SQL*Loader FAQ

From Oracle FAQ

SQL*Loader FAQ:

Contents

What is SQL*Loader and what is it used for?

SQL*Loader is a bulk loader utility used for moving data from external files into the Oracle database. Its syntax is similar to that of the DB2 load utility, but comes with more options. SQL*Loader supports various load formats, selective loading, and multi-table loads.

SQL*Loader (sqlldr) is the utility to use for high performance data loads. The data can be loaded from any text file and inserted into the database.

How does one use the SQL*Loader utility?

One can load data into an Oracle database by using the **sqlldr** (sqlload on some platforms) utility. Invoke the utility without arguments to get a list of available parameters. Look at the following example:

```
sqlldr username@server/password control=loader.ctl
sqlldr username/password@server control=loader.ctl
```

This sample control file (loader.ctl) will load an external data file containing delimited data:

```
load data
 infile 'c:\data\mydata.csv'
 into table emp
 fields terminated by "," optionally enclosed by '"'
 ( empno, empname, sal, deptno )
```

The mydata.csv file may look like this:

```
10001,"Scott Tiger", 1000, 40
10002,"Frank Naude", 500, 20
```

Optionally, you can work with tabulation delimited files by using one of the following syntaxes:

```
fields terminated by "\t"
fields terminated by X'09'
```

Additionally, if your file was in Unicode, you could make the following addition.

```
load data
 CHARACTERSET UTF16
 infile 'c:\data\mydata.csv'
 into table emp
 fields terminated by "," optionally enclosed by '"'
 ( empno, empname, sal, deptno )
```

Another Sample control file with in-line data formatted as fix length records. The trick is to specify "*" as the name of the data file, and use BEGINDATA to start the data section in the control file:

```
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
```

How does one 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.

Possible problems and workarounds:

The spreadsheet may contain cells with newline characters (ALT+ENTER). SQL*Loader expects the entire record to be on a single line. Run the following macro to remove newline characters (Tools -> Macro -> Visual Basic Editor):

```
' Removing tabs and carriage returns from worksheet cells
Sub CleanUp()
 Dim TheCell As Range
 On Error Resume Next

 For Each TheCell In ActiveSheet.UsedRange
   With TheCell
     If .HasFormula = False Then
       .Value = Application.WorksheetFunction.Clean(.Value)
     End If
   End With
 Next TheCell
End Sub
```

Tools:

If you need a utility to load Excel data into Oracle, download *quickload* from sourceforge at http://sourceforge.net/projects/quickload

Is there a SQL*Unloader to download data to a flat file?

Oracle does not supply any data unload utilities. Here are some workarounds:

Using SQL*Plus

You can use SQL*Plus to select and format your data and then spool it to a file. This example spools out a CSV (comma separated values) file that can be imported into MS-Excel:

```
set echo off newpage 0 space 0 pagesize 0 feed off head off trimspool on
spool oradata.txt
```

```
select col1 || ',' || col2 || ',' || col3
  from tab1
 where col2 = 'XYZ';
spool off
```

Warning: if your data contains a comma, choose another separator that is not in the data. You can also enclose the column that contains the comma between ".

You can also use the "set colsep" command if you don't want to put the commas in by hand. This saves a lot of typing. Example:

```
set colsep ','
set echo off newpage 0 space 0 pagesize 0 feed off head off trimspool on
spool oradata.txt
select col1, col2, col3
  from tab1
 where col2 = 'XYZ';
spool off
```

Using PL/SQL

PL/SQL's UTL_FILE package can also be used to unload data. Example:

```
declare
  fp utl_file.file_type;
begin
  fp := utl_file.fopen('c:\oradata','tab1.txt','w');
  utl_file.putf(fp, '%s, %sn', 'TextField', 55);
  utl_file.fclose(fp);
end;
/
```

Using Oracle SQL Developer

The freely downloadable Oracle SQL Developer (http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html) application is capable of exporting data from Oracle tables in numerous formats, like Excel, SQL insert statements, SQL loader format, HTML, XML, PDF, TEXT, Fixed text, etc.

It can also import data from Excel (.xls), CSV (.csv), Text (.tsv) and DSV (.dsv) formats directly into a database.

Third-party programs

You might also want to investigate third party tools to help you unload data from Oracle. Here are some examples:

- WisdomForce FastReader - http://www.wisdomforce.com
- IxUnload from ixionsoftware.com - http://www.ixionsoftware.com/products/
- FAst extraCT (FACT) for Oracle from CoSort - http://www.cosort.com/products/FACT
- Unicenter (also ManageIT or Platinum) Fast Unload for Oracle from CA
- Keeptool's Hora unload/load facility (part v5 to v6 upgrade) can export to formats such as Microsoft Excel, DBF, XML, and text.
- TOAD from Quest
- SQLWays from Ispirer Systems
- PL/SQL Developer from allroundautomation

Can one load variable and fixed length data records?

Loading delimited (variable length) data

In the first example we will show how delimited (variable length) data can be loaded into Oracle:

```
LOAD DATA
 INFILE *
 INTO TABLE load_delimited_data
 FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
 TRAILING NULLCOLS
 (  data1,
    data2
 )
BEGINDATA
11111,AAAAAAAAAA
22222,"A,B,C,D,"
```

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 big enough buffer to hold the entire column, thus eliminating potential "Field in data file exceeds maximum length" errors. Example:

```
...
resume char(4000),
...
```

Loading positional (fixed length) data

If you need to load positional data (fixed length), look at the following control file example:

```
LOAD DATA
  INFILE *
  INTO TABLE load_positional_data
  (  data1 POSITION(1:5),
     data2 POSITION(6:15)
  )
BEGINDATA
11111AAAAAAAAAA
22222BBBBBBBBBB
```

For example, *position(01:05)* will give the 1st to the 5th character (11111 and 22222).

Can one skip header records while loading?

One can skip unwanted header records or continue an interrupted load (for example if you run out of space) by specifying the "SKIP=n" keyword. "n" specifies the number of logical rows to skip. Look at these examples:

```
OPTIONS (SKIP=5)
LOAD DATA
INFILE *
INTO TABLE load_positional_data
(  data1 POSITION(1:5),
   data2 POSITION(6:15)
)
BEGINDATA
11111AAAAAAAAAA
22222BBBBBBBBBB
...
```

```
sqlldr userid=ora_id/ora_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.

Can one modify 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)"
 )
```

Can one load data from multiple files/ into multiple tables at once?

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
  )
```

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 a 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. In delimited formats, use "POSITION(1)" after the first column to reset the pointer.

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
 )
```

Can one selectively load only the records that one needs?

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
  )
```

Can one skip 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"
)
```

How does one load multi-line records?

One can create one logical record from multiple physical records using one of the following two clauses:

- CONCATENATE - use when SQL*Loader should combine the same number of physical records together to form one logical record.
- CONTINUEIF - use if a condition indicates that multiple records should be treated as one. Eg. by having a '#' character in column 1.

How does one load records with multi-line fields?

Using Stream Record format, you can define a record delimiter, so that you're allowed to have the default delimiter ('\n') in the field's content.

After the INFILE clause set the delimiter:

```
load data
infile "test.dat" "str '|\n'"
into test_table
fields terminated by ';' TRAILING NULLCOLS
(
  desc,
```

```
  txt
)
```

test.dat:

```
one line;hello dear world;|
two lines;Dear world,
hello!;|
```

Note that this doesn't seem to work with inline data (INFILE * and BEGINDATA).

How can one get SQL*Loader to COMMIT only at the end of the load file?

One cannot, but by setting the ROWS= parameter to a large value, committing can be reduced. Make sure you have big rollback segments ready when you use a high value for ROWS=.

Can one improve the performance of SQL*Loader?

- A very simple but easily overlooked hint is not to have any indexes and/or constraints (primary key) on your load tables during the load process. This will significantly slow down load times even with ROWS= set to a high value.
- Add the following option in the command line: DIRECT=TRUE. This will effectively bypass most of the RDBMS processing. However, there are cases when you can't use direct load. For details, refer to the FAQ about the differences between the conventional and direct path loader below.
- Turn off database logging by specifying the UNRECOVERABLE option. This option can only be used with direct data loads.
- Run multiple load jobs concurrently.

What is the difference between the conventional and direct path loader?

The conventional path loader essentially loads the data by using standard INSERT statements. The direct path loader (DIRECT=TRUE) bypasses much of the logic involved with that, and loads directly into the Oracle data files. More information about the restrictions of direct path loading can be obtained from the Oracle Server Utilities Guide.

Some of the restrictions with direct path loads are:

- Loaded data will not be replicated
- 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)"). Details are in Metalink Note:230120.1.

How does one use SQL*Loader to load 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. Look at this example:

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.jpg
```

How does one load EBCDIC 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,
 field7    POSITION (74:74)   INTEGER EXTERNAL,
 field8    POSITION (75:75)   INTEGER EXTERNAL,
 field9    POSITION (76:86)   INTEGER EXTERNAL
)
```