8

SQL*Loader Command-Line Reference

To start regular SQL*Loader, use the command-line parameters.

> **Note:**
>
> Regular SQL*Loader and SQL*Loader Express mode share some of the same parameters, but the behavior of these parameters can be different for each utility. The parameter descriptions described here are for regular SQL*Loader. For SQL*Loader Express options, refer to the SQL*Loader Express parameters.

- Starting SQL*Loader
  Learn how to start SQL*Loader, and how to specify parameters that manage how the load is run.
- Command-Line Parameters for SQL*Loader
  Manage SQL*Loader by using the command-line parameters.
- Exit Codes for Inspection and Display
  Oracle SQL*Loader provides the results of a SQL*Loader run immediately upon completion.

8.1 Starting SQL*Loader

Learn how to start SQL*Loader, and how to specify parameters that manage how the load is run.

To display a help screen that lists all SQL*Loader parameters, enter `sqlldr` at the prompt. and press **Enter**. The output shows each parameter, including default values for parameters, and a brief description of each parameter.

- Specifying Parameters on the Command Line
  When you start SQL*Loader, you specify parameters to establish various characteristics of the load operation.
- Alternative Ways to Specify SQL*Loader Parameters
  Learn how you can move some command-line parameters into the control file, or place commonly used parameters in a parameter file.
- Using SQL*Loader to Load Data Across a Network
  To use SQL*Loader to load data across a network connection, you can specify a connect identifier in the connect string when you start the SQL*Loader utility.

8.1.1 Specifying Parameters on the Command Line

When you start SQL*Loader, you specify parameters to establish various characteristics of the load operation.

To see how to specify SQL*Loader parameters, refer to the following examples:

You can separate the parameters by commas. However, it is not required to delimit parameters by commas:

```
> sqlldr CONTROL=ulcase1.ctl LOG=ulcase1.log
Username: scott
Password: password
```

Specifying by position means that you enter a value, but not the parameter name. In the following example, the username `scott` is provided, and then the name of the control file, `ulcase1.ctl`. You are prompted for the password:

```
> sqlldr scott ulcase1.ctl
Password: password
```

After a parameter name is used, you must supply parameter names for all subsequent specifications. No further positional specification is allowed. For example, in the following command, the `CONTROL` parameter is used to specify the control file name, but then the log file name is supplied without the `LOG` parameter, even though the LOG parameter was previously specified. Submitting this command now results in an error, even though the position of `ulcase1.log` is correct:

```
> sqlldr scott CONTROL=ulcase1.ctl ulcase1.log
```

For the command to run, you must enter the command with the `LOG` parameter specifically specified:

```
> sqlldr scott CONTROL=ulcase1.ctl LOG=ulcase1.log
```

8.1.2 Alternative Ways to Specify SQL*Loader Parameters

Learn how you can move some command-line parameters into the control file, or place commonly used parameters in a parameter file.

If the length of the command line exceeds the maximum line size for your system, then you can put certain command-line parameters in the control file by using the `OPTIONS` clause.

You can also group parameters together in a parameter file. You specify the name of this file on the command line using the `PARFILE` parameter when you start SQL*Loader.

These alternative ways of specifying parameters are useful when you often use the same parameters with the same values.

Parameter values specified on the command line override parameter values specified in either a parameter file or in the `OPTIONS` clause.

Related Topics

- OPTIONS Clause for Schema Data
  The following SQL*Loader command-line parameters can be specified using the `OPTIONS` clause.

- PARFILE
  The `PARFILE` SQL*Loader command-line parameter specifies the name of a file that contains commonly used command-line parameters.

8.1.3 Using SQL*Loader to Load Data Across a Network

To use SQL*Loader to load data across a network connection, you can specify a connect identifier in the connect string when you start the SQL*Loader utility.

This identifier can specify a database instance that is different from the current instance identified by the setting of the `ORACLE_SID` environment variable for the current user. The connect identifier can be an Oracle Net connect descriptor or a net service name (usually defined in the `tnsnames.ora` file) that maps to a connect descriptor. Use of a connect identifier requires that you have Oracle Net Listener running (to start the default listener, enter `lsnrctl start`). The following example starts SQL*Loader for user `scott` using the connect identifier `inst1`:

```
> sqlldr CONTROL=ulcase1.ctl
Username: scott@inst1
Password: password
```

The local SQL*Loader client connects to the database instance defined by the connect identifier `inst1` (a net service name), and loads the data, as specified in the `ulcase1.ctl` control file.

> **Note:**
>
> To load data into a pluggable database (PDB), simply specify its connect identifier on the connect string when you start SQL*Loader.

> **See Also:**
>
> - *Oracle Database Net Services Administrator's Guide* for more information about connect identifiers and Oracle Net Listener
> - *Oracle Database Concepts* for more information about PDBs

8.2 Command-Line Parameters for SQL*Loader

Manage SQL*Loader by using the command-line parameters.

The defaults and maximum values listed for these parameters are for Linux and Unix-based systems. They can be different on your operating system. Refer to your operating system documentation for more information.

- BAD
  The `BAD` command-line parameter for SQL*Loader specifies the name or location, or both, of the bad file associated with the first data file specification.

- BINDSIZE
  The BINDSIZE command-line parameter for SQL*Loader specifies the maximum size (in bytes) of the bind array.
- COLUMNARRAYROWS
  The COLUMNARRAYROWS command-line parameter for SQL*Loader specifies the number of rows to allocate for direct path column arrays.
- COMPRESS_STREAM
  The COMPRESS_STREAM SQL*Loader command-line parameter specifies Direct Path API stream data sent from the client to servers should be compressed.
- CONTROL
  The CONTROL command-line parameter for SQL*Loader specifies the name of the SQL*Loader control file that describes how to load the data.
- CREDENTIAL
  The CREDENTIAL command-line parameter for SQL*Loader enables reading data stored in object stores.
- DATA
  The DATA command-line parameter for SQL*Loader specifies the names of the data files containing the data that you want to load.
- DATE_CACHE
  The DATE_CACHE command-line parameter for SQL*Loader specifies the date cache size (in entries).
- DEFAULTS
  The DEFAULTS command-line parameter for SQL*Loader controls evaluation and loading of default expressions.
- DEGREE_OF_PARALLELISM
  The DEGREE_OF_PARALLELISM command-line parameter for SQL*Loader specifies the degree of parallelism to use during the load operation.
- DIRECT
  The DIRECT command-line parameter for SQL*Loader specifies the load method to use, either conventional path or direct path.
- DIRECT_PATH_LOCK_WAIT
  The DIRECT_PATH_LOCK_WAIT command-line parameter for SQL*Loader controls direct path load behavior when waiting for table locks.
- DISCARD
  The DISCARD command-line parameter for SQL*Loader lets you optionally specify a discard file to store records that are neither inserted into a table nor rejected.
- DISCARDMAX
  The DISCARDMAX command-line parameter for SQL*Loader specifies the number of discard records to allow before data loading is terminated.
- DNFS_ENABLE
  The DNFS_ENABLE SQL*Loader command-line parameter lets you enable and disable use of the Direct NFS Client on input data files during a SQL*Loader operation.
- DNFS_READBUFFERS
  The DNFS_READBUFFERS SQL*Loader command-line parameter lets you control the number of read buffers used by the Direct NFS Client.
- EMPTY_LOBS_ARE_NULL
  The EMPTY_LOBS_ARE_NULL SQL*Loader command-line parameter specifies that any LOB column for which there is no data available is set to NULL, rather than to an empty LOB.

- ERRORS
  The ERRORS SQL*Loader command line parameter specifies the maximum number of allowed insert errors.
- EXTERNAL_TABLE
  The `EXTERNAL_TABLE` parameter instructs SQL*Loader whether to load data using the external tables option.
- FILE
  The `FILE` SQL*Loader command-line parameter specifies the database file from which the extents are allocated.
- GRANULE_SIZE
  The `GRANULE_SIZE` SQL*Loader command-line parameter specifies a size for granules of data for automatic parallel loading.
- GSM_HOST
  The `GSM_HOST` SQL*Loader command-line parameter specifies the host on which the Global Service Manager is located, which is required for loading shards in parallel.
- GSM_NAME
  The `GSM_NAME` SQL*Loader command-line parameter specifies the Global Service Manager name, which is required for loading shards in parallel.
- GSM_PORT
  The `GSM_PORT` SQL*Loader command-line parameter specifies the listener port number for the Global Service Manager, which is required for loading shards in parallel.
- HELP
  The `HELP` SQL*Loader command-line parameter displays online help for the SQL*Loader utility.
- LOAD
  The `LOAD` SQL*Loader command-line parameter specifies the maximum number of records to load.
- LOAD_SHARDS
  The `LOAD_SHARDS` SQL*Loader command-line parameter specifies a specific list of shards to load from a sharded table.
- LOG
  The `LOG` SQL*Loader command-line parameter specifies a directory path, or file name, or both for the log file where SQL*Loader stores logging information about the loading process.
- MULTITHREADING
  The `MULTITHREADING` SQL*Loader command-line parameter enables stream building on the client system to be done in parallel with stream loading on the server system.
- NO_INDEX_ERRORS
  The `NO_INDEX_ERRORS` SQL*Loader command-line parameter specifies whether indexing errors are tolerated during a direct path load.
- OPTIMIZE PARALLEL
  The SQL*Loader `OPTIMIZE_PARALLEL` parameter specifies whether automatic parallel loads should enable SQL*Loader to choose the optimal parallel loading option.
- PARALLEL
  The SQL*Loader `PARALLEL` parameter specifies whether loads that use direct path can operate in multiple concurrent sessions to load data into the same table.

- PARFILE
  The PARFILE SQL*Loader command-line parameter specifies the name of a file that contains commonly used command-line parameters.
- PARTITION_MEMORY
  The PARFILE SQL*Loader command-line parameter specifies the amount of memory that you want to have used when you are loading many partitions.
- READER_COUNT
  The READER_COUNT SQL*Loader command-line parameter specifies the number of input data file reader threads for automatic parallel loads.
- READSIZE
  The READSIZE SQL*Loader command-line parameter specifies (in bytes) the size of the read buffer, if you choose not to use the default.
- RESUMABLE
  The RESUMABLE SQL*Loader command-line parameter enables and disables resumable space allocation.
- RESUMABLE_NAME
  The RESUMABLE_NAME SQL*Loader command-line parameter identifies a statement that is resumable.
- RESUMABLE_TIMEOUT
  The RESUMABLE_TIMEOUT SQL*Loader command-line parameter specifies the time period, in seconds, during which an error must be fixed.
- ROWS
  For conventional path loads, the ROWS SQL*Loader command-line parameter specifies the number of rows in the bind array, and in direct path loads, the number of rows to read from data files before a save.
- SDF_PREFIX
  The SDF_PREFIX SQL*Loader command-line parameter specifies a directory prefix, which is added to file names of LOBFILEs and secondary data files (SDFs) that are opened as part of a load operation.
- SILENT
  The SILENT SQL*Loader command-line parameter suppresses some of the content that is written to the screen during a SQL*Loader operation.
- SKIP
  The SKIP SQL*Loader command-line parameter specifies the number of logical records from the beginning of the file that should not be loaded.
- SKIP_INDEX_MAINTENANCE
  The SKIP_INDEX_MAINTENANCE SQL*Loader command-line parameter specifies whether to stop index maintenance for direct path loads.
- SKIP_UNUSABLE_INDEXES
  The SKIP_UNUSABLE_INDEXES SQL*Loader command-line parameter specifies whether to skip an index encountered in an Index Unusable state and continue the load operation.
- STREAMSIZE
  The STREAMSIZE SQL*Loader command-line parameter specifies the size (in bytes) of the data stream sent from the client to the server.
- TRIM
  The TRIM SQL*Loader command-line parameter specifies whether you want spaces trimmed from the beginning of a text field, the end of a text field, both, or neither.

- USERID
  The `USERID` SQL*Loader command-line parameter provides your Oracle username and password for SQL*Loader.

8.2.1 BAD

The `BAD` command-line parameter for SQL*Loader specifies the name or location, or both, of the bad file associated with the first data file specification.

Default

The name of the data file, with an extension of `.bad`.

Purpose

Specifies the name or location, or both, of the bad file associated with the first data file specification.

Syntax and Description

```
BAD=[directory/][filename]
```

The bad file stores records that cause errors during insert, or that are improperly formatted. If you specify the `BAD` parameter, then you must supply either a directory, or file name, or both. If there are rejected records, and you have not specified a name for the bad file, then the name defaults to the name of the data file with an extension or file type of `.bad`.

The value you provide for *`directory`* specifies the directory where you want the bad file to be written. The specification can include the name of a device or network node. The value of *`directory`* is determined as follows:

- If the `BAD` parameter is not specified at all, and a bad file is needed, then the default directory is the one in which the SQL*Loader control file resides.
- If the `BAD` parameter is specified with a file name, but without a directory, then the directory defaults to the current directory.
- If the `BAD` parameter is specified with a directory, but without a file name, then the specified directory is used, and the name defaults to the name of the data file, with an extension or file type of `.bad`.

The value you provide for *`filename`* specifies a file name that is recognized as valid on your platform. You must specify only a name (and extension, if you want to use one other than `.bad`). Any spaces or punctuation marks in the file name must be enclosed within single quotation marks.

A bad file specified on the command line becomes the bad file associated with the first `INFILE` statement (if there is one) in the control file. You can also specify the of the bad file in the SQL*Loader control file by using the `BADFILE` clause. If the bad file is specified in both the control file and by command line, then the command-line value is used. If a bad file with that name already exists, then it is either overwritten, or a new version is created, depending on your operating system.

Example

The following specification creates a bad file named `emp1.bad` in the current directory:

```
BAD=emp1
```

Related Topics

- Understanding and Specifying the Bad File
  When SQL*Loader executes, it can create a file called a *bad* file, or reject file, in which it places records that were rejected because of formatting errors or because they caused Oracle errors.

8.2.2 BINDSIZE

The `BINDSIZE` command-line parameter for SQL*Loader specifies the maximum size (in bytes) of the bind array.

Default

```
256000
```

Purpose

Specifies the maximum size (in bytes) of the bind array.

Syntax and Description

```
BINDSIZE=n
```

A **bind array** is an area in memory where SQL*Loader stores data that is to be loaded. When the bind array is full, the data is transmitted to the database. The bind array size is controlled by the parameters `BINDSIZE` and `READSIZE`.

The size of the bind array given by `BINDSIZE` overrides the default size (which is system dependent) and any size determined by `ROWS`.

Restrictions

- The `BINDSIZE` parameter is used only for conventional path loads.

Example

The following `BINDSIZE` specification limits the maximum size of the bind array to 356,000 bytes.

```
BINDSIZE=356000
```

Related Topics

- Differences Between Bind Arrays and Conventional Path Loads
  With bind arrays, you can use SQL*Loader to load an entire array of records in one operation.

- READSIZE
  The `READSIZE` SQL*Loader command-line parameter specifies (in bytes) the size of the read buffer, if you choose not to use the default.
- ROWS
  For conventional path loads, the `ROWS` SQL*Loader command-line parameter specifies the number of rows in the bind array, and in direct path loads, the number of rows to read from data files before a save.

8.2.3 COLUMNARRAYROWS

The `COLUMNARRAYROWS` command-line parameter for SQL*Loader specifies the number of rows to allocate for direct path column arrays.

Default

`5000`

Purpose

Specifies the number of rows that you want to allocate for direct path column arrays.

Syntax and Description

```
COLUMNARRARYROWS=n
```

The value for this parameter is not calculated by SQL*Loader. You must either specify it or accept the default.

Example

The following example specifies that you want to allocate 1000 rows for direct path column arrays.

```
COLUMNARRAYROWS=1000
```

Related Topics

- Using CONCATENATE to Assemble Logical Records
  Use `CONCATENATE` when you want SQL*Loader to always combine the same number of physical records to form one logical record.
- Specifying the Number of Column Array Rows and Size of Stream Buffers
  The number of column array rows determines the number of rows loaded before the stream buffer is built.

8.2.4 COMPRESS_STREAM

The `COMPRESS_STREAM` SQL*Loader command-line parameter specifies Direct Path API stream data sent from the client to servers should be compressed.

Default

`FALSE`

Syntax and Description

```
COMPRESS_STREAM=[TRUE|FALSE]
```

The `COMPRESS_STREAM` parameter is used with automatic parallel loads, starting with Oracle Database 23ai. It enables you to specify that you want Direct Path API stream data to be compressed when it is sent from the client to servers. Setting this parameter to `TRUE` can improve performance when loading distant servers.

If you are loading files remotely from a client to a server, you can use this parameter to see if load performance is improved. If you use this parameter, then it can override the value you specify with the `STREAMSIZE` parameter.

Restrictions

- This parameter can only be used in direct path loading.
- Setting `MULTITHREADING=TRUE` disables this option. To obtain the potential performance benefits from `COMPRESS_STREAM`, ensure that multithreading is set to `FALSE`.

Example

The following example specifies to compress Direct Path API stream data:

```
COMPRESS_STREAM=TRUE
```

8.2.5 CONTROL

The `CONTROL` command-line parameter for SQL*Loader specifies the name of the SQL*Loader control file that describes how to load the data.

Default

There is no default.

Purpose

Specifies the name of the SQL*Loader control file that describes how to load the data.

Syntax and Description

```
CONTROL=control_file_name
```

If you do not specify a file extension or file type, then it defaults to `.ctl`. If the `CONTROL` parameter is not specified, then SQL*Loader prompts you for it.

If the name of your SQL*Loader control file contains special characters, then your operating system can require that you enter the control file name preceded by an escape character. Also, if your operating system uses backslashes in its file system paths, then you can be required to use multiple escape characters, or you can be required to enclose the path in quotation marks. Refer to your operating system documentation for more information about how to use special characters.

Example

The following example specifies a control file named `emp1`. It is automatically given the default extension of `.ctl`.

```
CONTROL=emp1
```

Related Topics

- SQL*Loader Control File Reference
  The SQL*Loader control file is a text file that contains data definition language (DDL) instructions for a SQL*Loader job.

8.2.6 CREDENTIAL

The `CREDENTIAL` command-line parameter for SQL*Loader enables reading data stored in object stores.

Default

none.

Purpose

Enables SQL*Loader to read object stores. For a data file, you can specify the URI for the data file that you want to read on the object store. The `CREDENTIAL` values specify credentials granted to the user running SQL*Loader. These permissions enable SQL*Loader to access the object store.

Syntax and Description

In the following syntax, the variable *`user-credential`* is the user credential (user name or password) that you specify SQL*Loader to use:

```
oracle.sqlldr.credential.user-credential.username
oracle.sqlldr.credential.user-credential.password
```

Usage Notes

If you specify the `CREDENTIAL` parameter, then SQL*Loader uses the values you provide for the keys as the username and password for the object store. Before you use CREDENTIAL, you must previously have created a valid credential by using `orapki`, or using the `mkstore` command.

> **Note:**
>
> The `mkstore` wallet management command line tool is deprecated with Oracle Database 23ai, and can be removed in a future release.
>
> To manage wallets, Oracle recommends that you use the `orapki` command line tool.

Restrictions

If you specify `CREDENTIAL`, and one of the following conditions are true, then you receive an error:

- One or both keys cannot be found in the Oracle Wallet
- The files specified for the `DATA` parameter are not a URI.
- The files specified for the `INFILE` clause in the control file are not URIs.

If a URI is specified for a data file, and the `CREDENTIAL` parameter is not specified, then you receive an error.

Example

To use the `CREDENTAL` parameter with SQL*Loader, you create a wallet, and define an access credential for the wallet for the target where you want to load data. Then you identify that credential with a user for whom you want to grant permissions to load data. After that task is complete, you can use the wallet credential to load data into the target database.

For example:

1. Where your wallet path is `/u01/app/oracle/product/wallets`, and the password is `cloud-pw-example` use the `orapki` utility to create a wallet:

```
% orapki wallet create -wallet /u01/app/oracle/product/wallets  -pwd cloud-
pw-example -auto_login
Oracle PKI Tool Release 20.0.0.0.0 - Production

Version 21.0.0.0.0

Copyright (c) 2004, 2019, Oracle and/or its affiliates. All rights
reserved.

Operation is successfully completed.
```

> **Note:**
>
> For an actual password, always ensure that you follow industry-standard practices for secure passwords.

2. Create the SQL*Loader credential `"obm_scott"` user. To do this, use the `mkstore` utility to define the database connection string (`oracle.sqlldr.credential.obm_scott` that can be used with the user ID `some_user`, with the password `some_password`:

```
% mkstore -wrl /u01/app/oracle/product/wallets -createEntry \
oracle.sqlldr.credential.obm_scott.username some_username

% mkstore -wrl wallet_location_directory -createEntry
oracle.sqlldr.credential.obm_scott.password \
some_password
```

> **Note:**
>
> For each credential, there can be only one user and password pair.

For both the `mkstore` commands, you are prompted to provide the password for the externally stored `obm_scott` credential, which in this example is `cloud-pw-example`.

3. Finally, you use SQL*Loader to load the data into the database, using the credential that you have created. For example:

```
% sqlldr sqlldr/cdb1_pdb6 dept.ctl credential=obm_scott log=dept.log  \
external_table=not_used proxy=https://www.example.com:80
```

You then load data, which in this example is `dept.csv`:

```
LOAD DATA
INFILE  'https://publickeyinfrastorage.example.com/v1/pkistore/dept.csv'
truncate
INTO TABLE DEPTOS
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(DEPTNO, DNAME, LOC)
```

8.2.7 DATA

The `DATA` command-line parameter for SQL*Loader specifies the names of the data files containing the data that you want to load.

Default

The same name as the control file, but with an extension of `.dat`.

Purpose

The `DATA` parameter specifies the name of the data file containing the data that you want to load.

Syntax and Description

```
DATA=data_file_name
```

If you do not specify a file extension, then the default is `.dat`.

The file specification can contain wildcards (only in the file name and file extension, not in a device or directory name). An asterisk (*) represents multiple characters and a question mark (?) represents a single character. For example:

```
DATA='emp*.dat'
```

```
DATA='m?emp.dat'
```

To list multiple data file specifications (each of which can contain wild cards), the file names must be separated by commas.

If the file name contains any special characters (for example, spaces, *, ?,), then the entire name must be enclosed within single quotation marks.

The following are three examples of possible valid uses of the `DATA` parameter (the single quotation marks would only be necessary if the file name contained special characters):

```
DATA='file1','file2','file3','file4','file5','file6'
DATA='file1','file2'
DATA='file3,'file4','file5'
DATA='file6'
```

> **Caution:**
>
> If multiple data files are being loaded and you are also specifying the `BAD` parameter, it is recommended that you specify only a directory for the bad file, not a file name. If you specify a file name, and a file with that name already exists, then it is either overwritten or a new version is created, depending on your operating system.

If you specify data files on the command line with the `DATA` parameter and also specify data files in the control file with the `INFILE` clause, then the first `INFILE` specification in the control file is ignored. All other data files specified on the command line and in the control file are processed.

If you specify a file processing option along with the `DATA` parameter when loading data from the control file, then a warning message is issued.

Example

The following example specifies that a data file named `employees.dat` is to be loaded. The `.dat` extension is assumed as the default because no extension is provided.

```
DATA=employees
```

8.2.8 DATE_CACHE

The `DATE_CACHE` command-line parameter for SQL*Loader specifies the date cache size (in entries).

Default

Enabled (for `1000` elements). To completely disable the date cache feature, set it to `0` (zero).

Purpose

Specifies the date cache size (in entries).

The date cache is used to store the results of conversions from text strings to internal date format. The cache is useful, because the cost of looking up dates is much less than converting from text format to date format. If the same dates occur repeatedly in the date file, then using the date cache can improve the speed of a direct path load.

Syntax and Description

```
DATE_CACHE=n
```

Every table has its own date cache, if one is needed. A date cache is created only if at least one date or timestamp value is loaded that requires data type conversion before it can be stored in the table.

The date cache feature is enabled by default. The default date cache size is 1000 elements. If the default size is used, and if the number of unique input values loaded exceeds 1000, then the date cache feature is automatically disabled for that table. However, if you override the default, and you specify a nonzero date cache size, and that size is exceeded, then the cache is not disabled.

To tune the size of the cache for future similar loads, use the date cache statistics (entries, hits, and misses) contained in the log file.

Restrictions

- The date cache feature is only available for direct path and external tables loads.

Example

The following specification completely disables the date cache feature.

```
DATE_CACHE=0
```

Related Topics

- Specifying a Value for DATE_CACHE
  To improve performance where the same date or timestamp is used many times during a direct path load, you can use the SQL*Loader date cache.

8.2.9 DEFAULTS

The `DEFAULTS` command-line parameter for SQL*Loader controls evaluation and loading of default expressions.

Default

`EVALUATE_ONCE`, unless a sequence is involved. If a sequence is involved, then the default is `EVALUATE_EVERY_ROW`.

Purpose

Controls evaluation and loading of default expressions.

The `DEFAULTS` parameter is only applicable to direct path loads.

Syntax and Description

```
DEFAULTS={IGNORE | IGNORE_UNSUPPORTED_EVALUATE_ONCE |
IGNORE_UNSUPPORTED_EVALUATE_EVERY_ROW |
EVALUATE_ONCE | EVALUATE_EVERY_ROW}
```

The behavior of each of the options is as follows:

- `IGNORE`: Default clauses on columns are ignored.
- `IGNORE_UNSUPPORTED_EVALUATE_ONCE`: Evaluate default expressions once at the start of the load. Unsupported default expressions are ignored. If the `DEFAULTS` parameter is not used, then default expressions are evaluated once, unless the default expression references a sequence, in which case every row is evaluated.
- `IGNORE_UNSUPPORTED_EVALUATE_EVERY_ROW`: Evaluate default expressions in every row, ignoring unsupported default clauses.
- `EVALUATE_ONCE`: Evaluate default expressions once at the start of the load. If the `DEFAULTS` parameter is not used, then default expressions are evaluated once, unless the default references a sequence, in which case every row is evaluated. An error is issued for unsupported default expression clauses. (This is the default option for this parameter.)
- `EVALUATE_EVERY_ROW`: Evaluate default expressions in every row, and issue an error for unsupported defaults.

Example

This example shows that a table is created with the name `test`, and a SQL*Loader control file named `test.ctl`:

```
create table test
(
   c0 varchar2(10),
   c1 number default '100'
)
;

test.ctl:

load data
infile *
truncate
into table test
fields terminated by ','
trailing nullcols
(
  c0 char
)
begindata
1,
```

To then load a NULL into `c1`, issue the following statement:

```
sqlldr scott/password t.ctl direct=true defaults=ignore
```

To load the default value of 100 into `c1`, issue the following statement:

```
sqlldr scott/password t.ctl direct=true
```

8.2.10 DEGREE_OF_PARALLELISM

The DEGREE_OF_PARALLELISM command-line parameter for SQL*Loader specifies the degree of parallelism to use during the load operation.

Default

NONE

Purpose

The DEGREE_OF_PARALLELISM parameter specifies the degree of parallelism to use during the load operation.

Syntax and Description

DEGREE_OF_PARALLELISM=[*degree-num*|DEFAULT|AUTO|NONE]

If a *degree-num* is specified, then it must be a whole number value from 1 to *n*.

If DEFAULT is specified, then the default parallelism *of the database* (not the default parameter value of AUTO) is used.

If AUTO is used, then Oracle Database automatically sets the degree of parallelism for the load.

If NONE is specified, then the load is not performed in parallel.

> **Note:**
>
> If AUTO or DEFAULT are used for conventional and direct path loads, then this results in no parallelism.

To optimize parallel reading and loading, Oracle recommends that you start by setting the parameters DEGREE_OF_PARALLELISM and READER_COUNT to a small value (for example, 4) and increase by a small amount to see if performance improves. The best value will depend on the client and server configuration. Too large a value can result in reduced performance. You should see a larger performance improvement when more work is required on the server (for example, if compression is being used).

For shard loading, Oracle recommends that you let SQL*Loader set DEGREE_OF_PARALLELISM. By default, that value by default is equal to the number of shards. If you have a large number of shards resulting in too many threads for the client to handle, then you can reduce the DEGREE_OF_PARALLELISM, resulting in multiple passes over the data.

Restrictions

- Automatic parallel loading is supported for a single table only. Multiple INTO clauses are not supported.
- Non-shard parallel loading of many partitions, especially with only a few rows per partition, may not perform well. The DEGREE_OF_PARALLELISM parameter should not be used for this case.

Example

The following example sets the degree of parallelism for the load to 4.

```
DEGREE_OF_PARALLELISM=4
```

Related Topics

- Parallel Execution Concepts in *Oracle Database VLDB and Partitioning Guide*

8.2.11 DIRECT

The `DIRECT` command-line parameter for SQL*Loader specifies the load method to use, either conventional path or direct path.

Default

```
FALSE
```

Purpose

The `DIRECT` parameter specifies the load method to use, either conventional path or direct path.

Syntax and Description

```
DIRECT=[TRUE | FALSE]
```

A value of `TRUE` specifies a direct path load. A value of `FALSE` specifies a conventional path load.

> **See Also:**
>
> Conventional and Direct Path Loads

Example

The following example specifies that the load be performed using conventional path mode.

```
DIRECT=FALSE
```

8.2.12 DIRECT_PATH_LOCK_WAIT

The DIRECT_PATH_LOCK_WAIT command-line parameter for SQL*Loader controls direct path load behavior when waiting for table locks.

Default

```
FALSE
```

Purpose

Controls direct path load behavior when waiting for table locks. Direct path loads must lock the table before the load can proceed. The `DIRECT_PATH_LOCK_WAIT` command controls the direct path API behavior while waiting for a lock.

Syntax and Description

```
DIRECT_PATH_LOCK_WAIT = {TRUE | FALSE}
```

- `TRUE`: Direct path waits until it can get a lock on the table before proceeding with the load.
- `FALSE`: (Default). When set to `FALSE`, the direct path API tries to lock the table multiple times and waits one second between attempts. The maximum number of attempts made is 30. If the table cannot be locked after 30 attempts, then the direct path API returns the error that was generated when trying to lock the table.

8.2.13 DISCARD

The `DISCARD` command-line parameter for SQL*Loader lets you optionally specify a discard file to store records that are neither inserted into a table nor rejected.

Default

The same file name as the data file, but with an extension of `.dsc`.

Purpose

The `DISCARD` parameter lets you optionally specify a discard file to store records that are neither inserted into a table nor rejected. They are not bad records, they simply did not match any record-selection criteria specified in the control file, such as a `WHEN` clause for example.

Syntax and Description

```
DISCARD=[directory/][filename]
```

If you specify the `DISCARD` parameter, then you must supply either a directory or file name, or both.

The *`directory`* parameter specifies a directory to which the discard file will be written. The specification can include the name of a device or network node. The value of directory is determined as follows:

- If the `DISCARD` parameter is not specified at all, but the `DISCARDMAX` parameter is, then the default directory is the one in which the SQL*Loader control file resides.
- If the `DISCARD` parameter is specified with a file name but no directory, then the directory defaults to the current directory.
- If the `DISCARD` parameter is specified with a directory but no file name, then the specified directory is used and the default is used for the name and the extension.

The *`filename`* parameter specifies a file name recognized as valid on your platform. You must specify only a name (and extension, if one other than .dsc is desired). Any spaces or punctuation marks in the file name must be enclosed in single quotation marks.

If neither the DISCARD parameter nor the DISCARDMAX parameter is specified, then a discard file is not created even if there are discarded records.

If the DISCARD parameter is not specified, but the DISCARDMAX parameter is, and there are discarded records, then the discard file is created using the default name and the file is written to the same directory in which the SQL*Loader control file resides.

> **Caution:**
>
> If multiple data files are being loaded and you are also specifying the DISCARD parameter, it is recommended that you specify only a directory for the discard file, not a file name. If you specify a file name, and a file with that name already exists, then it is either overwritten or a new version is created, depending on your operating system.

A discard file specified on the command line becomes the discard file associated with the first INFILE statement (if there is one) in the control file. If the discard file is also specified in the control file, then the command-line value overrides it. If a discard file with that name already exists, then it is either overwritten or a new version is created, depending on your operating system.

> **See Also:**
>
> Discarded and Rejected Records for information about the format of discard files

Example

Assume that you are loading a data file named employees.dat. The following example supplies only a directory name so the name of the discard file will be employees.dsc and it will be created in the mydir directory.

```
DISCARD=mydir/
```

8.2.14 DISCARDMAX

The DISCARDMAX command-line parameter for SQL*Loader specifies the number of discard records to allow before data loading is terminated.

Default

```
ALL
```

Purpose

The DISCARDMAX parameter specifies the number of discard records to allow before data loading is terminated.

Syntax and Description

```
DISCARDMAX=n
```

To stop on the first discarded record, specify a value of 0.

If `DISCARDMAX` is specified, but the `DISCARD` parameter is not, then the name of the discard file is the name of the data file with an extension of `.dsc`.

Example

The following example allows 25 records to be discarded during the load before it is terminated.

```
DISCARDMAX=25
```

8.2.15 DNFS_ENABLE

The `DNFS_ENABLE` SQL*Loader command-line parameter lets you enable and disable use of the Direct NFS Client on input data files during a SQL*Loader operation.

Default

```
TRUE
```

Purpose

The `DNFS_ENABLE` parameter lets you enable and disable use of the Direct NFS Client on input data files during a SQL*Loader operation.

Syntax and Description

```
DNFS_ENABLE=[TRUE|FALSE]
```

The Direct NFS Client is an API that can be implemented by file servers to allow improved performance when an Oracle database accesses files on those servers.

SQL*Loader uses the Direct NFS Client interfaces by default when it reads data files over 1 GB. For smaller files, the operating system input/output (I/O) interfaces are used. To use the Direct NFS Client on *all* input data files, use `DNFS_ENABLE=TRUE`.

To disable use of the Direct NFS Client for all data files, specify `DNFS_ENABLE=FALSE`.

The `DNFS_READBUFFERS` parameter can be used to specify the number of read buffers used by the Direct NFS Client; the default is 4.

> **See Also:**
>
> - Oracle Grid Infrastructure Installation Guide for your platform for more information about enabling the Direct NFS Client

Example

The following example disables use of the Direct NFS Client on input data files during the load.

```
DNFS_ENABLE=FALSE
```

8.2.16 DNFS_READBUFFERS

The `DNFS_READBUFFERS` SQL*Loader command-line parameter lets you control the number of read buffers used by the Direct NFS Client.

Default

4

Purpose

The `DNFS_READBUFFERS` parameter lets you control the number of read buffers used by the Direct NFS Client. The Direct NFS Client is an API that can be implemented by file servers to allow improved performance when an Oracle database accesses files on those servers.

Syntax and Description

```
DNFS_READBUFFERS=n
```

The value for *n* is the number of read buffers you specify. It is possible that you can compensate for inconsistent input/output (I/O) from the Direct NFS Client file server by increasing the number of read buffers. However, using larger values can result in increased memory usage.

Restrictions

- To use this parameter without also specifying the `DNFS_ENABLE` parameter, the input file must be larger than 1 GB.

Example

The following example specifies 10 read buffers for use by the Direct NFS Client.

```
DNFS_READBUFFERS=10
```

Related Topics

- *Oracle Grid Infrastructure Installation Guide* for your platform

8.2.17 EMPTY_LOBS_ARE_NULL

The `EMPTY_LOBS_ARE_NULL` SQL*Loader command-line parameter specifies that any LOB column for which there is no data available is set to NULL, rather than to an empty LOB.

Default

```
FALSE
```

Purpose

If the SQL*Loader `EMPTY_LOBS_ARE_NULL` parameter is specified, then any Large Object (LOB) columns for which there is no data available are set to NULL, rather than to an empty LOB. Setting LOB columns for which there is no data available to NULL negates the need to make that change through post-processing after the data is loaded.

Syntax and Description

```
EMPTY_LOBS_ARE_NULL = {TRUE | FALSE}
```

You can specify the `EMPTY_LOBS_ARE_NULL` parameter on the SQL*Loader command line, and also on the `OPTIONS` clause in a SQL*Loader control file.

Restrictions

None.

Example

In the following example, as a result of setting `empty_lobs_are_null=true`, the LOB columns in `c1` are set to `NULL` instead of to an empty LOB.

```
create table t
(
   c0 varchar2(10),
   c1 clob
)
;

sqlldr control file:

options (empty_lobs_are_null=true)
load data
infile *
truncate
into table t
fields terminated by ','
trailing nullcols
(
  c0 char,
  c1 char
)
begindata
1,,
```

8.2.18 ERRORS

The ERRORS SQL*Loader command line parameter specifies the maximum number of allowed insert errors.

Default

50

Purpose

The `ERRORS` parameter specifies the maximum number of insert errors to allow.

Syntax and Description

```
ERRORS=n
```

If the number of errors exceeds the value specified for `ERRORS`, then SQL*Loader terminates the load. Any data inserted up to that point is committed.

To permit no errors at all, set `ERRORS=0`. To specify that all errors be allowed, use a very high number.

SQL*Loader maintains the consistency of records across all tables. Therefore, multitable loads do not terminate immediately if errors exceed the error limit. When SQL*Loader encounters the maximum number of errors for a multitable load, it continues to load rows to ensure that valid rows previously loaded into tables are loaded into all tables and rejected rows are filtered out of all tables.

In all cases, SQL*Loader writes erroneous records to the bad file.

Example

The following example specifies a maximum of 25 insert errors for the load. After that, the load is terminated.

```
ERRORS=25
```

8.2.19 EXTERNAL_TABLE

The `EXTERNAL_TABLE` parameter instructs SQL*Loader whether to load data using the external tables option.

Default

```
NOT_USED
```

Syntax and Description

```
EXTERNAL_TABLE=[NOT_USED | GENERATE_ONLY | EXECUTE]
```

The possible values are as follows:

- `NOT_USED` - the default value. It means the load is performed using either conventional or direct path mode.
- `GENERATE_ONLY` - places all the SQL statements needed to do the load using external tables, as described in the control file, in the SQL*Loader log file. These SQL statements can be edited and customized. The actual load can be done later without the use of SQL*Loader by executing these statements in SQL*Plus.
- `EXECUTE` - attempts to execute the SQL statements that are needed to do the load using external tables. However, if any of the SQL statements returns an error, then the attempt to load stops. Statements are placed in the log file as they are executed. This means that if a SQL statement returns an error, then the remaining SQL statements required for the load will not be placed in the log file.

  If you use `EXTERNAL_TABLE=EXECUTE` and also use the `SEQUENCE` parameter in your SQL*Loader control file, then SQL*Loader creates a database sequence, loads the table

using that sequence, and then deletes the sequence. The results of doing the load this way will be different than if the load were done with conventional or direct path. (For more information about creating sequences, see `CREATE SEQUENCE` in *Oracle Database SQL Language Reference.)*

> **Note:**
>
> When the `EXTERNAL_TABLE` parameter is specified, any datetime data types (for example, `TIMESTAMP`) in a SQL*Loader control file are automatically converted to a `CHAR` data type and use the external tables `date_format_spec` clause. See date_format_spec.

Note that the external table option uses directory objects in the database to indicate where all input data files are stored and to indicate where output files, such as bad files and discard files, are created. You must have `READ` access to the directory objects containing the data files, and you must have `WRITE` access to the directory objects where the output files are created. If there are no existing directory objects for the location of a data file or output file, then SQL*Loader will generate the SQL statement to create one. Therefore, when the `EXECUTE` option is specified, you must have the `CREATE ANY DIRECTORY` privilege. If you want the directory object to be deleted at the end of the load, then you must also have the `DROP ANY DIRECCTORY` privilege.

> **Note:**
>
> The `EXTERNAL_TABLE=EXECUTE` qualifier tells SQL*Loader to create an external table that can be used to load data and then executes the `INSERT` statement to load the data. All files in the external table must be identified as being in a directory object. SQL*Loader attempts to use directory objects that already exist and that you have privileges to access. However, if SQL*Loader does not find the matching directory object, then it attempts to create a temporary directory object. If you do not have privileges to create new directory objects, then the operation fails.
>
> To work around this, use `EXTERNAL_TABLE=GENERATE_ONLY` to create the SQL statements that SQL*Loader would try to execute. Extract those SQL statements and change references to directory objects to be the directory object that you have privileges to access. Then, execute those SQL statements.

When using a multi-table load, SQL*Loader does the following:

1. Creates a table in the database that describes all fields in the input data file that will be loaded into any table.
2. Creates an `INSERT` statement to load this table from an external table description of the data.
3. Executes one `INSERT` statement for every table in the control file.

To see an example of this, run case study 5, but add the `EXTERNAL_TABLE=GENERATE_ONLY` parameter. To guarantee unique names in the external table, SQL*Loader uses generated names for all fields. This is because the field names may not be unique across the different tables in the control file.

> **See Also:**
>
> - "SQL*Loader Case Studies" for information on how to access case studies
> - External Tables Concepts
> - The ORACLE_LOADER Access Driver

Restrictions

- Julian dates cannot be used when you insert data into a database table from an external table through SQL*Loader. To work around this, use `TO_DATE` and `TO_CHAR` to convert the Julian date format, as shown in the following example:

  ```
  TO_CHAR(TO_DATE(:COL1, 'MM-DD-YYYY'), 'J')
  ```

- Built-in functions and SQL strings cannot be used for object elements when you insert data into a database table from an external table.

Example

```
EXTERNAL_TABLE=EXECUTE
```

8.2.20 FILE

The `FILE` SQL*Loader command-line parameter specifies the database file from which the extents are allocated.

Default

There is no default.

Purpose

The `FILE` parameter specifies the database file from which the extents are allocated.

> **See Also:**
>
> Parallel Data Loading Models

Syntax and Description

```
FILE=tablespace_file
```

By varying the value of the `FILE` parameter for different SQL*Loader processes, data can be loaded onto a system with minimal disk contention.

Restrictions

- The `FILE` parameter is used only for direct path parallel loads.

8.2.21 GRANULE_SIZE

The `GRANULE_SIZE` SQL*Loader command-line parameter specifies a size for granules of data for automatic parallel loading.

Default

If you do not specify a granule size, then SQL*Loader calculates the optimal default granule size for each file, depending on the number of readers, and their size.

Syntax and Description

```
GRANULE_SIZE=n
```

The `GRANULE_SIZE` parameter is used with automatic parallel loads, starting with Oracle Database 23ai. It enables you to specify the maximum size, in bytes, of data granules. For data file formats that can support being divided into multiple granules of data, such as `csv` files, SQL*Loader divides data files for parallel reading and loading using an optimal granule size for the file. Oracle recommends that you accept this default. However, you can specify a specific granule size to see if that improves load performance.

> **Note:**
>
> The granule size should be greater than or equal to the `READSIZE` parameter.

Restrictions

The `GRANULE_SIZE` parameter is ignored when a file cannot be split into granules.

Example

The following example specifies a granule size of 16000000 bytes:

```
GRANULE_SIZE=16000000
```

8.2.22 GSM_HOST

The `GSM_HOST` SQL*Loader command-line parameter specifies the host on which the Global Service Manager is located, which is required for loading shards in parallel.

Default

There is no default.

Purpose

The `GSM_HOST` parameter specifies the host on which the Global Service Manager is located. This hostname is required for loading shards in parallel. Global Data Service clients use the Global Service Manager to perform all GDS configuration and client connection operations to sharded tables.

> **See Also:**
>
> Parallel Data Loading Models

Syntax and Description

```
GSM_HOST=name-of-host
```

Example

The host on which the Global Service Manager resides, `myhost1`, is specified in this SQL*Loader command line by the `GSM_HOST` parameter:

```
sqlldr scott/tiger t.ctl gsm_name=shdsrv.shpool.oradbcloud gsm_host=myhost1
gsm_port=4338
```

8.2.23 GSM_NAME

The `GSM_NAME` SQL*Loader command-line parameter specifies the Global Service Manager name, which is required for loading shards in parallel.

Default

There is no default.

Purpose

The `GSM_NAME` parameter specifies the Global Service Manager name, which is required for loading shards in parallel. Global Data Service clients use the Global Service Manager to perform all GDS configuration and client connection operations to sharded tables.

> **See Also:**
>
> Parallel Data Loading Models

Syntax and Description

```
GSM_NAME=name-of-gsm-manager
```

Example

The Global Service Manager name `shdsrv.shpool.oradbcloud` is specified in this SQL*Loader command line by the `GSM_HOST` parameter:

```
sqlldr scott/tiger t.ctl gsm_name=shdsrv.shpool.oradbcloud gsm_host=myhost1
gsm_port=4338
```

8.2.24 GSM_PORT

The `GSM_PORT` SQL*Loader command-line parameter specifies the listener port number for the Global Service Manager, which is required for loading shards in parallel.

Default

There is no default.

Purpose

The `GSM_PORT` parameter specifies the Global Service Manager Listener port, which is required for loading shards in parallel. Global Data Service clients use the Global Service Manager to perform all GDS configuration and client connection operations to sharded tables.

> **See Also:**
>
> Parallel Data Loading Models

Syntax and Description

```
GSM_PORT=gsm-manager-port-number
```

Example

The Global Service Manager Listener port, `4338`, is specified in this SQL*Loader command line by the `GSM_PORT` parameter:

```
sqlldr scott/tiger t.ctl gsm_name=shdsrv.shpool.oradbcloud gsm_host=myhost1
gsm_port=4338
```

8.2.25 HELP

The `HELP` SQL*Loader command-line parameter displays online help for the SQL*Loader utility.

Default

```
FALSE
```

Syntax and Description

```
HELP = [TRUE | FALSE]
```

If `HELP=TRUE` is specified, then SQL*Loader displays a summary of all SQL*Loader command-line parameters.

You can also display a summary of all SQL*Loader command-line parameters by entering `sqlldr -help` on the command line.

8.2.26 LOAD

The `LOAD` SQL*Loader command-line parameter specifies the maximum number of records to load.

Default

All records are loaded.

Purpose

Specifies the maximum number of records to load.

Syntax and Description

```
LOAD=n
```

To test that all parameters you have specified for the load are set correctly, use the `LOAD` parameter to specify a limited number of records rather than loading all records. No error occurs if fewer than the maximum number of records are found.

Example

The following example specifies that a maximum of 10 records be loaded.

```
LOAD=10
```

For external tables method loads, only successfully loaded records are counted toward the total. So if there are 15 records in the input data file and records 2 and 4 are bad, then the following records are loaded into the table, for a total of 10 records: 1, 3, 5, 6, 7, 8, 9, 10, 11, and 12.

For conventional and direct path loads, both successful and unsuccessful load attempts are counted toward the total. So if there are 15 records in the input data file, and records 2 and 4 are bad, then only the following 8 records are actually loaded into the table: 1, 3, 5, 6, 7, 8, 9, and 10.

8.2.27 LOAD_SHARDS

The `LOAD_SHARDS` SQL*Loader command-line parameter specifies a specific list of shards to load from a sharded table.

Default

If no list of shards is specified, then all shards are loaded.

Purpose

The `LOAD_SHARDS` parameter specifies a comma-delimited list of shard identifiers (shard names). If you do not specify a list, then SQL*LOADER loads all shards.

For sharded tables, use this parameter after attempting automatic parallel loading where some shards failed to load. To resolve the issue, you can perform an automatic parallel load, and use the `LOAD_SHARDS` parameter to provide a list to SQL*Loader of any shards that failed to load in

the previous load attempt. SQL*Loader will ignore the shards that you do not list with `LOAD_SHARDS`.

> **See Also:**
>
> Parallel Data Loading Models

Syntax and Description

```
LOAD_SHARDS=shard1,shard2,shard3 . . .
```

Example

In this SQL*Loader command line, the `LOAD_SHARDS` parameter specifies to load only the `dbs7` and `dbs23` shards:

```
sqlldr scott/tiger t.ctl gsm_name=shdsrv.shpool.oradbcloud gsm_host=example1
gsm_port=4338 load_shards=dbs7,dbs23
```

8.2.28 LOG

The `LOG` SQL*Loader command-line parameter specifies a directory path, or file name, or both for the log file where SQL*Loader stores logging information about the loading process.

Default

The current directory, if no value is specified.

Purpose

Specifies a directory path, or file name, or both for the log file that SQL*Loader uses to store logging information about the loading process.

Syntax and Description

```
LOG=[[directory/][log_file_name]]
```

If you specify the `LOG` parameter, then you must supply a directory name, or a file name, or both.

If no directory name is specified, it defaults to the current directory.

If a directory name is specified without a file name, then the default log file name is used.

Example

The following example creates a log file named `emp1.log` in the current directory. The extension `.log` is used even though it is not specified, because it is the default.

```
LOG=emp1
```

8.2.29 MULTITHREADING

The `MULTITHREADING` SQL*Loader command-line parameter enables stream building on the client system to be done in parallel with stream loading on the server system.

Default

`TRUE` on multiple-CPU systems, `FALSE` on single-CPU systems

Syntax and Description

```
MULTITHREADING=[TRUE | FALSE]
```

By default, the multithreading option is always enabled (set to `TRUE`) on multiple-CPU systems. In this case, the definition of a multiple-CPU system is a single system that has more than one CPU.

On single-CPU systems, multithreading is set to `FALSE` by default. To use multithreading between two single-CPU systems, you must enable multithreading; it will not be on by default.

Restrictions

> **Note:**
>
> This option is normally disabled for automatic parallel loading. If enabled, it is possible that it can improve performance, but be aware that this option adds an additional thread for each direct path parallel loading thread.

- The `MULTITHREADING` parameter is available only for direct path loads.
- Multithreading functionality is operating system-dependent. Not all operating systems support multithreading.

Example

The following example enables multithreading on a single-CPU system. On a multiple-CPU system it is enabled by default.

```
MULTITHREADING=TRUE
```

Related Topics

- Optimizing Direct Path Loads on Multiple-CPU Systems
  If you are performing direct path loads on a multiple-CPU system, then SQL*Loader uses multithreading by default. A multiple-CPU system in this case is defined as a single system that has two or more CPUs.

8.2.30 NO_INDEX_ERRORS

The `NO_INDEX_ERRORS` SQL*Loader command-line parameter specifies whether indexing errors are tolerated during a direct path load.

Default

`FALSE`

Syntax and Description

```
NO_INDEX_ERRORS=[TRUE | FALSE]
```

A setting of `NO_INDEX_ERRORS=FALSE` means that if a direct path load results in an index becoming unusable, then the rows are loaded, and the index is left in an unusable state. This is the default behavior.

A setting of `NO_INDEX_ERRORS=TRUE` means that if a direct path load results in any indexing errors, then the load is stopped. No rows are loaded, and the indexes are left as they were.

Restrictions

The `NO_INDEX_ERRORS` parameter is valid only for direct path loads. If it is specified for conventional path loads, then it is ignored.

Example

```
NO_INDEX_ERRORS=TRUE
```

8.2.31 OPTIMIZE PARALLEL

The SQL*Loader `OPTIMIZE_PARALLEL` parameter specifies whether automatic parallel loads should enable SQL*Loader to choose the optimal parallel loading option.

Default

`TRUE`

Purpose

Specifies whether you want to enable SQL*Loader to choose the fastest parallel load option available to your data automatically, or if you want to specify a particular automatic parallel load mode. Oracle recommends that you accept the default.

Syntax and Description

```
OPTIMIZE_PARALLEL=[TRUE|FALSE]
```

Starting with Oracle Database 23ai, SQL*Loader can perform parallel loads automatically, and select the fastest mode available for your tables, depending on whether they are non-sharded or sharded tables. This is the default option for automatic parallel loading. Oracle recommends that you accept the default. However, you can use this parameter to override SQL*Loader selecting the parallel loading mode, so that you can try an alternate client parallel mode to see if it can run faster.

Example

The following example specifies that SQL*Loader will not select the optimal parallel load option on its own, and instead let you specify the load option.

```
OPTIMIZE_PARALLEL=FALSE
```

Related Topics

- Loading Modes for Automatic Parallel Loads

8.2.32 PARALLEL

The SQL*Loader `PARALLEL` parameter specifies whether loads that use direct path can operate in multiple concurrent sessions to load data into the same table.

Default

```
FALSE
```

Purpose

Specifies whether loads that use direct path can operate in multiple concurrent sessions to load data into the same table.

> **Note:**
>
> The default for `PARALLEL` is `FALSE`, but if you use direct path automatic parallel loading and set the parameter `DEGREE_OF_PARALLELISM`, then `PARALLEL` is automatically set to `TRUE` for direct path if parallelism can be implemented, so you do not need to specify `PARALLEL`.

Syntax and Description

```
PARALLEL=[TRUE | FALSE]
```

Restrictions

- The `PARALLEL` parameter is not valid in conventional path loads.

Example

The following example specifies that the load will be performed in parallel.

```
PARALLEL=TRUE
```

Related Topics

- About SQL*Loader Parallel Data Loading Models
  There are three basic models of concurrency that you can use to minimize the elapsed time required for data loading.

8.2.33 PARFILE

The `PARFILE` SQL*Loader command-line parameter specifies the name of a file that contains commonly used command-line parameters.

Default

There is no default.

Syntax and Description

```
PARFILE=file_name
```

Instead of specifying each parameter on the command line, you can simply specify the name of the parameter file. For example, a parameter file named `daily_report.par` might have the following contents:

```
USERID=scott
CONTROL=daily_report.ctl
ERRORS=9999
LOG=daily_report.log
```

For security reasons, do not include your `USERID` password in a parameter file. After you specify the parameter file at the command line, SQL*Loader prompts you for the password. For example:

```
sqlldr PARFILE=daily_report.par
Password: password
```

Restrictions

- On some systems it can be necessary to have no spaces around the equal sign (=) in the parameter specifications.

Example

See the example in the Syntax and Description section.

8.2.34 PARTITION_MEMORY

The PARFILE SQL*Loader command-line parameter specifies the amount of memory that you want to have used when you are loading many partitions.

Default

`0` (zero) This setting limits memory use based on the value of the `PGA_AGGREGATE_TARGET` initialization parameter. When memory use approaches that value, loading of some partitions is delayed.

Purpose

Specifies the amount of memory that you want to have used when you are loading many partitions. This parameter is helpful in situations in which the number of partitions you are

loading use up large amounts of memory, perhaps even exceeding available memory. (This scenario can occur, especially when the data is compressed).

After the specified limit is reached, loading of some partition rows is delayed until memory use falls below the limit.

Syntax and Description

```
PARTITION_MEMORY=n
```

The parameter value `n` is in kilobytes.

If `n` is set to `0` (the default), then SQL*Loader uses a value that is a function of the `PGA_AGGREGATE_TARGET` initialization parameter.

If `n` is set to `-1` (minus 1), then SQL*Loader makes no attempt to use less memory when loading many partitions.

Restrictions

- This parameter is only valid for direct path loads.
- This parameter is available only in Oracle Database 12c Release 1 (12.1.0.2) and later releases.

Example

The following example limits memory use to 1 GB.

```
> sqlldr hr CONTROL=t.ctl DIRECT=true PARTITION_MEMORY=1000000
```

8.2.35 READER_COUNT

The `READER_COUNT` SQL*Loader command-line parameter specifies the number of input data file reader threads for automatic parallel loads.

Default

```
1
```

Syntax and Description

```
READER_COUNT=n
```

The use case for the `READER_COUNT` parameter depends on the mode of automatic parallel loading that you use.

For non-sharded tables, Mode One parallel loading is the fastest option. The `READER_COUNT` parameter is ignored with this mode, because SQL*Loader automatically divides up data files into granules of data, and the threads parse and load these granules.

When using Mode Two parallel loading, `DEGREE_OF_PARALLELISM` determines the number of loader threads. This is the fastest mode that you can use when loading sharded tables in parallel. When loading non-sharded tables, however, this is the non-optimized mode. In Mode Two, reader and loader threads appear separately in the log file, either as reader or as loader threads.

When using Mode Three automatic parallel loads, SQL*Loader Reader/Loaders read all files (no granules) for sharded tables.

The `READER_COUNT` parameter determines the number of readers available to read files.

Restrictions

Example

The following example sets the number of reader threads to five.

```
READER_COUNT=5
```

Related Topics

- Loading Modes for Automatic Parallel Loads

8.2.36 READSIZE

The `READSIZE` SQL*Loader command-line parameter specifies (in bytes) the size of the read buffer, if you choose not to use the default.

Default

```
1048576
```

Syntax and Description

```
READSIZE=n
```

In the conventional path method, the bind array is limited by the size of the read buffer. Therefore, the advantage of a larger read buffer is that more data can be read before a commit operation is required.

For example, setting `READSIZE` to `1000000` enables SQL*Loader to perform reads from the data file in chunks of 1,000,000 bytes before a commit is required.

> **Note:**
>
> If the `READSIZE` value specified is smaller than the `BINDSIZE` value, then the `READSIZE` value is increased.

For automatic parallel loading, to increase the read buffer when loading shards, you can use the `READSIZE` parameter to set a higher buffer value.

Restrictions

- The `READSIZE` parameter is used *only* when reading data from data files. When reading records from a control file, a value of 64 kilobytes (KB) is *always* used as the `READSIZE`.
- The `READSIZE` parameter has no effect on Large Objects (LOBs). The size of the LOB read buffer is fixed at 64 kilobytes (KB).
- The maximum size allowed is platform-dependent.

Example

The following example sets the size of the read buffer to 500,000 bytes, which means that commit operations will be required more often than if the default or a value larger than the default were used.

```
READSIZE=500000
```

Related Topics

- BINDSIZE

8.2.37 RESUMABLE

The `RESUMABLE` SQL*Loader command-line parameter enables and disables resumable space allocation.

Default

```
FALSE
```

Purpose

Enables and disables resumable space allocation.

Syntax and Description

```
RESUMABLE=[TRUE | FALSE]
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about resumable space allocation.

Restrictions

- Because this parameter is disabled by default, you must set `RESUMABLE=TRUE` to use its associated parameters, `RESUMABLE_NAME` and `RESUMABLE_TIMEOUT`.

Example

The following example enables resumable space allocation:

```
RESUMABLE=TRUE
```

8.2.38 RESUMABLE_NAME

The `RESUMABLE_NAME` SQL*Loader command-line parameter identifies a statement that is resumable.

Default

```
'User USERNAME(USERID), Session SESSIONID, Instance INSTANCEID'
```

Syntax and Description

```
RESUMABLE_NAME='text_string'
```

This value is a user-defined text string that is inserted in either the `USER_RESUMABLE` or `DBA_RESUMABLE` view to help you identify a specific resumable statement that has been suspended.

Restrictions

- This parameter is ignored unless the `RESUMABLE` parameter is set to `TRUE` to enable resumable space allocation.

Example

```
RESUMABLE_NAME='my resumable sql'
```

8.2.39 RESUMABLE_TIMEOUT

The `RESUMABLE_TIMEOUT` SQL*Loader command-line parameter specifies the time period, in seconds, during which an error must be fixed.

Default

`7200` seconds (2 hours)

Syntax and Description

```
RESUMABLE_TIMEOUT=n
```

If the error is not fixed within the timeout period, then execution of the statement is terminated, without finishing.

Restrictions

- This parameter is ignored unless the `RESUMABLE` parameter is set to `TRUE` to enable resumable space allocation.

Example

The following example specifies that errors must be fixed within ten minutes (600 seconds).

```
RESUMABLE_TIMEOUT=600
```

8.2.40 ROWS

For conventional path loads, the ROWS SQL*Loader command-line parameter specifies the number of rows in the bind array, and in direct path loads, the number of rows to read from data files before a save.

Default

Specifies the number of rows in the bind array. The Conventional path default is 64. Direct path default is all rows.

Purpose

For conventional path loads the ROWS parameter specifies the number of rows in the bind array.For direct path loads, the ROWS parameter specifies the number of rows that SQL*Loader reads from the data files before a data save.

Syntax

```
ROWS=n
```

Conventional Path Loads Description

In conventional path loads only, the ROWS parameter specifies the number of rows in the bind array. The maximum number of rows is 65534.

Direct Path Loads Description

In direct path loads only, the ROWS parameter identifies the number of rows that you want to read from the data file before a data save. The default is to read all rows and save data once at the end of the load. The actual number of rows loaded into a table on a save is approximately the value of ROWS minus the number of discarded and rejected records since the last save.

> **Note:**
>
> If you specify a low value for ROWS, and then attempt to compress data using table compression, then the compression ratio probably will be degraded. When compressing the data, Oracle recommends that you either specify a high value, or accept the default value.

Restrictions

- The ROWS parameter is ignored for direct path loads when data is loaded into an Index Organized Table (IOT), or into a table containing VARRAY types, XML columns, or Large Objects (LOBs). This means that the load still takes place, but no save points are done.
- For direct path loads, because LONG VARCHAR data type data are stored as LOBs, you cannot use the ROWS parameter. If you attempt to use the ROWS parameter with LONG VARCHAR data in direct path loads, then you receive an ORA-39777 error (Data saves are not allowed when loading LOB columns).

Example

In a conventional path load, the following example would result in an error because the specified value exceeds the allowable maximum of 65534 rows.

```
ROWS=65900
```

Related Topics

- Using Data Saves to Protect Against Data Loss
  When you have a savepoint, if you encounter an instance failure during a SQL*Loader load, then use the `SKIP` parameter to continue the load.

8.2.41 SDF_PREFIX

The `SDF_PREFIX` SQL*Loader command-line parameter specifies a directory prefix, which is added to file names of LOBFILEs and secondary data files (SDFs) that are opened as part of a load operation.

Default

There is no default.

Purpose

Specifies a directory prefix, which is added to file names of LOBFILEs and secondary data files (SDFs) that are opened as part of a load operation.

> **Note:**
>
> The `SDF_PREFIX` parameter can also be specified in the `OPTIONS` clause in the SQL Loader control file.

Syntax and Description

```
SDF_PREFIX=string
```

If `SDF_PREFIX` is specified, then the string value must be specified as well. There is no validation or verification of the string. The value of `SDF_PREFIX` is prepended to the filenames used for all LOBFILEs and SDFs opened during the load. If the resulting string is not the name of as valid file, then the attempt to open that file fails and an error is reported.

If `SDF_PREFIX` is not specified, then file names for LOBFILEs and SDFs are assumed to be relative to the current working directory. Using `SDF_PREFIX` allows those files names to be relative to a different directory.

Quotation marks are only required around the string if it contains characters that would confuse the command line parser (for example, a space).

The file names that are built by prepending `SDF_PREFIX` to the file names found in the record are passed to the operating system to open the file. The prefix can be relative to the current working directory from which SQL*Loader is being executed or it can be the start of an absolute path.

Restrictions

- The `SDF_PREFIX` parameter should not be used if the file specifications for the LOBFILEs or SDFs contain full file names.

Example

The following SQL*Loader command looks for LOB files in the `lobdir` subdirectory of the current directory

```
sqlldr control=picts.ctl log=picts.log sdf_prefix=lobdir/
```

8.2.42 SILENT

The `SILENT` SQL*Loader command-line parameter suppresses some of the content that is written to the screen during a SQL*Loader operation.

Default

There is no default.

Syntax and Description

```
SILENT=[HEADER | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL]
```

Use the appropriate values to suppress one or more of the following (if more than one option is specified, they must be separated by commas):

- `HEADER`: Suppresses the SQL*Loader header messages that normally appear on the screen. Header messages still appear in the log file.
- `FEEDBACK`: Suppresses the "commit point reached" messages and the status messages for the load that normally appear on the screen. But `"XX Rows successfully loaded."` even prints on the screen.
- `ERRORS`: Suppresses the data error messages in the log file that occur when a record generates an Oracle error that causes it to be written to the bad file. A count of rejected records still appears.
- `DISCARDS`: Suppresses the messages in the log file for each record written to the discard file.
- `PARTITIONS`: Disables writing the per-partition statistics to the log file during a direct load of a partitioned table.
- `ALL`: Implements all of the suppression values: `HEADER, FEEDBACK, ERRORS, DISCARDS,` and `PARTITIONS`. But `"XX Rows successfully loaded."` even prints on the screen.

Example

You can suppress the header and feedback messages that normally appear on the screen with the following command-line argument:

```
SILENT=HEADER, FEEDBACK
```

But `"XX Rows successfully loaded."` even prints on the screen.

8.2.43 SKIP

The `SKIP` SQL*Loader command-line parameter specifies the number of logical records from the beginning of the file that should not be loaded.

Default

`0` (No records are skipped.)

Purpose

Specifies the number of logical records from the beginning of the file that should not be loaded. Using this specification enables you to continue loads that have been interrupted for some reason, without loading records that have already been processed.

Syntax and Description

```
SKIP=n
```

You can use the `SKIP` parameter for all conventional loads, for single-table direct path loads, and for multiple-table direct path loads when the same number of records was loaded into each table. You cannot use `SKIP` for multiple-table direct path loads when a different number of records was loaded into each table.

If a `WHEN` clause is also present, and the load involves secondary data, then the secondary data is skipped only if the `WHEN` clause succeeds for the record in the primary data file.

Restrictions

- The `SKIP` parameter cannot be used for external table loads.

Example

The following example skips the first 500 logical records in the data files before proceeding with the load:

```
SKIP=500
```

Related Topics

- Interrupted SQL*Loader Loads
  Learn about common scenarios in which SQL*Loader loads are interrupted or discontinued, and what you can do to correct these issues.

8.2.44 SKIP_INDEX_MAINTENANCE

The `SKIP_INDEX_MAINTENANCE` SQL*Loader command-line parameter specifies whether to stop index maintenance for direct path loads.

Default

`FALSE`

Purpose

Specifies whether to stop index maintenance for direct path loads.

Syntax and Description

```
SKIP_INDEX_MAINTENANCE=[TRUE | FALSE]
```

If set to `TRUE`, this parameter causes the index partitions that would have had index keys added to them to instead be marked Index Unusable because the index segment is inconsistent with respect to the data it indexes. Index segments that are unaffected by the load retain the state they had before the load.

The `SKIP_INDEX_MAINTENANCE` parameter:

- Applies to both local and global indexes
- Can be used (with the `PARALLEL` parameter) to perform parallel loads on an object that has indexes
- Can be used (with the `PARTITION` parameter on the `INTO TABLE` clause) to do a single partition load to a table that has global indexes
- Records a list (in the SQL*Loader log file) of the indexes and index partitions that the load set to an Index Unusable state

Restrictions

- The `SKIP_INDEX_MAINTENANCE` parameter does not apply to conventional path loads.
- Indexes that are unique and marked Unusable are not allowed to skip index maintenance. This rule is enforced by DML operations, and enforced by the direct path load to be consistent with DML.

Example

The following example stops index maintenance from taking place during a direct path load operation:

```
SKIP_INDEX_MAINTENANCE=TRUE
```

8.2.45 SKIP_UNUSABLE_INDEXES

The `SKIP_UNUSABLE_INDEXES` SQL*Loader command-line parameter specifies whether to skip an index encountered in an Index Unusable state and continue the load operation.

Default

The value of the Oracle Database configuration parameter, `SKIP_UNUSABLE_INDEXES`, as specified in the initialization parameter file. The default database setting is `TRUE`.

Purpose

Specifies whether to skip an index encountered in an Index Unusable state and continue the load operation.

Syntax and Description

```
SKIP_UNUSABLE_INDEXES=[TRUE | FALSE]
```

A value of `TRUE` for `SKIP_UNUSABLE_INDEXES` means that if an index in an Index Unusable state is encountered, it is skipped and the load operation continues. This allows SQL*Loader to load a table with indexes that are in an Unusable state before the beginning of the load. Indexes that are not in an Unusable state at load time will be maintained by SQL*Loader. Indexes that are in an Unusable state at load time will not be maintained, but instead will remain in an Unusable state at load completion.

Both SQL*Loader and Oracle Database provide a `SKIP_UNUSABLE_INDEXES` parameter. The SQL*Loader `SKIP_UNUSABLE_INDEXES` parameter is specified at the SQL*Loader command line. The Oracle Database `SKIP_UNUSABLE_INDEXES` parameter is specified as a configuration parameter in the initialization parameter file. It is important to understand how they affect each other.

If you specify a value for `SKIP_UNUSABLE_INDEXES` at the SQL*Loader command line, then it overrides the value of the `SKIP_UNUSABLE_INDEXES` configuration parameter in the initialization parameter file.

If you do not specify a value for `SKIP_UNUSABLE_INDEXES` at the SQL*Loader command line, then SQL*Loader uses the Oracle Database setting for the `SKIP_UNUSABLE_INDEXES` configuration parameter, as specified in the initialization parameter file. If the initialization parameter file does not specify a setting for `SKIP_UNUSABLE_INDEXES`, then the default setting is `TRUE`.

The `SKIP_UNUSABLE_INDEXES` parameter applies to both conventional and direct path loads.

Restrictions

- Indexes that are unique and marked Unusable are not allowed to skip index maintenance. This rule is enforced by DML operations, and enforced by the direct path load to be consistent with DML.

Example

If the Oracle Database initialization parameter has a value of `SKIP_UNUSABLE_INDEXES=FALSE`, then setting `SKIP_UNUSABLE_INDEXES=TRUE` on the SQL*Loader command line overrides it. Therefore, if an index in an Index Unusable state is encountered after this parameter is set, then it is skipped, and the load operation continues.

```
SKIP_UNUSABLE_INDEXES=TRUE
```

8.2.46 STREAMSIZE

The `STREAMSIZE` SQL*Loader command-line parameter specifies the size (in bytes) of the data stream sent from the client to the server.

Default

256000

Purpose

Specifies the size (in bytes) of the data stream sent from the client to the server.

Syntax and Description

```
STREAMSIZE=n
```

The `STREAMSIZE` parameter specifies the size of the direct path stream buffer. The number of column array rows (specified with the `COLUMNARRAYROWS` parameter) determines the number of rows loaded before the stream buffer is built. The optimal values for these parameters vary, depending on the system, input data types, and Oracle column data types used. When you are using optimal values for your particular configuration, the elapsed time in the SQL*Loader log file should go down.

Restrictions

- The `STREAMSIZE` parameter applies only to direct path loads.
- The minimum value for `STREAMSIZE` is 65536. If a value lower than 65536 is specified, then 65536 is used instead.

Example

The following example specifies a direct path stream buffer size of 300,000 bytes.

```
STREAMSIZE=300000
```

Related Topics

- Specifying the Number of Column Array Rows and Size of Stream Buffers
  The number of column array rows determines the number of rows loaded before the stream buffer is built.

8.2.47 TRIM

The `TRIM` SQL*Loader command-line parameter specifies whether you want spaces trimmed from the beginning of a text field, the end of a text field, both, or neither.

Default

```
LDRTRIM
```

Purpose

Specifies that spaces should be trimmed from the beginning of a text field, the end of a text field, both, or neither. Spaces include blanks and other nonprinting characters, such as tabs, line feeds, and carriage returns.

Syntax and Description

```
TRIM=[LRTRIM | NOTRIM | LTRIM | RTRIM | LDRTRIM]
```

The valid values for the `TRIM` parameter are as follows:

- `NOTRIM` indicates that you want no characters trimmed from the field. This setting generally yields the fastest performance.
- `LRTRIM` indicates that you want both leading and trailing spaces trimmed from the field.
- `LTRIM` indicates that you want leading spaces trimmed from the field
- `RTRIM` indicates that you want trailing spaces trimmed from the field.
- `LDRTRIM` is the same as `NOTRIM` except in the following cases:
  - If the field is not a delimited field, then spaces are trimmed from the right.
  - If the field is a delimited field with `OPTIONALLY ENCLOSED BY` specified, and the optional enclosures are missing for a particular instance, then spaces are trimmed from the left.

> **Note:**
>
> If trimming is specified for a field that consists only of spaces, then the field is set to `NULL`.

Restrictions

- The `TRIM` parameter is valid only when the external table load method is used.

Example

The following example specifies a load operation for which no characters are trimmed from any fields:

```
TRIM=NOTRIM
```

8.2.48 USERID

The `USERID` SQL*Loader command-line parameter provides your Oracle username and password for SQL*Loader.

Default

There is no default.

Purpose

Provides your Oracle user name and password for SQL*Loader, so that you are not prompted to provide them. If it is omitted, then you are prompted for them. If you provide as the value a slash (`/`), then `USERID` defaults to your operating system login.

Syntax and Description

```
USERID=[username | / | SYS]
```

Specify a user name. For security reasons, Oracle recommends that you specify only the user name on the command line. SQL*Loader then prompts you for a password.

If you do not specify the `USERID` parameter, then you are prompted for it. If you use a forward slash (virgule), then `USERID` defaults to your operating system login.

If you connect as user `SYS`, then you must also specify `AS SYSDBA` in the connect string.

Restrictions

- Because the string `AS SYSDBA`, contains a blank, some operating systems can require that you place the entire connect string inside quotation marks, or marked as a literal by some other method. Some operating systems also require that quotation marks on the command line are preceded by an escape character, such as backslashes.

  Refer to your operating system-specific documentation for information about special and reserved characters on your system.

Example

The following example specifies a user name of `hr`. SQL*Loader then prompts for a password. Because it is the first and only parameter specified, you do not need to include the parameter name `USERID`:

```
> sqlldr hr
Password:
```

Related Topics

- Specifying Parameters on the Command Line
  When you start SQL*Loader, you specify parameters to establish various characteristics of the load operation.

8.3 Exit Codes for Inspection and Display

Oracle SQL*Loader provides the results of a SQL*Loader run immediately upon completion.

Usage Notes

In addition to recording the results in a log file, SQL*Loader may also report the outcome in a process exit code. This Oracle SQL*Loader functionality allows for checking the outcome of a SQL*Loader invocation from the command line or a script. The following table shows the exit codes for various results:

Table 8-1 Exit Codes for SQL*Loader

Result	Exit Code
All rows loaded successfully	`EX_SUCC`
All or some rows rejected	`EX_WARN`
All or some rows discarded	`EX_WARN`
Discontinued load	`EX_WARN`
Command-line or syntax errors	`EX_FAIL`
Oracle errors nonrecoverable for SQL*Loader	`EX_FAIL`
Operating system errors (such as file open/close and malloc)	`EX_FTL`

Examples

For Linux and Unix operating systems, the exit codes are as follows:

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  3
```

For Windows operating systems, the exit codes are as follows:

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  4
```

If SQL*Loader returns any exit code other than zero, then consult your system log files and SQL*Loader log files for more detailed diagnostic information.

On Unix platforms, you can check the exit code from the shell to determine the outcome of a load.