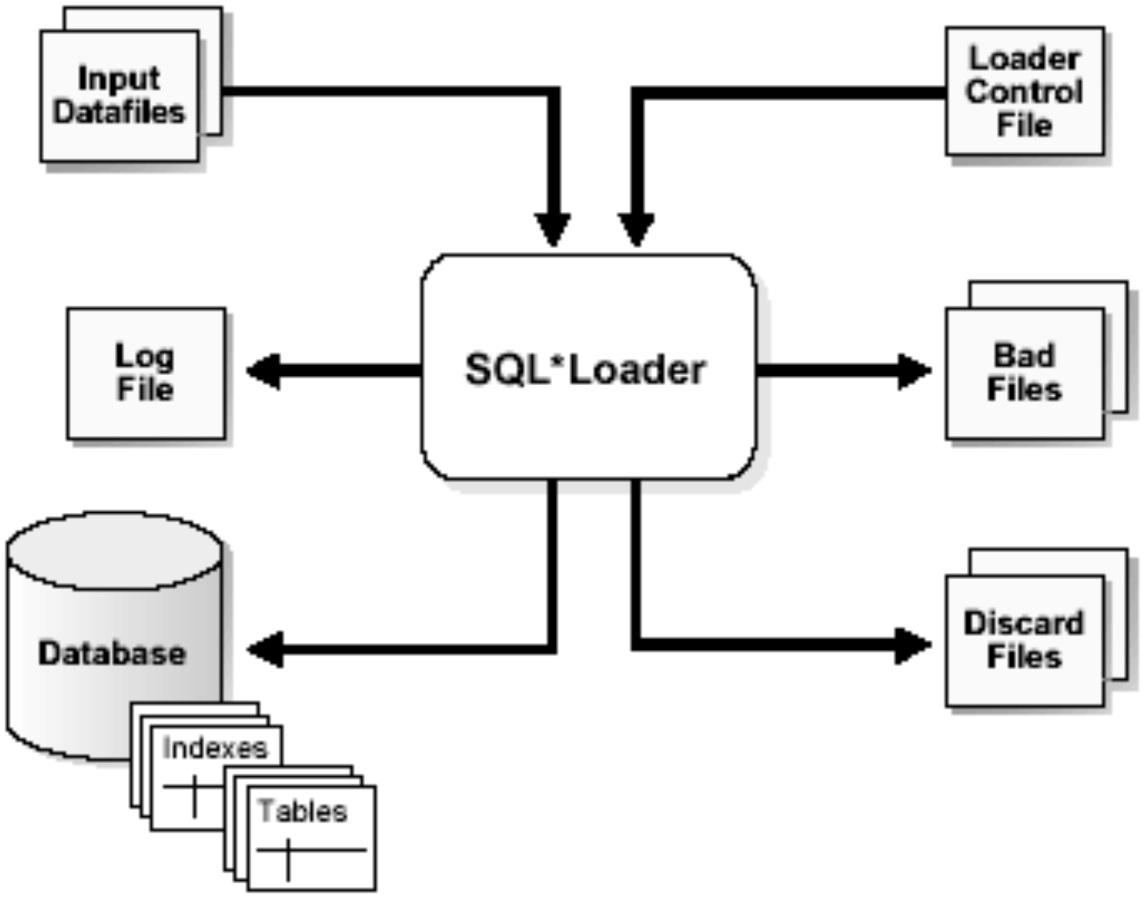
SQL Loader

Introduction

- SQL*Loader loads data from external files into tables of an Oracle database.
- You can use SQL*Loader to do the following:
  - Load data from multiple datafiles during the same load session.
  - Load data into multiple tables during the same load session.
  - Specify the character set of the data.
  - Selectively load data (you can load records based on the records' values).
  - Manipulate the data before loading it, using SQL functions.
  - Generate unique sequential key values in specified columns.
  - Use the operating system's file system to access the datafiles.
  - Load data from disk, tape, or named pipe.
  - Generate sophisticated error reports, which greatly aids troubleshooting.
  - Load arbitrarily complex object-relational data.
  - Use secondary datafiles for loading LOBs and collections.
  - Use either conventional or direct path loading.
  - Use a DB2 Load Utility control file as a SQL*Loader control file with few or no changes.

Introduction

- A typical SQL*Loader session takes as input a control file, which controls the behavior of SQL*Loader, and one or more datafiles. The output of SQL*Loader is an Oracle database (where the data is loaded), a log file, a bad file, and potentially, a discard file.
- An example of the flow of a SQL*Loader session is shown



SQL*Loader Control File

- The control file is a text file written in a language that SQL*Loader understands.
- The control file tells SQL*Loader where to find the data, how to parse and interpret the data, where to insert the data, and more.
- Although not precisely defined, a control file can be said to have three sections.
- The first section contains session-wide information, for example:
    - Global options such as bindsize, rows, records to skip, and so on
    - INFILE clauses to specify where the input data is located
    - Data to be loaded
- The second section consists of one or more INTO TABLE blocks. Each of these blocks contains information about the table into which the data is to be loaded, such as the table name and the columns of the table.
- The third section is optional and, if present, contains input data.

Input Data and Datafiles

- SQL*Loader reads data from one or more files (or operating system equivalents of files) specified in the control file.
- From SQL*Loader’s perspective, the data in the datafile is organized as *records*. A particular datafile can be in fixed *record format, variable record format, or stream record format*.
- The record format can be specified in the control file with the INFILE parameter. (default is stream record format)

Input Data and Datafiles

Fixed Record Format

- A file is in fixed record format when all records in a datafile are the same byte length.
- Fixed format is simple to specify. For example:

```
INFILE datafile_name "fix n"
```

- This example specifies that SQL*Loader should interpret the particular datafile as being in fixed record format where every record is n bytes long.
- Consider the example control file data :

```
load data
infile ’example.dat’ "fix 11"
into table example
fields terminated by ’,’ optionally enclosed by ’"’
(col1, col2)
```

- The file ‘example.dat’ has the data as below:

```
   001, cd, 0002,fghi,
   00003,lmn,
   1, "pqrs",
   0005,uvwx,
```

Input Data and Datafiles

Variable Record Format

- A file is in variable record format when the length of each record in a character field is included at the beginning of each record in the datafile.
- This format provides some added flexibility over the fixed record format and a performance advantage over the stream record format.

```
INFILE "datafile_name" "var n"
```

- In this example, n specifies the number of bytes(< 40) in the record length field. (Default 5)
- Consider the example control file data :

```
load data
infile ’example.dat’ "var 3"
into table example
fields terminated by ’,’ optionally enclosed by ’"’
(col1 char(5),col2 char(7))
```

- The file ‘example.dat’ has the data as below:

```
009hello,cd,010world,im,
012my,name is,
```

- The control file specification that tells SQL*Loader to look for data in the datafile example.dat and to expect variable record format where the record length fields are 3 bytes long.

Input Data and Datafiles

Stream Record Format

- A file is in stream record format when the records are not specified by size; instead SQL*Loader forms records by scanning for the record terminator.
- The specification of a datafile to be interpreted as being in stream record format :

```
INFILE datafile_name ["str terminator_string"]
```

- The terminator_string is specified as either ’ char_string’ or X’hex_ string’ where:
  - ’ char_string’ is a string of characters enclosed in single or double quotation marks
  - X’hex_string’ is a byte string in hexadecimal format
- When the terminator_string contains special (nonprintable) characters, it should be specified as a X’hex_string’. Some nonprintable characters can be specified as (’ char_string’) by using a backslash. For example:

\n linefeed	\t horizontal tab
\r carriage return	\v vertical tab

```
load data
infile ’example.dat’ "str ’|\n’"
into table example
fields terminated by ’,’ optionally enclosed by ’"’
(col1 char(5),col2 char(7))
```

- The following is the data in the ‘example.dat’ file:

```
hello,world,|
james,bond,|
```

Input Data and Datafiles

Logical Records

- SQL*Loader organizes the input data into physical records, according to the specified record format.
- By default a physical record is a logical record, but for added flexibility, SQL*Loader can be instructed to combine a number of physical records into a logical record.
- SQL*Loader can be instructed to follow one of the following two logical record forming strategies:
    - Combine a fixed number of physical records to form each logical record
    - Combine physical records into logical records while a certain condition is true

Input Data and Datafiles

Data Fields

- Once a logical record is formed, field setting on the logical record is done.
- Field setting is a process in which SQL*Loader uses control-file field specifications to determine which parts of logical record data correspond to which control-file fields.
- Most control-file field specifications claim a particular part of the logical record. This mapping takes the following forms:
  - The byte position of the data field’s beginning, end, or both, can be specified.
  - The strings delimiting (enclosing and/or terminating) a particular data field can be specified. A delimited data field is assumed to start where the last data field ended, unless the byte position of the start of the data field is specified.
  - The byte offset and/or the length of the data field can be specified. This way each field starts a specified number of bytes from where the last one ended and continues for a specified length.
  - Length-value datatypes can be used. In this case, the first n number of bytes of the data field contain information about how long the rest of the data field is.

Data Conversion and Datatype Specification

- During a conventional path load, *data fields* in the datafile are converted into *columns* in the database (direct path loads are conceptually similar, but the implementation is different). There are two conversion steps:
  1. SQL*Loader uses the field specifications in the control file to interpret the format of the datafile, parse the input data, and populate the bind arrays that correspond to a SQL INSERT statement using that data.
  2. The Oracle database server accepts the data and executes the INSERT statement to store the data in the database.
- The Oracle database server uses the datatype of the column to convert the data into its final, stored form.

Discarded and Rejected Records

- Records read from the input file might not be inserted into the database. Such records are placed in either a bad file or a discard file.
- **The Bad File**

  The bad file contains records that were rejected, either by SQL*Loader or by the Oracle database server. Given below are some of the reasons why records are rejected.
- **SQL*Loader Rejects**

  Records are rejected by SQL*Loader when the input format is invalid. For example, if the second enclosure delimiter is missing, or if a delimited field exceeds its maximum length, SQL*Loader rejects the record. Rejected records are placed in the bad file.
- **Oracle Rejects**

  After a record is accepted for processing by SQL*Loader, a row is sent to the Oracle database server for insertion. If Oracle determines that the row is valid, then the row is inserted into the database. If not, the record is rejected, and SQL*Loader puts it in the bad file. The row may be rejected, for example, because a key is not unique, because a required field is null, or because the field contains invalid data for the Oracle datatype.
- **The Discard File**

  As SQL*Loader executes, it may create a file called the discard file. The discard file contains records that were filtered out of the load because they did not match any record-selection criteria specified in the control file. The discard file therefore contains records that were not inserted into any table in the database.

Log File and Logging Information

- When SQL*Loader begins execution, it creates a *log file.* If it cannot create a log file, execution terminates.
- The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.

SQL*Loader Command-Line Reference

Invoking SQL*Loader

- When you invoke SQL*Loader, you can specify certain parameters to establish session characteristics. You specify values for parameters, or in some cases, you can accept the default without entering a value.
- For example:

-
  ```
  SQLLDR CONTROL=foo.ctl, LOG=bar.log,
  BAD=baz.bad, DATA=etc.dat USERID=scott/tiger,
  ERRORS=999, LOAD=2000,
  DISCARD=toss.dis,DISCARDMAX=5
  ```
- If you invoke SQL*Loader without specifying any parameters, SQL*Loader displays a help screen which lists the available parameters and their default values.

Command-Line Parameters

BAD (bad file)

- Default: The name of the datafile, with an extension of `.bad`.
- `BAD` specifies the name of the bad file created by SQL*Loader to store records that cause errors specified during insert or that are improperly formatted. If a filename is not, the default is used.

- **CONTROL (control file)**
- Default: none
- `CONTROL` specifies the name of the SQL*Loader control file that describes how to load data. If a file extension or file type is not specified, it defaults to `.ctl`. If the filename is omitted, SQL*Loader prompts you for it.

DATA (datafile)

- Default: The name of the control file, with an extension of `.dat`.
- `DATA` specifies the name of the datafile containing the data to be loaded. If you do not specify a file extension or file type, the default is `.dat.`

ERRORS (errors to allow)

- `ERRORS` specifies the maximum number of insert errors to allow.
- If the number of errors exceeds the value specified for `ERRORS`, then SQL*Loader terminates the load.
- To permit no errors at all, set `ERRORS=0.` To specify that all errors be allowed,use a very high number.
- On a single-table load, SQL*Loader terminates the load when errors exceed this error limit. Any data inserted up that point, however, is committed.

LOAD (records to load)

- Default: All records are loaded.
- `LOAD` specifies the maximum number of logical records to load
- No error occurs if fewer than the maximum number of records are found.

LOG (log file)

- Default: The name of the control file, with an extension of `.log`.
- `LOG` specifies the log file that SQL*Loader will create to store logging information about the loading process.

DISCARD (filename)

- Default: The name of the datafile, with an extension of `.dsc`.
- `DISCARD` specifies a discard file (optional) to be created by SQL*Loader to store records that are neither inserted into a table nor rejected.
- A discard file filename specified on the command line becomes the discard file associated with the first `INFILE` statement in the control file.
- If the discard file filename is specified also in the control file, the command-line value overrides it.

DISCARDMAX (integer)

- Default: `ALL`
- `DISCARDMAX` specifies the number of discard records to allow before data loading is terminated. To stop on the first discarded record, specify one (1).

Control File Contents

- The SQL*Loader control file is a text file that contains data definition language(DDL) instructions.
- DDL is used to control the following aspects of a SQL*Loader session:

1. Where SQL*Loader will find the data to load
2. How SQL*Loader expects that data to be formatted
3. How SQL*Loader will be configured (memory management, rejecting records,interrupted load handling, and so on) as it loads the data
4. How SQL*Loader will manipulate the data being loaded

```
-- This is a sample control file
LOAD DATA
INFILE 'sample.dat'
BADFILE 'sample.bad'
DISCARDFILE 'sample.dsc'
APPEND
INTO TABLE emp
WHEN (57) = '.'
TRAILING NULLCOLS
(hiredate SYSDATE,
  deptno POSITION(1:2)  INTEGER EXTERNAL(2)
         NULLIF deptno=BLANKS,
   job    POSITION(7:14)  CHAR  TERMINATED BY WHITESPACE
          NULLIF job=BLANKS  "UPPER(:job)",
   mgr    POSITION(28:31) INTEGER EXTERNAL
          TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
   ename  POSITION(34:41) CHAR
          TERMINATED BY WHITESPACE  "UPPER(:ename)",
   empno  POSITION(45) INTEGER EXTERNAL
          TERMINATED BY WHITESPACE,
   sal    POSITION(51) CHAR  TERMINATED BY WHITESPACE
          "TO_NUMBER(:sal,'$99,999.99')",
   comm   INTEGER EXTERNAL  ENCLOSED BY '(' AND '%'
          ":comm * 100"
)
```

1.Comments in the Control File

- Comments can appear anywhere in the command section of the file, but they should not appear within the data. Precede any comment with two hyphens, for example:

```
--This is a comment
```

2. The `LOAD DATA` statement tells SQL*Loader that this is the beginning of a new data load.

3. The `INFILE` clause specifies the name of a datafile containing data that you want to load.

4. The `BADFILE` parameter specifies the name of a file into which rejected records are placed.

5. The `DISCARDFILE` parameter specifies the name of a file into which discarded records are placed.

6. The `APPEND` parameter is one of the options you can use when loading data into a table that is not empty.
7. The `INTO TABLE` clause allows you to identify tables, fields, and datatypes. Itdefines the relationship between records in the datafile and tables in the database.
8. The `WHEN` clause specifies one or more field conditions, based upon which SQL*Loader decides whether or not to load the data.

- For example, the following clause indicates that any record with the value "q" in the fifth column position should be loaded:

```
WHEN (5) = ’q’
```

- A `WHEN` clause can contain several comparisons, provided each is preceded by `AND.`Parentheses are optional, but should be used for clarity with multiple comparisons joined by `AND,` for example:

```
WHEN (deptno = ’10’) AND (job = ’SALES’)
```

9. The `TRAILING NULLCOLS` clause tells SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.

- For example, consider the following data:

```
10 Accounting
```

- Assume that the preceding data is read with the following control file and the record ends after `dname`:

```
INTO TABLE dept
TRAILING NULLCOLS
( deptno CHAR TERMINATED BY " ",
dname CHAR TERMINATED BY WHITESPACE,
loc CHAR TERMINATED BY WHITESPACE
)
```

In this case, the remaining `loc` field is set to null. Without the `TRAILING NULLCOLS` clause, an error would be generated due to missing data.

10. The remainder of the control file contains the field list, which provides information about column formats in the table being loaded.

- `POSITION` specifies the position of a data field.

  For example:

  ```
  ename POSITION (1:20) CHAR
  empno POSITION (22-26) INTEGER EXTERNAL
  allow POSITION (*+2) INTEGER EXTERNAL
  TERMINATED BY "/"
  ```

- Column `ename` is character data in positions 1 through 20, followed by column `empno`, which is presumably numeric data in columns 22 through 26. Column `allow` is offset from the end of `empno` by +2. Therefore, it starts in column 29 and continues until a slash is encountered.

Specifying the Datatype of a Data Field

- The datatype specification of a field tells SQL*Loader how to interpret the data in the field.
- For example, a datatype of `INTEGER` specifies binary data, while `INTEGER EXTERNAL` specifies character data that represents a number.
- A `CHAR` field can contain any character data.
- Only *one* datatype can be specified for each field; if a datatype is not specified, `CHAR` is assumed.

- If the record satisfies the `WHEN` clauses for the table, or no `WHEN` clauses are specified, SQL*Loader checks each field for a `NULLIF` clause.
- If a `NULLIF` clause exists, SQL*Loader evaluates it.
- If the `NULLIF` clause is satisfied, SQL*Loader sets the field to `NULL`.
- If the `NULLIF` clause is not satisfied, or if there is no `NULLIF` clause,
- SQL*Loader checks the length of the field from field evaluation. If the field has a length of 0 from field evaluation (for example, it was a null field, or whitespace trimming resulted in a null field), SQL*Loader sets the field to `NULL`.

- The `TERMINATED BY WHITESPACE` clause is one of the delimiters it is possible to specify for a field.
- If `TERMINATED BY WHITESPACE` is specified, data is read until the first occurrence of a whitespace character (spaces, tabs, blanks, line feeds, form feeds, or carriage returns).
- Then the current position is advanced until no more adjacent whitespace characters are found. This allows field values to be delimited by varying amounts of whitespace.

- The `ENCLOSED BY` clause is another possible field delimiter.
- If a field is enclosed, or terminated and enclosed, then any whitespace outside the enclosure delimiters is not part of the field.
- Any whitespace between the enclosure delimiters belongs to the field,whether it is leading or trailing whitespace.