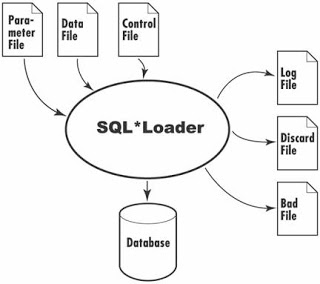
SQL\*Loader (***sqlldr***) is, the utility, to use for high performance data loads, which has a powerful data parsing engine which puts little limitation on the format of the data in the datafile. The data can be loaded from any flat file and inserted into the Oracle database.  
  
SQL\*Loader is a bulk loader utility used for moving data from external files into the Oracle database. SQL\*Loader supports various load formats, selective loading, and multi-table loads.  
  
SQL\*Loader reads data file(s) and description of the data which is defined in the control file. Using this information and any additional specified parameters (either on the command line or in the PARFILE), SQL\*Loader loads the data into the database.  
  
During processing, SQL\*Loader writes messages to the log file, bad rows to the bad file, and discarded rows to the discard file.  
  
[](http://3.bp.blogspot.com/_f48bGre1-7k/Si0Rl41MqYI/AAAAAAAAI7M/E0MkBeM3oG4/s1600-h/clip_image001.jpg)

**The Control File**  
The SQL\*Loader control file, is a flat file or text file, contains information that describes how the data will be loaded. It contains the table name, column data types, field delimiters, bad file name, discard file name, conditions to load, SQL functions to be applied and may contain data or infile name.  
  
Control file can have three sections:  
  
1. The first section contains session-wide information.  
e.g.: \* global options such as bindsize, rows, records to skip, etc.  
\* INFILE clauses to specify where the input data is located  
\* data character set specification  
2. 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.  
3. The third section is optional and, if present, contains input data.  
  
Some control file syntax considerations are:  
\* The syntax is free-format (statements can extend over multiple lines).  
\* It is case insensitive, however, strings enclosed in single or double quotation marks are taken literally, including case.  
\* In control file syntax, comments extend from the two hyphens (--), which mark the beginning of the comment, to the end of the line. Note that the optional third section of the control file is interpreted as data rather than as control file syntax; consequently, comments in this section are not supported.  
\* Certain words have special meaning to SQL\*Loader and are therefore reserved. If a particular literal or a database object name (column name, table name, etc.) is also a reserved word (keyword), it must be enclosed in single or double quotation marks.  
  
Options in SQL\*Loader while loading the data.  
(a) **INSERT**: Specifies that you are loading into an empty table. SQL\*Loader will abort the load if the table contains data to start with. *This is the default*.  
(b) **APPEND**: If we want to load the data into a table which is already containing some rows.  
(c) **REPLACE**: Specifies that, we want to replace the data in the table before loading. Will 'DELETE' all the existing records and replace them with new.  
(d) **TRUNCATE**: This is same as 'REPLACE', but SQL\*Loader will use the 'TRUNCATE' command instead of 'DELETE' command.  
  
This sample control file will load an external data file containing delimited data:

|  |
| --- |
| load data infile 'c:\data\emp.csv' into table emp //here INSERT is default fields terminated by "," optionally enclosed by '"' (empno, empname, sal, deptno) |

The emp.csv file may look like this:

|  |
| --- |
| 10001,"Scott Tiger", 1000, 40 10002,"Frank Naude", 5000, 20 |

Another Sample control file with in-line data formatted as fix length records.

|  |
| --- |
| load data infile \* replace into table departments (dept position (02:05) char(4), deptname position (08:27) char(20) ) begindata COSC COMPUTER SCIENCE ENGL ENGLISH LITERATURE MATH MATHEMATICS POLY POLITICAL SCIENCE |

"infile \*" means, the data is within the control file; otherwise we’ve to specify the file name and location. The trick is to specify "\*" as the name of the data file, and use BEGINDATA to start the data section in the control file:  
  
**Loading variable length(delimited) data**  
  
In the first example we will see how delimited (variable length) data can be loaded into Oracle.  
Example 1:

|  |
| --- |
| LOAD DATA INFILE \* CONTINUEIF THIS (1) = '\*' INTO TABLE delimited\_data FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS (data1 "UPPER(:data1)", data2 "TRIM(:data2)" data3 "DECODE(:data2, 'hello', 'goodbye', :data1)" ) BEGINDATA \*11111,AAAAAAAAAA,hello \*22222,"A,B,C,D,",Testttt |

**NOTE:**The default data type in SQL\*Loader is CHAR(255). To load character fields longer than 255 characters, code the type and length in your control file. By doing this, Oracle will allocate a bigger buffer to hold the entire column, thus eliminating potential "Field in data file exceeds maximum length" errors.  
e.g.:  
...  
resume char(4000),  
...  
  
Example 2:

|  |
| --- |
| LOAD DATA INFILE 'table.dat' INTO TABLE ‘table-name’ FIELDS TERMINATED BY ',' TRAILING NULLCOLS ( COL1 DECIMAL EXTERNAL NULLIF (COL1=BLANKS), COL2 DECIMAL EXTERNAL NULLIF (COL2=BLANKS), COL3 CHAR NULLIF (COL3=BLANKS), COL4 CHAR NULLIF (COL4=BLANKS), COL5 CHAR NULLIF (COL5=BLANKS), COL6 DATE "MM-DD-YYYY" NULLIF (COL6=BLANKS) ) |

Numeric data should be specified as type ‘external’, otherwise, it is read as characters rather than as binary data.

Decimal numbers need not contain a decimal point; they are assumed to be integers if not specified.

The standard format for date fields is DD-MON-YY.

**Loading fixed length (positional) data**  
  
The control file can also specify that records are in fixed format. A file is in fixed record format when all records in a datafile are the same length. The control file specifies the specific starting and ending byte location of each field. This format is harder to create and less flexible but can yield performance benefits. A control file specifying a fixed format could look like the following.  
Example 1:

|  |
| --- |
| LOAD DATA INFILE \* INTO TABLE positional\_data (data1 POSITION(1:5), data2 POSITION(6:15) ) BEGINDATA 11111AAAAAAAAAA 22222BBBBBBBBBB |

In the above example, position(01:05) will give the 1st to the 5th character (11111 and 22222).  
  
Example 2:

|  |
| --- |
| LOAD DATA INFILE 'table.dat' INTO TABLE ‘table-name’ ( COL1 POSITION(1:4) INTEGER EXTERNAL, COL2 POSITION(6:9) INTEGER EXTERNAL, COL3 POSITION(11:46) CHAR, COL4 POSITION(48:83) CHAR, COL5 POSITION(85:120) CHAR, COL6 POSITION(122:130) DATE "MMDDYYYY" ) |

**SQL\*Loader Options**  
  
Usage: sqlldr keyword=value [keyword=value ...]

* Data File

* Flexible

* Download Pdf

* Samples

* Social networking site

* Facebook

* Google+

Invoke the utility without arguments to get a list of available parameters.  
$ sqlldr  
  
To check which options are available in any release of SQL\*Loader uses this command:  
$ sqlldr help=y  
  
Look at the following example:  
$ sqlldr username@server/password control=loader.ctl  
$ sqlldr username/password@server control=loader.ctl  
  
SQL\*Loader provides the following options, which can be specified either on the command line or within a parameter file:  
  
**userid** – The Oracle username and password.  
  
**control** – The name of the control file. This file specifies the format of the data to be loaded.  
  
**log** – The name of the file used by SQL\*Loader to log results. The log file contains information about the SQL\*Loader execution. It should be viewed after each SQL\*Loader job is completed. Especially interesting is the summary information at the bottom of the log, including CPU time and elapsed time. It has details like no. of lines readed, no. of lines loaded, no. of rejected lines (full data will be in discard file), no. of bad lines, actual time taken load the data.  
  
**bad** – A file that is created when at least one record from the input file is rejected. The rejected data records are placed in this file. A record could be rejected for many reasons, including a non-unique key or a required column being null.  
  
**data**– The name of the file that contains the data to load.  
  
**discard**– The name of the file that contains the discarded rows. Discarded rows are those that fail the WHEN clause condition when selectively loading records.  
  
**discardmax** – [ALL] The maximum number of discards to allow.  
  
**skip** – [0] Allows the skipping of the specified number of logical records.  
  
**load** – [ALL] The number of logical records to load.  
  
**errors** – [50] The number of errors to allow on the load. SQL\*Loader will tolerates this many errors (50 by default). After this limit, it'll abort the loading and rollbacks the already inserted records.  
  
**rows**– [64] The number of rows to load before a commit is issued (in conventional path).  
[ALL] For direct path loads, rows are the number of rows to read from the data file before saving the data in the datafiles. Committing less frequently (higher value of ROWS) will improve the performance of SQL\*Loader.  
  
**bindsize** – [256000] The size of the conventional path bind array in bytes. Larger bindsize will improve the performance of SQL\*Loader.  
  
**silent** – Suppress messages/errors during data load. A value of ALL will suppress all load messages. Other options include DISCARDS, ERRORS, FEEDBACK, HEADER, and [PARTITIONS](http://satya-dba.blogspot.com/2009/07/partitioning-in-oracle.html).  
  
**direct** – [FALSE] Specifies whether or not to use a direct path load or conventional. Direct path load (DIRECT=TRUE) will load faster than conventional.  
  
**\_synchro** – internal testing.  
  
**parfile** – [Y] The name of the file that contains the parameter options for SQL\*Loader.  
  
**parallel** – [FALSE] do parallel load. Available with direct path data loads only, this option allows multiple SQL\*Loader jobs to execute concurrently and will improve the performance.  
  
**file** – Used only with parallel loads, this parameter specifies the file to allocate extents from. Specify a filename that contains index creation statements.  
  
**skip\_unusable\_indexes** – [FALSE] Determines whether SQL\*Loader skips the building of indexes or [index partitions](http://satya-dba.blogspot.com/2009/07/partitioning-in-oracle.html)that are in an unusable state.  
  
**skip\_index\_maintenance** – [FALSE] Stops index maintenance for direct path loads only. Do not maintain indexes, mark affected indexes as unusable.  
  
**commit\_discontinued** – [FALSE] commit loaded rows when load is discontinued. *This is from 10g.*  
  
**\_display\_exitcode** – Display exit code for SQL\*Loader execution.  
  
**readsize** – [1048576] The size of the read buffer used by SQL\*Loader when reading data from the input file. This value should match that of bindsize.  
  
**external\_table** – [NOT\_USED] Determines whether or not any data will be loaded using external tables. The other valid options include GENERATE\_ONLY and EXECUTE.  
  
**columnarrayrows**– [5000] Specifies the number of rows to allocate for direct path column arrays.  
  
**streamsize** – [256000] Specifies the size of direct path stream buffer size in bytes.  
  
**multithreading** – use multithreading in direct path. The default is TRUE on multiple CPU systems and FALSE on single CPU systems.  
  
**resumable** – [FALSE] Enables and disables resumable space allocation. When “TRUE”, the parameters resumable\_name and resumable\_timeout are utilized.  
  
**resumable\_name** – User defined string that helps identify a resumable statement that has been suspended. This parameter is ignored unless resumable = TRUE.  
  
**resumable\_timeout** – [7200 seconds] The time period in which an error must be fixed. This parameter is ignored unless resumable = TRUE.  
  
**date\_cache** – [1000] Size (in entries) of date conversion cache.  
  
**no\_index\_errors** - [FALSE] abort load on any index errors (This is from [Oracle 11g release2](http://satya-dba.blogspot.com/2009/09/whats-new-in-11g-release-2.html)).  
  
**\_testing\_ncs\_to\_clob** – test non character scalar to character lob conversion. *This is from Oracle 10g.*  
  
**\_parallel\_lob\_load** – allow direct path parallel load of lobs. *This is from Oracle 10g.*  
  
**\_trace\_events** – Enable tracing during run by specifying events and levels (SQLLDR\_LOWEST,...). *This is from Oracle 11g.*  
  
**\_testing\_server\_slot\_size** – test with non default direct path server slot buffer size. *This is from Oracle 11g.*  
  
**\_testing\_server\_ca\_rows** – test with non default direct path server column array rows. *This is from Oracle 11g.*  
  
**\_testing\_server\_max\_rp\_ccnt** – test with non default direct path max row piece columns. *This is from Oracle 11g.*

* Free Web

* Mathematics

* Locations

* Data File

* Flexible

* Download Pdf

* Samples

*Note: values within parenthesis are the default values.*

PLEASE NOTE: Command-line parameters may be specified either by position or by keywords. An example of the former case is 'sqlldr scott/tiger foo'; an example of the latter is 'sqlldr control=foo userid=scott/tiger'. One may specify parameters by position before but not after parameters specified by keywords.  
For example, 'sqlldr scott/tiger control=foo logfile=log' is allowed, but 'sqlldr scott/tiger control=foo log' is not, even though the position of the parameter 'log' is correct.  
  
**Miscellaneous**  
  
**1.** To load MS-Excel data into Oracle, Open the MS-Excel spreadsheet and save it as a CSV (Comma Separated Values) file. This file can now be copied to the Oracle machine and loaded using the SQL\*Loader utility.  
  
**2.** Oracle does not supply any data unload utilities (like SQL\*Unloader), to get the data from database. You can use SQL\*Plus to select and format your data and then spool it to a file or you have to use any third party tool.  
  
**3. Skipping header records while loading**  
We can skip unwanted header records or continue an interrupted load (e.g. run out of space) by specifying the "SKIP=n" keyword. "n" specifies the number of logical rows to skip. Look at this example:

|  |
| --- |
| OPTIONS (SKIP=5) LOAD DATA INFILE \* INTO TABLE load\_positional\_data (data1 POSITION(1:5), data2 POSITION(6:15) ) BEGINDATA 11111AAAAAAAAAA 22222BBBBBBBBBB ... |

$ sqlldr userid=id/passwd control=control\_file\_name.ctl skip=4  
  
If you are continuing a multiple table direct path load, you may need to use the CONTINUE\_LOAD clause instead of the SKIP parameter. CONTINUE\_LOAD allows you to specify a different number of rows to skip for each of the tables you are loading.  
  
**4. Modifying data as the database gets loaded**  
Data can be modified as it loads into the Oracle Database. One can also populate columns with static or derived values. However, this only applies for the conventional load path (and not for direct path loads). Here are some examples:

|  |
| --- |
| LOAD DATA INFILE \* INTO TABLE modified\_data (rec\_no "my\_db\_sequence.nextval", region CONSTANT '31', time\_loaded "to\_char(SYSDATE, 'HH24:MI')", data1 POSITION(1:5) ":data1/100", data2 POSITION(6:15) "upper(:data2)", data3 POSITION(16:22)"to\_date(:data3, 'YYMMDD')" ) BEGINDATA 11111AAAAAAAAAA991201 22222BBBBBBBBBB990112 |

|  |
| --- |
| LOAD DATA INFILE 'mail\_orders.txt' BADFILE 'bad\_orders.txt' APPEND INTO TABLE mailing\_list FIELDS TERMINATED BY "," (addr, city, state, zipcode, mailing\_addr "decode(:mailing\_addr,null, :addr, :mailing\_addr)", mailing\_city "decode(:mailing\_city,null, :city, :mailing\_city)", mailing\_state, move\_date "substr(:move\_date, 3, 2) || substr(:move\_date, 7, 2)" ) |

**5. Loading from multiple input files**  
One can load from multiple input files provided they use the same record format by repeating the INFILE clause. Here is an example:

|  |
| --- |
| LOAD DATA INFILE file1.dat INFILE file2.dat INFILE file3.dat APPEND INTO TABLE emp (empno POSITION(1:4) INTEGER EXTERNAL, ename POSITION(6:15) CHAR, deptno POSITION(17:18) CHAR, mgr POSITION(20:23) INTEGER EXTERNAL ) |

**6. Loading into multiple tables**  
One can also specify multiple "INTO TABLE" clauses in the SQL\*Loader control file to load into multiple tables. Look at the following example:

|  |
| --- |
| LOAD DATA INFILE \* INTO TABLE tab1 WHEN tab = 'tab1' (tab FILLER CHAR(4), col1 INTEGER ) INTO TABLE tab2 WHEN tab = 'tab2' (tab FILLER POSITION(1:4), col1 INTEGER ) BEGINDATA tab1|1 tab1|2 tab2|2 tab3|3 |

The "tab" field is marked as FILLER as we don't want to load it.  
Note the use of "POSITION" on the second routing value (tab = 'tab2'). By default field scanning doesn't start over from the beginning of the record for new INTO TABLE clauses. Instead, scanning continues where it left off. POSITION is needed to reset the pointer to the beginning of the record again.  
  
Another example:

|  |
| --- |
| LOAD DATA INFILE 'mydata.dat' REPLACE INTO TABLE emp WHEN empno != ' ' (empno POSITION(1:4) INTEGER EXTERNAL, ename POSITION(6:15) CHAR, deptno POSITION(17:18) CHAR, mgr POSITION(20:23) INTEGER EXTERNAL ) INTO TABLE proj WHEN projno != ' ' (projno POSITION(25:27) INTEGER EXTERNAL, empno POSITION(1:4) INTEGER EXTERNAL ) |

**7.** In SQL\*Loader, one cannot COMMIT only at the end of the load file, but by setting the ROWS parameter to a large value, committing can be reduced. Make sure you have big [rollback segments](http://satya-dba.blogspot.com/2009/08/rollback-segments-in-oracle.html) ready when you use a high value for ROWS.  
  
**8. Selectively loading filtered records**  
Look at this example, (01) is the first character, (30:37) are characters 30 to 37:

|  |
| --- |
| LOAD DATA INFILE 'mydata.dat' BADFILE 'mydata.bad' DISCARDFILE 'mydata.dis' APPEND INTO TABLE my\_selective\_table WHEN (01) <> 'H' and (01) <> 'T' and (30:37) = '20031217' ( region CONSTANT '31', service\_key POSITION(01:11) INTEGER EXTERNAL, call\_b\_no POSITION(12:29) CHAR ) |

**NOTE:**SQL\*Loader does not allow the use of OR in the WHEN clause. You can only use AND as in the example above! To workaround this problem, code multiple "INTO TABLE ... WHEN" clauses. Here is an example:

|  |
| --- |
| LOAD DATA INFILE 'mydata.dat' BADFILE 'mydata.bad' DISCARDFILE 'mydata.dis' APPEND INTO TABLE my\_selective\_table WHEN (01) <> 'H' and (01) <> 'T' ( region CONSTANT '31', service\_key POSITION(01:11) INTEGER EXTERNAL, call\_b\_no POSITION(12:29) CHAR ) INTO TABLE my\_selective\_table WHEN (30:37) = '20031217' ( region CONSTANT '31', service\_key POSITION(01:11) INTEGER EXTERNAL, call\_b\_no POSITION(12:29) CHAR ) |

**9. Skipping certain columns while loading data**  
One cannot use POSITION(x:y) with delimited data. Luckily, from Oracle 8i one can specify FILLER columns. FILLER columns are used to skip columns/fields in the load file, ignoring fields that one does not want. Look at this example:

|  |
| --- |
| LOAD DATA TRUNCATE INTO TABLE T1 FIELDS TERMINATED BY ',' (field1, field2 FILLER, field3 ) |

BOUNDFILLER (available with Oracle 9i and above) can be used if the skipped column's value will be required later again. Here is an example:

|  |
| --- |
| LOAD DATA INFILE \* TRUNCATE INTO TABLE sometable FIELDS TERMINATED BY "," trailing nullcols ( c1, field2 BOUNDFILLER, field3 BOUNDFILLER, field4 BOUNDFILLER, field5 BOUNDFILLER, c2 ":field2 || :field3", c3 ":field4 + :field5" ) |

**10. Loading images, sound clips and documents**  
SQL\*Loader can load data from a "primary data file", SDF (Secondary Data file - for loading nested tables and VARRAYs) or LOBFILE. The LOBFILE method provides an easy way to load documents, photos, images and audio clips into BLOB and CLOB columns.  
Given the following table:  
  
CREATE TABLE image\_table (  
image\_id NUMBER(5),  
file\_name VARCHAR2(30),  
image\_data BLOB);  
  
Control File:

|  |
| --- |
| LOAD DATA INFILE \* INTO TABLE image\_table REPLACE FIELDS TERMINATED BY ',' ( image\_id INTEGER(5), file\_name CHAR(30), image\_data LOBFILE (file\_name) TERMINATED BY EOF ) BEGINDATA 001,image1.gif 002,image2.jpg 003,image3.bmp |

**11. Loading EBCDIC data**  
SQL\*Loader is character set aware (you can specify the character set of the data). Specify the character set WE8EBCDIC500 for the EBCDIC data. The following example shows the SQL\*Loader controlfile to load a fixed length EBCDIC record into the Oracle Database:

|  |
| --- |
| LOAD DATA CHARACTERSET WE8EBCDIC500 INFILE data.ebc "fix 86 buffers 1024" BADFILE 'data.bad' DISCARDFILE 'data.dsc' REPLACE INTO TABLE temp\_data ( field1 POSITION (1:4) INTEGER EXTERNAL, field2 POSITION (5:6) INTEGER EXTERNAL, field3 POSITION (7:12) INTEGER EXTERNAL, field4 POSITION (13:42) CHAR, field5 POSITION (43:72) CHAR, field6 POSITION (73:73) INTEGER EXTERNAL ) |

**12. Reading multiple rows per record**

If the data is in fixed format the number of rows of data to be read for each record can be specified using the concatenate clause,

e.g.: *concatenate 3*

reads 3 rows of data for every record. The data rows are literally concatenated together so that positions 81 to 160 are used to specify column positions for data in the second row (assuming that the record length of the file is 80). You should also specify the record length (240 in this case) with a reclen clause when concatenating data, that is:

*reclen 240*

The continueif clause may specify more than one character.

e.g.: *continueif this (1:4) = 'two'*

specifies that if the first four characters of the current line are 'two', the next line is a continuation of this line. In this case the first four columns of each record are assumed to contain only a continuation indicator and are not read as data.

When using fixed format data, the continuation character may be in the last column of the data record. For example:

*continueif last = '+'*

specifies that if the last non-blank character in the line is '+', the next line is a continuation of the current line. This method does not work with free format data because the continuation character is read as the value of the next field.

**Common Errors** in SQL\*Loader are  
(i) Foreign key is not found.  
(ii) Length of the column in the infile/datafile may be bigger than the target table column size.  
(iii) Mismatching of datatypes.  
  
**How to improve SQL\*Loader Performance**  
  
SQL\*Loader is flexible and offers many options that should be considered to maximize the speed of data loads.  
  
**1. Use Direct Path Loads** - The conventional path loader essentially loads the data by using standard insert statements. Direct path load builds blocks of data in memory and saves these blocks directly into the extents allocated for the table being loaded. The direct path loader (DIRECT=TRUE) loads directly into the Oracle datafiles and creates blocks in Oracle database block format. This will effectively bypass most of the RDBMS processing. The fact that SQL is not being issued makes the entire process much less taxing on the database. There are certain cases, in which direct path loads cannot be used (clustered tables).

To prepare the database for direct path loads, the script $ORACLE\_HOME/rdbms/admin/catldr.sql must be executed (no need to run this, if you ran catalog.sql at the time of database creation).  
  
**Differences between direct path load and conventional path load**

|  |  |
| --- | --- |
| **Direct Path Load** | **Conventional Path Load** |
| The direct path loader (DIRECT=TRUE) bypasses much of the overhead involved, and loads directly into the Oracle datafiles | The conventional path loader essentially loads the data by using standard INSERT statements |
| Loads directly to datafile | Loads via buffers |
| No redolog | Redolog will be generated |
| Will not use Oracle instance and RAM | Will use Oracle instance and RAM |
| Can't disable constraints and indexes | Can disable constraints and indexes |
| --- | Default buffer size is 256KB(bindsize) or 64 rows |
| Can do parallel load | Can't do parallel load |
| Will use streams (default stream buffer size - 256K) | Will not use streams |
| Will use multithreading | Will not use multithreading |
| --- | Data can be modified as it loads into the Oracle Database, also populate columns with static or derived values |
| Can use UNRECOVERABLE option | Can't use UNRECOVERABLE option |

Several restrictions associated with direct path loading:

• Tables and associated indexes will be locked during load.

• SQL\*Net access is available only under limited circumstances, and its use will slow performance.

• Clustered tables cannot be loaded.

• Loaded data will not be replicated.

• Constraints that depend on other tables are disabled during load and applied when the load is completed.

• SQL functions are not available. Cannot always use SQL strings for column processing in the control file (something like this will probably fail: col1 date "ddmonyyyy" "substr(:period,1,9)").

**2. External Table Load**- An External table load creates an external table for data in a datafile and executes INSERT statements to insert the data from datafile into target table.  
  
Advantages over conventional & direct:  
(i) An external table load attempts to load datafiles in parallel, if datafile is big enough.  
(ii) An external table load allows modification of the data being loaded by using SQL and PL/SQL functions as part of the insert statement that is used to create external table.  
  
**3. Disable Indexes and Constraints** - For only conventional data loads, the disabling of indexes and constraints can greatly enhance the performance of SQL\*Loader. This will significantly slow down load times even with ROWS set to a high value.  
  
**4. Use a Larger Bind Array**. For conventional data loads only, larger bind arrays limit the number of calls to the database and increase performance. The size of the bind array is specified using the bindsize parameter. The bind array's size is equivalent to the number of rows it contains (rows=) times the maximum length of each row.  
  
**5. Use ROWS=n** to commit less frequently. For conventional data loads only, the rows parameter specifies the number of rows per commit. Issuing fewer commits will enhance performance.  
  
**6. Use Parallel Loads.** Available with direct path data loads only, this option allows multiple SQL\*Loader jobs to execute concurrently.  
$ sqlldr control=first.ctl parallel=true direct=true  
$ sqlldr control=second.ctl parallel=true direct=true  
  
**7. Use Fixed Width Data**. Fixed width data format saves Oracle some processing when parsing the data. The savings can be tremendous, depending on the type of data and number of rows.  
  
**8. Disable Archiving During Load.**While this may not be feasible in certain environments, disabling database archiving can increase performance considerably.  
  
**9. Use unrecoverable.**The UNRECOVERABLE option (unrecoverable load data) disables the writing of the data to the redo logs. This option is available for direct path loads only.  
  
**Benchmarking**  
The following benchmark tests were performed with the various SQL\*Loader options. The table was truncated after each test.

|  |  |  |
| --- | --- | --- |
| **SQL\*Loader Option** | **Elapsed Time(Seconds)** | **Time Reduction** |
| direct=false rows=64 | 135 | - |
| direct=false bindsize=512000 rows=10000 | 92 | 32% |
| direct=false bindsize=512000 rows=10000 database in noarchivelog mode | 85 | 37% |
| direct=true | 47 | 65% |
| direct=true unrecoverable | 41 | 70% |
| direct=true unrecoverable fixed width data | 41 | 70% |

The results above indicate that conventional path loads take the longest. However, the bindsize and rows parameters can aid the performance under these loads. The test involving the conventional load didn’t come close to the performance of the direct path load with the unrecoverable option specified.  
  
It is also worth noting that the fastest import time achieved (earlier) was 67 seconds, compared to 41 for SQL\*Loader direct path – a 39% reduction in execution time. This proves that SQL\*Loader can load the same data faster than import.  
  
These tests did not compensate for indexes. All database load operations will execute faster when indexes are disabled.