data record is to use *delimiters*—markers that follow data fields or enclose them. The server data loader trims delimiters from input data records; it doesn't load delimiters into user tables. Input data records with delimiters are called *delimited data*. There are two types of delimited data: terminated and enclosed.

Note: You can't use delimiters for the following data types: BINARY (and synonyms RAW and BYTE), DECIMAL (and synonym NUMERIC), DOUBLE, FLOAT, INTEGER, SMALLINT, VARBINARY (and synonyms VARRAW and VARBYTE), and VARCHAR.

Terminated data

Terminated data are data fields followed by a *termination delimiter*—any single character or one or more blank characters. For example, a termination delimiter could be a comma:

2	8	3	9	,	M	c	G	u	i	r	e	,	J	a	c	k	,	
2	3	8	8	,	C	o	r	n	f	l	a	k	e	,	S	u	e	,

Terminated data is read from the starting position of the data field up to, but not including, the termination delimiter. The end of the input data record will serve as the delimiter (if needed).

Enclosed data

Enclosed data are data fields preceded and followed by *enclosure delimiters*, such as parentheses:

(2	8	3	9)	(M	c	G	u	i	r	e)	(J	a	c	k)	
(2	3	8	8)	(C	o	r	n	f	l	a	k	e)	(S	u	e)

This type of data is read by skipping any characters until the first enclosure delimiter, then reading data until the second enclosure delimiter. Enclosure delimiters can be defined as optional, meaning that not all data fields have to be enclosed by those enclosure delimiters.

Are blank characters loaded?

Character-type data fields can contain blank characters, also called *spaces*, *whitespace*, or just *blanks*. *Leading blanks* are blanks at the beginning of a data

field. *Trailing blanks* are blanks at the end of a data field. Rules about the trimming of blanks follow.

- Blanks that are part of a data field that's enclosed by enclosure delimiters are not trimmed.
- Leading blanks may be trimmed from a character-type data field when optional enclosure delimiters are not present.
- Trailing blanks are not trimmed from character-type data fields.
- Leading and trailing blanks in VARCHAR data fields are not trimmed.

SYSSQL dataset

This chapter describes the format and provides examples of the LOAD DATA, LOAD INDEX, and MERGE statements. You include these statements in a SYSSQL dataset.

About the SYSSQL dataset

The host dataset named by the SYSSQL DD statement in the JCL can contain one LOAD DATA statement, one LOAD INDEX statement, or one MERGE statement. You can also include other SQL statements in this dataset. For example, you can include a CREATE TABLE statement to create a user table before you load it or include a CREATE INDEX statement to create a deferred index before you load it. You cannot use the FileTek MVS Data Loader utility to issue queries, so a SYSSQL dataset cannot contain SELECT statements.

The client data loader adds the contents of the SYSSQL dataset to the data stream that it writes to a StorHouse VRAM file. In a subsequent step, the server data loader reads the VRAM file to obtain the necessary information for the operation.

Character set of SYSSQL

The character set of the SYSSQL dataset is EBCDIC. Your input data can have a different character set, for instance, you can load ASCII data from an ANSI-format tape. If the character set of your input data is not EBCDIC, the server data loader automatically converts any character string literals supplied on a LOAD

DATA statement to the character set of the input data. See page 4-28 for an example.

SYSSQL guidelines

Guidelines for entering statements in a SYSSQL dataset are as follows:

- Only one LOAD DATA, LOAD INDEX, or MERGE statement is allowed in a SYSSQL dataset, and the statement must follow any StorHouse SQL statements.
- For compatibility, a LOAD DATA statement can contain DB2 and Oracle clauses (for example, Oracle OPTIONS clause) that are not defined in the StorHouse syntax. The FileTek MVS Data Loader utility accepts but does not process these clauses.
- Statements can span more than one line, and any new line can begin with any keyword.
- Character strings can be enclosed in single or double quotes, for example, 'A' or "A".
- Hexadecimal strings are preceded with X and enclosed in single or double quotes, for example, X'0001' or X"0001".
- Case is significant only in quoted strings.
- Spaces are significant only in quoted strings. For example, 'A' is a different value from 'A '.
- You can include comments by starting them with two dashes (--).
- Statements must end with a semicolon. The semicolon must appear at the end of a record after any comment.

Statement formats

This section contains the conventions and syntax of the LOAD DATA, LOAD INDEX, and MERGE statements.

Format conventions

The LOAD DATA, LOAD INDEX, and MERGE statements use the same format conventions as StorHouse SQL. Those conventions are:

Convention	Description
UPPERCASE	Uppercase terms indicate keywords that are part of the syntax. Type keywords in any case.
lowercase	Lowercase terms refer to grammar elements (like field_spec) and user-supplied values (like segment_tag). When supplying values, only quoted strings are case sensitive.
() , / * - ; : . + '	These characters are part of the syntax. Type them as shown.
{ }	Braces indicate that the item is required. When a list of items is enclosed in braces and separated by a vertical bar, you must choose one item.
[]	Brackets indicate that the item is optional.
	Vertical bar separates alternatives. You can specify one of the alternatives shown.
...	Ellipsis points indicate that you can repeat the part of the statement preceding them any number of times.

SQL identifiers

The SQL identifiers that you supply on a statement—like unquoted table, column, and field names—must follow these rules:

- Start with a letter
- Cannot exceed 32 characters
- Must be a contiguous string of characters (no blanks)
- Can consist of the characters a–z, A–Z, 0–9, _ (underscore)
- Cannot be an SQL reserved word
- Are case insensitive (you can type them in any case) unless delimited

You must delimit SQL identifiers that don't follow the rules. The delimiters are double quotes. For example:

- "95orders" (doesn't start with a letter)
- SUE."ORDER DETAILS" (contains a blank)
- "july\$" (contains a special character)
- "GROUP" (is a reserved word)

Note that:

- Period(s) used for qualified table names must be outside the quotes, like in

SUE."ORDER DETAILS"

- The preceding colon for a delimited field name in a field_spec must be outside the quotes. For example:

:"RECORD CODE" POSITION (1) CHAR

LOAD DATA

```
LOAD [ DATA ]
[ CHARACTERSET { cset_name | ccsid } ]
[ { INFILE | INDDN } { [ NOENVIRON ] (infile_list) | * | - } ]
[ load_options ]
{ into_table_spec }...
```

Argument	Format
cset_name	WE8EBCDIC500 WE8PC850 WE8ISO8859P1
ccsid	500 850 819
(infile_list)	sm_file_name [/group] [NOENVIRON] [, sm_file_name [/group] [NOENVIRON]]...
load_options	[{ DISCARDFILE DISCARDN } sth_file_name [/group]] [{ DISCARDS DISCARDMAX } num_discards] [CONCATENATE num_lines] [CONTINUEIF continueif_condition] [PRESERVE BLANKS] [SUBSPACE ROTATE] [ESCAPED BY [DELIMITER 'char' NONE]]
continueif_condition	{ [THIS NEXT] (position) LAST } operator { any_string BLANKS }
position	start_column [{ : - } end_column]
operator	= != ^= <> < > <= >=
any_string	string Xstring
into_table_spec	INTO TABLE [owner.] table_name [WHEN field_condition [{ AND OR } field_condition]...] [FIELDS fields_specs] [TRAILING [NULLCOLS]] [SAME SEGMENT] [DIFF[ERENT] SEGMENT] [SEGMENT segment_tag] [REPLACE SEGMENT [[owner.] table_name.] segment_tag] [[TABLE VALUE HASH LOB] SUBSPACE number]... [(field_spec [, field_spec]...)]

Argument	Format
field_condition	{ (position) column_name field_name } operator { any_string BLANKS }
fields_specs	[CHAR] [NULLFLAGS] [delimiter_spec] [NULLIF (EMPTY BLANK)] [DEFAULTIF (EMPTY BLANK)]
delimiter_spec	[TERMINATED [BY] { WHITESPACE 'char' X'hexbyte' }] [[OPTIONALLY] ENCLOSED [BY] { 'char' X'hexbyte' }] [AND { 'char' X'hexbyte' }]]
field_spec	{ :field_name column_name } data_spec
data_spec	RECNUM SEQUENCE (start_num [,increment]) SYSDATE CONSTANT any_value position_spec
any_value	any_string identifier n
position_spec	[POSITION (position * [+num])] [datatype_spec] [NULLIF field_condition] [DEFAULTIF field_condition]
datatype_spec	See page 4-86 for loader data types.

Note the following:

- The LOAD keyword and INTO TABLE clause are the only required arguments.
- You can specify load_options, into_table_spec, fields_specs, delimiter_spec, and position_spec clauses in any order.
- If you omit a field_spec list and a FIELDS clause, the server data loader generates a field_spec for every column in the named table using the CREATE TABLE data types and lengths. You can do this only when the input data is

relatively positioned and has the same order and same data types as the CREATE TABLE definition.

- If you omit a field_spec list but include a FIELDS NULLFLAGS clause, the server data loader generates a field_spec for every column in the named table using the CREATE TABLE data types and lengths.
- If you omit a field_spec list but include a FIELDS CHAR clause, the server data loader generates a field_spec for every column in the named table using the CHARACTER loader data type (with any associated delimiter_spec).

The following table lists the function of each clause and keyword in the LOAD DATA statement. See the listed page for more information.

To	Use	See page
Load data already on StorHouse	INFILE or INDDN	4-8
Collect discarded records	DISCARDFILE or DISCARDN	4-14
Limit the number of discarded records	DISCARDS or DISCARDMAX	4-17
Identify the character set of the input data	CHARACTERSET	4-18
Combine a fixed number of physical records into one logical record	CONCATENATE	4-19
Combine a varied number of physical records into one logical record	CONTINUEIF	4-21
Retain blank characters in input data	PRESERVE BLANKS	4-30
Rotate among subspaces	SUBSPACE ROTATE	4-32
Identify the name of the user table	INTO TABLE	4-38
Choose which records to load	WHEN	4-42
Generate field_specs, identify NULL flags, specify default delimiters and other defaults	FIELDS	4-50
Identify an escape character	ESCAPED BY	4-59

To	Use	See page
Load missing data fields with null values	TRAILING NULLCOLS	4-59
Load one or more segments at a time	SAME SEGMENT and DIFFERENT SEGMENT	4-63
Name a segment that may be replaced later	SEGMENT	4-66
Replace a segment	REPLACE SEGMENT	4-68
Select subspaces	SUBSPACE number	4-70
Load a record number into a column	RECNUM	4-81
Generate a sequence of values	SEQUENCE	4-81
Load the current date into a column	SYSDATE	4-82
Load a constant value into a column	CONSTANT	4-82
Specify the position of a data field	POSITION	4-83
Set a column to a null value	NULLIF	4-108
Set a column to the default value	DEFAULTIF	4-109

LOAD INDEX

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

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

MERGE

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

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

Loading data already on StorHouse

You can load data that's already on StorHouse, for instance:

- Data copied to a VRAM file during a previous successful load operation
- Discarded records in a discard file (which is a VRAM file)
- Data in one or more StorHouse VRAM files

Specify the INFILE (synonym INDDN) clause with a VRAM file name or a list of file names to load data already on StorHouse. In order to use the INFILE clause to load data already on StorHouse, you *must* include the FROMDD keyword with a null value on the LOAD control statement. You don't have to include the SYSREC DD statement on the JCL. Conversely, if you use or omit INFILE * or INFILE - to load data from a host input dataset, then the SYSREC DD statement is required.

You can use the INFILE clause in combination with the DISCARDFILE clause to collect discarded records while loading data in a host dataset or in a VRAM file.

See “Collecting discarded records in a discard file” on page 4-14 for more information about using the DISCARDFILE clause.

When loading data that’s already in a VRAM file on StorHouse, you must specify the name of that VRAM file in the INFILE clause. Your StorHouse system or database administrator can obtain a VRAM file name by using the StorHouse Command Language SHOW FILE command. You can also get the name of a VRAM file created during a previous load operation by looking at the SYSPRINT listings. The FileTek MVS Data Loader utility lists the names of these temporary VRAM files in message LDL689I, for example:

```
DID NOT DELETE TEMP SM FILE STHLDR.TEMPF.L003434.K45081.N00000
```

Note: Message LDL689I is generated only when the TEMP_FILE keyword on the LOAD control statement is KEEP.

See “Temporary VRAM file names” on page 1-39 for more information about the file name format of temporary VRAM files created during load operations. Note that the prefix—STHLDR.TEMPF—is the default prefix. Your prefix may differ if you specified one at the FNPREFIX keyword on the LOAD control statement.

Format of INFILE clause

```
{ INFILE | INDDN } { [ NOENVIRON ] (infile_list) | * | - }
```

where infile_list is:

```
sm_file_name [ /group ] [ NOENVIRON ] [, sm_file_name [ /group ]  
[ NOENVIRON ] ]...
```


Argument	Description
NOENVIRON	(optional) Keyword to use when loading discarded records or data in a VRAM file created by another application program. If specifying a list of files, you can place this keyword before the list if it applies to all files in the list. Otherwise, place the keyword after any applicable file name and group.
sm_file_name	(required) Name of the StorHouse VRAM file that contains the data you are loading. If specifying a list of files, enclose the list in parentheses and use a comma to separate each file name and group. Parentheses are not required when specifying a single file.
/group	(optional) Name of the StorHouse file access group to which the VRAM file belongs. You can omit the group name if it is the default group of the StorHouse account ID you use to log in the StorHouse FTP server.
* or -	(optional) Argument for loading data from a host input dataset (DDname SYSREC) instead of a VRAM file that's already on StorHouse. This is the default if you omit the INFILE clause.

Example INFILE clauses

This section contains example INFILE clauses.

To load data from a previous load operation

You can load data in a VRAM file created during a previous successful load operation by using the INFILE clause. In order to do this, the TEMP_FILE keyword on the LOAD control statement from the previous load operation must be KEEP; otherwise, the VRAM file is deleted after the successful completion of the load operation.

When using INFILE to load data from a previous load operation, the LOAD DATA statement from the previous load operation is ignored. For this load, include all of the necessary clauses and field_specs in the LOAD DATA statement.

For example, assume that yesterday you loaded a user table, and now today, you want to load a different user table with some of the same logical records that were copied to the VRAM file called `STHLDR.TEMPF.L003434.K45081.N00000`. You would specify this INFILE clause:

```
INFILE STHLDR.TEMPF.L003434.K45081.N00000
```

Now if the VRAM file is not in the default group for the account ID at the ACCT keyword on the SMDEF control statement, then you must also specify the StorHouse group name. For example, if this VRAM file belongs to group `STH`, which isn't your default group, then you would specify this INFILE clause with the group name:

```
INFILE STHLDR.TEMPF.L003434.K45081.N00000/STH
```

To load data from any other VRAM file

You can load data from a VRAM file created by any application program interface (API) program that can write to StorHouse. In this case, however, you must use the `NOENVIRON` keyword in the INFILE clause. Additionally, if the VRAM file is not in the default group of the StorHouse account ID on the SMDEF control statement, then you must specify the StorHouse group name.

For example, assume you're loading data in a StorHouse VRAM file called `SEP2195` in a group called `ATM`. You would use this INFILE clause if `ATM` is your default group:

```
INFILE SEP2195 NOENVIRON
```

Or you would use this INFILE clause if `ATM` is not your default group:

```
INFILE SEP2195/ATM NOENVIRON
```


To load data from multiple VRAM files

You can specify a list of VRAM files to load data from multiple files into a table. Enclose the list in parentheses and use a comma to separate each file name. You can place the NOENVIRON keyword before the file list if it applies to all files. For example:

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

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

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

To load discarded records

You can load discarded records in a discard file by using the INFILE clause with the NOENVIRON keyword and by specifying the name of the discard file. If the discard file is not in the default group of the StorHouse account ID provided for the ACCT keyword on the SMDEF control statement, then you must also specify the StorHouse group name after the discard file name.

For instance, to load discarded records in a discard file called FILE1 in group STH, you would use this INFILE clause if STH is your default group:

```
INFILE FILE1 NOENVIRON
```

Or you would use this INFILE clause if STH is not your default group:

```
INFILE FILE1/STH NOENVIRON
```

Now if you also wanted to collect any discarded records generated while loading these discarded records, you would use the DISCARDFILE clause and specify the name of the discard file to collect any new discarded records. For example, if you

wanted to load discarded records in FILE1 and collect discarded records in FILE2, you would specify these clauses:

```
INFILE FILE1 NOENVIRON  
DISCARDFILE FILE2
```

To load data from a host input dataset and collect discarded records

You can load data from a host input dataset and collect discarded records by using INFILE * and specifying a DISCARDFILE clause. See “Collecting discarded records in a discard file” on page 4-14 for more information about the DISCARDFILE clause.

For example, assume you’re loading data from a host input dataset named by the SYSREC DD statement on the JCL, and you want to collect discarded records in a discard file called FILE1 in group STH. You would use these clauses if STH is your default group:

```
INFILE *  
DISCARDFILE FILE1
```

Or you would specify these clauses if STH is not your default group:

```
INFILE *  
DISCARDFILE FILE1/STH
```

Collecting discarded records in a discard file

Discarded records are logical records that do not meet the selection criteria in a WHEN clause. If you’re using a WHEN clause and you want to collect discarded records in a discard file, then use the DISCARDFILE (synonym DISCARDDN) clause.

If you're loading data from a host input dataset and want to collect discarded records, then use `INFILE *` or `INDDN -` with the `DISCARDFILE` clause. If you're loading data from a VRAM file on StorHouse, then specify `INFILE` with the VRAM file name, followed by the `DISCARDFILE` clause. See "Identifying the user table to load" on page 4-38 for more information about the `INFILE` clause.

Note the following:

- A discard file is a VRAM file on StorHouse. Someone must create this file on StorHouse before you can collect discarded records. A load fails if the discard file does not exist on StorHouse.
- You must use your own API program to view discarded records in a discard file or to determine which records were discarded.
- You discard logical records, but the records in the discard file appear as a set of physical records (the original physical records that constituted the discarded logical record).
- If the input data is loaded from a host input dataset, then discarded records are in blocked format. If the input data is in a StorHouse VRAM file, the discarded records are blocked if the VRAM file data was blocked.
- If you forget to include a `DISCARDFILE` clause and any logical records do not satisfy the selection criteria of a `WHEN` clause, you'll receive a warning message and the discarded records will not be saved.
- If you specify the same discard file name for different loads, then new discarded records are appended to existing discarded records in that discard file. Discarded records are not overwritten.
- A `LOAD DATA` statement contains only one `DISCARDFILE` clause; therefore, all discarded records are placed into one discard file no matter how many `INTO TABLE` clauses or `WHEN` clauses you use.

- Whenever discarded records are created, the server data loader issues an informational message indicating the number of records discarded.
- If you want to set a maximum number of discarded records, use the DISCARDS clause (see page 4-17).

Caution: If a load fails and you restart it, the discard results won't be reliable because the discard file may contain duplicate records.

Format of DISCARDFILE clause

{ DISCARDFILE | DISCARDDN } sth_file_name [/group]

Argument	Description
sth_file_name	(required) Name of the StorHouse VRAM file to contain discarded records during the load.
/group	(optional) Name of the StorHouse group to which the discard file belongs. If the discard file is not in the default group of the StorHouse account ID supplied on the SMDEF control statement, then you must also specify the StorHouse group name.

Example DISCARDFILE clause

To collect discarded records in a discard file called FILE1 and a group called STH, you would use one of these clauses if STH is your default group:

```
DISCARDFILE FILE1
DISCARDDN FILE1
```

Or you would specify one of these clauses if STH is not your default group:

```
DISCARDFILE FILE1/STH
DISCARDDN FILE1/STH
```


Limiting the number of discarded records

You can limit the number of discarded records by using the DISCARDS (synonym DISCARDMAX) clause. The number of discarded records as well as the records themselves are logical records.

Note the following:

- A load fails when the number of discarded records exceeds this limit.
- If you include a DISCARDFILE clause but omit a DISCARDS clause, then there's no limit to the number of discarded records.
- If you include a DISCARDS clause but omit a DISCARDFILE clause, then discarded records will be limited but not saved.

Format of DISCARDS clause

{ DISCARDS | DISCARDMAX } num_discards

Argument	Description
num_discards	(required) Maximum number of discarded records for each SYSREC dataset. Valid values are 0 (no limit) through 2147483646.

Example DISCARDS clause

To limit the number of discarded records to 500, specify one of these clauses:

DISCARDS 500
DISCARDMAX 500

Specifying the character set of the input data

You can specify the character set of the input data. When you do, any character-based data and padding are converted to this character set. StorHouse supports ISO, EBCDIC, and PC character sets. The default is EBCDIC.

You have four options for specifying the character set of the input data you're loading. You can:

- Take the default (EBCDIC)
- Provide a value for the CCSID keyword on a LOAD control statement (described on page 5-6)
- Use the CHARACTERSET clause in a LOAD DATA statement
- Use the CHARSET keyword in a datatype_spec for an individual data field of type CHAR , CLOB, or any EXTERNAL (described on page 4-105)

Note the following rules for specifying a character set:

- If you specify the character set for *both* the CHARACTERSET clause and the CCSID keyword, then the CHARACTERSET clause overrides the CCSID keyword.
- If you don't include the CHARACTERSET clause, CCSID keyword, or CHARSET keyword, then the default character set is EBCDIC.
- The CHARSET value overrides both the CHARACTERSET clause and the CCSID keyword for an individual data field.

Format of CHARACTERSET clause

CHARACTERSET { cset_name | ccsid }

Argument	Description
cset_name	(required if no ccsid is specified) Name of the character set. Valid values: <ul style="list-style-type: none">■ WE8EBCDIC500 for EBCDIC character set■ WE8ISO8859P1 for ISO 8859-1 character set■ WE8PC850 for PC character set
ccsid	(required if no cset_name is specified) CCSID of the character set. Valid values: <ul style="list-style-type: none">■ 500 for EBCDIC character set■ 819 for ISO 8859-1 character set■ 850 for PC character set

Example CHARACTERSET clause

Assume the input data is the PC character set. You could specify one of these CHARACTERSET clauses in a LOAD DATA statement:

```
CHARACTERSET 850
CHARACTERSET WE8PC850
```

Concatenating a fixed number of physical records into a logical record

You can create logical records from the *same* number of physical records by using the CONCATENATE clause. For example, when every pair of data records corresponds to a row in a user table, you could use this clause to combine the first

pair of physical records into one logical record, the second pair of physical records into another logical record, and so on. If the number of physical records to be combined varies, then use the CONTINUEIF clause described on page 4-21 instead of the CONCATENATE clause.

Note: The CONCATENATE clause applies to *all* into_table_specs in a LOAD DATA statement. The CONCATENATE clause does not apply to LOB records in the input dataset.

Format of CONCATENATE clause

CONCATENATE num_lines

Argument	Description
num_lines	(required) Number of physical records to combine. This number must be greater than 1.

Example CONCATENATE clause

Assume that two physical records in an input dataset make up one row in a user table. You would use this CONCATENATE clause to combine records 1 and 2, records 3 and 4, records 5 and 6, and so on:

CONCATENATE 2

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4

Then these are the logical records:

1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4

Combining a varied number of physical records into a logical record

Use the CONTINUEIF clause when the number of physical records to be combined varies, or to combine physical records that satisfy a certain *condition*. A condition is true when a continuation field contains or does not contain a *comparison value* in a specified *location*.

For example, with a CONTINUEIF clause you can combine physical records that contain an * (comparison value) in column 80 (location). Or you can combine physical records that don't contain blanks (comparison value) in columns 1 and 2 (location).

Note: The CONTINUEIF clause applies to *all* into_table_specs in a LOAD DATA statement. The CONTINUEIF clause does not apply to LOB records in the input dataset.

Format of CONTINUEIF clause

CONTINUEIF continueif_condition

where continueif_condition:

{ [THIS | NEXT] (position) | LAST } operator { any_string | BLANKS }

Argument	Description
THIS	(optional and the default) If the condition is true in the current record, then append the next physical record to it, continuing until the condition is false. If the condition is false in the current record, then it is the last physical record of the current logical record. The continuation field is removed from the physical record.
NEXT	(optional) If the condition is true in the next record, then append it to the previous (current) physical record. If the condition is false in the next record, then the current physical record is the last physical record of the current logical record. The continuation field is removed from the physical record.
(position)	<p>Starting (required) and ending (optional) column numbers of the continuation field in the physical record. The format is:</p> <pre>start_column [{ : - } end_column]</pre> <p>For example (1:2) or (1-2). The position is required if you specify THIS or NEXT. The first position in a record is 1. This range of columns is removed from every physical record when the logical records are assembled. This is the only place where you'll specify column numbers in the physical record instead of the logical record.</p>
LAST	(required if you omit (position)) If the condition is true in the last non-blank character(s) in the current physical record, then append the next physical record to it, continuing until the condition is false. If the condition is false in the current record, then the current record is the last physical record of the current logical record. The continuation field is <i>not</i> removed from the physical record; therefore, it remains in the logical record.
operator	<p>(required) Comparison operator. Specify one of the following:</p> <p>= (equal), != or ^= or <> (not equal), < (less than), > (greater than), <= (less than or equal), >= (greater than or equal)</p>

Argument	Description
any_string	(required if you omit BLANKS) Comparison value. The format is: string Xstring
string	String of characters enclosed in single or double quotes. Use string for continuation fields containing character data. This character string must match the case of the input data.
Xstring	String of hex digits preceded with X and enclosed in single or double quotes, for example, x'01ff'. Use X string for continuation fields containing binary data.
BLANKS	(required if you omit any_string) Keyword to specify a blank as the comparison value.

Example CONTINUEIF clauses

This section contains example CONTINUEIF clauses.

To combine the current physical record with the next one

You can combine the current physical record with the next one by using the THIS keyword in a CONTINUEIF clause. When you use CONTINUEIF THIS, the continuation field is removed from every physical record before the logical records are assembled. Note that THIS is the default if you do not specify NEXT, LAST, or THIS.

For example, assume that the comparison value is an * located in column 1. You would use one of these CONTINUEIF clauses:

```
CONTINUEIF THIS (1) = '*'
```

or

```
CONTINUEIF (1) = '*'
```

If record 1 contains an asterisk in column 1, then record 2 will be combined with record 1. If record 2 also contains an asterisk in column 1, then record 3 will be

combined with records 1 and 2. If record 2 doesn't contain an asterisk in column 1, it is still combined with record 1, but record 3 starts a new logical record.

So if these are the physical records:

*	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2
*	3	3	3	3	3	3	3	3	3	3
*	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2									
3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5

Notice that the continuation field (column 1) is removed from all physical records. The * and blanks are not part of the logical records.

To combine the next physical record with the previous one

You can combine the next physical record with the previous one by using the NEXT keyword in a CONTINUEIF clause. When you use CONTINUEIF NEXT, the continuation field is removed from every physical record before the logical records are assembled.

For example, assume that the comparison value is ABC located in columns 5, 6, and 7. You would use this CONTINUEIF NEXT clause:

```
CONTINUEIF NEXT (5:7) = 'ABC'
```

If record 2 contains ABC in columns 5, 6, and 7, then record 2 will be appended to record 1. If record 2 does not contain the comparison value in the indicated columns, then it begins a new logical record.

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	A	B	C	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	A	B	C	5	5	5

Then these are the assembled logical records:

1	1	1	1	1	1	1			
2	2	2	2	2	2	2	3	3	3
4	4	4	4	4	4	4	5	5	5

Notice that the continuation field (columns 5 through 7) is removed from all physical records.

To use the last non-blank data column as the comparison value

The LAST keyword in a CONTINUEIF clause is similar to the THIS keyword in that you use it to combine the current physical record with the next one. LAST differs from THIS as follows:

- The test is *always* made against the last non-blank character(s).
- You don't specify starting and ending column numbers to identify the location of the continuation field.
- The continuation field is not removed from the physical records.

For instance, assume that the comparison value is a comma located in the last non-blank data column. You would use this CONTINUEIF LAST clause:

CONTINUEIF LAST = ','

So if these are the physical records:

```

1 1 1 1 1 1 1 1 1 1 ,
2 2 2 2 2 2 2 2 2 2 
3 3 3 3 3 3 3 3 3 3 ,
4 4 4 4 4 4 4 4 4 4 ,
5 5 5 5 5 5 5 5 5 5 

```

Then these are the assembled logical records:

```

1 1 1 1 1 1 1 1 1 1 , 2 2 2 2 2 2 2 2 2 2 
3 3 3 3 3 3 3 3 3 3 , 4 4 4 4 4 4 4 4 4 4 , 5 5 5 5 5 5 5 5 5 5 

```

Notice that the continuation field is *not* removed from the physical records. The commas and blanks are part of the logical records.

To specify the starting column number of a continuation field

You must specify a starting column number with the THIS or NEXT keyword to identify the starting location of the continuation field in a physical record. Enclose the column number in parentheses. Note that column numbers start with 1. When you specify only a starting column number, the length of the continuation field is equal to the length of the comparison value that you provide.

For example, assume the comparison value is the letter C, located in column 79. You would use this CONTINUEIF clause:

```
CONTINUEIF THIS (79) = 'C'
```

In this example, the length of the continuation field is 1 because the comparison value is the letter C. If the current physical record contains the letter C in column 79, then the next physical record is combined with it.

To specify starting and ending column numbers of a continuation field

You can specify both the starting and ending column numbers with the THIS or NEXT keyword to identify the location of the continuation field in the physical record. Enclose the column numbers in parentheses, and use a colon or dash to separate the starting column number from the ending column number.

For example, assume the comparison value is ABC located in columns 5, 6, and 7. You want to combine the next physical record with the previous one, so you would specify the following CONTINUEIF clause:

```
CONTINUEIF NEXT (5:7) = 'ABC'
```

If the comparison value is shorter than the length defined by the starting and ending column numbers, then the server data loader pads the value on the right with blanks (if a character string) or zeros (if hex digits). For example, if

```
CONTINUEIF NEXT (5:7) = 'AB'
```

then the actual selection criteria is 'AB ' (blank padded on the right).

If the comparison value is longer than the length defined by the starting and ending column numbers, then the server data loader trims the value on the right. For instance, if

```
CONTINUEIF NEXT (5:7) = 'ABCD'
```

then the D is trimmed and not considered part of the comparison value. You will receive a warning message when any non-blank characters or non-zero bytes are trimmed.

To use a character string as a comparison value

If the type of data in the continuation field is character, then specify the comparison value as a character string enclosed in single or double quotes. The character string *must* match the case (UPPER, lower, or Mixed) of the input data.

For instance, if the comparison value is the letter c, located in column 79 of the physical record, then you would specify the comparison value as a character string:

```
CONTINUEIF (79) = 'c'
```

If the value of the CCSID keyword on the LOAD control statement differs from EBCDIC (which is the character set of the SYSSQL dataset), then all comparison values that are character strings are automatically converted by the server data loader to the CCSID of the compared data field.

For example, suppose you're loading ASCII data from an ANSI-format tape. You can load this data directly, without conversion, by specifying an ASCII CCSID (819) through the CCSID keyword on the LOAD control statement. However, SYSSQL is always EBCDIC (CCSID 500). The server data loader will convert any character constants in SYSSQL to CCSID 819, so comparisons will work as expected.

To use a hex string as a comparison value

If the type of data in the continuation field is binary, then specify the comparison value as an even number of hex digits preceded with X and enclosed in single or double quotes.

For example, if the comparison value is the number 1, data type SMALLINT, located in column 1 of the physical record, then you would specify this comparison value as a string of hex digits:

```
CONTINUEIF (1:2) = X'0001'
```


To use blank characters as a comparison value

You can specify one or more blank characters as a comparison value by using the **BLANKS** keyword in the **CONTINUEIF** clause. You can use **BLANKS** with the **THIS** and **NEXT** keywords; **LAST** tests non-blank data only. The continuation field is removed from every physical record before the logical records are assembled.

Note: The default length of **BLANKS** is 1; so, if you specify a starting column only, the length of the continuation field is one blank.

For example, to combine the next physical record with the current one whenever column 10 and column 11 contain a blank character, then you would use this **CONTINUEIF** clause:

CONTINUEIF THIS (10:11) = BLANKS

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1		
2	2	2	2	2	2	2	2	2	2		
3	3	3	3	3	3	3	3	3	3		
4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5		
6	6	6	6	6	6	6	6	6	6	6	6

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1			
2	2	2	2	2	2	2	2	2	3	3	3
5	5	5	5	5	5	5	5	5	6	6	6

Notice that the continuation field (columns 10 and 11) is removed from all physical records. Also note that record 1 does not contain a blank character in column 10; therefore, the condition is false and record 2 is not appended to record 1.

To use a not equal comparison operator

You can specify that a continuation field *not equal* a specific comparison value by using any one of these “not equal” comparison operators in a CONTINUEIF clause: != or ^= or <>

For instance, to combine the next physical record with the current one whenever column 10 *does not* contain a blank character, then you would use this CONTINUEIF clause:

CONTINUEIF THIS (10) <> BLANKS

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	
3	3	3	3	3	3	3	3	3	
4	4	4	4	4	4	4	4	4	
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3								
4	4	4	4	4	4	4	4	4								
5	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6

Preserving blanks

The server data loader automatically trims leading blanks from a delimited data field that’s supposed to be enclosed but isn’t. If the previous data field is not

TERMINATED BY WHITESPACE, you can retain the leading blanks by using the PRESERVE BLANKS clause.

Note: Blanks within enclosure delimiters are always preserved.

Before reading this section, it may be helpful to have a basic understanding of delimited data and blanks. Consider reading the following sections first:

- “What’s delimited data?” (see page 3-6)
- “Are blank characters loaded?” (see page 3-7)
- “Generating field_specs, identifying NULL flags, specifying default delimiters and other defaults” (see page 4-50)

Format of PRESERVE BLANKS clause

PRESERVE BLANKS

Example PRESERVE BLANKS clause

Suppose you specified this FIELDS clause:

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ''

But in this input data record, the optional enclosure delimiter (double quote) isn't present in the first two data fields:

2839,McGuire,"Jack",

Without the PRESERVE BLANKS clause, the leading blanks would be trimmed and the following data fields would be loaded:

2839McGuireJack

2839	McGuire	Jack
data field 1	data field 2	data field 3

With the PRESERVE BLANKS clause, the leading blanks would be retained and the following data fields would be loaded:

	2	8	3	9		M	c	G	u	i	r	e	J	a	c	k	
data field 1					data field 2					data field 3							

Rotating among subspaces

You can rotate user table components (table data, value indexes, hash indexes, and LOB data) among subspaces by using the SUBSPACE ROTATE clause. You might do this if you're loading multiple segments of a table at a time and do not need to select certain subspaces for components. Rotation is not necessary when loading one segment or when a user tablespace contains one subspace for all component types or one subspace for each component type. You use SUBSPACE ROTATE with multiple INTO TABLE clauses and DIFFERENT SEGMENT clauses. You can also use SUBSPACE ROTATE to rotate among subspaces for successive result segments of a LOAD INDEX or MERGE operation.

Note the following:

- You must specify the SUBSPACE ROTATE clause before the INTO TABLE clauses of the LOAD DATA and MERGE statements.
- You cannot use both a SUBSPACE ROTATE and a SUBSPACE number clause in a LOAD DATA, LOAD INDEX, or MERGE statement. Only one of these clauses is allowed.
- If you omit both the SUBSPACE ROTATE and the SUBSPACE number clause, then the server data loader uses the lowest-numbered subspace that allows the component type.

- Rotation of subspaces occurs independently for each component type, that is, the server data loader rotates among applicable subspaces for table data, for hash indexes, for value indexes, and for LOB data.
- If an INTO TABLE clause specifies (or defaults to) SAME SEGMENT, it shares the subspace of the most recent INTO TABLE clause for the same table.

For example, in the following LOAD DATA statement, the second INTO TABLE clause shares the same subspace as the first INTO TABLE clause, but the third INTO TABLE clause uses the next subspace.

```
LOAD
SUBSPACE ROTATE

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '1/1/2000'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '1/31/2000'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '2/1/2000'
DIFF SEGMENT;
```

- If a user table has multiple indexes (for instance, two hash indexes), and those indexes are assigned to the same user tablespace, each *index* does not rotate among subspaces but rather both index *files* (for instance, both hash index files) for an INTO TABLE clause use the same subspace.
- When indexes for a user table are assigned to different user tablespaces, each user tablespace can contain a different number of subspaces for each index type. For example, one user tablespace may contain two subspaces for hash indexes and three subspaces for value indexes, while the other user tablespace may contain four subspaces for hash indexes and five subspaces for value indexes.

Format of SUBSPACE ROTATE clause

SUBSPACE ROTATE

Example SUBSPACE ROTATE clauses

This section contains example SUBSPACE ROTATE clauses.

To rotate among subspaces in a user tablespace

Assume you're loading two segments. The user table has a hash index and a value index assigned to the same user tablespace as the table. The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	blank (all component types)
2	T (table data only)
3	V (value indexes only)

The following LOAD DATA statement creates two table data files, two hash index files, and two value index files for the ATM.TRANSACTIONS table:

```
LOAD
SUBSPACE ROTATE

INTO TABLE ATM.TRANSACTIONS
WHEN (1) = 'A'

INTO TABLE ATM.TRANSACTIONS
WHEN (1) = 'B'
DIFF SEGMENT;
```


For the first INTO TABLE clause (segment 1), the server data loader uses subspace 1 for the table data file, hash index file, and value index file (because subspace 1 is the lowest-numbered subspace that allows all component types). For the second INTO TABLE clause (segment 2), the server data loader uses the following subspaces:

- Subspace 2 for the table data file (because it's the next subspace that allows table data)
- Subspace 1 for the hash index file (because it's the only subspace that allows hash indexes)
- Subspace 3 for the value index file (because it's the next subspace that allows value indexes)

To rotate among subspaces in multiple user tablespaces

Assume you're loading two segments. The user table has two hash indexes, two value indexes, and two LOB columns (to be stored out-of-line). The table, hash index 1, and value index 1 are assigned to user tablespace 1. Hash index 2, value index 2, and both LOB columns are assigned to user tablespace 2.

User tablespace 1 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	H (hash indexes only)
4	H (hash indexes only)
5	V (value indexes only)
6	V (value indexes only)

User tablespace 2 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	H (hash indexes only)
2	V (value indexes only)
3	L (LOB data only)
4	L (LOB data only)

The following LOAD DATA statement creates two table data files, four hash index files, four value index files, and two LOB subsegment files for the POS.TRANSACTIONS table:

```
LOAD
SUBSPACE ROTATE

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '1/1/2000'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '2/1/2000'
DIFF SEGMENT;
```

For the first INTO TABLE clause (segment 1), the server data loader uses the following subspaces:

- Subspace 1 in user tablespace 1 for the table data file (because it's the lowest-numbered subspace that allows table data)
- Subspace 3 in user tablespace 1 for the hash index file of hash index 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 5 in user tablespace 1 for the value index file of value index 1 (because it's the lowest-numbered subspace that allows value indexes)

- Subspace 1 in user tablespace 2 for the hash index file of hash index 2 (because it's the only subspace that allows hash indexes)
- Subspace 2 in user tablespace 2 for the value index file of value index 2 (because it's the only subspace that allows value indexes)
- Subspace 3 in user tablespace 2 for both LOB subsegment files (because it's the lowest-numbered subspace that allows LOB data)

For the second INTO TABLE clause (segment 2), the server data loader uses the following subspaces:

- Subspace 2 in user tablespace 1 for the table data file (because it's the next subspace that allows table data)
- Subspace 4 in user tablespace 1 for the hash index file of hash index 1 (because it's the next subspace that allows hash indexes)
- Subspace 6 in user tablespace 1 for the value index file of value index 1 (because it's the next subspace that allows value indexes)
- Subspace 1 in user tablespace 2 for the hash index file of hash index 2 (because it's the only subspace that allows hash indexes)
- Subspace 2 in user tablespace 2 for the value index file of value index 2 (because it's the only subspace that allows value indexes)
- Subspace 4 in user tablespace 2 for both LOB subsegment files (because it's the next subspace that allows LOB data)

Identifying the user table to load

An INTO TABLE clause specifies the name of the user table you are loading. You can:

- Provide the fully qualified table name (with the owner name)
- Omit the owner name if the ACCT keyword on the SMDEF control statement is the owner
- Use a symbolic variable to substitute an owner name, table name, or both

One INTO TABLE clause is required for each user table. For instance, to load two user tables with the same data, you would specify two INTO TABLE clauses. The maximum number of INTO TABLE clauses you can include in a LOAD DATA statement depends on the value of the StorHouse SQL_LDR_MAXINTO system parameter set for your organization. For more information about this system parameter, refer to the *StorHouse Database Administration Guide*.

Caution: Your load will fail if the number of INTO TABLE clauses exceeds the maximum number.

Format of INTO TABLE clause

INTO TABLE [owner.] table_name

Argument	Description
owner.	(optional) Owner of the user table. You can omit the owner name if the ACCT keyword on the SMDEF control statement is the owner, or you can use a symbolic variable to substitute the owner name in the LOAD control statement or the EXEC statement of the JCL.
table_name	(required) Name assigned to the user table on the CREATE TABLE or CREATE SYNONYM statement. A table name can be a synonym but it cannot be a view. You can use a symbolic variable to substitute a full or partial table name in the LOAD control statement or the EXEC statement of the JCL.

Example INTO TABLE clauses

This section contains example INTO TABLE clauses.

To use the fully qualified table name

The format of a fully qualified StorHouse user table name is:

owner.table_name

where owner is the 1–32 character name of the owner of the user table, and table_name is the 1–32 character table name assigned to the user table when it was created with the CREATE TABLE or CREATE SYNONYM statement.

For example, assume your account ID is JULIETTE and you're loading a user table called ORDERS owned by JACK. You would specify the fully qualified user table name as follows:

INTO TABLE JACK.ORDERS

To omit the owner name

If you omit the owner name in the two-part table name, a StorHouse engine assumes that the owner is the account ID supplied for the ACCT keyword on the SMDEF control statement. The engine checks that this account ID is the owner of the user table or has INSERT privilege on the user table.

For instance, if the ACCT keyword on the SMDEF control statement is JACK, then you can identify the user table name without the owner name, as follows:

INTO TABLE ORDERS

Note: The maximum length of the account ID in the ACCT keyword is 12 characters, but for DB2 users the maximum length is 8 characters. If an owner name contains more than 12 characters, you must use the fully qualified table name (provide the owner name).

To use a symbolic variable to substitute an owner name, table name, or both

Rather than providing the owner name or table name or both in an INTO TABLE clause, you can use a *symbolic variable*—for instance, &&0—and supply a *substitution string* in the Pn keyword of the LOAD control statement or in the PARM of the EXEC statement of the JCL. You can also substitute part of a table name. In fact, you can use a symbolic variable to replace *any* character string in a SYSSQL dataset. You can include up to nine symbolic variables—&&0 through &&8—in a SYSSQL dataset. Chapter 5, “Control statements,” describes the Pn keyword in detail. Some example substitutions follow.

To substitute an owner name. To substitute the owner name in a user table called ORDERS, you would create an INTO TABLE clause as follows:

INTO TABLE &&0.ORDERS

Then you would supply the substitution string for the Pn keyword in the LOAD control statement. For example, if the owner is JACK, supply this substitution string:

```
LOAD P0=JACK
```

Or you would supply the substitution string in the PARM of the EXEC statement as follows:

```
LOADER EXEC PGM=LDLSLDR,PARM='LOAD P0=JACK'
```

In this example, the complete name is JACK.ORDERS.

To substitute both an owner name and a table name. To substitute both the owner name (JACK) and the table name (ORDERS) in the SYSSQL dataset, you would create an INTO TABLE clause and Pn keyword as follows:

```
INTO TABLE &&0.&&1  
LOAD P0=JACK P1=ORDERS
```

Or you would create an INTO TABLE clause and EXEC statement as follows:

```
INTO TABLE &&0.&&1  
LOADER EXEC PGM=LDLSLDR,PARM='LOAD P0=JACK P1=ORDERS'
```

In this example, the complete name is JACK.ORDERS.

To substitute part of a table name. To substitute part of a table name in the SYSSQL dataset, for instance, to insert a date into a table name, you would create an INTO TABLE clause and Pn keyword as follows:

```
INTO TABLE JACK.ORDERS&&0  
LOAD P0=JULY4
```


Or you would create an INTO TABLE clause and EXEC statement as follows:

```
INTO TABLE JACK.ORDERS&&0  
LOADER EXEC PGM=LDLSLDR,PARM='LOAD P0=JULY4'
```

In this example, the complete name is JACK.ORDERSJULY4.

Choosing which rows to load

You can choose to load or discard a logical record by using the WHEN clause. This clause tests one or more conditions in a logical record. If you omit the WHEN clause, which you can because it's optional, then *all* logical records are loaded into the specified user table. If you include the WHEN clause, then it must follow the INTO TABLE clause.

The WHEN clause gives you a number of options for selecting the data that you want to load, whether you're loading one user table or multiple user tables. With one input dataset you can:

- Load logical records that meet one or more conditions, that is, where certain data fields contain certain values.
- Load *all* logical records into multiple user tables. In other words, you can load multiple (different) user tables with the same input data records.
- Load *all* logical records into some user tables and *some* logical records into other user tables.
- Load *some* logical records into some user tables and *some* logical records into other user tables.

With the WHEN clause, you specify criteria for selecting a logical record. This selection criteria includes the location of a value you want tested as well as the value itself. Logical records that satisfy the selection criteria are loaded.

Note the following:

- You cannot use a WHEN clause for delimited data or relatively positioned data fields.
- When specifying multiple criteria (AND and OR), ANDs have a higher precedence than ORs. You can use parentheses to alter or force precedence.
- For clarity, you can use parentheses to enclose a field condition.
- You can use the DISCARDFILE clause to collect logical records that don't satisfy the selection criteria in any WHEN clause. See "Collecting discarded records in a discard file" on page 4-14 for more information.

By using the SKIP and TAKE keywords on the LOAD control statement (see pages 5-8 and 5-9) you can select specific physical records to be transferred to StorHouse. The WHEN clause then applies to the remaining records.

For example, if an input dataset contains 100 physical records, and if TAKE=50, then the WHEN clause applies to the first 50 records in the input dataset. Now assume that SKIP=50. In this case, the WHEN clause applies to the last 50 records because the first 50 records are not transferred to StorHouse.

Remember that the client data loader processes the SKIP and TAKE keywords, and the server data loader processes the LOAD DATA statement. The SKIP and TAKE keywords apply to physical records. The WHEN clause applies to logical records.

Format of WHEN clause

WHEN field_condition [{ AND | OR } field_condition]...

Argument	Description
field_condition	{ (position) column_name field_name } operator { any_string BLANKS }
(position)	<p>(required if not using column_name or field_name) Starting (required) and ending (optional) column numbers of the column or field that contains the criteria you are testing. The format is:</p> <p>start_column [{ : - } end_column]</p> <p>For example: (1:3) or (1-3) indicates the criteria starts in column 1 and ends in column 3.</p> <p>Note that when specifying a (position) with a character (not a hex) string, the data is assumed to be in the default data CCSID.</p>
column_name	(required if not using (position) or field_name) Name of a column that contains the criteria you are testing.
field_name	(required if not using (position) or column_name) Name of a field that contains the criteria you are testing.
operator	<p>(required) Comparison operator. Specify one of the following:</p> <p>= (equal), != or ^= or <> (not equal), < (less than), > (greater than), <= (less than or equal), >= (greater than or equal)</p>
any_string	<p>(required if not using BLANKS) Criteria you are testing. Specify one of the following, depending on the type of data in the data field:</p> <ul style="list-style-type: none"> ■ string – string of characters enclosed in single or double quotes. Use string for data fields containing character data. This character string must match the case (UPPER, lower, or Mixed) of the input data. ■ Xstring – string of hexadecimal digits preceded with X and enclosed in single or double quotes. Use Xstring for data fields containing binary data.
BLANKS	(required if not using any_string) Keyword to test one or more blanks.
AND	(optional) Keyword for testing multiple values. A logical record is loaded when all conditions specified for AND are true.
OR	(optional) Keyword for testing multiple values. A logical record is loaded when at least one condition specified for OR is true.

Example WHEN clauses

This section contains example WHEN clauses.

To specify the starting column number of the selection criteria

You can specify a starting column number to identify the starting location of the value you want tested as selection criteria. Enclose the column number in parentheses. When you specify a starting column number only, the length of the selection criteria is derived from the comparison string.

For example, assume the selection criteria is the number 3, data type SMALLINT, located at column 1 of the logical record. You would use this WHEN clause:

```
WHEN (1) = X'0003'
```

In this example, because there's no ending column number, the length is derived from the comparison value, which is 2.

To specify starting and ending column numbers of the selection criteria

You can specify both the starting and ending column numbers to identify the location of the value you want tested as selection criteria. Enclose the column numbers in parentheses, and use a colon or a dash to separate the starting column number from the ending column number.

For example, assume the value you want tested is the character string ABC located in columns 1 through 3. You would specify this WHEN clause:

```
WHEN (1:3) = 'ABC'
```

Padding selection criteria. If the value is shorter than the length defined by the starting and ending column numbers, then that value is padded on the right

with blanks (if a character string) or zeros (if a hexadecimal digits). For example, if

```
WHEN (1:3) = 'AB'
```

then the actual selection criteria is 'AB ' (blank padded on the right).

Trimming selection criteria. If the value is longer than the length defined by the starting and ending column numbers, then it is truncated on the right. For instance, if

```
WHEN (1:3) = 'ABCD'
```

then the D is trimmed and not considered part of the selection criteria. You'll receive a warning message when any non-blank characters or non-zero bytes are trimmed.

To use a column name to identify the selection criteria

You can use a column name instead of the starting and ending column numbers or a field name to identify the value you want tested as selection criteria. This column name must *exactly* match the column name in the table's CREATE TABLE statement. You must delimit column names that do not follow the conventions of StorHouse SQL identifiers (see page 4-4 for conventions).

When you use a column name in a WHEN clause, the field_spec for that column identifies the location and length of the value. For example, assume the name of the column that contains the value you want tested is CUSTOMER_NUMBER. Here's the WHEN clause:

```
WHEN CUSTOMER_NUMBER = '3392003900'
```

Here's the field_spec for the CUSTOMER_NUMBER column:

```
CUSTOMER_NUMBER POSITION (6) INTEGER EXTERNAL(10)
```


In this example, the WHEN clause identifies the value that you are testing and the field_spec identifies the location and length of that value. All logical records with the integer 3392003900 starting in position 6 will be loaded.

To use a field name to identify the selection criteria

You can use a field name instead of the starting and ending column numbers or a column name to identify the value you want tested as selection criteria. You must delimit field names that do not follow the conventions of StorHouse SQL identifiers (see page 4-4 for conventions).

When you use a field name in a WHEN clause, the field_spec for that field identifies the location and length of the value. For example, assume the name of the field that contains the value you want tested is RECORD_CODE and the value you are testing is B. Here's the WHEN clause:

```
WHEN RECORD_CODE = 'B'
```

Here's the field_spec for the RECORD_CODE field:

```
:RECORD_CODE POSITION (1) CHAR
```

In this example, the WHEN clause identifies the value that you are testing and the field_spec identifies the location and length of that value. Field names in a WHEN clause must *not* be preceded by a colon. Field names in a field_spec must be preceded by a colon. All logical records with the letter B in position 1 will be loaded.

To use a character string as selection criteria

When the value you are testing as selection criteria is character data, enclose that value in single or double quotes. Note the following:

- The character string *must* match the case (UPPER, lower, or Mixed) of the input data.
- If the selection column or field is of VARCHAR type, then the comparison is made against the character data only; the length part (first 2 bytes) is not included. If the actual length of the data field differs from the length of the tested value, the test can be true only if non-significant (blank) bytes are trimmed from the larger value.
- If the value of the CCSID keyword on the LOAD control statement differs from EBCDIC (the character set of the SYSSQL dataset), then all comparison values that are character strings are automatically converted by the server data loader to the CCSID of the compared data field.

For example, if the selection criteria is the letter A, data type CHAR, located in column 1 of the logical record, then you would specify this selection criteria as a character string enclosed in quotes:

```
WHEN (1) = 'A'
```

To use a hexadecimal string as selection criteria

When the value you are testing as selection criteria is binary data, precede the value with X and enclose the value in single or double quotes. For example, assume the selection criteria is the number 1, data type SMALLINT, located at column 1 of the logical record. You would specify this selection criteria as a string of hexadecimal digits preceded with X and enclosed in quotes:

```
WHEN (1:2) = X'0001'
```


To test blanks

When the value you are testing as selection criteria is one or more blank characters, then use the **BLANKS** keyword in a **WHEN** clause. The default length of **BLANKS** is 1, so if you specify a starting column only, the selection criteria is one blank.

For instance, you would specify this **WHEN** clause to load all logical records that contain a blank character in column 80:

```
WHEN (80) = BLANKS
```

To test multiple values (using AND)

You can test multiple values by using the **AND** keyword in a **WHEN** clause. A logical record is loaded when *all* conditions are true. For example:

```
WHEN (1) = 'A' AND STOCK_NUMBER = '1023992'
```

In this example, all records with the letter A in column 1 and where the column **STOCK_NUMBER** contains a value of 1023992 will be loaded.

To test one value or another (using OR)

You can test one value or another by using the **OR** keyword in a **WHEN** clause. A logical record is loaded when at least one of the conditions is true. For example:

```
WHEN (11:13) = '800' OR (11:13) = '888'
```

In this example, a logical record with either 800 or 888 in columns 11 through 13 will be loaded.

To test one value or another and multiple values (using OR and AND)

You can specify multiple selection criteria by using both the AND and OR keywords in a WHEN clause. A logical record is loaded when *all* AND conditions are true and when at least one of the OR conditions is true. You can use parentheses to enforce precedence. For example:

```
WHEN ((1:3)= '301' OR (1:3)= '410') AND ((11:13)= '301' OR (11:13)= '410')
```

In this example, a logical record with either 301 or 410 in columns 1 through 3 *and* with either 301 or 410 in columns 11 through 13 will be loaded.

Generating field_specs, identifying NULL flags, specifying default delimiters and other defaults

You can use a FIELDS clause to:

- Generate a field_spec as the CHARACTER loader data type for every column in the named user table. To do this, include the CHAR keyword with the FIELDS clause and omit the field_spec for all data fields.
- Indicate each input data record starts with a sequence of 1-byte NULL flags (T for NULL or F for NOT NULL). To do this, include the NULLFLAGS keyword with the FIELDS clause and omit the field_spec for all data fields.
- Specify a default delimiter for character-based data fields (CHARACTER and all EXTERNAL data types). See "Guidelines for specifying a default delimiter" on page 4-52 for more information. You can also specify a delimiter in a field_spec for an individual data field.
- Load a NULL value for any data field that is empty or contains blanks. An *empty data field* consists of two adjacent delimiters. To do this, include the

NULLIF clause with the FIELDS clause. You can also include a NULLIF clause with a field_spec for an individual data field.

- Load a column's default value for any data field that is empty or contains blanks. To do this, include a DEFAULTIF clause with the FIELDS clause. You can also include a DEFAULTIF clause with a field_spec for an individual data field.
- If you are loading LOB data that never exceeds 32705, you can use FIELDS CHAR and place the LOB data in the data stream.
- You can include the CHAR and NULLFLAGS keywords in a FIELDS clause only when you omit a field_spec for all data fields.
- If you omit a field_spec for all data fields and you omit the FIELDS clause, the server data loader generates a field_spec for every column in the named table using the corresponding the CREATE TABLE data type.
- When you use the NULLFLAGS keyword, the server data loader appends a NULLIF clause to each generated field_spec when the NULL flag is T (for NULL). This loads a NULL into the corresponding table column.
- When you use FIELDS CHAR, FileTek recommends you include a delimiter_spec.
- NULLIF and/or DEFAULTIF clauses in the FIELDS clause apply to any generated field_specs or to any data fields that do not have NULLIF and/or DEFAULTIF clauses in their field_specs.
- If you specify FIELDS CHAR and the input dataset contains LOB data fields, all fields must still fit in a legal (32K-1-max) record size.
- If you omit the field_specs and the FIELDS CHAR clause, the FileTek MVS Data Loader utility places any LOB data fields at the end of the record in CREATE TABLE order.

Guidelines for specifying a default delimiter

You can delimit data fields composed of character data. These data fields include CHARACTER and all of the EXTERNAL data types. Note the following:

- You cannot delimit VARCHAR data fields.
- A delimiter can be only one character.
- The maximum length of a data field does not include the delimiter(s).
- You cannot use delimited data fields in a WHEN clause.
- If a data field contains a character that is also used as an enclosure delimiter, you can double that character in the data to protect it. The server data loader converts the doubled character to a single character. Or you can identify an escape character with the ESCAPED BY clause. See “Identifying an escape character” on page 4-59 for more information.
- If a data field contains a character that is also used as a termination delimiter, you cannot double that character in the data; but you can use an enclosure delimiter or identify an escape character with the ESCAPED BY clause.

- For instance, if the data field is:

SAN ANTONIO, TX

and the termination delimiter is a comma:

SAN ANTONIO,TX,

then use an enclose delimiter (such as parentheses) to protect the data field:

(SAN ANTONIO,TX),

Or specify an ESCAPED BY clause:

ESCAPED BY ','

- If data fields are enclosed with a single delimiter (like a single quote) and are not separated by a terminator or blanks, adjacent delimiters would be interpreted as data. For example, the following data fields are enclosed by single quotes. The end delimiter of the first data field and the start delimiter of the second data field would be interpreted as data:

'2839"Jack'

- You can use the **OPTIONALLY** keyword only when using the **TERMINATED** keyword. You can include both keywords in the **FIELDS** clause, or both in a **field_spec**, or one in a **FIELDS** clause and the other in a **field_spec**.

See “What’s delimited data?” on page 3-6 for basic information about delimited data and types of delimiters. See “Specifying a delimiter for an individual data field” on page 4-106 for more information about overriding the default delimiter.

Format of FIELDS clause

FIELDS fields_specs

where fields_specs:

```
[ CHAR ]
[ NULLFLAGS ]
[ delimiter_spec ]
[ NULLIF ( EMPTY | BLANK ) ]
[ DEFAULTIF ( EMPTY | BLANK ) ]
```

and where delimiter_spec:

```
[ TERMINATED [ BY ] { WHITESPACE | 'char' | X'hexbyte' } ]
[ [ OPTIONALLY ] ENCLOSED [BY] { 'char' | X'hexbyte' } ]
[ AND { 'char' | X'hexbyte' } ] ]
```

Argument	Description
CHAR	(optional) Keyword for generating a field_spec as the CHARACTER loader data type for all columns in the named table. A delimiter_spec is highly suggested. An error occurs if you include FIELDS CHAR and a field_spec list in a LOAD DATA statement.
NULLFLAGS	(optional) Keyword for generating a field_spec for all columns as the corresponding CREATE TABLE data type (if CHAR is omitted) and for identifying NULL flags (T for NULL and F for NOT NULL) at the beginning of each input data record for each data field. An error occurs if you include FIELDS NULLFLAGS and a field_spec list in a LOAD DATA statement.
delimiter_spec	(optional but recommended) Keywords for describing delimited data.
TERMINATED	Keyword for describing terminated data fields.
BY	Keyword for readability only.

Argument	Description
WHITESPACE	Keyword for indicating that the delimiter is one or more blank characters. You can use this keyword with the TERMINATED keyword, not the ENCLOSED keyword.
'char'	Value of a character delimiter, consisting of exactly one character and enclosed in single or double quotes. This value must match the case (UPPER or lower) of the input data.
X'hexbyte'	Value of a hexadecimal delimiter, consisting of exactly two hex digits.
OPTIONALLY	Keyword for indicating that some data fields may be enclosed with the specified delimiter(s). You can specify OPTIONALLY only when using TERMINATED. You can include both keywords in the FIELDS clause, or both in a field_spec, or one in a FIELDS clause and the other in a field_spec.
ENCLOSED	Keyword for describing enclosed data fields.
AND	Keyword for describing data fields enclosed with different starting and ending delimiters. If omitted, the enclosure delimiters are the same. The AND keyword must follow the ENCLOSED keyword.
NULLIF	(optional) Option to load a NULL value for any data field that is empty or contains blanks.
EMPTY	Keyword for indicating an empty condition, that is, two adjacent delimiters. Using the EMPTY keyword is equivalent to specifying an empty pair of delimiters in the NULLIF clause.
BLANK	Keyword for indicating a data field contains zero or more blanks.
DEFAULTIF	(optional) Option to load a column's default value for any data field that is empty or contains blanks. See the preceding EMPTY and BLANK keywords for more information.

Example FIELDS clauses

This section contains example FIELDS clauses.

To describe data fields terminated with a character

When data fields are terminated by a single character, use the **TERMINATED** keyword and specify the value of the delimiter in the **FIELDS** clause. You can specify a delimiter's value in character or hexadecimal format. Enclose the value in single or double quotes. The end of a logical record always serves as a termination delimiter for the last data field if no delimiter is present.

For example, to specify the delimiter for these data fields:

2	8	3	9	!	M	c	G	u	i	r	e	!	J	a	c	k	!	
2	3	8	8	!	C	o	r	n	f	l	a	k	e	!	S	u	e	!

You would include this **FIELDS** clause:

FIELDS TERMINATED BY '!'

To describe data fields terminated by a blank

When data fields are terminated by a blank, use the **TERMINATED** keyword with the **WHITESPACE** keyword in the **FIELDS** clause. For example, to specify the delimiter for these data fields:

2	8	3	9		M	c	G	u	i	r	e					J	a	c	k									
2	3	8	8		C	o	r	n	f	l	a	k	e				S	u	e									

You would include this **FIELDS** clause:

FIELDS TERMINATED BY WHITESPACE

Note: The next field's data will not include any of the whitespace from the previous data field unless the next data field has a fixed-start position.

To describe data fields enclosed by the same delimiter

When data fields are enclosed by the same delimiter, then use the ENCLOSED keyword and specify the value of the enclosure delimiter. For example, to specify that data fields are enclosed by double quotes and terminated with blanks:

```
"2839" "McGuire" "Jack"
"2388" "Cornflake" "Sue"
```

You would include this FIELDS clause:

```
FIELDS TERMINATED BY WHITESPACE ENCLOSED BY '"'
```

To describe data fields enclosed by different delimiters

When data fields are enclosed by different enclosure delimiters, then use the ENCLOSED keyword and specify the delimiter characters with the AND keyword. For example, to specify that data fields are enclosed by parentheses:

```
(2839)(McGuire)(Jack)
(2388)(Cornflake)(Sue)
```

You would include this FIELDS clause:

```
FIELDS ENCLOSED BY '(' AND ')'
```

To describe data fields that are both terminated and enclosed

When data fields are both terminated and enclosed, then use the TERMINATED keyword followed by the ENCLOSED keyword in a FIELDS clause. You can also use the OPTIONALLY keyword when some data fields are enclosed. For example,

to specify that all data fields are terminated by a comma and some are optionally enclosed by parentheses:

```
2839,(McGuire),Jack,
2388,(Cornflake),Sue,
```

You would include this FIELDS clause:

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '(' AND ')'

To generate CHAR field_specs

You can generate field_specs as the CHARACTER loader data type by including the CHAR keyword with the FIELDS clause and by omitting field_specs from the LOAD DATA statement. For example, to generate field_specs for these data fields:

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

include this FIELDS clause:

FIELDS CHAR TERMINATED BY WHITESPACE

To identify NULL flags in input data records

You can identify NULL flags in input data records by using the NULLFLAGS keyword in a FIELDS clause and by omitting field_specs in a LOAD DATA statement. In the following example, the server data loader generates field_specs using the CHARACTER data type.

For example, to describe these data fields:

```
FFF2839  McGuire Jack
FFF2388  Cornflake Sue
```


include this FIELDS clause:

FIELDS CHAR NULLFLAGS TERMINATED BY WHITESPACE

To load NULL values for empty data fields

You can load a NULL value for an empty data field (two adjacent delimiters) by including the NULLIF clause with the EMPTY keyword. For example, to load a NULL value for the empty data field in the first record below and to generate field_specs using the CHARACTER data type:

```
( 2 8 3 9 ) ( M c G u i r e ) ( J a c k ) ( )  
( 2 3 8 8 ) ( C o r n f l a k e ) ( S u e ) ( L )
```

include this FIELDS clause:

FIELDS CHAR ENCLOSED BY '(' AND ')' NULLIF EMPTY

Identifying an escape character

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

Format of ESCAPED BY clause

ESCAPED BY [DELIMITER | 'char' | NONE]

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

Note the following:

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

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

Example ESCAPED BY clause

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

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

Loading missing data fields with null values

A record that's missing a data field is called a *short record*. By using the TRAILING NULLCOLS clause, you can load a null value for a data field that's defined in a field_spec but missing in a record. If you omit this clause, then the FileTek MVS

Data Loader utility generates an error message when a data record is missing a data field.

Note the following:

- The column in the user table must allow null values. The load will fail if you try to load a null value into a column defined as NOT NULL.
- If the field_spec refers to a field (not a column), no error is generated and no null value is loaded. Fields are not loaded into user tables.
- If an input data record ends in the middle of a data field (of character-type data) and you specify the TRAILING NULLCOLS clause, then the record is padded.
- If you specify the TRAILING NULLCOLS clause and the data record ends before the start of a data field, then that data field is loaded with a null value.

If a data field is not missing, but you want to replace the data value with a null value, then you can use the NULLIF clause in a field_spec. See “Loading a column with a null value” on page 4-108 for more information about the NULLIF clause.

Format of TRAILING NULLCOLS clause

TRAILING [NULLCOLS]

The NULLCOLS keyword is optional.

Example TRAILING NULLCOLS clause

Assume that you are loading a user table with three columns. Below is the field_spec:

```
(ORDER_NUM POSITION (1) INT EXTERNAL(4),  
REP_LASTNAME POSITION (5) CHAR(15),  
REP_FIRSTNAME POSITION (20) CHAR(15))
```

These are some of the input data records. Notice that record 4 is missing the representative's first name.

[illegible]

If you included the TRAILING NULLCOLS clause and if the user table's column allowed null values, then the server data loader would load a null value into the REP_FIRSTNAME column. If you omitted the TRAILING NULLCOLS clause, then the FileTek MVS Data Loader utility would generate an error because data's missing. The load operation would fail if the REP_FIRSTNAME column in the user table was defined as NOT NULL.

Loading one or more segments

You can load data into multiple segments of a user table by including multiple INTO TABLE clauses and specifying the SAME SEGMENT and DIFFERENT SEGMENT clauses. You can also load multiple segments of different user tables during one load. See “Loads and segments” on page 1-8 for more information about segments.

SAME SEGMENT is the default, which means that the server data loader loads the data into one segment. Simply omit the SAME SEGMENT and DIFFERENT SEGMENT clauses when you want to load data into one new segment.

Note the following:

- The value of the StorHouse SQL_LDR_MAXINTO system parameter must be greater than 1 to load multiple segments during a load.
- When there are multiple INTO TABLE clauses, then SAME SEGMENT loads data into the same segment as the most recent INTO TABLE clause for the same user table.
- You can name segments if you need or plan to replace them later. See page 4-66 for more information about using the SEGMENT clause to name a segment.

Format of SAME and DIFFERENT SEGMENT clauses

SAME SEGMENT

DIFF[ERENT] SEGMENT

Example SAME and DIFFERENT SEGMENT clauses

This section contains examples that show how to load data into multiple segments of the same user table as well as into different user tables.

To load multiple segments of the same user table

The following example loads two segments in the ATM.TRANSACTIONS table. Records containing A in column 1 will be loaded into one segment and those containing B in column 1 will be loaded into a different segment.

```
LOAD  
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'A'
```

```
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'B'  
DIFF SEGMENT;
```

To load multiple segments of different user tables

The following example loads one segment of one user table and multiple segments of another user table.

```
LOAD  
INTO TABLE ATM.TRANSACTIONS
```

```
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE= '5/1/1999'
```

```
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE = '6/1/1999'  
DIFF SEGMENT;
```

Note the following:

- All records will be loaded into one segment of the ATM.TRANSACTIONS user table. SAME SEGMENT (omitted from the into_table_spec) is the default.
- Records containing 5/1/1999 in the TRANS_DATE column will be loaded into one segment of the POS.TRANSACTIONS table. SAME SEGMENT (omitted from the into_table_spec) is the default.

- Records containing 6/1/1999 in the TRANS_DATE column will be loaded into a different segment of the POS.TRANSACTIONS table.

Naming a segment

You can name the current segment you're loading in the event you need to replace it in the future. This name is called a *segment tag*. You specify a segment tag with the SEGMENT clause. Note the following:

- If you omit the SEGMENT clause, the default segment tag is the load ID that's automatically generated during a load. The FileTek MVS Data Loader utility reports the load ID on message LDL757I.
- Segment tags need not be unique, but this could result in several segments being invalidated by a subsequent replace operation.
- If there are multiple INTO TABLE clauses for the same segment, you can specify only one SEGMENT clause for one of the INTO TABLE clauses. An error will occur if you specify different segment tags for the same segment.
- Segment tags are stored in the SYSSTHSEGMENTS system table.
- StorHouse/RM normalizes the segment tag on a SEGMENT clause to uppercase. This means you can use any case when providing a segment tag on a REPLACE clause. Segment tags are case sensitive only when delimited.

Format of SEGMENT clause

SEGMENT segment_tag

Argument	Description
segment_tag	(required) Name of the segment that you're loading into the user table specified on the INTO TABLE clause. This name cannot exceed 40 characters and must follow SQL identifier conventions (see page 4-4) or be quoted.

Example SEGMENT clauses

This section contains example SEGMENT clauses.

To use the load ID as the segment tag

Omit the SEGMENT clause to use the load ID generated by the FileTek MVS Data Loader utility. For example, there's no SEGMENT clause for either INTO TABLE clause below, so the load ID is the segment tag for both segments.

Note: Load IDs start with a digit, so if you issue a subsequent REPLACE clause, you must delimit the load ID with double quotes.

```
LOAD  
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'A'
```

```
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'B'  
DIFF SEGMENT;
```


To assign different segment tags for multiple segments of the same user table

In the following example, the segment tag in the second INTO TABLE clause is different from the one in the first INTO TABLE clause.

```
LOAD
INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '5/1/1999'
SEGMENT MAYSEGMENT
```

```
INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '6/1/1999'
DIFF SEGMENT
SEGMENT JUNESEGMENT;
```

Replacing a segment

You can invalidate an existing segment by using the REPLACE SEGMENT clause. When you replace a segment, a StorHouse engine updates the SYSSTHSEGMENTS system table, setting the INVALID_FLAG column to Y and the INVALID_TIME column to the current time for all segments with the given segment tag, owner, and table name.

Note the following:

- You can obtain the segment tag from the SYSSTHSEGMENTS system table.
- If you named a segment with the SEGMENT clause, you can use any case when providing the segment tag on the REPLACE clause. StorHouse/RM normalizes these segment tags to uppercase. For example:

```
SEGMENTcallsload
REPLACE SEGMENT CALLSLOAD or REPLACE SEGMENT Callsload
```


- If you delimited the segment tag on the SEGMENT clause, you must use the same case and delimit it on the REPLACE clause. For example:

```
SEGMENT "1seg"  
REPLACE SEGMENT "1seg"
```

- If you omitted the SEGMENT clause when the segment was loaded, the load ID is the segment tag. Load IDs start with a digit, so you must delimit them with double quotes. For example:

```
REPLACE SEGMENT "00343445081"
```

- If you omit the owner and table name, the default is the owner and table name specified on the INTO TABLE clause.
- If you include both the SEGMENT and REPLACE SEGMENT clauses and specify the same segment tag for both, the current (new) segment will not be invalidated.
- If there are multiple INTO TABLE clauses for the same segment, you need to specify only one REPLACE SEGMENT clause for one of the INTO TABLE clauses. An error occurs if you specify different segment tags for the same segment.
- When replacing segments *without* loading data, the SYSREC DD statement in the JCL must name an empty dataset. You can use DD* followed by /*.
- No error occurs if there are no segments with the specified segment tag.

Caution: Do not run MERGE and REPLACE SEGMENT operations at the same time against the same table. StorHouse/RM may not merge the segments because the REPLACE SEGMENT operation indicates a change to the table. This does not damage the data but is an inefficient use of time and resources.

Format of REPLACE SEGMENT clause

REPLACE SEGMENT [[owner.] table_name.] segment_tag

Argument	Description
owner.	(optional) StorHouse account ID of the owner of the user table containing the segment you are replacing. If you omit the owner name, the server data loader uses the owner name on the corresponding INTO TABLE clause.
table_name.	(optional) Name of the user table containing the segment you are replacing. If you omit the table name, the server data loader uses the table name on the corresponding INTO TABLE clause.
segment_tag	(required) Name assigned to the segment (on the SEGMENT clause) when you loaded the user table. If you omitted the SEGMENT clause, the load ID is the segment tag.

Example REPLACE SEGMENT clause

The following example loads data into a new segment called JULYSEGMENT and invalidates the segment called JUNESEGMENT. The server data loader will replace all segments named JUNESEGMENT in the POS.TRANSACTIONS table.

```
LOAD  
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE= '7/1/1999'  
SEGMENT JULYSEGMENT  
REPLACE JUNESEGMENT;
```

Selecting subspaces

You can select specific subspaces for user table components (table data, value indexes, hash indexes, and LOB data) by including one or more SUBSPACE

number clauses after an INTO TABLE clause of a LOAD DATA or MERGE statement or on a LOAD INDEX statement. Note the following:

- You cannot use both a SUBSPACE ROTATE and a SUBSPACE number clause in a LOAD DATA, LOAD INDEX, or MERGE statement. Only one of these clauses is allowed.
- If you omit both the SUBSPACE ROTATE and the SUBSPACE number clause, the server data loader uses the lowest-numbered subspace that allows the component type.
- You can include a SUBSPACE number clause for any component types. For instance, for a data load, you can select a subspace for just table data and use the default selection (lowest-numbered subspace) for indexes and LOB data. For an index load, you can select different subspaces for value and hash indexes. And for a merge operation, you can select different subspaces for table data, value indexes, and hash indexes. LOB data is not reprocessed in a merge operation, so it remains in its current location.
- If you omit the component type (for example, TABLE or VALUE), the clause applies to all component types (unless one or more is overridden by a later SUBSPACE number clause). For example, if you specify SUBSPACE 2, then the server data loader uses subspace 2 for table data, hash indexes, value indexes, and LOB data.
- If you specify multiple SUBSPACE number clauses in the same INTO TABLE clause, later ones that specify the same component type(s) supersede earlier ones.
- Subspace(s) must exist for the applicable component type. For example, you can select subspace 3 for table data if the user tablespace contains a subspace 3 defined with OBJECT_TYPE T (for table data) or blank (for all component types). An error occurs if you select a subspace that does not exist or is not valid for a component type.

- The server data loader uses the table data subspace and the user table tablespace for LOB values that fit within a row, that is, for in-line LOBs.

Format of SUBSPACE number clause

[[TABLE | VALUE | HASH | LOB] SUBSPACE number]...

Argument	Description
TABLE	(optional) Keyword to select a subspace for table data.
VALUE	(optional) Keyword to select a subspace for value indexes.
HASH	(optional) Keyword to select a subspace for hash indexes.
LOB	(optional) Keyword to select a subspace for LOB data. StorHouse/RM uses LOB subspaces only for out-of-line LOBs.
number	(required) Subspace number. If you omit TABLE, VALUE, HASH or LOB, all component types use the same subspace number.

See page 4-135 for the SUBSPACE number clause of the LOAD INDEX statement and page 4-138 for the SUBSPACE number clause of the MERGE statement.

Example SUBSPACE number clauses

This section contains example SUBSPACE number clauses.

To select subspaces when loading one segment

You can select a specific subspace for each component type. For example, assume you're loading one segment. The user table has three hash indexes, and two value

indexes assigned to the same user tablespace as the table. The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	H (hash indexes only)
4	H (hash indexes only)
5	V (value indexes only)
6	V (value indexes only)

With the following LOAD DATA statement, the server data loader uses the following subspaces:

- Subspace 2 for the table data file
- Subspace 4 for all three hash index files
- Subspace 6 for both value index files

```
LOAD
INTO TABLE TOLLFREE
TABLE SUBSPACE 2
HASH SUBSPACE 4
VALUE SUBSPACE 6
(FROMNUM POSITION(1) BINARY EXTERNAL(10),
TONUM BINARY EXTERNAL(10),
ACCOUNT CHAR(12))
WHEN (11:13) = '800' OR (11:13) = '888' ;
```


To select subspaces when loading multiple segments

When loading multiple segments, you can use the same subspace or different subspaces for each component type. For example, assume you're loading two segments. The user table has one hash index and one value index assigned to the same user tablespace as the table.

The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	T (table data only)
4	H (hash indexes only)
5	H (hash indexes only)
6	H (hash indexes only)
7	V (value indexes only)
8	V (value indexes only)
9	V (value indexes only)

Assume you want to use a different subspace for each table data file but the same subspace for both hash index files and the same subspace for both value index files. With the following LOAD DATA statement, the server data loader uses the following subspaces:

- Subspace 1 for the table data file in segment 1
- Subspace 2 for the table data file in segment 2
- Subspace 5 for the hash index files in segments 1 and 2
- Subspace 9 for the value index files in segments 1 and 2


```
LOAD
INTO TABLE MARCHCALLS
TABLE SUBSPACE 1
HASH SUBSPACE 5
VALUE SUBSPACE 9
(FROMNUM POSITION(1) BINARY EXTERNAL(10),
TONUM BINARY EXTERNAL(10),
ACCOUNT CHAR(12))
WHEN (11:13) = '800' OR (11:13) = '888'

INTO TABLE MARCHCALLS
DIFFERENT SEGMENT
TABLE SUBSPACE 2
HASH SUBSPACE 5
VALUE SUBSPACE 9
(FROMNUM POSITION(1) BINARY EXTERNAL(10),
TONUM BINARY EXTERNAL(10),
ACCOUNT CHAR(12))
WHEN (11:13) != '800' AND (11:13) != '888';
```

To select subspaces in multiple user tablespaces

If any indexes or LOB columns for a user table are assigned to different user tablespaces, you can select different subspaces in different user tablespaces. If the subspace number is the same in all user tablespaces, then you can omit the TABLE | HASH | VALUE | LOB keywords and just specify the SUBSPACE keyword with the subspace number.

For example, assume you're loading one segment. The user table, assigned to user tablespace 1, has two hash indexes, two value indexes, and two LOB columns. The hash indexes are assigned to user tablespace 2, the value indexes are assigned to user tablespace 3, and the LOB columns are assigned to user tablespace 4.

User tablespace 1 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)

User tablespace 2 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	H (hash indexes only)
2	H (hash indexes only)

User tablespace 3 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	V (value indexes only)
2	V (value indexes only)

User tablespace 4 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	L (LOB data only)
2	L (LOB data only)

With the following LOAD DATA statement, the server data loader uses the following subspaces:

- Subspace 2 in user tablespace 1 for the table data file
- Subspace 2 in user tablespace 2 for both hash index files
- Subspace 2 in user tablespace 3 for both value index files
- Subspace 2 in user tablespace 4 for both LOB subsegment files

```
LOAD
INTO TABLE CALLSDBA.BILLSUMMARY
SUBSPACE 2
(BILL_ACCOUNT CHAR(10),
BILL_DATE DATE EXTERNAL(10),
BILL_CATEGORY CHAR(1),
NUMBER_CALLS INT EXTERNAL(3),
PHOTO_ID VARBINARY,
BILL_IMAGE VARBINARY);
```

Describing data fields

A *field_spec* describes a data field in a logical record. Include a *field_spec* to:

- Describe a field in a WHEN clause.

You must provide a *field_spec* for any field name specified in a WHEN clause. Remember that fields aren't loaded into user tables. You use fields only to assign a name to a portion of the logical record to be used in a condition.

- Generate a value to be loaded into a column. You can generate:
 - the record number of a logical record (RECNUM)
 - a sequence of values (SEQUENCE)
 - the current date (SYSDATE)
 - a constant value (CONSTANT)

- Describe a data field to be loaded into a column. You can describe:
 - the position of the data field in the logical record (`position_spec`)
 - the name and length of the data type (`datatype_spec`)
 - the character set for an individual data field (`CHARSET`)
 - a delimiter for a `CHAR` or `EXTERNAL` data field (`delimiter_spec`)
 - a condition that causes a column to be loaded with a null value (`NULLIF`) or a default value (`DEFAULTIF`)

You can omit all or some `field_specs` when describing data fields to be loaded into columns.

- If you include a `FIELDS CHAR` clause, you must omit all `field_specs`. The server data loader generates a `field_spec` for every column with the `CHARACTER` loader data type and determines the length of the data fields using the default-length rules for `CHARACTER`.
- You can omit a `FIELDS CHAR` clause and all `field_specs` if the input data is in the same order as the `CREATE TABLE` definition and all data fields are relatively positioned and of the same data type. The server data loader uses the `CREATE TABLE` definition (data types and lengths) to determine how to interpret the input data.
- If you omit a `FIELDS CHAR` clause and *some* `field_specs`, the server data loader loads the omitted columns with the default value. If no default value was assigned when the user table was created, a null value is loaded.

Format of `field_spec`

(`field_spec` [, `field_spec`]...)

where field_spec:

{ :field_name | column_name } data_spec

Argument	Description
:field_name	(required if using a field name in a condition) Name of the field you are using in a condition. Precede this field name with a colon (:). This name must adhere to the conventions of StorHouse SQL identifiers (see page 4-4 for conventions) or be quoted.
column_name	(required to generate data or to describe a column to be loaded) Name of the column in the input data. This name must match the column name specified in the CREATE TABLE statement.
data_spec	(optional) Value to be generated or description of the data field. The format is: RECNUM SEQUENCE (start_num [,increment]) SYSDATE CONSTANT any_value position_spec
position_spec	(optional) Position of the data field in a logical record, name and length of the data type, condition that loads a null value or a default value into a column. You can specify position_spec clauses in any order. The format is: [POSITION (position * [+num])] [datatype_spec] [NULLIF field_condition] [DEFAULTIF field_condition]
datatype_spec	(optional) Name of the data type and length of the column or field in the logical record. Depending on the data type, you can also specify a character set or a delimiter for an individual data field.

Providing a field name

If you used a field name in a condition (like in a WHEN clause), then you must provide a field_spec for that field. For instance, for the following WHEN clause:

```
WHEN RECORD_CODE = 'A'
```

you would provide this field_spec:

```
:RECORD_CODE POSITION (1) CHAR
```

Remember that:

- A field name in a WHEN clause does *not* start with a colon.
- A field name in a field_spec *must* start with a colon.
- Fields aren't loaded, only columns are loaded into user tables.
- If a field name is delimited, the preceding colon (in the field_spec) must be outside the quotes, for example, :“RECORD CODE” POSITION (1) CHAR

Providing a column name

When generating data or describing data fields to be loaded, you must provide a column name in a field_spec. Note the following:

- Column names must match the names assigned on the CREATE TABLE statement. For instance, if a column name in the CREATE TABLE statement is ORDER_NUM, then specify ORDER_NUM (not ORDER_NUMBER or ORDERNUMBER) in the field_spec.
- Column names are not case sensitive unless quoted.

Loading a record number into a column

Use the RECNUM keyword after a column name to load a column with the record number of the logical record from which that row was loaded. Record numbers start at 1 and increment for each logical record, including discarded records. You can load this record number into a table column of data type SMALLINT or INTEGER.

For example, to generate record numbers for a column called RECORD_NUMBER, specify:

```
RECORD_NUMBER RECNUM
```

Generating a sequence of values

Use the SEQUENCE clause after a column name to generate a sequence of unique values for each logical record that is not discarded. You can load these values into a table column of data type SMALLINT or INTEGER.

The format of SEQUENCE is:

```
SEQUENCE ( start_num [ ,increment ] )
```

Argument	Description
start_num	(required) Starting value, which must be a positive integer or 0.
increment	(optional) Number to increment subsequent logical records that are not discarded. If you omit the increment, the default is 1.

For example, to generate unique values for a column named CUSTOMER_NUM, starting with 1 for the first logical record not discarded and incrementing by 1 for each subsequent logical record not discarded, type:

```
CUSTOMER_NUM SEQUENCE (1,1)
```


Loading the current date into a column

Use the SYSDATE keyword after a column name to load the current date into a column, specifically, the date that the load or restart operation *began* (even if the load continues past midnight). The table column must be of data type CHAR, CLOB, DATE, or VARCHAR.

For example, to load the current date into a column called ORDER_DATE, specify:

```
ORDER_DATE SYSDATE
```

Loading a constant value into a column

Use the CONSTANT clause after a column name to load a constant value into a column. Note the following:

- The constant value can be a quoted character string or hexadecimal string, an unquoted identifier, or a number.
- The table column can be any data type.
- The server data loader treats the constant value as a string and converts it to the data type of the table column.
- If you specify a character string and the data field has a fixed position (specified by a POSITION clause), the server data loader pads or truncates that string if it is shorter or longer than the fixed position.

The format of CONSTANT is:

CONSTANT any_value

Argument	Description
any_value	Value to load into the column. The format is: any_string identifier num
any_string	Character string enclosed in single or double quotes or hexadecimal string preceded with X and enclosed in quotes
identifier	Unquoted string
n	Unsigned integer optionally followed by K (x1024) or M (xKK)

For example, to load the value CA into a column called STATE, specify:

STATE CONSTANT 'CA'

Specifying the position of a data field

To load a column or use a field, the server data loader must know where to locate it in a logical record. You specify the location of a data field by including a POSITION clause in a field_spec. You can specify:

- A *fixed position* with a starting (and optional ending) column number
- A *relative position* as a continuation or an offset from the previous data field

The POSITION clause is optional. If you omit it, the default is POSITION (*) or relative position as a continuation from the previous data field. Typically, you omit the POSITION clause when the input data contains VAR-type or delimited data fields. You must use relative positioning (the default POSITION (*)) for BLOB and CLOB data fields.

For VARCHAR or VARBINARY data fields, the position must include the 2-byte field containing the actual length of the data field. Any relatively positioned data

fields (using * or *+num) *after* a VARCHAR data field will start at varying positions based on the actual length of the VARCHAR data field in each logical record.

The format of POSITION is:

POSITION (position | * [+num])

Argument	Description
position	Starting (required) and ending (optional) column numbers of the data field in the logical record. The format is: start_column [{ : - } end_column]
start_column	Starting position of the data field in the logical record. The first position in a logical record is 1.
: or -	Character that separates the starting column from the ending column. Either character is valid.
end_column	Ending position of the data field in the logical record. If you omit the ending column, the length comes from the datatype_spec of the data field. If you include the ending position, the length of the datatype_spec is ignored.
*	The data field immediately follows the previous data field. When you include *, the length comes from the datatype_spec of the data field.
+num	An offset from the previous data field, where num is the number of positions. Use +num with *.

For example:

- To specify the starting column number of a data field:

POSITION (1)

In this example, the data field starts in position 1 of the logical record. The length comes from the datatype_spec of the data field. For instance, if the data type is SMALLINT, which has a length of 2 bytes, then the data field starts in position 1 and ends in position 2.

- To specify the starting and ending column numbers of a data field:

POSITION (1:3)

In this example, the data field starts in position 1 and ends in position 3 of the logical record. Another way to specify this is POSITION (1-3). If you also specify the length of the data type, the length is ignored.

- To specify continuation from the previous data field:

POSITION (*)

In this example, the data field is located one record position after the previous data field. So if the previous data field ends in position 15, then this data field starts in position 16 of the logical record. The length of the data field comes from the datatype_spec. POSITION (*) is the default.

Note: When a LOAD DATA statement contains multiple INTO TABLE clauses, the position used when the first field_spec of an INTO TABLE clause is relative is:

- column 1 of the logical record if no prior INTO TABLE clauses have been selected for loading (as determined by their WHEN clauses),
- or the column following the last data field in the last INTO TABLE clause that was selected for loading.

See page 4-121 for an example of relative positioning with multiple INTO TABLE clauses.

- To specify an offset from the previous data field:

POSITION (*+5)

In this example, the data field is located 5 positions plus 1 position after the previous data field. Therefore, if the previous data field ends in position 10,

then this data field starts in position 16 of the logical record. `POSITION (*+0)` is the same as `POSITION (*)`. The length of the data field comes from the `datatype_spec`.

Specifying the data type

A `datatype_spec` provides the data type name and length of a data field in a logical record. The server data loader uses this information to interpret a data field. For some data types, you can also specify a character set and a delimiter. Note that:

- The data type name is always required in a `datatype_spec`.
- The length, character set, and delimiters are optional in a `datatype_spec`.

See page 4-102 for more information about how the server data loader calculates the length of a data field if you omit the length in a `datatype_spec`.

The data type of an input data field does not have to match the data type of the column in the user table, but those data types must be compatible. The server data loader automatically performs any necessary conversions. See page 4-102 for a summary of compatible data types.

In a `datatype_spec`, the data type must be one of the data types supported by the server data loader. Those data types are described in the following tables.

Definitions of the data type specifications are as follows:

Definitions of data type specifications

LOAD DATA syntax	Format of the data type in a LOAD DATA datatype_spec
Input field size	Length, default length, length ranges of the input data
Input field format	Acceptable format and/or contents of the input data
Range	Minimum and maximum data values allowed for the data type
Storage size	Amount of space when stored on StorHouse. Note that null data takes no space (in the corresponding CREATE TABLE data type).
Target types	Valid CREATE TABLE data types for this input type, in other words, the supported conversions. See page 4-102 for a summary of conversions.

BINARY data type

LOAD DATA syntax	BINARY [(length)] RAW [(length)] BYTE [(length)] CHAR[ACTER] [(length)] CHARSET 65535
Input field size	Default length: CREATE TABLE length if BINARY or CHAR, else 1 Length range: 1 to 32765
Input field format	Any bytes
Storage size	CREATE TABLE length (1 to 256)
Target types	BINARY, BLOB, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none">■ If the LOAD length is greater than the CREATE TABLE length, the excess characters are silently truncated, that is, the server data loader does not report or log whether any data is trimmed.■ The input data is copied directly. There is no conversion, even when the target type is CHAR, CLOB, or VARCHAR.

BINARY EXTERNAL data type

LOAD DATA syntax	BINARY EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec] RAW EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec] BYTE EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Input field size	Default length: 512 (with delimiter_spec) or else CREATE TABLE length*2 if BINARY or CHAR, else 2 Length range: 1 to 32765
------------------	-------------------------------------------------------------------------------------------------------------------------------

Input field format	Hexits (0-9, a-f, A-F), where 2 hexits=1 byte
--------------------	-----------------------------------------------

Target types	All except TIME, DATE, TIMESTAMP, and DECIMAL
--------------	-----------------------------------------------

Notes	<ul style="list-style-type: none">■ If an odd number of hexits is supplied, a leading 0 is added.■ If the LOAD length/2 is greater than the CREATE TABLE length, the excess characters are silently truncated.
-------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

BLOB data type

LOAD syntax	BLOB [(max_length [K M G])] CLOB [(max_length [K M G])] CHARSET 65535
Input field size	Contents of 64-bit length field Length range: 0+8 (minimum) to 2G-9+8 (maximum) Default max_length = CREATE TABLE max_length + 8
Input field format	64-bit length field followed by the data
Storage size	In-line LOB: Size of BLOB plus 4 bytes Out-of-line LOB: Size of BLOB in the LOB subsegment file plus 22 bytes for the object identifier (OID) in the table data file
Target type	BLOB, CLOB
Notes	<ul style="list-style-type: none">■ The length of the data is read from a 64-bit field preceding the data. The native values key affects the interpretation of these bytes.■ If a BLOB data value exceeds the LOAD DATA max_length, the load fails.■ If a BLOB data value exceeds the CREATE TABLE length, the data is silently truncated.■ When the target type is CLOB, there are no conversions, that is, the input characters are copied directly rather than converted to hexits or from any data character set.■ See “Specifying a BLOB or CLOB data type” on page 4-107 for additional considerations.

CHARACTER data type

LOAD DATA syntax	CHAR[ACTER] [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: MIN (CREATE TABLE length, 32705) if BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR; or else 256 (with a delimiter_spec) or else 1 Length range: 1 to 32765
Input field format	Any characters
Storage size	CREATE TABLE length (1 to 256)
Target types	All
Notes	<ul style="list-style-type: none"> ■ If the ccsid is not 819, then the data will be converted. ■ If ccsid=65535, the data type is changed to BINARY and any delimiter_spec is ignored. ■ The CHAR length is not enforced on the LOAD DATA statement. This is helpful if you need to load delimited data with a size greater than 256 into a VARCHAR table column. In this case, you must use a CHAR data type to define that delimited data. ■ If the LOAD length is greater than the CREATE TABLE length, the excess characters are silently truncated. ■ If the target type is [VAR]BINARY, there is no conversion, that is, the input characters are copied directly. ■ When the target is a numeric type, a data field of only blanks is an error.

CLOB data type

LOAD syntax	CLOB [(max_length [K M G])] [CHARSET ccsid]
Input field size	Contents of 64-bit length field Length range: 0+8 (minimum) to 2G-9+8 (maximum) Default max_length = CREATE TABLE max_length + 8
Input field format	64-bit length field followed by the data
Storage size	In-line LOB: Size of CLOB plus 4 bytes Out-of-line LOB: Size of CLOB in the LOB subsegment file plus 22 bytes for the object identifier (OID) in the table data file
Target type	CLOB, BLOB
Notes	<ul style="list-style-type: none">■ If a CLOB data value exceeds the LOAD DATA max_length, the load fails.■ If a CLOB data value exceeds the CREATE TABLE length, the value is silently truncated.■ If the ccsid is 65535, the data type is changed to BLOB.■ When the target type is BLOB, there is no conversion, that is, the input characters are copied directly rather than interpreted as hexits.■ See "Specifying a BLOB or CLOB data type" on page 4-107 for additional considerations.

DATE or DATE EXTERNAL data type

LOAD DATA syntax	DATE [EXTERNAL] [(length)] ["mask"] [CHARSET ccsid] [delimiter_spec]
------------------	-------------------------------------------------------------------------

Input field size	Default length: 10 (without delimiter_spec or 256 (with delimiter_spec) Length range: 1 to 32765
------------------	--------------------------------------------------------------------------------------------------------

Input field format	Character representation of a date. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for dates.
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Storage size	4
--------------	---

Range	1/1/0001 to 12/31/9999
-------	------------------------

Target types	DATE
--------------	------

Note	The mask (its length or content) is currently not used.
------	---------------------------------------------------------

DECIMAL or NUMERIC data type

LOAD DATA syntax	DEC[IMAL] [PACKED] NUMERIC [PACKED] DEC[IMAL] (precision[, scale]) NUMERIC (precision[, scale])
Input field size	(precision + 2) / 2 Default precision and scale: CREATE TABLE precision and scale Default scale (if precision supplied): 0 Precision range: 1 to 31 Scale range: 0 to precision
Input field format	Number in the form <i>ddd...ds</i> , where <i>d</i> is a decimal digit represented by 4 bits and <i>s</i> is a 4-byte sign value (where a plus sign (+) is represented by A, C, E, or F and a minus sign (-) is represented by B or D)
Storage size	(precision + 2) / 2
Target types	DECIMAL, INTEGER, SMALLINT
Note	If you omit the precision but include a starting and ending column in the POSITION clause for the data field, the server data loader will calculate the precision from the POSITION clause.

DECIMAL EXTERNAL data type

LOAD DATA syntax	DEC[IMAL] EXTERNAL (length[, scale]) [CHARSET ccsid] [delimiter_spec] NUMERIC EXTERNAL (length[, scale]) [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 256 (with delimiter_spec) or else CREATE TABLE precision + 3 if DECIMAL, else 11 Default scale: 0 Length range: 1 to 32765
Input field format	String of characters that represents a number, with or without a sign or a decimal point. <i>E-notation</i> (floating-point literal) is also accepted. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for E-notation format.
Target types	DECIMAL, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"> ■ If no decimal point exists in the input and E-notation is not used, there is an implied decimal point at the position indicated by the scale (after any trailing blanks are trimmed). ■ A data field of only blanks is an error.

DOUBLE data type

LOAD DATA syntax	FLOAT FLOAT (bits) DOUBLE [PRECISION]
Input field size	8
Input field format	Double precision floating-point binary number
Storage size	8
Range	IEEE limits
Target types	REAL, DOUBLE, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"> ■ FLOAT with no length is FLOAT(53). ■ bits = 22 to 53 ■ Floating-point is interpreted as Proprietary S/370.

FLOAT (REAL) data type

LOAD DATA syntax	FLOAT FLOAT (bits) REAL
Input field size	4
Input field format	Single precision floating-point binary number
Storage size	4
Range	IEEE limits
Target types	REAL, DOUBLE, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none">■ When bits = 1 to 21, the number is single precision.■ When bits = 22 to 53, the number is double precision.■ FLOAT with no length is double precision (FLOAT(53)).■ Floating-point is interpreted as Proprietary S/370.

FLOAT EXTERNAL data type

LOAD DATA syntax	FLOAT EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 24 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Integer value, decimal value, or character representation of a floating-point number (E-notation). Refer to the <i>StorHouse SQL Reference Manual</i> for E-notation format. Leading and trailing blanks are allowed.
Target types	REAL, DOUBLE, DECIMAL, INTEGER, SMALLINT
Note	A data field of only blanks is an error.

4

SYSSQL dataset

Describing data fields

INTEGER data type

LOAD DATA syntax	INT[EGER]
Input field size	2 if nvk=DOS, else 4
Input field format	Signed binary integer
Storage size	4
Range	-2147483648 to 2147483647
Target types	INTEGER, SMALLINT
Note	The byte order is interpreted as most significant byte first, big-endian.

INTEGER EXTERNAL data type

LOAD DATA syntax	INT[EGER] EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 11 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Array of characters that represents a number, with or without a sign, without a decimal point. Leading and trailing blanks are allowed.
Target types	INTEGER, SMALLINT
Note	A data field of only blanks is an error.

SMALLINT data type

LOAD DATA syntax	SMALLINT
Input field size	2
Input field format	Signed binary integer
Storage size	2
Range	-32768 to 32767
Target types	INTEGER, SMALLINT
Note	The byte order is interpreted as most significant byte first, big-endian.

TIME EXTERNAL data type

LOAD DATA syntax	TIME EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 12 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Character representation of a time. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for time.
Storage size	4
Range	0:0:0.000 through 24:0:0.000 (leap seconds (up to 62 seconds) are also allowed)
Target types	TIME
Note	The server data loader accepts a time data field without milliseconds.

TIMESTAMP EXTERNAL data type

LOAD DATA syntax	TIMESTAMP EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 26 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Character representation of a timestamp. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for timestamp.
Storage size	8
Range	See TIME and DATE, except TIMESTAMP contains microseconds
Target types	TIMESTAMP

VARBINARY data type

LOAD DATA syntax	VARBINARY [(max_length)] VARRAW [(max_length)] VARBYTE [(max_length)] VARCHAR [(max_length)] CHARSET 65535
Input field size	Contents of SMALLINT length field + 2 Default max_length: CREATE TABLE max_length if VARBINARY or VARCHAR, else 256 Length range: 1 to 32765
Input field format	SMALLINT field followed by any bytes
Storage size	Length of data + 2 (unless compressed)
Target types	BINARY, BLOB, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none">■ The length of the data is read from the SMALLINT field preceding the data.■ If the data exceeds the LOAD max_length or runs off the end of the record, an error is generated and the load is terminated.■ If the data exceeds the CREATE TABLE length, the data is silently truncated.■ If the CREATE TABLE length is greater than 4096, the data is compressed before being written.■ If the target type is [VAR]CHAR, there is no conversion, that is, the input characters are copied directly.

VARCHAR data type

LOAD DATA syntax	VARCHAR [(max_length)] [CHARSET ccsid]
Input field size	Contents of SMALLINT length field + 2 Default max_length: CREATE TABLE max_length if VARCHAR or VARBINARY, else 256 Length range: 1 to 32765
Input field format	SMALLINT field followed by any characters
Storage size	Length of data + 2 (unless compressed)
Target types	All
Notes	<ul style="list-style-type: none"> ■ The length of the data is read from the SMALLINT field preceding the data. ■ If the data exceeds the LOAD max_length or runs off the end of the record, an error is generated and the load is terminated. ■ If the data exceeds the CREATE TABLE length, the data is silently truncated. ■ If CHARSET=65535, the data type is changed to VARBINARY. ■ If the CREATE TABLE length is greater than 4096, the data is compressed before being written. ■ If the target type is [VAR]BINARY, there is no conversion, that is, the input characters are copied directly. ■ When the target is a numeric type, a data field of only blanks is an error.

Converting data types

The data type in a datatype_spec does not have to match the data type in the column definition of a CREATE TABLE statement, but the data types must be compatible. The server data loader automatically performs any necessary conversions, but you need to be sure that the data type you provide in a datatype_spec can be converted to the data type of the user table.

For instance, if the data type of the input data is SMALLINT, but the data type of the column in the user table is INTEGER, then the server data loader converts the input data from SMALLINT to INTEGER.

The server data loader trims input data fields that are longer than the length provided in a column definition of a CREATE TABLE statement. The server data loader also rescales DECIMAL or NUMERIC columns. For example, if the input data is DECIMAL(7,3) but the column definition for the user table is DECIMAL(7,2), then the server data loader rescales the input data and stores it as DECIMAL(7,2) in the user table.

The following table identifies the allowable data type conversions. In the table:

- The *Input type* is the data type of the input data (LOAD data type).
- The *Target type* is the data type of the table being loaded (CREATE TABLE data type).

Synonyms are not listed but are supported. For instance, BYTE, a synonym for BINARY, has the same conversions as BINARY.

Data type conversions for loading

Input type	Target type													
	BINARY	BLOB	CHARACTER	CLOB	DATE	DOUBLE PRECISION	INTEGER	NUMERIC (DECIMAL)	REAL	SMALLINT	TIME	TIMESTAMP	VARBINARY	VARCHAR
BINARY	X	X	X	X									X	X
BINARY EXTERNAL	X	X	X	X		X	X		X	X			X	X
BLOB		X		X										
CHARACTER	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CLOB		X		X										
DATE					X									
DECIMAL							X	X		X				
DECIMAL EXTERNAL							X	X		X				
DOUBLE						X	X		X	X				
FLOAT(REAL)						X	X		X	X				
FLOAT EXTERNAL						X	X	X	X	X				
INTEGER							X			X				
INTEGER EXTERNAL							X			X				
SMALLINT							X			X				
TIME EXTERNAL											X			
TIMESTAMP EXTERNAL												X		
VARBINARY	X	X	X	X									X	X
VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Calculating the length of a data field

In a datatype_spec, you can optionally provide *explicit lengths* for most of the data types. For example, CHAR(10) describes a data field that consists of 10 bytes. Specifying CHAR without a length is also valid.

The data types DOUBLE, INTEGER, REAL, and SMALLINT have *implied lengths*. You don't specify lengths for these data types. Your host determines the size of these implied lengths. For example, INTEGER for an IBM S/370 or a SPARC implementation has an implied length of 4 bytes; for a DOS implementation, 2 bytes.

The server data loader calculates the length of a data field:

- With the starting and ending column numbers in the POSITION clause of a field_spec
- With the explicit or implied length of the data type in a datatype_spec

Because you can provide both a POSITION clause and a datatype_spec, it's possible that those lengths can conflict. You will receive a warning message when there is a conflict with the length of a data field. The server data loader determines the length of a data field in one of the following ways, *in the order listed*:

- For FLOAT, DOUBLE, INTEGER, REAL, and SMALLINT, the implied length is used. For instance, the implied length of SMALLINT is 2.
- If you specify a length for VARCHAR, VARBINARY, VARBYTE, or VARRAW, for instance, VARCHAR(25), then that length is the maximum number of characters or bytes in the data field; but, the actual length is in the 2-byte SMALLINT field preceding the data. The total length of a data field in a logical record is the actual data length plus 2. If you specify starting and ending column numbers in a POSITION clause, be sure to include the 2-byte SMALLINT field containing the actual length.
- For delimited data, if you specify a length for the data type or if you specify starting and ending column numbers, then that length is the maximum length of the data field. The actual length may vary based on the presence of the delimiter, but it cannot exceed the maximum length.

- If you don't specify an ending column number in the POSITION clause, then the length of the data type is used.

For example, the length of the following data field is 18 characters:

```
REP_LASTNAME POSITION (4) CHAR(18)
```

- If you specify starting and ending column numbers in the POSITION clause *and* the length of a data type, then the starting and ending column numbers are used. The length of the data type is ignored. For instance, the length of the following data field is 15 characters:

```
REP_LASTNAME POSITION (4:18) CHAR(18)
```

- If you don't specify an ending column number in the POSITION clause, and you specify a data type name but not an explicit length, then depending on the data type either the CREATE TABLE length or the *default length* of the loader data type determines the length of the data field. See the data type specifications to determine when the CREATE TABLE or default length is used.

For instance, the default length of the INTEGER EXTERNAL loader data type—which is 11—would determine the length of the following data field:

```
ORDER_NUM POSITION (4) INT EXTERNAL
```

The CREATE TABLE length would determine the length of the following data field if the column data type is CHAR or BINARY:

```
REP_LASTNAME POSITION (4) CHAR
```

- If you omit both the POSITION clause and the datatype_spec, the CREATE TABLE data type determines the data type and length.

Caution: In all cases, if the length of a data field in a datatype_spec is greater than the length of the data type defined in the CREATE TABLE statement, then that data field will be trimmed. For example, if a column is defined as CHAR(10)

in a CREATE TABLE statement, but the length of a data type in a datatype_spec is 11 characters, then the last character of the data field will not be loaded.

Specifying a character set for an individual data field

For CHAR, CLOB, VARCHAR, and EXTERNAL data fields, you can specify a character set for an individual data field by using the CHARSET keyword with a CCSID value in a datatype_spec. This character set overrides the character set in the CHARACTERSET clause of the LOAD DATA statement and in the CCSID keyword on the LOAD control statement. Use the CHARSET keyword only when you want to override one of these character sets or the default character set (EBCDIC) for an individual data field.

The CCSIDs you can specify with the CHARSET keyword are as follows:

- 500 for EBCDIC
- 819 for ISO 8859-1
- 850 for PC

For CHAR and VARCHAR data fields, you can also specify CCSID 65535 with the CHARSET keyword. Specifying CCSID 65535 with the CHARSET keyword for these data fields is a synonym for BINARY and VARBINARY, respectively. Specifying CCSID 65535 with any of the EXTERNAL data types is invalid.

For example, assume the character set in the CHARACTERSET clause is 819 (ISO 8859-1), but the character set of an ORDER_NUM data field in the input data is EBCDIC. In the datatype_spec, you would include this CHARSET keyword to specify a different character set for this data field only:

```
ORDER_NUM POSITION (1) INT EXTERNAL(4) CHARSET 500
```


Specifying a delimiter for an individual data field

For CHAR and any of the EXTERNAL data fields, you can include a `delimiter_spec` in a `datatype_spec` to:

- Specify a delimiter for an individual data field
- Override the default delimiter in a FIELDS clause for an individual data field
- Supplement the FIELDS clause for a particular data field

See “Generating `field_specs`, identifying NULL flags, specifying default delimiters and other defaults” on page 4-50 for the format of the delimiter specification.

For example, assume that the data fields are terminated. The default delimiter in the FIELDS clause is a comma, but an INTEGER EXTERNAL column called TRANSNUMBER is terminated by a colon instead of a comma. In this case, you would include a `delimiter_spec` in the `datatype_spec` to override the default delimiter in the FIELDS clause. Here’s the FIELDS clause that sets the default delimiter to a comma:

FIELDS TERMINATED BY ','

Here’s the `field_spec` that overrides the FIELDS clause for the TRANSNUMBER column only:

TRANSNUMBER INT EXTERNAL(8) TERMINATED BY ':'

Now assume the FIELDS clause is the same (comma as a terminator), but the TRANSNUMBER data field is also enclosed by double quotes. In this case, the FIELDS clause would specify the terminator delimiter and the `delimiter_spec` in the `datatype_spec` would specify the enclosure delimiter. Here’s the `field_spec` that supplements the FIELDS clause for the TRANSNUMBER column only:

TRANSNUMBER INT EXTERNAL(8) ENCLOSED BY '"'

Specifying a BLOB or CLOB data type

When specifying a BLOB or CLOB data type to describe LOB data fields in the input data file, note the following guidelines:

- LOB data fields must be specified last in a field specification list of a LOAD DATA statement.
- Only BLOB and CLOB column (target) data types are valid for the BLOB and CLOB loader data types.
- LOB data fields must be relatively positioned, that is, POSITION(*).
- The CONTINUEIF and CONCATENATE clauses are supported if LOB data records are present, but these clauses do not apply to LOB records.
- LOB records are not collected in discard files.
- LOB data fields cannot be sent to multiple output segments, that is you can't specify a fixed start position with multiple INTO TABLE clauses for different tables. This also applies to using the same table name with the DIFFERENT SEGMENT clause. For example, if you have LOB data in the input data file, you can't specify the following:

```
LOAD
INTO TABLE table_1 (f1 POSITION(1) char(5), lobcol CLOB(20M))
...
INTO TABLE table_2 (f1POSITION(1) char(5), anotherlobcol CLOB(20M))
...
```

- You cannot use the NULLIF and DEFAULTIF clauses to specify a condition for LOB data fields. For instance, you cannot specify the following:

```
INTO TABLE x
( not_a_lob POSITION(5) CHAR(1),
```



```
lobcol CLOB(4M) NULLIF lobcol = 'abc',
```

```
...
```

But you can include a NULLIF or DEFAULTIF clause for a LOB data field if the condition specifies a non-LOB data field or a NULL flag. For example, you can specify the following:

```
INTO TABLE x
( not_a_lob POSITION(5) CHAR(1),
  lobcol CLOB(4M) NULLIF not_a_lob = 'N',
```

```
...
```

Loading a column with a null value

You can set a column's value to a null value by using the NULLIF clause and specifying a field condition in a field_spec. A column is loaded with a null value if the condition is true. The column in the user table must be defined as NULL, otherwise the load operation will fail.

The field condition that you specify with the NULLIF clause is the same as the field condition described at “Format of WHEN clause” on page 4-43. For clarity, you can optionally enclose the field condition in parentheses.

Note: The NULLIF clause differs from the TRAILING NULLCOLS clause described on page 4-59. Use the NULLIF clause to replace the value of a data field that's in a logical record. Use the TRAILING NULLCOLS clause when that data field is missing. A NULLIF clause in a FIELDS clause applies to any data field that doesn't have its own NULLIF clause in a field_spec.

For example, suppose you want to load a null value into the INITIAL column of a user table whenever a data field consists of blanks. You could specify the following field_spec, using the column name and the BLANKS keyword with the NULLIF clause:

```
INITIAL POSITION (15) CHAR NULLIF INITIAL=BLANKS
```


Or you could specify the starting column number and the BLANKS keyword with the NULLIF clause:

INITIAL POSITION (15) CHAR NULLIF (15)=BLANKS

Setting a column to the default value

When creating a user table, you can optionally define a default value for each column. If no default value was specified, the default is a null value. A default value can be:

- A literal—string, numeric, or binary—for a column of like data type
- A null value for a column of any data type
- The current date
- The current time
- The StorHouse account ID used to load the data for CHAR or VARCHAR columns

The server data loader loads a column's default value when you use the DEFAULTIF clause in a FIELDS clause or in a field_spec or when you omit the field_spec for the data field. A DEFAULTIF clause describes a field condition that when true, sets the column to the default value. The field condition is the same as the one described at “Format of WHEN clause” on page 4-43. For clarity, you can enclose the DEFAULTIF field condition in parentheses.

For example, suppose you want to load a default value into a STATE column whenever a data field consists of blanks. You could specify the column name and the BLANKS keyword with the DEFAULTIF clause:

STATE POSITION (20:21) CHAR DEFAULTIF (STATE=BLANKS)

Or you could specify the column numbers and the BLANKS keyword with the DEFAULTIF clause:

STATE POSITION (20:21) CHAR DEFAULTIF (20:21)=BLANKS

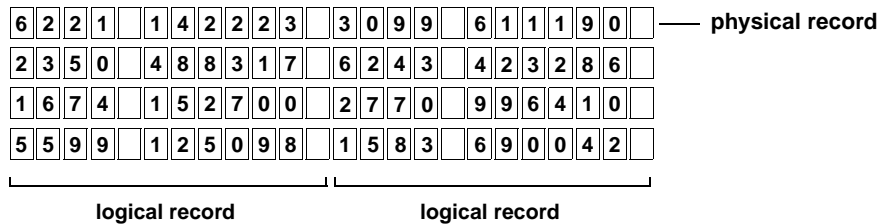
Using multiple into_table_specs

By using multiple into_table_specs in a LOAD DATA statement, you can:

- Create multiple logical records from one physical record
- Load data from the same input dataset into different user tables
- Load different segments of the same user table or multiple user tables (see page 4-63)

Creating multiple logical records from one physical record

You can create multiple logical records from one physical record by using multiple into_table_specs in a LOAD DATA statement. For instance, suppose you want to split each physical record into two logical records. Each logical record will contain a TRANSNUMBER column and an ACCOUNT_NUMBER column.



You could use the following into_table_specs to split each physical record into two logical records and load them into one user table called ATM.TRANSACTIONS:

```
LOAD
  INTO TABLE ATM.TRANSACTIONS
  (TRANSNUMBER POSITION (1) INT EXTERNAL(4),
  ACCOUNT_NUMBER POSITION (6) INT EXTERNAL(6))
```



```

INTO TABLE ATM.TRANSACTIONS
(TRANSACTION_POSITION (13) INT EXTERNAL(4),
ACCOUNT_NUMBER_POSITION (18) INT EXTERNAL(6));

```

Using the same input dataset to load multiple user tables

You can load multiple user tables with the same or different records in an input dataset by including multiple `into_table_specs` in one `LOAD DATA` statement. For instance, suppose:

- You're loading data into two user tables: JACK.ORDERS and SUE.ORDERS.
- Input data records contain a field in position 1 called RECORD_CODE.
- You'll load all input data records into Jack's table.
- You'll load only those records with a RECORD_CODE of B into Sue's table.

Below are some of the input data records:

A	2	8	3	9	M	c	G	u	i	r	e							J	a	c	k										
B	2	3	8	8	C	o	r	n	f	l	a	k	e						S	u	e										
A	1	4	3	9	M	c	G	u	i	r	e								J	a	c	k									
B	3	0	9	5	C	o	r	n	f	l	a	k	e							S	u	e									

You would use the following LOAD DATA statement to load these two user tables:

```
LOAD
INTO TABLE &&0.ORDERS
(ORDER_NUM POSITION (2:5) INT EXTERNAL,
REP_LASTNAME POSITION (6:20) CHAR,
REP_FIRSTNAME POSITION (21:35) CHAR)
```



```
INTO TABLE &&1.ORDERS
WHEN RECORD_CODE = 'B'
(:RECORD_CODE POSITION (1) CHAR,
ORDER_NUM POSITION (2:5) INT EXTERNAL,
REP_LASTNAME POSITION (6:20) CHAR,
REP_FIRSTNAME POSITION (21:35) CHAR);
```

Note the following:

- Each INTO TABLE clause contains a symbolic variable for the owner name. In the LOAD control statement, you would supply the following substitution string in the Pn keyword:

```
LOAD P0=JACK P1=SUE
```

- The WHEN clause contains the field name RECORD_CODE. The field_spec describes the :RECORD_CODE field.
- The starting and ending column numbers in the POSITION clauses determine the lengths of the data fields.

Example LOAD DATA statements

This section contains examples of the LOAD DATA statement.

- Example 1 describes how to load all data records into one user table. This LOAD DATA statement contains one into_table_spec.
- Example 2 describes how to combine a fixed number of records and then load some of them into one user table. This LOAD DATA statement contains a CONCATENATE clause, one WHEN clause, and one into_table_spec.
- Example 3 describes how to load delimited data into two different user tables. This LOAD DATA statement contains two into_table_specs and two FIELDS clause.

- Example 4 explains how to combine a variable number of records and then load all of them into one user table. This LOAD DATA statement contains the following clauses and keyword: CONTINUEIF, INTO TABLE, TRAILING NULLCOLS, and NULLIF.
- Example 5 illustrates how to load SMALLINT, DECIMAL, and VARCHAR data. This LOAD DATA statement explains how to use relative positioning and illustrates numeric and variable-length data fields in data records.
- Example 6 describes how to use relative positioning to load delimited data into multiple user tables. This LOAD DATA statement contains four into_table_specs and three WHEN clauses.
- Example 7 illustrates how to use multiple selection criteria to load specific records into multiple user tables. This LOAD DATA statement contains three into_table_specs and three WHEN clauses with the AND and OR keywords.
- Example 8 explains how to replace a segment without loading data. This LOAD DATA statement contains three INTO TABLE clauses and three REPLACE SEGMENT clauses.
- Example 9 shows how to include SQL statements in the SYSSQL dataset. This example includes one CREATE TABLE statement, two CREATE INDEX statements, and one LOAD DATA statement.
- Example 10 describes how to load two segments of the same user table, selecting subspaces for each component type. This LOAD DATA statement contains two into_table_specs, six SUBSPACE number clauses, one DIFF SEGMENT clause, and two WHEN clauses.
- Example 11 shows how to load LOB data fields using a default field list and specifying the NULLFLAGS keyword.

Example 2: Combining a fixed number of records and loading some of them into one user table

Assume you're loading data into a user table called SUE.ORDER_DETAILS. Each pair of physical records must be combined. You'll load only those records that contain a REP_LASTNAME of Cornflake.

CREATE TABLE statement

```
CREATE TABLE SUE.ORDER_DETAILS
  (ORDER_NUM SMALLINT NOT NULL,
  REP_LASTNAME CHAR(15) NOT NULL,
  REP_FIRSTNAME CHAR(10) NOT NULL,
  BUYER_LASTNAME CHAR(15) NOT NULL,
  BUYER_FIRSTNAME CHAR(10) NOT NULL,
  STATE CHAR(2) NOT NULL)
TABLE SPACE ORDERS2000
```

Data records

Two physical records make up one logical record.

2	8	3	9	M	c	G	u	i	r	e							J	a	c	k							
W	h	i	t	e													S	a	n	d	y					C	A
2	3	8	8	C	o	r	n	f	l	a	k	e						S	u	e							
H	i	l	i														R	i	c	h	a	r	d			N	M
1	4	3	9	M	c	G	u	i	r	e							J	a	c	k							
B	a	y	l	i	g	h	t										M	a	u	r	e	e	n			P	A

LOAD DATA statement

```
LOAD
  CONCATENATE 2
  INTO TABLE SUE.ORDER_DETAILS
  WHEN REP_LASTNAME = 'Cornflake'
  (ORDER_NUM POSITION (1) INT EXTERNAL(4),
  REP_LASTNAME POSITION (5) CHAR(15),
  REP_FIRSTNAME POSITION (20) CHAR(10),
  BUYER_LASTNAME POSITION (30) CHAR(15),
  BUYER_FIRSTNAME POSITION (45) CHAR(10),
  STATE POSITION (55) CHAR(2));
```


Example 3: Loading delimited data into multiple user tables

Assume you're loading terminated data into two user tables: JACK.ORDERS and SUE.ORDERS. You'll load all records into both tables.

CREATE TABLE statements

```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS2000
```

```
CREATE TABLE SUE.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS2000
```

Data records

The data fields are terminated by a comma.

```
2839,McGuire,Jack
2388,Cornflake,Sue
1439,McGuire,Jack
3095,Cornflake,Sue
```

LOAD DATA statement

```
LOAD
INTO TABLE JACK.ORDERS
FIELDS TERMINATED BY ','
(ORDER_NUM INT EXTERNAL(4),
 REP_LASTNAME CHAR(15),
 REP_FIRSTNAME CHAR(15))

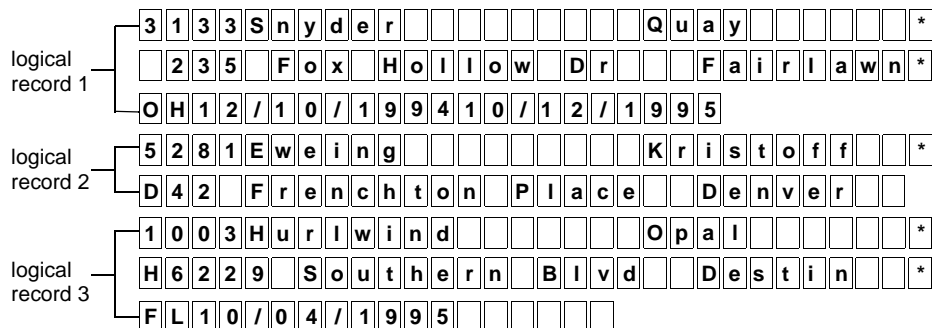
INTO TABLE SUE.ORDERS
FIELDS TERMINATED BY ','
(ORDER_NUM POSITION(1) INT EXTERNAL(4),
 REP_LASTNAME CHAR(15),
 REP_FIRSTNAME CHAR(15));
```


Example 4: Combining a variable number of records and loading null values

Assume that you're loading all data records into a user table called SUE.ACCOUNTS.

CREATE TABLE statement	CREATE TABLE SUE.ACCOUNTS (ACCOUNT_NUM SMALLINT NOT NULL, LASTNAME CHAR(15) NOT NULL, FIRSTNAME CHAR(10) NOT NULL, INITIAL CHAR(1), ADDRESS CHAR(20) NOT NULL, CITY CHAR(8) NOT NULL, STATE CHAR(2), FIRST_ORDER DATE, LAST_ORDER DATE) TABLE SPACE ORDERS2000
-----------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Data records A variable number of physical records make up one logical record. Some data fields in logical records 1 and 3 contain blanks. Some data fields in logical record 2 are missing.



**LOAD DATA
statement**

```
LOAD
CONTINUEIF (30) = '*'
INTO TABLE SUE.ACCOUNTS
TRAILING NULLCOLS
(ACCOUNT_NUM POSITION (1) INT EXTERNAL(4),
LASTNAME POSITION (5) CHAR(15),
FIRSTNAME POSITION (20) CHAR(10),
INITIAL POSITION (30) CHAR(1) NULLIF INITIAL=BLANKS,
ADDRESS POSITION (31) CHAR(20),
CITY POSITION (51) CHAR(8),
STATE POSITION (59) CHAR(2),
FIRST_ORDER POSITION (61) DATE EXTERNAL(10) NULLIF
FIRST_ORDER=BLANKS,
LAST_ORDER POSITION (71) DATE EXTERNAL(10) NULLIF
LAST_ORDER=BLANKS);
```

In this example:

- CONTINUEIF THIS is the default, which means that the current physical record will be combined with the next one.
- The INITIAL data field in logical record 1 is blank; but a null value will be loaded because the NULLIF condition is true.
- Logical record 2 has three missing data fields—STATE, FIRST_ORDER, LAST_ORDER—but null values will be loaded because of the TRAILING NULLCOLS clause.
- The LAST_ORDER data field in logical record 3 is blank; but a null value will be loaded because the NULLIF condition is true.

Example 5: Loading SMALLINT, DECIMAL, and VARCHAR data

Suppose you're loading all data records into a user table called SALES_BY_LOCATION.

CREATE TABLE statement	<pre>CREATE TABLE SALES_BY_LOCATION (LOCATION_ID SMALLINT, COST DECIMAL(5,3), LAST_NAME VARCHAR(64), FIRST_NAME CHAR(12)) TABLE SPACE FY2000</pre>
-------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------

LOAD DATA statement	<pre>LOAD DATA INTO TABLE SALES_BY_LOCATION (LOCATION_ID POSITION(1) SMALLINT, COST POSITION(3) DECIMAL(4,2), LAST_NAME POSITION(6) VARCHAR(60), FIRST_NAME POSITION(*+2) VARCHAR(12));</pre>
----------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Note these differences between the CREATE TABLE statement and the LOAD DATA statement:

- In CREATE TABLE, COST is data type DECIMAL(5,3).
In LOAD DATA, COST is data type DECIMAL(4,2).

The server data loader will rescale the length of the input data to DECIMAL(5,3).

- In CREATE TABLE, LAST_NAME is data type VARCHAR(64).
In LOAD DATA, LAST_NAME is data type VARCHAR(60).

A length in a LOAD DATA statement can be less than the length in a CREATE TABLE statement.

4

SYSSQL dataset

Example LOAD DATA statements

- In CREATE TABLE, FIRST_NAME is data type CHAR(12).
In LOAD DATA, FIRST_NAME is data type VARCHAR(12).

You can load a VARCHAR data field into a CHAR column as long as the data field isn't longer than the target column.

Data records Data is represented in hex, two digits per column. Character data is represented in EBCDIC (CCSID 500) coding. For simplicity, only the characters A (x'C1'), b (x'82'), c (x'83') and D (x'C4') are used. All data fields are small to simplify the example.

column numbers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																			
record 1	0	4	F	2	0	1	2	3	4	C	0	0	0	2	C	1	8	2	4	0	4	0	0	0	0	1	8	3						
record 2	0	0	1	2	0	5	6	7	8	C	0	0	0	4	C	4	8	3	8	2	8	2	4	0	4	0	0	0	0	2	C	1	C	1

Below are descriptions of the data fields in record 1:

Column/field name	Position	Input value	Loaded value
LOCATION_ID	1 and 2	04F2	1266
COST	3 through 5	01234C	+12.340
LAST_NAME	6 through 9	0002C182	Ab
*	10 and 11	4040	not loaded
FIRST_NAME	12 through 14	000183	c

Below are descriptions of the data fields in record 2:

Column/field name	Position	Input value	Loaded value
LOCATION_ID	1 and 2	0012	18
COST	3 through 5	05678	+56.780
LAST_NAME	6 through 11	0004C4838282	Dcbb
*	12 and 13	4040	not loaded
FIRST_NAME	14 through 15	0002C1C1	AA

Notice that the COST columns are rescaled to DECIMAL(5,3). Also notice the relative positioning of FIRST_NAME. In record 1, LAST_NAME ends in column 9; therefore, the relative position field * starts in column 10. So *+2 is column 12. The length field of FIRST_NAME (hex 0001) begins in column 12. In record 2, LAST_NAME ends in column 11; therefore, the length field of FIRST_NAME (hex 0002) begins in column 14.

Example 6: Using relative positioning to load delimited data into multiple user tables

Assume you're loading data into two user tables: EMPLOYEE and PROJECT.

CREATE TABLE statements

```
CREATE TABLE EMPLOYEE
(EMP_NUM SMALLINT NOT NULL,
EMP_INITIALS CHAR(3) NOT NULL,
DEPT_NUM SMALLINT NOT NULL)
TABLE SPACE EMPLOYEE
```

```
CREATE TABLE PROJECT
(EMP_NUM SMALLINT NOT NULL,
PROJ_ID SMALLINT NOT NULL)
TABLE SPACE EMPLOYEE
```



```
LOAD
  INTO TABLE EMPLOYEE
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM POSITION(4) INTEGER EXTERNAL(2),
  EMP_INITIALS CHAR(3),
  DEPT_NUM INTEGER EXTERNAL(2))

  INTO TABLE PROJECT
  WHEN (1:1) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3))

  INTO TABLE PROJECT
  WHEN (2:2) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3))

  INTO TABLE PROJECT
  WHEN (3:3) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3));
```

The server data loader checks each `into_table_spec` for each logical data record. If the `WHEN` clause is true or there is no `WHEN` clause, data is loaded into the table named in the `INTO TABLE` clause. The server data loader then sets the relative position to the column number that follows the last field. The next selected `into_table_spec` will (if relative) start at this position.

So in this example:

1. The server data loader processes the first data record.
 - a. The server data loader checks the first `into_table_spec`, which doesn't contain a `WHEN` clause. Starting at column 4, the server data loader then

loads the EMP_NUM (value of 39), the EMP_INITIALS (value of WTC), and the DEPT_NUM (value of 12) into the EMPLOYEE table.

b. The server data loader checks the remaining three into_table_specs, whose WHEN clauses are false (the record doesn't contain the Y flag in at least one of the first three characters).

2. The server data loader then processes the second data record.

a. The server data loader checks the first into_table_spec, which doesn't contain a WHEN clause. Starting at column 4, the server data loader then loads the EMP_NUM (value of 40), the EMP_INITIALS (value of SSC), and the DEPT_NUM (value of 12) into the EMPLOYEE table.

b. The server data loader checks the second into_table_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a DEPT_NUM value of 12), the server data loader then loads the EMP_NUM (value of 40) and the PROJ_ID (value of 944) into the PROJECT table.

c. The server data loader checks the third and fourth into_table_specs, whose WHEN clauses are both false.

3. The server data loader then processes the third data record.

a. The server data loader checks the first into_table_spec, which doesn't contain a WHEN clause. Starting at column 4, the server data loader then loads the EMP_NUM (value of 41), the EMP_INITIALS (value of JED), and the DEPT_NUM (value of 01) into the EMPLOYEE table.

b. The server data loader checks the second into_table_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a DEPT_NUM value of 01), the server data loader then loads the

EMP_NUM (value of 41) and the PROJ_ID (value of 908) into the PROJECT table.

- c. The server data loader checks the third and fourth into_table_specs, whose WHEN clauses are both false.
4. The server data loader finally processes the fourth record.
 - a. The server data loader checks the first into_table_spec, which doesn't contain a WHEN clause. Starting at column 4, the server data loader then loads the EMP_NUM (value of 42), the EMP_INITIALS (value of KAK), and the DEPT_NUM (value of 01) into the EMPLOYEE table.
 - b. The server data loader checks the second into_table_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a DEPT_NUM value of 01), the server data loader then loads the EMP_NUM (value of 42) and the PROJ_ID (value of 968) into the PROJECT table.
 - c. The server data loader checks the third into_table_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a PROJ_ID value of 968), the server data loader then loads the EMP_NUM (value of 42) and the PROJ_ID (value of 911) into the PROJECT table.
 - d. The server data loader checks the fourth into_table_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a PROJ_ID value of 911), the server data loader then loads the EMP_NUM (value of 42) and the PROJ_ID value of 912) into the PROJECT table.

Example 7: Using multiple selection criteria

Assume you're loading data into three user tables: TOLLFREE, TOLLCALL, and JUSTLOCALS.

**CREATE TABLE
statements**

```
CREATE TABLE TOLLFREE
(FROMNUM BINARY(5) NOT NULL,
TONUM BINARY(5) NOT NULL,
ACCOUNT CHAR(12) NOT NULL)
TABLE SPACE MONTHLYCALLS
```

```
CREATE TABLE TOLLCALL
(FROMNUM BINARY(5) NOT NULL,
TONUM BINARY(5) NOT NULL,
ACCOUNT CHAR(12) NOT NULL)
TABLE SPACE MONTHLYCALLS
```

```
CREATE TABLE JUSTLOCALS
(FROMNUM BINARY(5) NOT NULL,
TONUM BINARY(5) NOT NULL,
ACCOUNT CHAR(12) NOT NULL)
TABLE SPACE MONTHLYCALLS
```

Data records

```
30155512348005551212acct1234567
30125110004105554321acct12345679
60755598768881234567acct98765432
30150512346075154321acct76456732
```

**LOAD DATA
statement**

This load data statement contains three into_table_specs. You can use any case (except in quoted strings). This example illustrates lowercase.

```
load
into table tollfree
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when (11:13) = '800' or (11:13) = '888'
```



```
into table tollcall
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when (11:13) != '800' and (11:13) != '888'
```

```
into table justlocals
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when ((1:3) = '301' or (1:3) = '410') and ((11:13) = '301' or (11:13) = '410');
```

In this example:

- The first into_table_spec specifies that a record containing 800 or 888 in positions 11 through 13 be loaded into the tollfree table.
- The second into_table_spec specifies that a record that *does not* contain 800 or 888 in positions 11 through 13 be loaded into the tollcall table.
- The third into_table_spec specifies that a record containing either 301 or 410 in positions 1 through 3 *and* containing either 301 or 410 in positions 11 through 13 be loaded into the justlocals table.

Load results

The server data loader would load the following data into the three tables.

Table	FROMNUM	TONUM	ACCOUNT
tollfree	3015551234	8005551212	acct1234567
	6075559876	8881234567	acct98765432
tollcall	3012511000	4105554321	acct12345679
	3015051234	6075154321	acct76456732
justlocals	3012511000	4105554321	acct12345679

Example 8: Replacing segments without loading

To replace a segment without loading, you must create a LOAD DATA statement with the following:

- LOAD keyword
- INTO TABLE clause(s)
- REPLACE clause(s)

Note: When replacing segments without loading data, the SYSREC DD statement must name an empty dataset. You can use DD* followed by /*.

Assume you need to replace the segments created when the tollfree, tollcall, and justlocals user tables were loaded with the LOAD DATA statement on page 4-126. The server data loader created one segment for each user table. The SEGMENT clause was omitted, so the load ID is the default segment tag for all segments. Assume the load ID was 00343445081.

Note: Because the load ID starts with a digit, you must delimit it with double quotes on the REPLACE clause.

LOAD DATA statement

```
load  
into table tollfree  
replace segment "00343445081"
```

```
into table tollcall  
replace segment "00343445081"
```

```
into table justlocals  
replace segment "00343445081";
```


Example 9: Including SQL statements in the SYSSQL dataset

SQL statements in a SYSSQL dataset must precede the LOAD DATA, LOAD INDEX, or MERGE statement. Each statement must end with a semicolon. For example, the following SYSSQL dataset contains CREATE TABLE and CREATE INDEX statements as well as a LOAD DATA statement. A StorHouse engine executes and commits each SQL statement when it completes.

SYSSQL dataset

```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
REP_LASTNAME CHAR(15) NOT NULL,
REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS2000;
```

```
CREATE HASH INDEX ORDER_IDX
ON JACK.ORDERS
(ORDER_NUM);
```

```
CREATE HASH INDEX REP_IDX
ON JACK.ORDERS
(REP_LASTNAME, REP_FIRSTNAME);
```

```
LOAD
INTO TABLE JACK.ORDERS
(ORDER_NUM POSITION (1) INT EXTERNAL(4),
REP_LASTNAME POSITION (5) CHAR(15),
REP_FIRSTNAME POSITION (20) CHAR(15));
```

Example 10: Selecting subspaces for each component type

Assume you're loading two segments of a user table called CALLSDBA.BILLSUMMARY.

**CREATE TABLE
SPACE
statement**

The tablespace contains six subspaces, two for each component type.

```
CREATE TABLE SPACE BILLING
(SUBSPACE 1 VSET TCAT1 FSET TCAT1 OBJECT_TYPE T,
SUBSPACE 2 VSET HCAT1 FSET HCAT1 OBJECT_TYPE H,
SUBSPACE 3 VSET VCAT1 FSET VCAT1 OBJECT_TYPE V,
SUBSPACE 4 VSET TCAT2 FSET TCAT2 OBJECT_TYPE T,
SUBSPACE 5 VSET HCAT2 FSET HCAT2 OBJECT_TYPE H,
SUBSPACE 6 VSET VCAT2 FSET VCAT2 OBJECT_TYPE V)
```

**CREATE TABLE
statement**

The user table is assigned to the BILLING user tablespace.

```
CREATE TABLE CALLSDBA.BILLSUMMARY
(BILL_ACCOUNT BINARY(5) NOT NULL,
BILL_DATE DATE NOT NULL,
BILL_CATEGORY CHAR(1) NOT NULL,
NUMBER_CALLS INTEGER)
TABLE SPACE BILLING
```

**CREATE INDEX
statements**

The indexes are assigned to the same user tablespace (BILLING) as the user table (no TABLE SPACE clause on CREATE INDEX).

```
CREATE HASH INDEX BILLACCOUNT
ON CALLSDBA.BILLSUMMARY (BILL_ACCOUNT)
```

```
CREATE HASH INDEX BILLCATEGORY
ON CALLSDBA.BILLSUMMARY (BILL_CATEGORY)
```

```
CREATE VALUE INDEX BILLDATE
ON CALLSDBA.BILLSUMMARY (BILL_DATE)
```

Data records

```
301792833901/31/20001004,
703274028301/31/20001002,
703872283901/31/20002001,
301339439201/31/20001003,
301340920301/31/20001002,
703419408301/31/20002003,
703229383901/31/20002001,
```


**LOAD DATA
statement**

This LOAD DATA statement creates two table data files, four hash index files, and two value index files. Data records containing a BILL_CATEGORY of 1 are loaded into the first segment and those containing a BILL_CATEGORY of 2 are loaded into the second segment.

```
LOAD
  INTO TABLE CALLSDBA.BILLSUMMARY
  TABLE SUBSPACE 1
  HASH SUBSPACE 2
  VALUE SUBSPACE 3
  (BILL_ACCOUNT BINARY EXTERNAL(10),
  BILL_DATE DATE EXTERNAL(10),
  BILL_CATEGORY CHAR(1),
  NUMBER_CALLS INT EXTERNAL(3))
  WHEN BILL_CATEGORY='1'

  INTO TABLE CALLSDBA.BILLSUMMARY
  DIFFERENT SEGMENT
  TABLE SUBSPACE 4
  HASH SUBSPACE 5
  VALUE SUBSPACE 6
  (BILL_ACCOUNT POSITION(1) BINARY EXTERNAL(10),
  BILL_DATE DATE EXTERNAL(10),
  BILL_CATEGORY CHAR(1),
  NUMBER_CALLS INT EXTERNAL(3))
  WHEN BILL_CATEGORY='2';
```

In this example, the server data loader uses the following subspaces for the first segment:

- Subspace 1 for the first table data file
- Subspace 2 for the two hash index files
- Subspace 3 for the value index file

The server data loader uses the following subspaces for the second segment:

- Subspace 4 for the table data file
- Subspace 5 for the two hash index files
- Subspace 6 for the value index file

Note: In this example, SUBSPACE ROTATE has the same effect.

Load results

The server data loader loads the following data into each segment:

Segment	BILL_ ACCOUNT	BILL_ DATE	BILL_ CATEGORY	NUMBER_ CALLS
1	3017928339	01/31/2000	1	004
	7032740283	01/31/2000	1	002
	3013394392	01/31/2000	1	003
	3013409203	01/31/2000	1	002
2	7038722839	01/31/2000	2	001
	7034194083	01/31/2000	2	003
	7032293839	01/31/2000	2	001

Example 11: Loading LOB data fields using a default field list and NULLFLAGS

This example shows how to load LOB data fields in the input data file using all the LOAD DATA statement defaults and specifying the NULLFLAGS keyword.

CREATE TABLE statement

The user table contains two CLOB columns and one CHAR column.

```
create table insteamlobs
(f1 clob, f2 clob, f3 char(5))
table space sm;
```


Data records The data is in var format. Binary data is displayed in hex and enclosed in brackets.

```
[002d]FFFline1[000000000000000f]f1lob1
contents[0000000000000006]f2lob1
[0008]FFTline2
[0008][000000000000000f]
[0006]f1lob2
[0011] contents[0000000000000000]
[001e]TTF    [0000000000000000][0000000000000006]f2row3
```

Note the following:

- The first record contains all of the data to be loaded into the first row of the table. All values fit within the record.
- The data to be loaded into the second row of the table is spread across four records. The first record contains the non-LOB field. The second, third, and fourth records contain a LOB value. The fourth record also contains a placeholder [0000000000000000] for a NULL value.
- The fifth record contains the data to be loaded into the third row of the table. Two data fields are NULL. The record contains data (blanks and [0000000000000000]) as a placeholder for those fields.

LOAD DATA statement The LOAD DATA statement simply contains the FIELDS clause with the NULLFLAGS keyword.

```
load data
into table instreamlobs
fields nullflags;
```


Load results Here's the content of the loaded table. All blanks indicate NULL data.

instreamlobs table

f1	f2	f3
f1lob1contents	f2lob1	line1
f1lob2contents		line2
2row3		

Loading a deferred index

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

Note the following guidelines:

- You can load multiple indexes for the same table in one operation.
- You can specify a range or list of segments to load.
- All metadata inserts and updates are performed after each segment is processed, rather than when the operation is confirmed.
- Only one LOAD INDEX statement is allowed in a SYSSQL dataset.
- Other non-load statements, such as CREATE INDEX, are allowed in a SYSSQL dataset but must precede the LOAD INDEX statement. You can use this feature to create the deferred indexes and then load them.
- You identify segments by segment ID with the SEGMENTS clause. Segment IDs are listed in the SYSSTHSEGMENTS system table. If you specify an

invalid segment, the data loader ignores it. The SEGMENTS clause applies to all types of indexes.

- The SUBSPACE ROTATE and SUBSPACE number clauses do not apply to range indexes.

Format of LOAD INDEX statement

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

where subspace_clause is:

```
SUBSPACE ROTATE |  
[ VALUE | HASH ] SUBSPACE number
```


Argument	Format
index_name	(required) Name of the index to be loaded. You can specify multiple index names to load multiple indexes (for the same table) in one operation.
SUBSPACE ROTATE	<p>(optional) Clause to rotate value indexes or hash indexes among subspaces.</p> <ul style="list-style-type: none"> ■ This clause is useful only when loading indexes for multiple segments. ■ SUBSPACE ROTATE does not apply to range indexes. ■ If you omit this clause and the SUBSPACE number clause, the server data loader uses the lowest-numbered subspace for the index type.
SUBSPACE number	<p>(optional) Clause to select a specific subspace for the index type, where number is the subspace number.</p> <ul style="list-style-type: none"> ■ If you omit the VALUE or HASH keyword, the server data loader uses the same subspace number for both value and hash indexes. ■ If you omit this clause and the SUBSPACE ROTATE clause, the server data loader uses the lowest-numbered subspace for the index type.
SEGMENTS segment_list	<p>(optional) One or more segment IDs to load indexes for specific segments. If you omit this clause, the server data loader creates index entries for all segments of the table and changes the status of the index from deferred to complete.</p> <p>The format of segment_list is:</p> <p>segment_list_item [, segment_list_item]...</p>
segment_list_item	segment_range segment
segment_range	first_segment - last_segment

Example LOAD INDEX statements

This section contains example LOAD INDEX statements.

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

```
LOAD INDEX ORDERS2000
```

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

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

```
LOAD INDEX ORDERS2000 SEGMENTS 0 SUBSPACE 2
```

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

```
LOAD INDEX ORDERS2000 SEGMENTS 0-5 SUBSPACE ROTATE
```

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

```
LOAD INDEX ORDERS2000, ORDERSDETAIL
```

Merging segments of a table

You can merge segments in a table by using the MERGE or COALESCE statement in a SYSSQL dataset. A *merge*, or *coalesce*, operation consolidates a group of segments into one segment (possibly more) and invalidates the input segments. You have full control over which segments are grouped, for instance, by specific segment IDs or segment tags or by size criteria. Merging segments enhances

performance because it reduces the number of file and extent opens and closes for a query.

Note the following guidelines:

- LOB subsegment files are not merged. The segment ID in a LOB file name, then, will be different from the other files in the segment. LOB file names have the original segment ID, and the table data file and index files have the new segment ID.
- The server data loader invalidates input segments after merging them but does not remove the invalidated segments. See “Replacing a segment” on page 4-68 for more information about invalidated segments.
- The server data loader automatically commits the operation after creating the result segment. You should still confirm the merge operation to delete the checkpoint.
- If the group of segments to be merged consists of only one segment, the server data loader does not merge the segment into a new one.

Caution: Do not run MERGE and REPLACE SEGMENT operations at the same time against the same table. StorHouse/RM may not merge the segments because the REPLACE SEGMENT operation indicates a change to the table.

Format of MERGE statement

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


Argument	Format
table_name	(required) Name of the table with the segments to be merged.
SUBSPACE ROTATE	(optional) Clause to rotate the components (table data, value indexes, hash indexes) among subspaces. This clause is useful for merge operations that produce multiple result segments. If you omit this clause and the SUBSPACE number clause, the server data loader uses the lowest-numbered subspace for the component type.
SUBSPACE number	<p>(optional) Clause to select specific subspaces for components in a result segment, where number is the subspace number.</p> <ul style="list-style-type: none"> ■ If you omit the TABLE, VALUE, or HASH keyword, the server data loader uses the same subspace number for all components. ■ If you omit this clause and the SUBSPACE ROTATE clause, the server data loader uses the lowest-numbered subspace for the component type.
SEGMENT segment_tag	(optional) Name of the result segment. A segment tag cannot exceed 40 characters and must follow SQL identifier conventions (see page 4-4) or be quoted. If you omit this clause, the server data loader uses the load ID as the segment tag.
SEGMENTS segment_list	<p>(optional) One or more input segments to be merged. You can identify the input segments by segment ID or segment tag. If you omit this clause, the server data loader considers all segments in the table. If you specify an invalidated or non-existent segment, the data loader issues a warning message but continues the operation.</p> <p>The format of segment_list is:</p> <p>IDS segment_list_item [, segment_list_item]... TAGS segment_tag [, segment_tag]...</p>
segment_list_item	segment_range segment
segment_range	first_segment - last_segment
EXCLUDE segment_list	(optional) One or more segments to exclude from the merge operation. You can identify excluded segments by segment ID or segment tag. If you omit this clause, the server data loader includes all segments in the table. The format of the segment_list is the same as for the SEGMENTS clause.

Argument	Format
MAXINSIZE n	(optional) Maximum size of input segments. This clause excludes segments that are larger than the specified number of bytes or a number followed by K(x1024), M(xKK), or G(xKM). If you omit this clause, the server data loader assumes no size limit on input segments.
MINOUTSIZE n	<p>(optional) Minimum size of the result segment. This clause groups the input list into a result segment that is no smaller than the specified value. If you omit this clause, the server data loader assumes no minimum size limit on result segments.</p> <ul style="list-style-type: none">■ If there aren't enough qualifying segments to create a group of the minimum size, the data loader does not merge them.■ When grouping segments, once the size of the result segment exceeds this value, the server data loader starts to group segments for another result segment.■ The size (n) is the number of bytes or a number followed by K(x1024), M(xKK), or G(xKM).■ If you specify MINOUTSIZE but not MAXINSIZE, the server data loader implicitly sets MAXINSIZE to the same limit. This prevents a segment that is already big enough from being added to a small segment or a group of segments.

Example MERGE statements

This section contains example MERGE statements.

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

```
MERGE INTO TABLE CALLSDetail
```

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

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


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

COALESCE INTO TABLE CALLSDetail EXCLUDE IDS 0

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

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

4

SYSSQL dataset

Merging segments of a table

4

SYSSQL dataset

Merging segments of a table

Control statements

This chapter describes the control statements that supply runtime information to the FileTek MVS Data Loader utility. Topics include:

- Control statement components and syntax
- Valid value types and their definitions
- LOAD control statement and its associated keyword/value pairs
- SMDEF control statement and its associated keyword/value pairs

About loader control statements

The FileTek MVS Data Loader utility reads two control statements from SYSIN to determine the operating parameters for the current load.

- The *LOAD control statement* contains information about the operation.
- The *SMDEF control statement* contains StorHouse information, such as the account ID used by the client data loader to connect to StorHouse and the volume set and file set names for the temporary VRAM file used for the data stream.

You can create these statements as instream data or in a host dataset. Both control statements are required for data load, index load, and merge operations.

Statement components

The LOAD and SMDEF control statements consist of two components: a command verb and a set of keyword/value pairs.

Command verb

A *command verb* identifies command action. LOAD and SMDEF are the command verbs. A command verb must appear as the first non-blank string in the first line of a control statement. It may be preceded by blanks and must be followed by at least one blank.

Keyword/Value pairs

A *keyword/value pair* consists of a keyword, followed by an equal sign, followed by a value. A *keyword* is a parameter that qualifies the command action according to the keyword *value*. An example of a keyword/value pair is CHECKPOINT=50.

Note: No spaces are permitted before or after the equal sign.

If any keyword/value pairs are specified in a control statement, at least one keyword/value pair must appear in the first statement line. You can specify multiple keyword/value pairs in one statement by separating the pairs with a comma.

Each keyword has an associated *keyword value type*. The format of a value depends on its keyword value type.

Keyword value types. Keywords can have one of four value types: string, quoted_string, numeric, and identifier. Command verb syntax descriptions on page 5-5 and on page 5-10 indicate valid keywords and their value types.

- A *string* value can contain any character except a quote (', the apostrophe), a comma, or a blank. You can enter characters in uppercase or lowercase letters. Case is not converted and may therefore be significant.

- A *quoted_string* is a string surrounded with single quotes ('). A *quoted_string* can contain commas and blanks. To embed a quote within a string, use two consecutive quotes as follows:

'Corporate "FINANCIAL" REPORT'

Note: These symbols " are not double quotation marks. They are two single quotes (') or apostrophes. The enclosing quotes are delimiters; they are not considered part of the value and do not count as characters in determining the string length.

If a keyword specifies that a *quoted_string* value is acceptable, a string value is also acceptable.

- A *numeric* value contains only digits (characters 0 - 9). Periods, commas, and signs are not allowed. The value may not be quoted. A numeric value may be null, which is the same as specifying zero.
- An *identifier* is a specific word such as YES or DIRECT. Identifiers may not be quoted. For keywords that have an identifier value type, this manual shows the valid identifiers rather than the word "identifier."

If you specify a keyword without a value following the equal sign, then the value defaults to "null." For any string-type value, a "null" value is an all-blank string. For a numeric, a "null" value is the same as specifying zero (digit 0).

Some keywords, such as ACCT (for the StorHouse account ID), specify values that have an associated password. You can specify the password by following the value with a slash and then the password. Any string preceded by a slash is never displayed in control statement processing listings.

General control statement syntax rules

The following syntax rules apply:

- You can code a control statement in a single SYSIN line or continue it across several lines.
- Each SYSIN line is expected to contain 80 characters, but columns 73-80 are reserved (for a possible sequence number). Information past column 72 is ignored.
- Any line that begins with an asterisk (*) in column 1 is considered a comment and is ignored.
- Verbs, keywords, and identifiers are recognized in uppercase, lowercase, or mixed case; but it is best to enter them as uppercase. No values (quoted or not) are changed from their entered case.
- Continuation is implied wherever a keyword/value pair is followed by a comma and a blank. Comments may follow the blank.

Command descriptions for LOAD and SMDEF follow.

LOAD

LOAD specifies information about the operation.

Command syntax

```
LOAD  CCSID=numeric,  
      CHECKPOINT=numeric,  
      DBNAME=string,  
      FNPREFIX=string,  
      FROMDD=string,  
      LOADID_OFFSET=numeric,  
      Pn=string,  
      SQLDD=string,  
      SKIP=numeric,  
      TAKE=numeric,  
      TEMP_FILE=KEEP | DELETE
```


Keyword	Definition
CCSID	<p>(optional) Coded Character Set Identifier (from the IBM Registry) for the input data. This CCSID must be one of the following:</p> <ul style="list-style-type: none"> ■ 500 (EBCDIC code page) ■ 819 (ISO 8859-1 code page) ■ 850 (PC code page) <p>The client data loader does not edit this value. However, the server data loader terminates the load operation if you specify an unsupported CCSID.</p> <p>If you omit CCSID, the default value is 500.</p>
CHECKPOINT	<p>(optional) Amount of data, in megabytes, that is transferred to StorHouse before the client data loader takes a checkpoint. This value affects a restart during the copy phase of a load. Small values minimize restart time but maximize checkpoint overhead. Never set this value larger than the size of one volume surface in the StorHouse volume set specified by VSET on SMDEF. Consult the StorHouse system administrator at your site for more information about StorHouse volume set surface sizes.</p> <ul style="list-style-type: none"> ■ A value of 0 indicates that checkpoints are to be taken every 100 megabytes. ■ The maximum CHECKPOINT value is 100. ■ If you specify CHECKPOINT on LOAD <i>and</i> SMDEF, the client data loader uses the smaller of the two values. ■ If you omit CHECKPOINT, the default is 100 megabytes. ■ You can abbreviate CHECKPOINT as CKPT.
DBNAME	<p>(required if omitted on SMDEF) Name of the StorHouse database. This database name must be in uppercase and should conform to the rules of your local database. For instance, a DB2 database name must be uppercase and cannot exceed 16 characters.</p> <p>You can supply the DBNAME on either the SMDEF or the LOAD control statement. If specified on both, the values must match or the operation fails. In addition, if you omit a DBNAME specification, the operation fails.</p> <p>You can abbreviate DBNAME as DBN.</p>

Keyword	Definition
FNPREFIX	(optional) File name prefix for the temporary StorHouse VRAM file used by the server data loader. The default value is STHLDR.TEMPF. Generally, you should use this default.
FROMDD	(optional) DDname for the host input dataset. Note the following: <ul style="list-style-type: none">■ If you omit FROMDD, the default value is SYSREC.■ If you specify a DDname other than SYSREC, be sure to match that DDname on the execution JCL.■ FROMDD should have a null value when you are submitting SQL statements only (not loading), loading data already on StorHouse, loading deferred indexes, and merging segments.■ If you specify a null value, the client data loader copies no data and the server data loader executes LOAD INDEX, MERGE, and SQL statements from the SYSSQL dataset only or loads data from a VRAM file specified on the INFILE clause.
LOADID_OFFSET	(optional depending on configuration) Number between 1 and 15 that is used to ensure uniqueness of load identifiers when data loads are run to a single StorHouse from multiple hosts that are not part of the same SYSPLEX. In this case, you should assign a different LOADID_OFFSET value to each of the hosts. <ul style="list-style-type: none">■ In a single processor or SYSPLEX-connected configuration, this parameter is not required.■ If you omit LOADID_OFFSET, the default value is 0.■ You can abbreviate LOADID_OFFSET as LOADID.

Keyword	Definition
Pn	<p>(optional) Substitution string that can be used to modify statements in the host SQL dataset (DDname SYSSQL) when those statements are copied to StorHouse for execution. Data in the host dataset is not changed.</p> <p>Maximum string length is 40 characters. No spaces are permitted.</p> <p>Any occurrence of the characters &&n (where n=0 through 8) in the host SQL dataset are replaced by the substitution string when a value is specified for Pn. If Pn is not defined, the &&n is left in the SQL statement.</p> <p>If you omit Pn, the default is "not specified." Note that you can specify Pn with a null value, but that is different from omitting Pn. A null value for Pn replaces any occurrences of &&n in the SQL with an empty string.</p> <p>If you use multiple Pn keywords, separate them with a space.</p> <p>Example:</p> <ul style="list-style-type: none"> ■ SQL dataset contains: LOAD DATA &&3.&&4 ■ LOAD contains: P3=MYID P4=JAN94 ■ Statement sent to StorHouse: LOAD DATA MYID.JAN94
SQLDD	<p>(optional) DDname for the host dataset that contains the LOAD DATA, LOAD INDEX, or MERGE statement. If you omit SQLDD, the default value is SYSSQL. If you specify a DDname other than SYSSQL, be sure to match that DDname on the execution JCL.</p>
SKIP	<p>(optional) Number of records in the input dataset that will be skipped, that is, not loaded. If you omit SKIP, the default number of records is 0, which indicates that loading begins at the first record in the input dataset.</p>

Keyword	Definition
TAKE	(optional) Number of records to be loaded from the input dataset. The default is “not specified.” All records in the input (except those skipped because of a non-zero SKIP value) are loaded.
TEMP_FILE	<p>(optional) Disposition of the temporary VRAM file.</p> <ul style="list-style-type: none">■ DELETE indicates the file is deleted when the operation completes successfully.■ KEEP indicates the file is not deleted. <p>To reuse this file, you must retain information that correlates the file name with the name of the user table.</p> <p>If the operation fails, the VRAM file is always kept to allow a subsequent restart.</p> <p>If you omit TEMP_FILE, the default is DELETE.</p>

SMDEF

SMDEF supplies values for StorHouse parameters that are required for the FileTek MVS Data Loader utility to connect to StorHouse and to write the data stream to the VRAM file.

For more information about StorHouse concepts such as volume sets, file sets, vulnerability time factor (VTF), accounts, and groups, refer to the *Command Language Reference Manual* and the *Concepts and Facilities Manual* in the StorHouse User Document Set.

Command syntax

```
SMDEF ACCT=string,  
      DBNAME=string,  
      CHECKPOINT=numeric,  
      FSET=string,  
      GROUP=string,  
      SM_NAME=string,  
      SUBS=string,  
      VSET=string,  
      VTF=string
```


Keyword	Description
ACCT	<p>(required) StorHouse account ID and password used by the client data loader to connect to StorHouse. The maximum length of the account ID is 12 characters, and the maximum length of the password is 32 characters. You provide the account ID and password as follows:</p> <p>Format: ACCT=string/password_string</p> <p>Example: ACCT=JACK/DAN987</p>
DBNAME	<p>(required if omitted on the LOAD control statement) Name of the StorHouse database. This database name must be in uppercase and should conform to the rules of your local database. For instance, a DB2 database name must be uppercase and cannot exceed 16 characters.</p> <p>You can supply the DBNAME on either the SMDEF or the LOAD control statement. If specified on both, the values must match or the load operation fails.</p> <p>You can abbreviate DBNAME as DBN.</p>
CHECKPOINT	<p>(optional) Amount of data, in megabytes, that is transferred to StorHouse before the client data loader takes a checkpoint. This value affects a restart during the copy phase of a load. Small values minimize restart time but maximize checkpoint overhead. Never set this value larger than the size of one volume surface in the StorHouse volume set specified by VSET. Consult your StorHouse system administrator for more information about StorHouse volume set surface sizes.</p> <ul style="list-style-type: none">■ A value of 0 indicates that checkpoints are to be taken after every 100 megabytes.■ The maximum CHECKPOINT value is 100.■ If you specify CHECKPOINT on LOAD and SMDEF, the client data loader uses the smaller of the two values.■ If you omit CHECKPOINT, the default is 100 megabytes.■ You can abbreviate CHECKPOINT as CKPT.
FSET	<p>(optional) Name of the file set to use for the temporary VRAM file on StorHouse. Maximum FSET name length is 8 characters. If you omit FSET, the default is the default FSET for the account ID specified in ACCT.</p>

Keyword	Description
GROUP	(optional) Name of the StorHouse file access group to use for the temporary VRAM file. Maximum GROUP length is 8 characters. If you omit GROUP, the default is the default GROUP for the account ID specified in ACCT.
SM_NAME	(optional) StorHouse identifier that is used whenever a StorHouse session is started. Maximum SM_NAME length is 6. If null, the default StorHouse name is used. If you omit SM_NAME, the default is blanks.
SUBS	(optional) StorHouse subsystem name. Maximum subsystem name length is 4 characters. If you omit SUBS, the default is blanks.
VSET	(optional) Name of the volume set to use for the temporary VRAM file. Maximum VSET name length is 8 characters. If you omit VSET, the default is the default VSET for the account ID specified in ACCT.
VTF	<p>StorHouse vulnerability time factor (VTF) value that determines the StorHouse backup priority for the temporary VRAM file. Valid values are:</p> <ul style="list-style-type: none"> ■ NEXT indicates to back up the temporary VRAM file during the next scheduled backup. ■ DIRECT indicates to bypass the StorHouse performance buffer and write the temporary VRAM file directly to its resident file set. ■ DEFAULT indicates to use the value of the StorHouse VTF system parameter. <p>If you are loading small amounts of data, either specify NEXT or omit VTF. If you are loading large amounts of data, always specify DIRECT.</p> <p>If you omit VTF, the StorHouse VTF system parameter value is used.</p>

Runtime

This chapter explains how to run the FileTek MVS Data Loader utility, including how to submit, restart, and abort data load, index load, and merge operations. This chapter also describes:

- What to do before running the FileTek MVS Data Loader utility
- Types of operations
- EXEC statement for the FileTek MVS Data Loader utility JCL
- DD statements
- Return codes

Preparing to run the utility

You can run the FileTek MVS Data Loader utility after:

- Installing it and initializing the checkpoint dataset (DDname CHKPT)
- Creating a sequential dataset (DDname SYSREC) containing the rows of data to load into one or more StorHouse user tables (for a load operation only)
- Creating a dataset (DDname SYSSQL) with a LOAD DATA, or LOAD INDEX, or MERGE statement
- Preparing the LOAD and SMDEF control statements instream or in a dataset (DDname SYSIN)
- Completing the StorHouse tasks described on page 1-7

Types of operations

You can run the following operations with the FileTek MVS Data Loader utility:

- Initialization operation – Described in Chapter 2, “Installation,” you run the FileTek MVS Data Loader utility as an initialization operation to create a checkpoint record in the checkpoint dataset. You should run this initialization operation once.
- Data load operation – You typically run a load operation to submit a normal production load job. The FileTek MVS Data Loader utility checks the checkpoint dataset first to ensure that the last load operation ended normally. You can run multiple data load operations concurrently, but each operation requires its own checkpoint dataset.
- Index load operation – This operation builds index entries for deferred indexes. For an index load, you omit the SYSREC DD statement and specify a null value for FROMDD on the LOAD control statement.
- Merge or replace operation – This operation consolidates existing segments or invalidates segments. You submit a merge or replace operation just like an index load operation. A replace operation may also be part of a data load operation.
- Restart operation – If a previous operation ended with a return code greater than 0, you can run a restart operation to complete the failed job. During a restart operation, the FileTek MVS Data Loader utility checks the checkpoint dataset to verify that the last operation ended abnormally and to obtain the checkpoint value, and then it continues where that operation left off.
- Abort operation – If an operation ends with a return code greater than 0, you can abort the operation to clean up the prior operation. An abort operation resets the checkpoint value in the checkpoint dataset, resets the checkpoint status on StorHouse, and for a data load, deletes and removes any segment files written to StorHouse.

EXEC statement

The EXEC statement of the FileTek MVS Data Loader utility JCL contains:

- The program name (PGM=), which is LDLSLDR
- One of six parameters (PARM=), enclosed in single quotes:
 - INIT – run an initialization operation, as described in Chapter 2
 - LOAD – submit an operation (data load, index load, replace-only, or merge) and/or a StorHouse SQL statement
 - RESTART – restart a failed load operation
 - RESTART_IGNORE – force reset the checkpoint record *without* loading data
 - RESTART_RELOAD – force reset the checkpoint record *and* reload the data
 - ABORT – terminate an operation
- A recommended region size (REGION=) of 4 megabytes

Below is a sample EXEC statement for submitting a data load operation:

```
//STEP EXEC PGM=LDLSLDR,PARM='LOAD',REGION=4M
```

You can use the following EXEC statement to restart a data load operation:

```
//STEP EXEC PGM=LDLSLDR,PARM='RESTART',REGION=4M
```

Note: You can provide a substitution string in the PARM and symbolic variables in the SYSSQL dataset to perform symbolic substitution. See page 4-40 for more information about symbolic substitution.

DD statements

The JCL for the FileTek MVS Data Loader utility contains the following DD statements:

DDname	Required for	Description
STEPLIB	All EXEC parameters	Refers to the FileTek MVS Data Loader utility load library (LDLLOAD) and the StorHouse load library (LSMLOAD). This DD statement is <i>optional</i> if the load libraries reside in the system LINKLIST.
CHKPT	All EXEC parameters	Refers to the checkpoint dataset that the client data loader uses to preserve operation state information.
SYSTEMM	All EXEC parameters	Receives SAS/C error messages. Typically, this DD statement is directed to a SYSOUT dataset.
SYSPRINT	All EXEC parameters	Receives run statistics and runtime messages, including the messages described in Appendix A. Typically, this DD statement is directed to a SYSOUT dataset.
SYSERR	All EXEC parameters	Receives error messages for those jobs with a return code greater than 0. See Appendix A for descriptions of these messages.
SYSSQL	LOAD and RESTART	Refers to the dataset that contains the LOAD DATA statement and any SQL statements.
SYSREC	LOAD and RESTART	Refers to the input sequential dataset that contains the rows of data you are loading into the user table. This DD statement is <i>optional</i> if the FROMDD keyword in the LOAD control statement has a value of null (for index load, merge, and replace operations)
SYSIN	All EXEC parameters	Refers to the dataset or the instream data containing the SMDEF and LOAD control statements. Both control statements are required.

Return codes

When an operation completes, the FileTek MVS Data Loader utility issues a message with one of the following return codes:

Code	Description
0	The operation completed successfully.
4	The initialization operation completed; no data was loaded.
8	Error during an operation. Error messages in SYSPRINT and SYSERR identify the specific failure. Some of the types of errors are as follows: <ul style="list-style-type: none">■ Cannot open, read from, or write to datasets■ Ambiguous restart state■ Mismatched checkpoint state■ Invalid SQL statements■ No database name■ Too many database names■ No SMDEF control statement matching a LOAD control statement
12	Error returned by StorHouse. Refer to the <i>StorHouse Messages and Codes Manual</i> for more information.
16	Error in control statement processing. Error messages in SYSPRINT and SYSERR identify the specific failure. Some of the types of errors are as follows: <ul style="list-style-type: none">■ The PARM value or syntax is incorrect■ LOADID_OFFSET is incorrect during a restart operation■ The SYSIN does not contain LOAD or SMDEF control statements■ Invalid keyword for a verb■ Checkpoint failures
20	Error in control statements (LOAD and SMDEF) in the SYSIN.
24	Missing DDname SYSERR. Add the DD statement to the JCL and then re-run the job.
28	Missing DDname SYSPRINT. Add the DD statement to the JCL and then re-run the job.

Submitting an operation

To submit an operation—data load, index load, merge, or replace—prepare the JCL and then submit the job.

▼ To customize the JCL for submitting an operation

1. Complete the JOB statement.
2. Specify PARM='LOAD' on the EXEC statement.
3. At the CHKPT DD statement, provide the name (DSN=) of the checkpoint dataset.
4. At the SYSSQL DD statement, provide the name (DSN=) of the dataset containing the LOAD DATA, LOAD INDEX, MERGE, and/or SQL statement(s).
5. At the SYSREC DD statement, provide the name (DSN=) of the input sequential dataset containing the data you are loading.

When replacing segments without loading data, the SYSREC DD statement must name an empty dataset. You can use DD* followed by /*.

Omit the SYSREC DD statement when submitting an index load and merge operation, or submitting SQL statements only, or if the input data already resides in a VRAM file on StorHouse. In other words, if the FROMDD keyword in the LOAD control statement is null, then don't include the SYSREC DD statement.

6. Prepare the dataset or instream data with the LOAD and SMDEF control statements, as described in Chapter 5, "Control statements."

Below is sample JCL for submitting a data load operation.

```
//LOAD          JOB    ...
//*
//* LOAD OPERATION
//*
//LOAD          EXEC   PGM=LDLSLDR,PARM='LOAD',REGION=4M
//*
//STEPLIB      DD      DSN=LDLLOAD,DISP=SHR
//              DD      DSN=LSMLOAD,DISP=SHR
//CHKPT        DD      DSN=checkpoint.dataset.name,DISP=OLD
//SYSTEM       DD      SYSOUT=*
//SYSPRINT     DD      SYSOUT=*
//SYSSQL       DD      DSN=load.data.dataset,DISP=SHR
//SYSREC       DD      DSN=input.dataset,DISP=SHR
//SYSERR       DD      SYSOUT=*
//SYSIN        DD      *
Sample instream — SMDEF  ACCT=account/password,
control          DBNAME=database_name,
statements       VSET=volsetid,FSET=fileset

LOAD
/*
```


The following JCL is an example for an index load operation. The SYSSQL DD identifies the dataset containing the LOAD INDEX statement, the SYSREC DD is omitted, and the FROMDD keyword on the LOAD control statement contains a null value.

```
//LOAD          JOB    ...
//*
//* INDEX LOAD OPERATION
//*
//LOAD          EXEC   PGM=LDLSLDR,PARM='LOAD',REGION=4M
//*
//STEPLIB       DD     DSN=LDLLOAD,DISP=SHR
//              DD     DSN=LSMLOAD,DISP=SHR
//CHKPT         DD     DSN=checkpoint.dataset.name,DISP=OLD
//SYSTEM        DD     SYSOUT=*
//SYSPRINT       DD     SYSOUT=*
//SYSSQL        DD     DSN=load.index.dataset,DISP=SHR
//SYSERR        DD     SYSOUT=*
//SYSIN         DD     *
SMDEF          ACCT=account/password,
               DBNAME=database_name,
               VSET=volsetid,FSET=fileset
LOAD           FROMDD=
/*
```

Restarting an operation

To restart an operation that failed, prepare the JCL and then submit the job. Typically, you run a restart operation when the original operation fails after you receive the message LDL757, Beginning Load Operation. If StorHouse is shut down while any operations are in progress, you can restart or abort the operation after StorHouse is started. Note the following:

- If the SYSSQL dataset contains a LOAD DATA statement with multiple into_table_specs or SQL statements, then recovery begins with the statement after the last successfully executed statement in the prior run.

- If you try to restart an operation but there is no checkpoint for that load ID, the FileTek MVS Data Loader utility treats the operation like a load instead of a restart.
- You must abort an operation if there are syntax errors in the SYSSQL dataset or an error occurs when loading LOB data.
- When restarting an operation, you can't change any of the records in the SYSSQL dataset. For instance, you can't:
 - Add or delete records. The SYSSQL dataset must contain the same number of records as the original operation.
 - Change the order of records. They must be in the exact order as the original operation.
 - Change a type of record, for instance, change a CREATE TABLE statement to a DROP TABLE statement.

If you need to make any of the above changes, then you must abort the operation.

▼ **To prepare the JCL for restarting an operation**

1. Complete the JOB statement.
2. Specify the desired PARM value on the EXEC statement. Valid values are:
 - PARM='RESTART'
 - PARM='RESTART_RELOAD'
 - PARM='RESTART_IGNORE'

Use PARM='RESTART' unless a FileTek MVS Data Loader utility message—such as LDL778I—specifically tells you to use one of the other values.

3. Use the same input dataset, LOAD DATA, LOAD INDEX, or MERGE statement, and control statements as the failed operation. The JCL setup of a

restart operation *must* match the setup of the original operation. If the JCL setup does not match, then abort the operation and start over.

Sample JCL for restarting a data load operation follows.

```
//RESTART      JOB    ...
//*
//* RESTART A LOAD OPERATION
//*
//LOAD          EXEC  PGM=LDLSLDR,PARM='RESTART',REGION=4M
//*
//STEPLIB      DD     DSN=LDLLOAD,DISP=SHR
//              DD     DSN=LSMLOAD,DISP=SHR
//CHKPT        DD     DSN=checkpoint.dataset.name,DISP=OLD
//SYSTEM       DD     SYSOUT=*
//SYSPRINT     DD     SYSOUT=*
//SYSSQL       DD     DSN=column.definition.dataset,DISP=SHR
//SYSREC       DD     DSN=input.dataset,DISP=SHR
//SYSERR       DD     SYSOUT=*
//SYSIN        DD     *
SMDEF          ACCT=account/password,
               DBNAME=database_name,
               VSET=volsetid,FSET=fileset

LOAD
/*
```

Aborting an operation

Instead of restarting an operation you can abort it. Aborting an operation deletes all checkpoints and removes any partially written segment files. You must abort an operation when you need to:

- Fix syntax errors in the SYSSQL dataset
- Change, add, or remove any SQL statements or LOAD DATA, LOAD INDEX, or MERGE clauses in the SYSSQL dataset

- Add or remove data records from the SYSREC dataset

You must also abort a data load if a segment file exceeded the maximum size or an error occurred when loading LOB data. To abort an operation that ended with a non-zero return code, prepare the JCL and then submit the job.

▼ To prepare the JCL for aborting an operation

1. Complete the JOB statement.
2. Specify PARM='ABORT' on the EXEC statement.
3. Keep all other DD statements as the previous operation with the exception of SYSREC and SYSSQL. These DD statements are not required for an abort operation.

Below is sample JCL for aborting a data load operation.

```
//ABORT      JOB    ...
//*
//*      ABORT A LOAD OPERATION
//*
//LOAD      EXEC  PGM=LDLSLDR,PARM='ABORT'
//*
//STEPLIB   DD     DSN=LDLPLOAD,DISP=SHR
//          DD     DSN=LSMLOAD,DISP=SHR
//CHKPT     DD     DSN=check.point.dataset.name,DISP=OLD
//SYSTEM    DD     SYSOUT=*
//SYSPRINT  DD     SYSOUT=*
//SYSERR     DD     SYSOUT=*
//SYSIN     DD     *
SMDEF      ACCT=account/password,
           DBNAME=database_name,
           VSET=volsetid,FSET=fileset
LOAD
/*
```


6

Runtime

Aborting an operation

Messages

This appendix contains FileTek MVS Data Loader utility error, warning, and informational messages. Messages are listed in sequential order by message number. Each message entry includes:

- The message number
- The message text (lowercase words indicate message variables)
- A brief explanation of why the utility issued the message
- The system action as a result of the error
- How you should respond to the message

LDL680I DDNAME ddname: OPEN FAILED

Explanation: An OPEN request failed for the dataset indicated by ddname.

System Action: Terminates the job and exits with a nonzero return code. If the failing DDname is SYSERR, the return code is 24. If the DDname is SYSPRINT, the return code is 28. If both of these DD statements are missing, this message cannot be written and only the return code (28) is available.

User Response: Add the required DD statement to the JCL, then re-run the job.

LDL681I FILETEK MVS DATA LOADER UTILITY V1R1M0 LOAD START DATE yyyy/mm/dd TIME hh:mm:ss

Explanation: The date and time you ran the operation.

System Action: Starts the operation.

User Response: None

LDL682I INPUT DDN=ddname REPOSITIONED: operation type RECORD record number

Explanation: For a restart operation, the input dataset is positioned to the correct record, or you specified the SKIP keyword on the LOAD control statement to bypass input data records.

System Action: None

User Response: None

LDL683I END INPUT REPOSITIONING, ELAPSED TIME=hh:mm:ss

Explanation: The input dataset has been repositioned in the hh:mm:ss indicated.

System Action: None

User Response: None

LDL684I RESTARTING AT SERVER LOAD PHASE

Explanation: The FileTek MVS Data Loader utility is restarting the operation. The prior load operation terminated while the server data loader was loading the user table on StorHouse.

System Action: Restarts the server load job.

User Response: None

LDL685I END SERVER LOAD PHASE, ELAPSED TIME=hh:mm:ss

Explanation: The server data loader finished loading the user table on StorHouse. The hh:mm:ss indicates the duration of the server load phase.

System Action: None

User Response: None

LDL686I END DATA COPY PHASE, ELAPSED TIME=hh:mm:ss

Explanation: The client data loader finished copying data from the host to StorHouse VRAM file. The hh:mm:ss indicates the duration of the copy phase of the load operation.

System Action: None

User Response: None

LDL687I INPUT DDN=ddname COPIED RECORD COUNT=number of records

Explanation: The client data loader finished copying the number of records from the SQL input dataset to the StorHouse VRAM file. The ddname is the value of the SQLDD keyword of the LOAD control statement.

System Action: None

User Response: None

LDL688I INVALID SQL STATEMENT

Explanation: The dataset with the LOAD DATA statement (DDname SYSSQL or other specified DDname for the SQLDD keyword in the LOAD control statement) contains an SQL statement named BEGINDATA. This record is not a valid LOAD DATA statement and not a valid StorHouse SQL statement.

System Action: Terminates the operation.

User Response: Remove the SQL statement from the dataset and then re-run the job.

LDL689I DID NOT DELETE TEMP SM FILE file name

Explanation: The temporary StorHouse VRAM file used to transport SQL and data to StorHouse for loading has not been deleted. This file is usually deleted after a load operation completes normally. The file was not deleted because you specified TEMP_FILE=KEEP on the LOAD control statement.

System Action: None

User Response: None. To make future use of this file, keep a log to correlate the name of the user table loaded and the StorHouse file name.

LDL690I END LOAD RUN, MAXIMUM RETURN CODE=return code

Explanation: This message is the last one issued by the FileTek MVS Data Loader utility; it indicates that the operation has completed. Return codes are:

- 0 – OK
- 4 – Warning issued for an initialization operation
- 8 – Error in utility control (usually PARM) processing
- 12 – Error returned by StorHouse
- 16 – Error in control statement processing
- 20 – Error in control statements (DDname SYSIN)
- 24 – DDname SYSERR missing
- 28 – DDname SYSPRINT missing

System Action: None

User Response: None if the return code is 0. For non-zero return codes, previous error messages indicate the processing failures.

LDL694I MEMORY ALLOCATION FOR storage_type FAILED length

Explanation: The client data loader was unable to allocate virtual memory because the region size is insufficient.

System Action: Terminates the job.

User Response: Increase the REGION size on the EXEC statement, then re-run the job.

LDL698I storHouse message id, storHouse message

Explanation: This message is from StorHouse.

System Action: None

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information.

LDL710I NO CONTROL STATEMENTS IN SYSIN

Explanation: The SYSIN dataset does not contain any control statements. At least one input record is required for all operations, with the exception of an initialization operation.

System Action: Terminates the job.

User Response: Ignore this message for an initialization operation. Otherwise, provide the control statement in the SYSIN and then re-run the job.

LDL711I REQUIRED PARAMETER NOT SPECIFIED

Explanation: A control statement does not contain a keyword/value pair required by the verb.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the control statement and then re-run the job.

LDL712I string NOT A VALID type

Explanation: The string is not a valid identifier for an object. The type may be VERB, KEYWORD, or VALUE. The control statement may contain a typographical error.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Verify the syntax and data in the control statement and then re-run the job.

LDL713D INVALID LENGTH OR TERMINATION IN DBCS STRING

Explanation: A control statement contains a DBCS string value that is too long or that does not have the required termination character.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Ensure that DBCS strings are enclosed in SOSI and then re-run the job.

LDL713Q INVALID LENGTH OR TERMINATION IN QUOTED STRING

Explanation: A control statement contains a quoted string value that is too long or that does not have the required termination character.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Ensure that a quoted string contains a close quote. Note that two consecutive quotes count as a single data character that is a quote (apostrophe).

LDL714I MISSING DELIMITER OR TERM DOES NOT END BEFORE COLUMN 73

Explanation: A control statement does not contain any delineating character (comma or equal sign).

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL715I NO VERB FOUND IN INPUT STATEMENT

Explanation: A control statement does not contain a valid verb or any verb. Valid verbs are LOAD, SMDEF, and SHOW. Note that SHOW is not supported in this release.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Check the spelling of the command verb on the control statement. Correct any errors and then re-run the job.

LDL716I SKIPPING TO NEXT INPUT STATEMENT

Explanation: The client data loader cannot continue scanning the current line because of errors.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL717I CANNOT LOCATE NEXT KEYWORD

Explanation: The client data loader cannot locate the next keyword following a value. A control statement may contain a typographical error.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL718I EXPECTED CONTINUATION NOT FOUND

Explanation: A control statement line following a line that indicated continuation did not begin with a recognized keyword. Any keyword/value pair that is followed by a comma and a blank indicates continuation.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Check that the last keyword/value pair is correctly terminated in the control statement. Correct any errors and then re-run the job.

LDL719I DATASET NAME SYNTAX ERROR

Explanation: A dataset name is invalid.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Review the MVS dataset name requirements. Note that you cannot enclose dataset names in quotes.

LDL720I STRING IS TOO LONG

Explanation: A string in a control statement is too long or a control statement may contain a typographical error.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL721I NON-DIGIT IN INTEGER

Explanation: An integer value field contains a non-digit character in a control statement, or a control statement contains a typographical error. Note that numeric values must be decimal and cannot contain commas, decimal points, or signs.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL722I VALUE TOO LARGE

Explanation: An integer value exceeded the maximum allowed by the associated keyword, or a control statement contains a typographical error.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct any errors in the control statement and then re-run the job.

LDL723I DUPLICATE KEYWORD

Explanation: A keyword appears twice in the same control statement. You cannot change a value by specifying the keyword twice on the same control statement.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the error in the control statement and then re-run the job.

LDL725I NULL VALUE NOT ALLOWED

Explanation: A control statement keyword was not followed by a value. The particular keyword does not allow a null value specification.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the error in the control statement and then re-run the job.

LDL726I CONTROL STATEMENT OR PARM ERRORS, RUN TERMINATED

Explanation: The control statement contains syntax errors in the SYSIN or there's an error with the PARM. Other messages identify the specific failures; this one just indicates that no loading is performed.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Respond to other messages that identify the specific error(s). Correct the errors and then re-run the job.

LDL727I INVALID LOAD/RESTART VALUE IN PARM: parameter

Explanation: The first field of the PARM in the EXEC statement contains a non-blank value other than one of the following: INIT, LOAD, RESTART, RESTART_IGNORE, RESTART_RELOAD, or ABORT.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the PARM and then re-run the job. If the PARM contains a substitution string, the first substitution value must be separated from the next substitution value by a blank.

LDL728I INVALID SUBSTITUTION STRING IN PARM: invalid string

Explanation: The substitution string in the PARM of the EXEC statement does not start with the string Pn=, where n is a digit 0-8.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the PARM and then re-run the job. Remember that substitution values in a substitution string must be separated by a blank.

LDL730I keyword= CANNOT BE SPECIFIED FOR statement type

Explanation: The listed keyword in the SYSIN is not supported for the type of control statement—LOAD or SMDEF.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Provide a valid keyword and then re-run the job.

LDL731I VALUE SPECIFIED FOR PARALLEL IS NOT YES OR NO

Explanation: The value for the keyword PARALLEL is not one of the allowed values. Possible values are YES or NO. This keyword is not supported in this release.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Provide a valid value for the keyword PARALLEL and then re-run the job.

LDL732I VALUE SPECIFIED FOR VTF IS NOT NEXT, NOW, DIRECT, OR DEFAULT

Explanation: The value for the keyword VTF on the SMDEF control statement is not one of the allowed values. Possible values are NEXT, DIRECT, or DEFAULT for the value of the StorHouse VTF system parameter.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Provide a valid value for the VTF keyword, then re-run the job.

LDL733I MULTIPLE TAGS SPECIFIED IN ONE SMDEF

Explanation: An SMDEF control statement contains more than one TAG. Only one TAG is allowed per SMDEF control statement.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the SMDEF control statement and then re-run the job.

LDL734I KEYWORD INCORRECTLY TERMINATED; USE = SIGN NOT BLANK

Explanation: A control statement contains a keyword followed by a blank rather than by an equal sign (=). The correct syntax is keyword=value. No blanks are allowed immediately preceding or following the equal sign.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Remove any blanks before or after the equal sign of the keyword and then re-run the job.

LDL735I PRIOR STATEMENT DID NOT INDICATE CONTINUATION

Explanation: A control statement begins with a keyword=value pair, not a verb. The prior line did not indicate continuation.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the error and then re-run the job.

LDL736I VERB INCORRECTLY TERMINATED; USE BLANK NOT = SIGN

Explanation: A control statement begins with a verb followed by an equal sign (=). The correct syntax is a verb followed by one or more blanks.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the control statement and then re-run the job.

LDL737I PRIOR STATEMENT MAY ERRONEOUSLY INDICATE CONTINUATION

Explanation: A control statement line begins with a verb, but the prior line indicated continuation. The control statement may contain a typographical error.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the error and then re-run the job.

LDL738I THIS PROGRAM SUPPORTS ONLY LOAD, SMDEF, AND SHOW REQUESTS

Explanation: The client data loader received a control statement with a verb that is not supported. The supported verbs are LOAD, SMDEF, and SHOW. The verb SHOW is not supported in this release.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the verb in the control statement and then re-run the job.

LDL739I TOO MANY SMTAG ENTRIES FOR ONE LOAD

Explanation: You specified the SMTAG keyword more than four times on the LOAD control statement. This control statement can be associated with a maximum of four SMDEF control statements. Note that the SMTAG keyword is not supported in this release.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the LOAD control statement and then re-run the job.

LDL740I VALUE SPECIFIED FOR LIST IS NOT CKPT OR NO

Explanation: The LIST keyword of the SHOW command contains an invalid value. Possible values are CKPT or NO. Note that the SHOW command is not supported in this release.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Provide a valid value for the LIST keyword and then re-run the job.

LDL741I DUPLICATE TAG ID IN SMDEF

Explanation: The identifier supplied with the TAG keyword on an SMDEF command duplicates a TAG used by another SMDEF control statement.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the identifier and then re-run the job.

LDL742I DUPLICATE SM TAG ID IN LOAD

Explanation: A LOAD control statement contains an SMTAG that has the same value as a prior tag. One SMDEF cannot be linked to a LOAD multiple times. This would result in duplicate files under the same group on the same StorHouse system.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Correct the LOAD control statement SMTAG list and then re-run the job.

**LDL743I RESTART AT LOAD OPERATION NUMBER=number
SYSIN DOES NOT CONTAIN THAT MANY LOAD STATEMENTS**

Explanation: The checkpoint dataset (DDname CHKPT) contains a checkpoint record that indicates a restart is necessary (prior run failed) and that load operation number is the restart point. However the current SYSIN dataset does not contain that number of LOAD control statements.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Restart the job with the same SQL, data, and SYSIN data as the prior failed run.

LDL744I CANNOT WRITE TO CHECKPOINT DATASET

Explanation: The client data loader encountered an error while writing to the checkpoint dataset (DDname CHKPT).

System Action: Terminates the job without updating the checkpoint record.

User Response: Repair the checkpoint dataset by running an initialization operation, and then re-run the job.

LDL745I CANNOT READ FROM CHECKPOINT DATASET

Explanation: The client data loader encountered an error while reading the checkpoint dataset (DDname CHKPT).

System Action: Terminates the job without updating the checkpoint record.

User Response: Repair the checkpoint dataset by running an initialization operation, and then re-run the job.

LDL746I DATA IN CHKPT DATASET IS NOT A VALID CHECKPOINT RECORD

Explanation: The checkpoint dataset (DDname CHKPT) contains an invalid checkpoint record. Probably, the dataset was not initialized.

System Action: Terminates the job without updating the checkpoint record.

User Response: If the dataset has not been initialized, run the FileTek MVS Data Loader utility with PARM='INIT'. Then re-run the load job.

LDL747I CHECKPOINT RECORD STATE DICTATES RESTART

Explanation: You ran the FileTek MVS Data Loader utility with PARM='LOAD', but the checkpoint record indicates that the prior run failed and the correct PARM is RESTART.

System Action: Terminates the job without updating the checkpoint record.

User Response: If a restart is required, then run the job with PARM='RESTART'. If the prior operation was at a checkpoint but is not to be restarted, run the job with PARM='ABORT', then rerun with LOAD. If RESTART or ABORT do not work, then rebuild the checkpoint with PARM='INIT' and then re-run the load with PARM='LOAD'.

LDL748I CHECKPOINT RECORD STATE DOES NOT ALLOW RESTART

Explanation: You ran the FileTek MVS Data Loader utility with PARM='RESTART', but the checkpoint record indicates that there was no prior run (record was initialized) or the prior run ended correctly. You cannot run the job with PARM='RESTART'.

System Action: Terminates the job without updating the checkpoint record.

User Response: Change the PARM value to LOAD and then run the job.

**LDL749I LOAD STATEMENT=number LOADID OFFSET=offset value
INCONSISTENT WITH RESTART LOADID=offset value**

Explanation: During a restart operation, the checkpoint record indicates that you ran the LOAD control statement with the specified offset value. However, the same LOAD control statement in the SYSIN has a different offset value. You cannot change control statements in SYSIN between the failed run and the restart run.

System Action: Terminates the job without updating the checkpoint record.

User Response: Correct the control statement and then re-run the job.

**LDL750I CHECKPOINT RECORD INITIALIZED. RETURN CODE FORCED TO 4, NO
DATA LOADING PERFORMED**

Explanation: You ran the FileTek MVS Data Loader utility with PARM='INIT'. The checkpoint record has been initialized. However an initialization operation never loads any data. The return code 4 simply indicates that initialization operations do not load data.

System Action: None

User Response: None

LDL751I SM DEFINITION “tag” ACCEPTED

Explanation: The FileTek MVS Data Loader utility parsed and analyzed the SMDEF control statement with the specified tag. No errors were found. The SMDEF control statement can be referenced by a LOAD control statement. If the tag is blank, then the SMDEF control statement does not contain a tag. This is acceptable if there is only one SMDEF control statement or if all other SMDEF control statements specify tag.

System Action: None

User Response: None

LDL752I OPERATION NUMBER id ASSIGNED TO THIS LOAD REQUEST

Explanation: The FileTek MVS Data Loader utility parsed and analyzed the LOAD control statement and assigned this id to the load request. No errors were found.

System Action: None

User Response: None

LDL753I RESTARTING AT OPERATION NUMBER id LOADID=load id phase

Explanation: The FileTek MVS Data Loader utility is restarting the LOAD command identified by the operation number id with the load id value generated by the prior failed job. The phase indicates the operation phase at which the load is being restarted—COPY or SERVER.

System Action: None

User Response: None

LDL754I NO command CONTROL STATEMENTS IN SYSIN

Explanation: The SYSIN does not contain an SMDEF or LOAD control statement. To perform a load operation, at least one SMDEF and one LOAD control statement is required in the SYSIN.

System Action: Terminates the job without updating the checkpoint record.

User Response: Correct the SYSIN and then re-run the job.

LDL755I NO SM DEFINITION WITH TAG=tag

Explanation: A LOAD control statement specified SMTAG=tag, but there is no SMDEF with that tag.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Verify that you specified the correct tag and then re-run the job.

LDL756I DBNAME CONFLICT first_dbname second_dbname

Explanation: You used the DBNAME keyword in multiple control statements—either LOAD and an associated SMDEF—or in multiple SMDEF control statements associated with the same LOAD. These multiple definitions did not specify the same database name.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Ensure that all DBNAME keywords are followed by the same database name, then re-run the job.

**LDL757I BEGINNING LOAD OPERATION number LOADID=load id
DBNAME=database name**

Explanation: The FileTek MVS Data Loader utility is processing the LOAD control statement for operation number. The load id and the database name are informational. Note that the load id is a unique number generated by the FileTek MVS Data Loader utility for restart purposes.

System Action: Processes the LOAD control statement.

User Response: None

LDL758I END LOAD OPERATION number RETURN CODE=return code

Explanation: The load operation specified by the LOAD control statement with operation number has completed with the indicated return code. A non-zero return code indicates that the load failed, and prior messages describe the failure.

System Action: None

User Response: None, if the return code is zero. Otherwise, correct the problems described by prior messages.

**LDL759I RETURN CODE=return code FROM PROCESSING CONTROL
STATEMENTS IN SYSIN**

Explanation: The FileTek MVS Data Loader utility parsed and analyzed the control statements in the SYSIN, which contains errors. The indicated return code is the highest return code value. Prior messages describe the errors.

System Action: Terminates the job.

User Response: Correct the problems described by prior messages and then re-run the job. Do not run a restart operation.

**LDL760I RETURN CODE=status code FROM STORHOUSE api function
OPERATION**

Explanation: A StorHouse operation has failed. The status code is a StorHouse status code, and the api function is the specific API function called during the process.

System Action: Terminates the job. The checkpoint record is in the “restart required” state.

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information about the status code. After correcting the StorHouse problem, run a restart operation (PARM='RESTART').

LDL761I type STATEMENT, REQUIRED FIELD keyword MISSING

Explanation: A control statement in the SYSIN does not contain a required field. The type is SMDEF or LOAD. The keyword is the missing keyword.

System Action: Analyzes all input control statements and parameters, but terminates the job and exits without processing any load requests or updating the checkpoint record.

User Response: Provide the missing keyword for the appropriate control statement and then re-run the job.

LDL762I RETURN CODE=return code FROM SEQUENTIAL I/O FUNCTION name

Explanation: A read operation to a host dataset has failed with the specified return code. The name is the I/O function performed.

System Action: Terminates the job. If the function is OPEN, the checkpoint record has *not* been changed. Otherwise, a restart operation may be required.

User Response: Correct the dataset and restart the job, specifying PARM='RESTART' if name is not OPEN.

LDL763I EOF ON DATA INPUT DURING REPOSITIONING

Explanation: The input dataset could not be positioned to the correct record for a restart operation or as requested by a SKIP keyword. End-of-data was encountered. For a restart operation, use the same input dataset as the original job. For a skip request, the count is too high and is past end-of-file.

System Action: Terminates the job without updating the checkpoint record.

User Response: Specify the correct input dataset for a restart, or correct the skip count.

**LDL764I RETURN STATUS=return code FROM CHECKPOINT WRITE;
ATTEMPTING TO CONTINUE**

Explanation: A checkpoint update could not be completed because of a write error. The client data loader is attempting to continue because all work may be lost if it terminates. A restart operation may be impossible.

System Action: Attempts to continue processing.

User Response: Repair the checkpoint record by running an initialization operation after the job completes.

**LDL765I ERROR WRITING TYPE record type RECORD, SM RETURN
CODE=status code**

Explanation: An error occurred while writing the data stream to StorHouse. The record type is the type of record being written—environment, SQL, data delimiter, or data. The status code is a StorHouse status code.

System Action: Terminates the job. The checkpoint record is in the “restart required” state.

User Response: Refer to the StorHouse *Messages and Codes Manual* for more information about the status code.

LDL766I SQL STATEMENT IS TOO LONG

Explanation: An SQL statement in the dataset associated with DDname SYSSQL is too long for StorHouse. The maximum size of a StorHouse SQL statement is 5,000 bytes.

System Action: Terminates the job. The checkpoint record is in the “restart required” state.

User Response: Correct the SQL statement and then restart the job with PARM='RESTART'.

LDL767I VALUE SPECIFIED FOR TEMP_FILE IS NOT KEEP OR DELETE

Explanation: The TEMP_FILE keyword on the LOAD control statement has a value other than KEEP or DELETE.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Correct the value for the TEMP_FILE keyword on the LOAD control statement and then re-run the job.

LDL768I CHECKPOINT RECORD DUMP FOLLOWS

Explanation: The header for the listing of a checkpoint record contents from the SHOW command.

System Action: None

User Response: None

LDL770I DATA COPY PHASE STATISTICS: RECORDS COPIED=number BYTES TRANSFERRED=number

Explanation: The load operation has successfully copied the indicated number of records and transferred the specified number of bytes to the VRAM file on StorHouse.

System Action: None

User Response: None

**LDL771I last function PRIOR=previous function STATUS=status code SM
FUNCTION=api function**

Explanation: Error logging information produced by StorHouse for error diagnosis. The last function is the name of support function last called, that is, the support function that encountered the error. The previous function is the name of the function that was called prior to the last function. Note that these are support module functions, not StorHouse/RM API functions. The status code is a StorHouse status code, and the api function is the name of the StorHouse/RM API call that produced the error.

System Action: Terminates the job after logging the information.

User Response: Fix the StorHouse problem and then restart the job with PARM='RESTART'.

LDL772I SM_NAME=name SUBSYS_ID=id ACCOUNT=aid GROUP=group name

Explanation: Additional logging information for a StorHouse error. This message follows message number LDL771I.

System Action: Terminates the job after logging the information.

User Response: Fix the StorHouse problem and then restart the job with PARM='RESTART'.

LDL773I SM FILE ATTRIBUTES: ATF=value VTF=value VSET=name FSET=name

Explanation: Additional logging information for a StorHouse error. This message follows message number LDL772I.

System Action: Terminates the job after logging the information.

User Response: Fix the StorHouse problem and then restart the job with PARM='RESTART'.

LDL774I SM FILE RECORD NUMBER=number FILENAME=file name

Explanation: Continuation of logging info from 771/772/773. This message may not be produced or it may contain irrelevant information if the failing operation doesn't involve a StorHouse file operation. Correct but irrelevant information may be provided if there was a prior StorHouse file operation but the specific failure was not in a file operation.

System Action: Terminates the job after logging the information.

User Response: Fix the StorHouse problem and then restart the job with PARM='RESTART'.

LDL775I command text

Explanation: Continuation of messages 771-774. If the failing operation was SM-CMD-INTE, then the command text is the command text sent to StorHouse.

System Action: Terminates the job after logging the information.

User Response: Fix the StorHouse problem and then restart the job with PARM='RESTART'.

LDL777I SM NO-CHECKPOINT RESPONSE; FILE WILL BE RELOADED

Explanation: During a restart operation, the server data loader received a no-such-checkpoint response from StorHouse. The operation was never started at StorHouse or RESTART_RELOAD was specified in the PARM.

System Action: Continues the restart operation, assuming that the user table was not loaded.

User Response: None

LDL778I SM NO-CHECKPOINT RESPONSE; RESTART IS NOT POSSIBLE

Explanation: During a restart operation, the server data loader received a no-such-checkpoint response from StorHouse. The client data loader does not know whether the data has or has not been loaded.

System Action: Terminates the job.

User Response: Inspect the user tables to determine the load status. Run an abort operation or re-initialize the client checkpoint dataset. You may have to re-run the load after correcting the checkpoint record. Report this failure to the StorHouse system or database administrator.

LDL779I RECORD NUMBER OF LAST RECORD COPIED TO SM=record number

Explanation: Message written after 778 to indicate the record number of the last data record copied to StorHouse.

System Action: None

User Response: Decide whether to use a PARM value of RESTART_IGNORE or RESTART_RELOAD and then restart the job.

LDL780I SQL STATEMENT DID NOT END WITH ;

Explanation: An SQL statement in the SYSSQL dataset does not end with a semicolon. The copy routine reached end-of-file but was still attempting to collect all of the text for a statement.

System Action: Terminates the job. The checkpoint record is in the “restart required” state.

User Response: Add a semicolon to the end of the SQL statement(s) in the SYSSQL dataset and then re-run the job.

LDL806I VALUE LIST CANNOT BE NESTED

Explanation: You specified the value for a keyword as a value list (keyword=(item_1, item_2,...)). Another open parenthesis was encountered in the list.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Fix the error and then re-run the job.

LDL807I UNMATCHED RIGHT (CLOSING) PARENTHESIS

Explanation: You specified the value for a keyword as a value list, and the closing parentheses is missing from that list.

System Action: Continues parsing the next keyword. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Add the closing parenthesis to the list and then re-run the job.

LDL808I MISSING RIGHT (CLOSING) PARENTHESIS

Explanation: An unmatched parenthesis was detected.

System Action: Continues parsing the next statement. The job fails when parsing, control statement analysis, and PARM checking is completed.

User Response: Add the closing parenthesis and then re-run the job.

A

Messages

LDL808I

Index

Symbols

() in SQL syntax 4-3, 4-43

... in SQL syntax 4-3

{ } in SQL syntax 4-3

| in SQL syntax 4-3

' in SQL syntax 4-3, 4-23

Numerics

500, 819, and 850 code pages 5-6

65535 CCSID 4-105

8859-1 code page 5-6

A

abort operation, description 6-2

ABORT PARM 6-3, 6-11

aborting a load operation 1-37, 6-11

ACCEPT process 2-6

access privileges 1-6

account ID 5-11

as owner name 4-40

for DB2 users 4-40

in SMDEF control statement 5-11

maximum length 4-40

ACCT keyword 5-11

administrator, StorHouse 1-7

ALDLLOAD dataset 2-4

allocating required datasets 2-4

AND keyword

in delimiter specification 4-55

ANSI-format tape 4-28

application program interface (API) 4-12

APPLY process 2-6

ASCII data 4-28

ATF command privilege 1-6

B

BINARY data type 4-87, 4-105

BINARY EXTERNAL data type 4-88

Index

C

BLANK keyword 4-55

BLANKS keyword
 in CONTINUEIF clause 4-29
 in DEFAULTIF clause 4-109
 in WHEN clause 4-44

BLOB data type 4-89

block requirements 2-4

block sizes for datasets 2-4

braces in SQL syntax 4-3

brackets in SQL syntax 4-3

building the checkpoint dataset 2-6

BY keyword in FIELDS clause 4-54

C

case

 column names 4-4, 4-80
 in keywords 5-4
 in SYSSQL dataset 4-2
 table names 4-4

CCSID 65535 as a synonym for BINARY or
 VARBINARY 4-105

CCSID keyword 5-6

CCSIDs 4-19

channel connection 1-3

CHAR field_specs 4-58

CHAR keyword 4-54

CHARACTER data type 4-90

character set names 4-19, 4-20

character set, specifying 4-18

character strings 4-2

CHARACTERSET clause
 description 4-18
 examples 4-19
 format 4-19
 rules 4-18

CHARSET keyword in data type specification 4-105

checkpoint dataset
 building 2-6
 DD statement 6-4
 DDname 6-1
 description 1-36
 how it's used 2-6
 initializing 2-6

CHECKPOINT keyword
 LOAD control statement 5-6
 SMDEF control statement 5-11

checkpoint record 1-36, 5-11

checkpointing
 before copy phase 1-35
 during copy phase 1-36
 during load phase 1-36

CHKPT DD statement 6-4

choosing which rows to load
 format of WHEN clause 4-43
 overview 4-42
 specifying multiple test values 4-49
 specifying starting and ending columns 4-45
 specifying the starting column 4-45
 testing selection criteria for blanks 4-49
 using a character string as selection criteria 4-48
 using a column name 4-46
 using a field name 4-47
 using a hexadecimal string as selection criteria

- 4-48
 - using AND 4-49
 - using OR 4-49
 - using OR and AND 4-50
 - when using SKIP and TAKE keywords 4-43
- CKPT 5-6, 5-11
- clauses
 - CHARACTERSET 4-18
 - CONCATENATE 4-19
 - CONSTANT 4-82
 - CONTINUEIF 4-21
 - DEFAULTIF 4-109
 - DIFFERENT SEGMENT 4-63
 - DISCARD DN 4-14
 - DISCARD FILE 4-14
 - DISCARD MAX 4-17
 - DISCARDS 4-17
 - EXCLUDE 4-139
 - FIELDS 4-52
 - IN DDN 4-9
 - IN FILE 4-9
 - INTO TABLE 4-38
 - MAX IN SIZE 4-140
 - MIN OUT SIZE 4-140
 - POSITION 4-83
 - PRESERVE BLANKS 4-30
 - REPLACE SEGMENT 4-68
 - SAME SEGMENT 4-63
 - SEGMENT 4-67, 4-139
 - SEGMENTS 4-136, 4-139
 - SEQUENCE 4-81
 - SUBSPACE number 4-70, 4-136, 4-139
 - SUBSPACE ROTATE 4-32, 4-136, 4-139
 - summary of 4-7
 - TRAILING NULLCOLS 4-61
 - WHEN 4-42
- client data loader 1-3, 4-43
- client/server 1-1
- CLOB data type 4-91
- CLUSTER name 2-5
- coalesce operation 4-137
- code pages 5-6
- coded character set identifier (CCSID) 5-6
- collecting discarded records
 - limiting the number of discarded records 4-17
 - overview 4-14
- column default values
 - account ID 4-109
 - current date 4-109
 - current time 4-109
 - literal 4-109
 - null value 4-109
- column definition in CREATE TABLE 4-101
- column name
 - case 4-4
 - in field specification 4-79, 4-80
 - in WHEN clause 4-44
- column numbers
 - in CONTINUEIF clause 4-22
 - in POSITION clause 4-84
- column, definition 3-3
- combining a varied number of records
 - current one with next one 4-23
 - format of CONTINUEIF 4-21
 - next one with previous one 4-24
 - overview 4-21
 - specifying a comparison value 4-28
 - specifying the starting and ending column numbers 4-27
 - specifying the starting column number 4-26

Index

C

- using a hexadecimal string 4-28
- using a not equal comparison operator 4-30
- using blanks 4-29
- using last non-blank column 4-25
- command privileges 1-6
- command syntax
 - LOAD control statement 5-5
 - SMDEF control statement 5-10
- command verb of a control statement 5-2
- commands
 - CREATE ACCOUNT 1-7
 - CREATE FILE 1-7
 - CREATE FSET 1-7
 - CREATE VSET 1-7
 - LOAD 5-5
 - SHOW FILE 4-10
 - SMDEF 5-10
- comments, in SYSSQL dataset 4-2
- comparison operators 4-22
- comparison value
 - BLANKS 4-23, 4-29
 - character 4-23, 4-28
 - converting 4-30
 - converting character sets 4-28
 - definition 3-5
 - hexadecimal 4-28
 - hexdigits 4-23
 - last non-blank column 4-25
 - padding 4-27
 - trimming 4-27
- CONCATENATE clause
 - description 4-19
 - example 4-20
 - format 4-20
- concatenating a fixed number of records
 - example 4-20
 - format of CONCATENATE 4-20
 - overview 4-19
- Concepts and Facilities Manual, StorHouse xvii
- concurrency 1-2
- condition, field 4-44
- conflict of data type lengths 4-103
- consolidated software index (CSI) 2-1
- CONSTANT clause 4-82
- contiguous string in SQL identifiers 4-4
- continuation field
 - definition 3-5
 - removing from physical records 4-22
 - specifying a comparison value 4-23, 4-44
 - specifying the location of 4-27
- CONTINUEIF clause
 - arguments 4-22
 - description 4-21
 - examples 4-28
 - format 4-21
 - LAST keyword 4-25
 - NEXT keyword 4-24
 - THIS keyword 4-23
- control statements
 - command verb 5-2
 - components 5-2
 - keyword value type 5-2
 - keyword/value pairs 5-2
 - LOAD 5-5
 - MERGE 4-9
 - SMDEF 5-10
 - syntax rules 5-4
 - types of 5-1
- conventions xvi

conversion

- character set 4-48
- comparison values 4-28
- data type 1-2, 4-101

copy phase of a load operation 1-3

CREATE FILE, StorHouse command 1-7

CREATE FSET, StorHouse command 1-7

CREATE INDEX statement 1-7

CREATE TABLE SPACE statement 1-7

CREATE TABLE statement 1-7

CREATE VSET, StorHouse command 1-7

creating

- account IDs 1-7
- discard files 1-7
- file sets 1-7
- input datasets 3-1
- multiple logical records from one physical record 4-110
- user tables 4-1
- volume sets 1-7

CSI 2-1, 2-5

CSI CLUSTER 2-5

customizing JCL

- abort operation 6-11
- installation 2-3, 2-4
- load operation 6-6
- restart operation 6-9
- SMP/E JCL procedure 2-5

D

data field

column 3-3

- description 3-1
- field 3-3
- truncation 4-104

data type conversion 4-101

data type specification

- providing the length of a data type 4-102
- specifying a character set 4-105
- specifying a delimiter 4-106

data types

- BINARY 4-87
- BINARY EXTERNAL 4-88
- BLOB 4-89
- CHARACTER 4-90
- CLOB 4-91
- DATE EXTERNAL 4-92
- DECIMAL 4-93
- DECIMAL EXTERNAL 4-94
- DOUBLE 4-94
- FLOAT EXTERNAL 4-95
- FLOAT or REAL 4-95
- INTEGER 4-96
- INTEGER EXTERNAL 4-96
- SMALLINT 4-97
- TIME EXTERNAL 4-97
- TIMESTAMP EXTERNAL 4-98
- VARBINARY 4-99
- VARCHAR 4-100

data types, specific

- DATE EXTERNAL 4-102
- TIMESTAMP EXTERNAL 4-102

database name 5-6, 5-11

datasets

- ALDLLOAD 2-4
- checkpoint 2-6
- CHKPT 6-4

Index

D

- LDLLOAD 2-4
- LDLS110.F1 2-2
- LDLS110.F2 2-2
- PROCLIB 2-5
- SAMPLES 2-3
- SMP/E 2-5
- SMPCSI 2-5
- SMPMCS 2-2
- SMPMTS 2-4
- SMPPTS 2-5
- SMPSCDS 2-5
- SMPSTS 2-5
- SYSIN 5-1, 5-4
- SYSOUT 6-4
- SYSREC 5-7
- SYSSQL 4-1, 5-8
- DATE EXTERNAL data type 4-92
- DBA privilege 1-6
- DBN 5-6, 5-11
- DBNAME keyword
 - LOAD control statement 5-6
 - SMDEF control statement 5-11
- DD statements
 - CHKPT 6-4
 - STEPLIB 6-4
 - SYSERR 6-4
 - SYSIN 6-4
 - SYSPRINT 6-4
 - SYSREC 4-9, 6-4
 - SYSSQL 4-1, 6-4
 - SYSTEM 6-4
- DDDEF statements 2-5
- DDnames
 - CHKPT 6-1, 6-4
 - input dataset 5-7
 - STEPLIB 6-4
 - SYSERR 6-4
 - SYSIN 6-4
 - SYSPRINT 6-4
 - SYSREC 5-7, 6-4
 - SYSSQL 5-8, 6-4
 - SYSTEM 6-4
- DECIMAL data type 4-93
- DECIMAL EXTERNAL data type 4-94
- default code page, EBCDIC 5-6
- default lengths of data types 4-104
- default StorHouse group 4-13
- default subspace 1-12
- default value, loading into a table 4-109
- DEFAULT, VTF 5-12
- DEFAULTIF clause 4-109
- DEFAULTIF keyword 4-55
- deferred index
 - definition 1-31
 - loading 1-31, 4-134
- definitions
 - abort operation 6-2
 - blanks 3-7
 - CCSID 5-6
 - CHARACTERSET clause 4-7
 - checkpoint 5-6
 - checkpoint dataset 2-6
 - checkpoint record 2-6
 - client data loader 1-3
 - column (in input data) 3-3
 - command verb 5-2
 - comparison value 3-5
 - CONCATENATE clause 4-7
 - condition 4-21

- CONSTANT keyword 4-8
- continuation field 3-5
- CONTINUEIF clause 4-7
- control file 1-4
- copy phase of a load operation 1-3
- data field 3-1
- data file 1-4
- default length 4-104
- default subspace 1-12
- DEFAULTIF clause 4-8
- deferred index 1-31
- delimited data 3-6
- delimiter 3-6
- DIFFERENT SEGMENT clause 4-63
- DISCARD DN clause 4-7
- discarded records 4-14
- DISCARD FILE clause 4-7
- DISCARD MAX clause 4-7
- DISCARDS clause 4-7
- enclosed data 3-7
- enclosure delimiters 3-7
- EXCLUDE clause 4-139
- explicit length 4-102
- field (in input data) 3-3
- field specification 4-77
- FIELDS clause 4-7
- FileTek MVS Data Loader utility 1-4
- fixed position 4-83
- function management ID (FMID) 2-1
- host input dataset 3-1
- identifier 5-3
- implied lengths 4-103
- INFILE clause 4-7
- initialization operation 6-2
- in-line LOB 1-8
- input data record 3-1
- input dataset 3-1
- INTO TABLE clause 4-7
- keyword 5-2
- keyword/value pair 5-2
- keyword/value type 5-2
- leading blank 3-7
- LOAD control statement 5-1
- load operation 6-2
- load phase of a load operation 1-3
- logical record 3-4
- MAXINSIZE clause 4-140
- MINOUTSIZE clause 4-140
- NULLIF clause 4-8
- numeric value 5-3
- out-of-line LOB 1-8
- owner name 4-39
- physical record 3-4
- POSITION clause 4-8
- PRESERVE BLANKS clause 4-7
- quoted string 5-3
- RECNUM keyword 4-8
- relative position 4-83
- restart operation 6-2
- SAME SEGMENT clause 4-63
- SEGMENT clause 4-8
- segment tag 4-66
- SEGMENTS clause 4-136, 4-139
- SEQUENCE clause 4-8
- server data loader 1-3
- SMDEF control statement 5-1
- space 3-7
- SQL identifier 4-4
- SQL_LDR_ENGINES system parameter 1-32
- SQL_LDR_MAXINTO system parameter 4-38
- SQL_LDR_MAXLOAD system parameter 1-32
- SQL_SESSIONS system parameter 1-32
- string value 5-2
- subspace 1-11
- SUBSPACE number clause 4-70
- SUBSPACE ROTATE clause 4-32
- substitution string 4-40
- symbolic variable 4-40

Index

E

- SYSDATE keyword 4-8
 - SYSSQL dataset 4-1
 - table name 4-39
 - target data types 4-101
 - terminated data 3-7
 - termination delimiter 3-7
 - trailing blank 3-8
 - TRAILING NULLCOLS clause 4-8
 - value 5-2
 - WHEN clause 4-7
 - whitespace 3-7
- DELETE command privilege 1-6
- delimited SQL identifiers 4-4
- DELIMITER keyword 4-60
- delimiter specification
- AND keyword 4-55
 - character delimiter 4-55
 - examples 4-56, 4-57
 - hexadecimal delimiter 4-55
 - in a data type specification 4-106
 - in a FIELDS clause 4-54
 - WHITESPACE keyword 4-55
- describing each column to load
- generating a sequence of values 4-81
 - loading a constant value 4-82
 - loading a default value 4-109
 - loading a null value 4-108
 - loading a record number 4-81
 - loading the system date 4-82
 - providing a column name 4-80
 - providing the data type length 4-102
 - providing the data type name 4-86
 - specifying a character set 4-105
 - specifying a delimiter 4-106
 - specifying the position 4-83
- dialogs, SMP/E 2-5
- DIFFERENT SEGMENT clause 4-63
- DIRECT, VTF 5-12
- discard file, creating 1-7
- discarded records
- accessing 4-15
 - collecting 4-14
 - definition 4-14
 - limiting the number of 4-17
 - loading 4-13
 - overwriting 4-15
 - when they're not saved 4-15
- DISCARDFILE/DISCARD DN clause
- description 4-14
 - discard file name 4-16
 - guidelines 4-16
 - StorHouse group name 4-16
- DISCARDS/DISCARD MAX clause
- description 4-17
 - example 4-17
 - format 4-17
 - guidelines 4-17
- disk space requirements 2-2
- disposition
- checkpoint dataset 2-7
 - VRAM file 5-9
- distribution tape 2-2
- distribution zone load library 2-4
- DOUBLE data type 4-94
- ## E
- EBCDIC character set 4-19

- EBCDIC code page 5-6
 - ellipsis points in SQL syntax 4-3
 - empty data field 4-50
 - EMPTY keyword 4-55
 - enclosed data 3-7
 - ENCLOSED keyword in FIELDS clause 4-55
 - enclosure delimiter 3-7
 - engine 1-5
 - E-notation 4-94, 4-95
 - error messages
 - in SYSERR 6-5
 - in SYSPRINT 6-5
 - list of A-1
 - runtime 6-4
 - SAS/C 6-4
 - error reporting 1-2
 - ESCAPED BY clause 4-59
 - ESCON 1-3
 - examples
 - CHARACTERSET clause 4-19
 - CHARSET keyword 4-105
 - CONCATENATE clause 4-20
 - CONSTANT clause 4-83
 - CONTINUEIF clause 4-23, 4-24, 4-25
 - delimiter specification for a data field 4-108
 - DIFFERENT SEGMENT clause 4-64
 - DISCARDS/DISCARDMAX clause 4-17
 - ESCAPED BY clause 4-61
 - FIELDS clause 4-56, 4-57, 4-59
 - INFILE clause 4-12
 - INFILE/INDDN clause 4-12, 4-13, 4-14
 - INTO TABLE clause 4-39, 4-40
 - LOAD DATA statement 4-112
 - LOAD INDEX statement 4-137
 - MERGE statement 4-140
 - multiple INTO TABLE specifications 4-110
 - NULLIF keyword 4-108
 - POSITION 4-84
 - PRESERVE BLANKS clause 4-31
 - RECNUM keyword 4-81
 - relative positioning 4-121
 - REPLACE SEGMENT clause 4-70
 - SAME SEGMENT clause 4-64
 - SEGMENT clause 4-67
 - SEQUENCE clause 4-81
 - SUBSPACE number clause 4-72
 - SUBSPACE ROTATE clause 4-34
 - SYSDATE keyword 4-82
 - TRAILING NULLCOLS clause 4-63
 - WHEN clause 4-45, 4-46, 4-47
 - exception processing 1-2
 - EXCLUDE clause 4-139
 - EXEC statement
 - contents 6-3
 - examples 6-3
 - PARM 6-3
 - PGM 6-3
 - REGION 6-3
 - executing
 - SMP/E ACCEPT 2-6
 - SMP/E APPLY 2-6
 - SMP/E RECEIVE 2-6
 - explicit lengths 4-102
 - extents 1-8
- ## F
- features, FileTek MVS Data Loader utility 1-1

Index

F

- field
 - in input data record 3-3
 - specification 4-77
- field condition 4-44
- field name
 - in field specification 4-79, 4-80
 - in WHEN clause 4-44
- field selection criteria
 - padding 4-45
 - specifying multiple test values 4-49
 - specifying starting and ending columns 4-45
 - specifying the starting column 4-45
 - testing for blanks 4-49
 - trimming 4-46
 - using a character string 4-48
 - using a column name 4-46
 - using a field name 4-47
 - using a hexadecimal string 4-48
 - with NULLIF keyword 4-108
- field specification
 - format 4-78
 - generating data 4-81, 4-82
 - identifying the position of a data field 4-83
 - including a delimiter specification 4-106
 - providing the data type length 4-102
 - specifying a column name 4-79
 - specifying a field name 4-79
 - using the CHARSET keyword 4-105
 - using the DEFAULTIF clause 4-109
 - using the NULLIF keyword 4-108
- FIELDS clause
 - AND keyword 4-57
 - BY keyword 4-54
 - ENCLOSED keyword 4-55, 4-57
 - examples 4-57
 - OPTIONALLY keyword 4-55, 4-57
 - overview 4-52
 - TERMINATED keyword 4-56
 - WHITESPACE keyword 4-56
- file access group 5-12
- file set name 5-11
- file set, VRAM file 1-7
- files on the distribution tape 2-2
- FileTek MVS Data Loader utility
 - client component 1-3
 - control statements 5-1
 - creating a LOAD DATA statement 4-1
 - definition 1-4
 - features 1-1
 - installing 2-1
 - return codes 6-5
 - running 6-1
 - server component 1-3
 - what to do before running 1-7, 6-1
- FLOAT (REAL) data type 4-95
- FLOAT EXTERNAL data type 4-95
- FMID 2-1
- FNPREFIX keyword 5-7
- format conventions 4-3
- formats
 - CHARACTERSET 4-19
 - CHARACTERSET clause 4-19
 - CONCATENATE clause 4-20
 - CONSTANT clause 4-83
 - CONTINUEIF clause 4-21
 - datatype_spec 4-87
 - DEFAULTIF clause 4-109
 - delimiter_spec 4-54
 - DIFFERENT SEGMENT clause 4-64
 - DISCARDFILE/DISCARDDN clause 4-16
 - DISCARDS/DISCARDMAX clause 4-17

- ESCAPED BY clause 4-60
- field_spec 4-78
- FIELDS clause 4-54
- INTO TABLE clause 4-39
- LOAD DATA statement 4-5
- LOAD INDEX statement 4-135
- MERGE statement 4-138
- POSITION clause 4-84
- RECNUM keyword 4-81
- SAME SEGMENT clause 4-64
- SEGMENT clause 4-64
- SEQUENCE clause 4-81
- SUBSPACE number clause 4-72
- SUBSPACE ROTATE clause 4-34
- SYSDATE keyword 4-82
- WHEN clause 4-43

FREE=CLOSE 3-2

FROMDD keyword 5-7

FSET keyword 5-11

fully qualified table name 4-39

function identifier 2-1

function management ID (FMID) 2-1

G

generating a sequence of values 4-81

GET command privilege 1-6

GRANT statement 1-7

GROUP keyword 5-12

group name, StorHouse 4-11

group, StorHouse 5-12

H

HASH keyword in SUBSPACE number clause 4-72

hexadecimal strings 4-2

hexdigits

- in CONTINUEIF clause 4-23
- in WHEN clause 4-44

host input dataset 3-1

host SQL dataset 5-8

I

IBM Registry 5-6

identifier of a keyword 5-3

identifiers, delimited 4-4

identifying the user table to load

- omitting the owner name 4-40
- overview 4-38
- using a symbolic variable 4-40
- using the fully qualified table name 4-39

implied lengths 4-103

in SQL syntax 4-3

index

- deferred 1-31
- loading 4-134

index load operation 4-134

index name 4-136

INFILE/INDDN clause

- loading data from a host data file 4-14
- loading data from a previous load operation 4-11

- loading data from a VRAM file 4-12
- loading discarded records 4-13
- NOENVIRON keyword 4-11, 4-13
- obtaining a VRAM file name 4-10
- StorHouse file name in 4-11
- when to specify a group name 4-12
- INIT PARM 6-3
- INITCKPT member 2-4
- initialization operation, description 6-2
- initializing
 - SMP/E CSI 2-5
 - the checkpoint dataset 2-6
- in-line LOB 1-8
- in-line LOBs 4-89, 4-91
- input data record
 - column 3-3
 - definition 3-1
 - field 3-3
 - logical 3-4
 - physical 3-4
 - record formats 3-2
- input dataset
 - considerations 3-2
 - creating 3-1
 - DDname 5-7
 - definition 3-1
 - how to specify for a load 3-2
 - multiple 3-2
 - on tape 3-2
- INSERT database component privilege 1-6
- INSERT privilege on user table 1-7
- installation procedure
 - 1. load the SAMPLES dataset 2-3
 - 2. allocate required datasets 2-4
 - 3. customize SMP/E JCL procedure 2-5
 - 4. initialize SMP/E CSI 2-5
 - 5. execute SMP/E RECEIVE 2-6
 - 6. execute SMP/E APPLY 2-6
 - 7. execute SMP/E ACCEPT 2-6
 - 8. build the checkpoint dataset 2-6
- installing the utility
 - allocate required datasets 2-4
 - build the checkpoint dataset 2-6
 - customize SMP/E JCL procedure 2-5
 - execute SMP/E ACCEPT 2-6
 - execute SMP/E APPLY 2-6
 - execute SMP/E RECEIVE 2-6
 - initialize SMP/E CSI 2-5
 - load the SAMPLES dataset 2-3
 - overview 2-1
 - software function identifier 2-1
 - system requirements 2-2
- instream dataset 6-4
- INTEGER data type 4-96
- INTEGER EXTERNAL data type 4-96
- INTO TABLE clause
 - description 4-38
 - example 4-39
 - format 4-39
 - maximum number 4-38
 - multiple 4-38, 4-39
 - owner name 4-39
 - substitution 4-40
 - table name 4-39
- INTO TABLE specification
 - multiple 4-110
 - syntax 4-5
- ISO 8859-1 character set 4-19
- ISO code page 5-6

ISPF 2-5

J

JCL

- in the SAMPLES dataset 2-2
- sample abort operation 6-11
- sample load operation 6-7, 6-8
- sample restart operation 6-10

K

keyword value type

- definition 5-2
- identifier 5-3
- numeric 5-3
- quoted string value 5-3
- string 5-2

keyword/value pairs 5-2

keywords, control statement

- ACCT 5-11
- CCSID (LOAD) 5-6
- CHECKPOINT 5-6
- CHECKPOINT (SMDEF) 5-11
- DBNAME (LOAD) 5-6
- DBNAME (SMDEF) 5-11
- FNPREFIX (LOAD) 5-7
- FROMDD(LOAD) 5-7
- FSET (SMDEF) 5-11
- GROUP (SMDEF) 5-12
- LOADID_OFFSET (LOAD) 5-7
- Pn (LOAD) 5-8
- SKIP (LOAD) 5-8
- SM_NAME (SMDEF) 5-12
- SQLDD (LOAD) 5-8

SUBS (SMDEF) 5-12

TAKE (LOAD) 5-9

TEMP_FILE (LOAD) 5-9

VSET (SMDEF) 5-12

VTF (SMDEF) 5-12

keywords, LOAD DATA statement

- AND in FIELDS clause 4-55
- BLANKS in WHEN clause 4-44
- BY in FIELDS clause 4-54
- CHAR in FIELDS clause 4-54
- CHARSET in data type specification 4-105
- CONCATENATE 4-20
- CONSTANT 4-82
- DEFAULTIF in a field specification 4-109
- DEFAULTIF in FIELDS clause 4-55
- DELIMITER 4-60
- DIFFERENT SEGMENT 4-64
- DISCARDFILE/DISCARDMAX 4-16
- DISCARDS/DISCARDMAX 4-17
- ENCLOSED in FIELDS clause 4-55
- ESCAPED BY 4-60
- HASH in SUBSPACE number clause 4-72
- INFILE/INDDN 4-11
- LAST in CONTINUEIF clause 4-22
- NEXT in CONTINUEIF clause 4-22
- NOENVIRON in INFILE/INDDN clause 4-11
- NONE 4-60
- NULLFLAGS in FIELDS clause 4-54
- NULLIF in a field specification 4-108
- NULLIF in FIELDS clause 4-55
- OPTIONALLY in FIELDS clause 4-55
- OR in WHEN clause 4-49
- PRESERVE BLANKS 4-31
- RECNUM 4-81
- REPLACE SEGMENT 4-70
- SAME SEGMENT 4-64
- SEGMENT 4-66
- SEQUENCE 4-81

Index

L

- SUBSPACE ROTATE 4-34
 - SYSDATE 4-82
 - TABLE in SUBSPACE number clause 4-72
 - TERMINATED in FIELDS clause 4-54
 - THIS in CONTINUEIF clause 4-22
 - TRAILING NULLCOLS 4-62
 - VALUE in SUBSPACE NUMBER clause 4-72
 - WHEN 4-44
 - WHITESPACE in FIELDS clause 4-55
 - keywords, LOAD INDEX statement
 - SEGMENTS 4-136
 - SUBSPACE 4-136
 - SUBSPACE ROTATE 4-136
 - keywords, MERGE statement
 - EXCLUDE 4-139
 - MAXINSIZE 4-140
 - MINOUTSIZE 4-140
 - SEGMENT 4-139
 - SEGMENTS 4-139
 - SUBSPACE 4-139
 - SUBSPACE ROTATE 4-139
- ## L
- large objects (LOBs)
 - BLOB data type 4-89
 - CLOB data type 4-91
 - in-line 1-8
 - loading 3-5
 - out-of-line 1-8
 - subsegment file 1-8
 - LAST keyword in CONTINUEIF clause 4-22, 4-25
 - LDLLOAD dataset 2-4
 - LDLS110.F1 dataset 2-2
 - LDLS110.F2 dataset 2-2
 - LDLSLDR program name 6-3
 - LDLSLDR utility 2-1
 - LDRSMPE member 2-4, 2-5
 - limiting the number of discarded records 4-17
 - LINKLIST 6-4
 - LOAD control statement
 - general
 - command syntax 5-5
 - command verb 5-2
 - keyword value types 5-2
 - keyword/value pairs 5-2
 - purpose 5-1
 - syntax rules 5-4
 - keywords
 - CCSID 5-6
 - CHECKPOINT 5-6
 - DBNAME 5-6
 - FNPREFIX 5-7
 - FROMDD 5-7
 - LOADID_OFFSET 5-7
 - Pn 5-8
 - SKIP 5-8
 - SQLDD 5-8
 - TAKE 5-9
 - TEMP_FILE 5-9
 - LOAD DATA statement
 - examples
 - all records, one user table 4-113
 - binary and variable-length data 4-119
 - combining records, null values 4-117
 - combining records, one user table 4-115
 - delimited data, multiple user tables 4-116
 - multiple logical records from one physical record 4-110
 - multiple selection criteria, data in control file 4-126
 - same input dataset, multiple user tables 4-111
 - selecting subspaces 4-129
 - general

- components 4-5
 - format 4-5
 - summary of clauses and keywords 4-7
- specifications
 - data_spec 4-79
 - datatype_spec 4-79, 4-86
 - delimiter_spec 4-54
 - field_spec 4-77
 - into_table_spec 4-39
 - position_spec 4-79
- syntax
 - CHARACTERSET clause 4-19
 - CONCATENATE clause 4-20
 - CONSTANT clause 4-82
 - CONTINUEIF clause 4-21
 - DEFAULTIF clause 4-109
 - DIFFERENT SEGMENT clause 4-64
 - DISCARDFILE/DISCARDDDN clause 4-16
 - DISCARDS/DISCARDMAX clause 4-17
 - ESCAPED BY 4-60
 - FIELDS clause 4-54
 - INFILE/INDDN clause 4-10
 - POSITION clause 4-84
 - PRESERVE BLANKS clause 4-31
 - REPLACE SEGMENT clause 4-70
 - SAME SEGMENT clause 4-64
 - SEGMENT clause 4-67
 - SEQUENCE clause 4-81
 - SUBSPACE number clause 4-72
 - SUBSPACE ROTATE clause 4-34
 - TRAILING NULLCOLS clause 4-62
 - WHEN clause 4-43
- tasks
 - collecting discarded records 4-14
 - combining a varied number of records 4-21
 - concatenating a fixed number of records 4-19
 - describing data fields 4-77
 - generating a sequence of values 4-81
 - identifying the user table 4-38
 - limiting the number of discarded records 4-17
 - loading a constant value 4-82
 - loading a record number 4-81
 - loading missing data fields with null values 4-61
 - loading one or more segments 4-63
 - loading the system date 4-82
 - naming a segment 4-66
 - preserving blanks 4-30
 - rotating among subspaces 4-32
 - selecting logical records 4-42
 - selecting subspaces 4-70
 - setting a column to the default value 4-109
 - specifying a default delimiter 4-52
 - specifying the character set 4-18
 - specifying the position of a data field 4-83
 - using multiple INTO TABLE specifications 4-110
- load ID 4-66
- load index operation 1-31
- LOAD INDEX statement
 - examples 4-137
 - format 4-135
 - guidelines 4-134
 - purpose 4-134
- load library 2-2
 - data loader 6-4
 - StorHouse 6-4
- load operation
 - copy phase 1-3
 - description 6-2
 - load phase 1-3
- LOAD PARM 6-3, 6-6
- load phase of a load operation 1-3
- LOADID 5-7
- LOADID_OFFSET keyword 5-7
- loading
 - a column with a null value 4-108
 - a constant value 4-82
 - a default value 4-109
 - a deferred index 4-134

Index

M

- a record number 4-81
- all records into one user table 4-114
- data from a host data file 4-14
- data from a previous load 4-11
- delimited data into multiple user tables 4-116
- different tables in multiple loads 1-33
- different tables in one load 1-32
- discarded records 4-13
- multiple segments of a table in one load 1-33
- multiple segments of multiple tables in multiple loads 1-34
- multiple segments of multiple tables in one load 1-33
- multiple table segments 4-63
- multiple user tables 4-111
- null values 4-117, 4-121
- SAMPLES dataset during installation 2-3
- SMALLINT, DECIMAL, and VARCHAR data 4-119
- some records into one user table 4-115
- the same table in multiple loads 1-34
- the system date 4-82
- locks 1-35
- logical record 3-4
- lowercase in SQL syntax 4-3

M

MAXINSIZE clause 4-140

member of a partitioned dataset 3-2

members, SAMPLES dataset

- INITCKPT 2-7
- LDRSMPE 2-5
- list of 2-4
- RUNLOAD 2-4

SMPACEPT 2-6

SMPALLOC 2-4

SMPAPPLY 2-6

SMPDDDEF 2-5

SMPRECV 2-6

SMPUCLIN 2-5

SQLSAMP 2-4

merge operation 1-11, 4-137

MERGE statement

- examples 4-140

- format 4-138

- guidelines 4-138

- purpose 4-137

merging segments 4-137

message numbers

- 680I A-1

- 681I A-1

- 682I A-2

- 683I A-2

- 684I A-2

- 685I A-2

- 686I A-3

- 687I A-3

- 688I A-3

- 689I A-4

- 690I A-4

- 694I A-4

- 698I A-5

- 710I A-5

- 711I A-5

- 712I A-6

- 713D A-6

- 713Q A-6

- 714I A-6

- 715I A-7

- 716I A-7

- 717I A-7

718I A-8
719I A-8
720I A-8
721I A-9
722I A-9
723I A-9
725I A-10
726I A-10
727I A-10
728I A-11
730I A-11
731I A-11
732I A-12
733I A-12
734I A-12
735I A-13
736I A-13
737I A-13
738I A-13
739I A-14
740I A-14
741I A-14
742I A-15
743I A-15
744I A-16
745I A-16
746I A-16
747I A-16
748I A-17
749I A-17
750I A-17
751I A-18
752I A-18
753I A-18
754I A-19
755I A-19
756I A-19
757I A-20
758I A-20

759I A-20
760I A-21
761I A-21
762I A-21
763I A-22
764I A-22
765I A-22
766I A-23
767I A-23
768I A-23
770I A-23
771I A-24
772I A-24
773I A-24
774I A-25
775I A-25
777I A-25
778I A-26
779I A-26
780I A-26
806I A-27
807I A-27
808I A-27

messages

error 6-4
in SYSERR 6-4
in SYSPRINT 6-4
list of A-1
runtime 6-4
SAS/C 6-4

Messages and Codes Manual, StorHouse xvii

metadata updates

data load operation 1-38
load index operation 1-39
merge operation 1-39
replace operation 1-38

MINOUTSIZE clause 4-140

multiple logical records in one physical record 4-110

N

naming a table segment 4-66

NEXT keyword in CONTINUEIF clause 4-22,
4-24

NEXT, VTF 5-12

NOENVIRON keyword in INFILE/INDDN clause
4-11

NONE keyword 4-60

not equal comparison operators 4-30

NOT NULL 4-62

notational conventions xvi

null value

- loading a column with (NULLIF keyword)
4-108

- loading missing data fields (TRAILING
NULLCOLS clause) 4-61

NULLFLAGS keyword 4-54, 4-58

NULLIF keyword 4-55, 4-108

number, subspace 4-72

numeric value of a keyword 5-3

O

object identifier 4-89, 4-91

OBJECT_TYPE parameter 1-12

OID 4-89, 4-91

omitting the owner name 4-40

operations, types of 6-2

operators, comparison 4-22

OPTIONALLY keyword in FIELDS clause 4-55

OR keyword in WHEN clause 4-49

out-of-line LOB 1-8

out-of-line LOBs 4-89, 4-91

owner name

- in REPLACE SEGMENT clause 4-70

- length of 4-39

- omitting 4-40

- specifying 4-39

- substituting 4-39

- when longer than 12 characters 4-40

P

padding

- comparison values 4-27

- selection criteria 4-45

parallelism

- loading different tables in multiple loads 1-33

- loading different tables in one load 1-32

- loading multiple segments of a table in one load
1-33

- loading multiple segments of multiple tables in
multiple loads 1-34

- loading multiple segments of multiple tables in
one load 1-33

- loading the same table in multiple loads 1-34

- querying a table while it's being loaded 1-35

- system parameters 1-31

parameters, EXEC statement 6-3

PARM

examples

- abort operation 6-11
- load operation 6-7, 6-8
- restart operation 6-10

values

- ABORT 6-3, 6-11
- INIT 2-7, 6-3
- LOAD 6-3, 6-6
- RESTART 6-3, 6-9
- RESTART_IGNORE 6-3, 6-9
- RESTART_RELOAD 6-3, 6-9

partitioned dataset 3-2

password, StorHouse account 1-6, 5-11

PC character set 4-19

PC code page 5-6

PDS 3-2

performance buffer 5-12

PGM 6-3

phases of a load operation

- copy 1-3
- load 1-3

physical record 3-4

Pn keyword 5-8

POSITION clause

- arguments 4-84
- description 4-83
- examples 4-84
- format 4-84

prefix, VRAM file name 5-7

preparing input 1-4

preparing to run the utility 6-1

PRESERVE BLANKS clause

example 4-31

format 4-31

privileges for loading 1-6, 4-40

PROCLIB dataset 2-5

PUT command privilege 1-6

Q

quick reference xvii

quoted string of a keyword 5-3

quoted strings 4-2

quotes

- in character strings 4-2
- in control statement keywords 5-3
- in delimited SQL identifiers 4-4
- in hexadecimal strings 4-2
- in PARM 6-3

R

RECEIVE process 2-6

RECNUM keyword 4-81

RECORD command privilege 1-6

record formats of input data records 1-2, 3-2

region size 6-3

relative positioning 4-121

REPLACE SEGMENT clause 4-70

replacing a segment 4-68

reserved words 4-4

Index

S

- RESOURCE privilege 1-6
 - restart operation, description 6-2
 - RESTART PARM 6-3, 6-9
 - RESTART_IGNORE PARM 6-3, 6-9
 - RESTART_RELOAD PARM 6-3, 6-9
 - restarting a load operation 1-37, 6-8
 - return codes 6-5
 - rotating among subspaces 1-24, 4-32
 - run statistics 6-4
 - RUNLOAD member 2-4
 - running the utility
 - aborting a load operation 6-11
 - DD statements 6-4
 - EXEC statement 6-3
 - restarting a load operation 6-8
 - return codes 6-5
 - sample abort JCL 6-11
 - sample load JCL 6-7, 6-8
 - sample restart JCL 6-10
 - submitting a load operation 6-6
 - types of operations 6-2
 - what to do before 6-1
 - runtime information 5-1
 - runtime messages 6-4
-
- ## S
- SAME SEGMENT clause 4-63
 - sample JCL
 - abort operation 6-11
 - installation 2-2
 - load operation 6-7, 6-8
 - restart operation 6-10
 - SAMPLES dataset 2-2
 - INITCKPT member 2-7
 - LDRSMPE member 2-5
 - list of members 2-4
 - SMPACEPT member 2-6
 - SMPALLOC member 2-4
 - SMPAPPLY member 2-6
 - SMPDDDEF member 2-5
 - SMPRECV 2-6
 - SMPUCLIN member 2-5
 - SQLSAMP member 2-4
 - unloading 2-3
 - SAS/C error messages 6-4
 - SEGMENT clause 4-67, 4-139
 - segment ID 1-38
 - segment list 4-136
 - segment tag 4-67, 4-70
 - segmentation 1-8
 - segments
 - description 1-8
 - loading multiple 4-63
 - merging 4-137
 - naming 4-66
 - replacing 4-68
 - size of 1-10
 - SEGMENTS clause 4-136, 4-139
 - SELECT statement 4-1
 - selecting subspaces 1-16, 4-70
 - SEQUENCE clause 4-81
 - sequential dataset 3-2, 6-4

- server data loader 1-3, 4-43
- SETGROUP command privilege 1-6
- shared lock 1-35
- short record 4-61
- SHOW FILE, StorHouse command 4-10
- shutdown, restart after 6-8
- sizes, segment 1-10
- SKIP keyword 4-43, 5-8
- skipping input data records 5-8
- SM_NAME keyword 5-12
- SMALLINT data type 4-97
- SMALLINT in VARCHAR data fields 3-5
- SMDEF control statement
 - general
 - command syntax 5-10
 - command verb 5-2
 - keyword value types 5-2
 - keyword/value pairs 5-2
 - purpose 5-1
 - syntax rules 5-4
 - keywords
 - ACCT 5-11
 - CHECKPOINT 5-11
 - DBNAME 5-11
 - FSET 5-11
 - GROUP 5-12
 - SM_NAME 5-12
 - SUBS 5-12
 - VSET 5-12
 - VTF 5-12
- SMP processing 2-4
- SMP/E
 - general
 - CSI 2-5
 - DDDEF statements 2-5
 - description 2-1
 - dialogs 2-5
 - JCL procedure 2-5
 - required datasets 2-5
 - processes
 - ACCEPT 2-6
 - APPLY 2-6
 - RECEIVE 2-6
- SMPACCEPT member 2-4, 2-6
- SMPALLOC member 2-4
- SMPAPPLY member 2-4, 2-6
- SMPCSI dataset 2-5
- SMPDDDEF member 2-4, 2-5
- SMPMCS dataset 2-2
- SMPMTS dataset 2-4
- SMPPTS dataset 2-5
- SMPRECV member 2-4, 2-6
- SMPSCDS dataset 2-5
- SMPSTS dataset 2-5
- SMPUCLIN member 2-4, 2-5
- software
 - function identifier 2-1
 - release number 2-1
 - version number 2-1
- spaces in SYSSQL dataset 4-2
- specifying a default delimiter
 - describing data fields enclosed by different delimiters 4-57
 - describing data fields enclosed by the same delimiter 4-57
 - describing data fields terminated by blanks 4-56
 - describing data fields terminated with a character

Index

S

- 4-56
- describing data fields that are both terminated and enclosed 4-57
- overview 4-52
- specifying keyword values
 - identifier 5-3
 - null 5-3
 - numeric value 5-3
 - quoted string 5-3
 - string 5-2
- specifying the character set of the input data
 - overview 4-18
 - with the CCSID keyword 5-6
 - with the CHARACTERSET clause 4-19
 - with the CHARSET keyword 4-105
- SQL
 - format conventions 4-3
 - host dataset 5-8
 - identifiers 4-4
 - in SYSSQL 4-1
 - reserved words 4-4
- SQL statements
 - CREATE INDEX 1-7, 4-130
 - CREATE TABLE 1-7, 4-130
 - CREATE TABLE SPACE 1-7, 4-130
 - GRANT 1-7
 - SELECT 4-1
- SQL syntax
 - braces { } 4-3
 - commas , 4-3
 - ellipsis points ... 4-3
 - lowercase terms 4-3
 - semicolon 4-2
 - single quotes ' 4-28
 - uppercase terms 4-3
 - vertical bar | 4-3
- SQL_LDR_ENGINES system parameter 1-32
- SQL_LDR_MAXINTO system parameter 1-32, 4-38, 4-64
- SQL_LDR_MAXLOAD system parameter 1-32
- SQL_SESSIONS system parameter 1-32
- SQLCOMMAND access privilege 1-6
- SQLDD keyword 5-8
- SQLEXECUTE access privilege 1-6
- SQLSAMP member 2-4
- STEPLIB DD statement 6-4
- STHLDR.TEMPF 5-7
- StorHouse 1-1, 3-1
 - account ID 5-11
 - group 5-12
 - group name 4-11, 4-16
 - identifier 5-12
 - password 5-11
 - privileges 1-6
 - product description xiii
 - subsystem name 5-12
 - system parameters 1-32
- StorHouse Database Administration Guide xvi
- StorHouse documentation
 - Concepts and Facilities Manual xvii
 - Messages and Codes Manual xvii
- StorHouse system administrator 1-7
- StorHouse/Control Center, description xiv
- StorHouse/RM Concepts xvi
- StorHouse/RM, description xiv
- StorHouse/SM, description xiii

- string value of a keyword 5-2
- strings 4-2
- submitting
 - a load operation 6-6
 - SQL statements 4-1
- SUBS keyword 5-12
- SUBSPACE 4-34
- SUBSPACE number clause
 - description 4-70, 4-136, 4-139
 - examples 4-72
 - format 4-72
- SUBSPACE ROTATE clause
 - description 1-24, 4-32, 4-136, 4-139
 - examples 4-34
 - format 4-34
- subspaces
 - default 1-12
 - definition 1-11
 - number 4-72
 - rotating among 1-24, 4-32, 4-136, 4-139
 - selecting 1-16, 4-70
- substituting
 - both owner and table names 4-41
 - owner name 4-40
 - part of a table name 4-41
- substitution string 4-40, 5-8
- symbolic variable 4-40
- syntax
 - CHARACTERSET clause 4-19
 - CONCATENATE clause 4-20
 - CONSTANT clause 4-83
 - CONTINUEIF clause 4-21
 - datatype_spec 4-87
 - DEFAULTIF clause 4-109
 - delimiter_spec 4-54
 - DIFFERENT SEGMENT clause 4-64
 - DISCARDFILE/DISCARD DN clause 4-16
 - DISCARDS/DISCARDMAX clause 4-17
 - ESCAPED BY clause 4-60
 - FIELDS clause 4-54
 - INTO TABLE clause 4-39
 - LOAD control statement 5-5
 - LOAD DATA statement 4-5
 - LOAD INDEX statement 4-135
 - MERGE statement 4-138
 - POSITION clause 4-84
 - PRESERVE BLANKS clause 4-31
 - RECNUM keyword 4-81
 - REPLACE SEGMENT clause 4-70
 - SAME SEGMENT clause 4-64
 - SEQUENCE clause 4-81
 - SMDEF control statement 5-10
 - SQL statements 4-3
 - SUBSPACE number clause 4-72
 - SUBSPACE ROTATE clause 4-34
 - SYSDATE keyword 4-82
 - SYSIN 5-4
 - TRAILING NULLCOLS clause 4-62
 - WHEN clause 4-43
- SYSDATE keyword 4-82
- SYSERR DD statement 6-4
- SYSIN dataset 5-4
- SYSIN DD statement 6-4
- SYSINDEXES system table 1-39
- SYSOUT dataset 6-4
- SYSPLEX 5-7
- SYSPRINT DD statement 6-4
- SYSREC dataset 5-7

Index

T

SYSREC DD statement 6-4

SYSSQL dataset 4-1, 4-2, 5-8

SYSSQL DD statement 6-4

SYSSTHFILES system table 1-38, 1-39

SYSSTHSEGMENTS system table 1-38

SYSTABLES system table 1-38

System Modification Program Extended (SMP/E)
2-1

system parameters

SQL_LDR_ENGINES 1-32

SQL_LDR_MAXINTO 1-32, 4-38, 4-64

SQL_LDR_MAXLOAD 1-32

SQL_MAX_EXT_VAL 1-36

SQL_SESSIONS 1-32

VTF 5-12

system requirements for installation 2-2

system table updates 1-38

SYSTEM DD statement 6-4

SYSTHSEGMENTS system table 1-38

T

table ID 1-38

TABLE keyword in SUBSPACE number clause 4-72

table name

case 4-4

fully qualified 4-39

in INTO TABLE clause 4-39

in MERGE statement 4-139

in REPLACE SEGMENT clause 4-70

partial 4-41

substituting 4-39

tables

block requirements and sizes of datasets 2-4

CHARACTERSET clause 4-19

CONCATENATE clause 4-20

CONTINUEIF clause 4-22

data type specifications 4-87

DD statements 6-4

DISCARDFILE/DISCARDDN clause 4-16

DISCARDS/DISCARDMAX clause 4-17

ESCAPED BY clause 4-60

field specification 4-79, 4-81, 4-83

FIELDS clause 4-54

files on the distribution tape 2-2

INFILE/INDDN clause 4-11

INTO TABLE clause 4-39

LOAD control statement keywords 5-6

LOAD INDEX arguments 4-136

LOAD INDEX format 4-8

members of the SAMPLES dataset 2-4

MERGE arguments 4-139

MERGE format 4-9

POSITION clause 4-84

REPLACE SEGMENT clause 4-70

return codes 6-5

SEGMENT clause 4-67

SMDEF control statement keywords 5-11

SQL format conventions 4-3

StorHouse tasks 1-7

SUBSPACE number clause 4-72

summary of LOAD DATA clauses 4-7

WHEN clause 4-44

TAKE keyword 4-43, 5-9

taking input records 5-9

target zone load library 2-4

TEMP_FILE keyword 5-9

temporary VRAM file 1-4

terminated data 3-7

TERMINATED keyword in FIELDS clause 4-54

termination delimiter 3-7

THIS keyword in CONTINUEIF clause 4-22, 4-23

TIME EXTERNAL data type 4-97

TIMESTAMP EXTERNAL data type 4-98

TLIB 2-5

TRAILING NULLCOLS clause 4-62

trimming

- comparison values 4-27

- data fields 4-104

- selection criteria 4-45

types of operations

- abort 6-2

- initialization 2-7, 6-2

- load 6-2

- replace 4-68

- restart 6-2

U

UCLIN 2-5

uppercase in SQL syntax 4-3

using data on StorHouse

- format of INFILE/INDDN clause 4-10

- loading data from a previous load 4-11

- loading data in a VRAM file 4-12

- loading discarded records 4-13

- obtaining a VRAM file name 4-10

using the fully qualified table name 4-39

V

VALUE keyword in SUBSPACE number clause 4-72

VARBINARY data type 4-99, 4-105

VARCHAR data type

- actual length 3-5

- as selection criteria 4-48

- considerations 3-5

- conversion 3-6

- description 4-100

- maximum length 4-103

- starting column 4-83

Variable-Blocked-Spanned (VBS) 3-2

verb of a control statement 5-2

vertical bar in SQL syntax 4-3

volume set name 5-12

volume set, VRAM file 1-7

VRAM file

- backup 5-12

- default prefix name 5-7

- deleting 5-9

- discard file 4-15

- file set name 5-11

- group 5-12

- keeping 5-9

- prefix 5-7

- volume set and file set 1-7

- volume set name 5-12

- VTF 5-12

- what it is 1-4

VSET 5-12

VSET keyword 5-12

VTF command privilege 1-6

VTF keyword 5-12

vulnerability time factor (VTF) 5-12

W

WHEN clause

- charstring 4-44

- column name 4-44

- creating 4-42

- field name 4-44

- format 4-43

- hexdigits 4-44

- options 4-42

- purpose 4-42

WHITESPACE keyword in FIELDS clause 4-55

Z

ZONE names 2-5

zones, CSI 2-5