**SQL \* Loader**

In many cases, you will be faced with a need to do a bulk upload of databases from files generated from some third-party system. This is where SQL\* Loader comes into picture. The input files may be generated by anything, but as long as the layout conforms to something that the SQL \* Loader can understand, it will upload the data successfully.

You need to use a SQL \* Loader control file that can interpret the contents of the input datafiles.

The differences between DATA PUMP and SQL \* Loader is that :

Data Pump reads and writes files in an Oracle proprietary format. It is used to transfer data into and out of, or between Oracle databases. SQL \* Loader can read any file no matter what its source.

Data Pump is a server side utility. SQL \* Loader is a user-process that establishes a session against the database instance via a server process.

SQL \* Loader uses a number of files for it’s operations :

The INPUT DATAFILES are the source data that it will upload into the database.

The CONTROLFILE is a text file with directives that tell SQL \* LOADER how to interpret the contents of the input files, and what to do with the rows it extracts from them.

The LOGFILES summarize the success or failure of the job, and shows us the errors during a failure.

The BADFILE. Rows extracted from the input files may be rejected by the SQL \* Loader (because they do not conform to the format expected by the control file) or may be rejected by the database (for example, the insertion might violate some integrity constraint). In either case, they are written out to a Bad file.

The REJECTED/Discard FILE contains rows that are successfully extracted from the input file but rejected because they did not match some record selection criterion given in the control file

The SQL \* Loader Control file is a text file instructing the SQL \* Loader on how to process the input datafiles. It is possible to include the actual data to be loaded on the control file, but normally this is not done. Usually, you will create one control file, and reuse it on a regular basis with different input datafiles.

The formats that the SQL \* Loader understands are Fixed Record Format, Variable Record Format, and Stream Record Format.

Fixed record format is the most straightforward, as well as the fastest to process. Each row of data is a fixed length, terminated with a carriage return character. Within the rows, you use some character as a field delimiter.

In the scott schema, create the following table :

Create table empnames(first varchar2(15),last varchar2(15));

Create a control file c1.ctl in the D:\photos folder and insert the following into it

load data

infile mydata.txt "fix 15"

into table empnames

fields terminated by ','

(first,last)

This will read the file ‘mydata.txt’, parse it into rows 15 bytes long(the 15 bytes must include two for the carriage return row terminator and fixed 13 for the text/data), divide the rows into two fields, and insert them into the columns “first” and “last” of the table named “empnames”. Create the input file “mydata.txt” in the D:\Photos folder with the following contents :

Hemant,Pangam

Hetal1,Pangam

Saloni,Pangam

Jairaj,Sarafi

Mahive,Pangam

Please note that ‘mydata.txt’ is used in control file and we are not giving absolute path, but relative path. So control file **c1.ctl** and mydata.txt should be in same directory.

Go to command prompt and in the folder ‘d:\photos’, run the SQL LOADER :

D:\photos>sqlldr scott/abc control=c1.ctl

The table must be empty, or else give the append option before the into clause. The column names must be mentioned in the control file as (first,last), even if there are only two fields. If there are more than two fields in the target table, they all get null values for the rows inserted from the text file.

Variable record format files include an entry at the head of each row to state how long the rows is(including 2 for the newline character). The length of this entry is included in the controlfile. Create a control file c2.ctl in the folder D:\Photos as follows :

load data

infile mydata1.txt "var 3"

append into table empnames

fields terminated by ','

(first,last)

Create a text file ‘mydata1.txt’ in the folder D:\Photos as follows :

015Hemant,Pangam

014Hetal,Pangam

015Saloni,Pangam

014Jairaj,Saraf

013Mahi,Pangam

The length of each row is a 3-digit number, as specified in the controlfile, and must include two characters for the carriage return. Variable record format is not as fast as fixed format, but is faster then stream format.

‘Append ‘ keyword is used to add the retrieved rows to the non-empty table.

Go to command prompt and in the folder ‘d:\photos’, run the SQL LOADER :

D:\photos>sqlldr scott/tiger control=c2.ctl

The Stream Format requires SQL \* Loader to do the most work. It is the most flexible format but also the slowest to process. Rather than being given any information about the length of the rows, the SQL \* Loader must scan each row for the record terminator.

Create a control file c3.ctl in the folder D:\Photos as follows :

load data

infile mydata2.txt "str '\n'"

append into table empnames

fields terminated by ','

(first,last)

Create a text file ‘mydata2.txt’ in the folder D:\Photos as follows :

Hemant,Pangam

Hetal,Pangam

Saloni,Pangam

Jairaj,Saraf

Mahi,Pangam

Go to command prompt and in the folder ‘d:\photos’, run the SQL LOADER:

D:\photos>sqlldr scott/tiger control=c3.ctl

Also try c31.ctl as follows :

load data

infile mydata2.txt "str '\n'"

truncate into table empnames

fields terminated by ','

(first,last)

Consider the following table :

create table enames(first\_name varchar2(15), middle\_name varchar2(15), last\_name varchar2(15));

Consider the following Input Data file named namesdata.txt with exactly 15-charactered first name, middle name and last name respectively.

Hemant Jayawant Pangam

Hetal Hemant Pangam

Saloni Hemant Pangam

Jairaj Vinayak Saraf

Consider the following Control file c5.ctl :

Load Data

Infile 'namesdata.txt'

truncate

Into table enames

(first\_name position(1:15) char,

Middle\_name position(16:30) char,

Last\_name position(31:45) char)

Sqlldr userid=scott/tiger control=c5.ctl

In this example, data will be loaded from the file ‘namesdata.txt’ into the table ‘enames’. The ‘namesdata.txt’ file contains data for all the three columns of the ‘enames’ table. Here, the middle\_name column value always begins at the position 16 in the file, even if the first\_name value is less than 15 characters. Each of the columns is listed, along with the position where its data resides in each physical record in the file. This format allows you to load data even if the source data’s column order does not match the order of columns in your table

Also try the following c51.ctl :

Load Data

Infile 'namesdata.txt'

append

Into table enames

(first\_name position(1:15) char,

Middle\_name position(16:30) char,

Last\_name position(31:45) char)

If the columns in your input file have variable lengths, you can use SQL \* Loader commands to tell Oracle how to determine when a value ends.

Consider the following Input Data file named names.txt with first name, middle name and last name respectively, separated by commas

Hari,Prasad,Sharma

Shaun,Tait,Pollock

Sachin,Ramesh,Tendulkar

Kapil,Dev,Nikhanj

Consider the following Control file c6.ctl :

Load Data

Infile 'names.txt'

truncate

Into table enames

Fields Terminated By ","

(First\_name, Middle\_name, Last\_name)

Be sure to select a delimiter that is not present within the values being loaded. In this example, a comma is the delimiter, so any comma present within any text string being loaded will be interpreted as end-of-field character

Thus, the input file does not have to be 45-characters wide for each row, as was the case in the previous example.

Here, we also use the TRUNCATE clause, which will truncate the table enames before the start of the load. However, in case the table is already locked by some update or delete command, then truncate will give an error ORA-00054 saying that the resource is busy, and will end the operation of SQL \* Loader. You may use the REPLACE clause instead of TRUNCATE. In case of REPLACE, in case the table is already locked by some update or delete command, then REPLACE will make the operation of SQL \* LOADER to go into a wait mode. The APPEND clause, if used, will add rows to the table. The INSERT(default) clause if used, requires that the table be empty, or else it will give an error.

Load Data

Infile 'names.txt'

replace

Into table enames

Fields Terminated By ","

(First\_name, Middle\_name, Last\_name)

Consider the following table :

Create table myemp(ename varchar2(10), ecity varchar2(20), age number(3) constraint xyz check(age>=21));

Consider the following Input Data file named emps1.txt with name, city and age of employees respectively separated by commas

Rahul Dravid,Bangalore,29

Sachin,Mumbai,25

Virat Koli,Delhi,17

TP Singh,Chennai,19

Consider the following control file c7.ctl :

Load data

Infile 'emps1.txt'

Into table myemp

Fields terminated by ','

(ename, ecity,age)

Sqlldr userid=scott@xe/tiger control=c7.ctl

Sqlldr userid=scott@xe/tiger control=c7.ctl direct=true

The option direct=true bypasses the constraint checking)

Only one record(Sachin,Mumbai,25) gets added successfully. A file named “emps1.bad” will get created(since the input data file was emps1.txt), and it will contain 3 records :

Rahul Dravid,Bangalore,29

Virat Koli,Delhi,17

TP Singh,Chennai,19

You may check the log file ‘c7.log’ which gets created (since the name of the control file is c7.ctl). This log file will give us the complete details of the operation, and number of records read, skipped, rejected, discarded. It also gives us the details of why each rejected record was rejected.

Because the rows that failed are isolated into the bad file, you can use that file as the input for a later load once the data has been corrected.

You may create a badfile and a logfile of your own choice as follows, by giving the following BADFILE clause in the control file as follows :

Consider c71.ctl as follows :

Load data

Infile 'emps1.txt'

badfile bad.txt

truncate

Into table myemp

Fields terminated by ','

(ename, ecity,age)

To create the log file, give the LOG clause with the SQLLDR command :

Sqlldr userid=scott/tiger control=c71.ctl log=log.txt

**Logical and Physical Records**

Logical row is a row that is inserted into the database. Physical row is the actual row/data retrieved from the input data file. Depending upon the structure of the input file, multiple physical rows may be combined to make one single logical row.

Example

Ramesh,Mumbai,32

In which case there is one physical record and it will create one logical record. But, if the datafile is as follows :

Ramesh,

Mumbai,

32

Consider the following table :

Create table myemp(ename varchar2(10), ecity varchar2(20), age number(3) check (age>=20));

Consider the following Input Data file named emp.txt with name, city and age of employees respectively separated by commas, but one item on each line :

Rahul,

Bangalore,

29

Sachin,

Mumbai,

25

Virat,

Delhi,

17

TP Singh,

Chennai,

19

Use the CONCATENATE clause in a Control file C8.ctl as follows :

Load data

Infile 'emp.txt'

badfile 'b.txt'

truncate

concatenate 3

Into table myemp

Fields terminated by ','

(ename, ecity,age)

In this case, we use **Concatenate 3** to create a single logical row from 3 physical rows.

You can use CONTINUEIF clause to specify the conditions that cause logical records to be continued.

Consider the following text file ‘Friends.txt’, where a ‘\*’ at the 14th position indicates that the next person is the spouse of the person with a ‘\*’ at the 14th position. Hence Aamir and Suhana are unmarried :

Ramesh 32 \*

Sita 26

Geeta 23 \*

Sudesh 32

Aamir 31

Gita 29 \*

Sohail 25

Suhana 22

Create the following table :

create table friends(frnd\_name varchar2(10),

frnd\_age number,

spouse\_name varchar2(10),

spouse\_age number);

Now, write a control file C91.ctl as follows :

Load Data

Infile 'friends.txt' "str '\n'"

truncate

continueif this (14)='\*'

Into table friends

(frnd\_name position(1:10) char,

frnd\_age position(11:12) ,

spouse\_name position(15:24) char,

spouse\_age position(25:26) )

sqlldr userid=scott@xe/abc control=c91.ctl

Now, write a control file C9.ctl as follows :

Load Data

Infile 'friends.txt' "str '\n'"

truncate

continueif this (14)='\*'

Into table friends

(frnd\_name position(1:10) char,

frnd\_age position(11:12) integer external,

spouse\_name position(15:24) char,

spouse\_age position(25:26) integer external)

sqlldr userid=scott@xe/abc control=c9.ctl

Now, write a control file C92.ctl as follows :

Load Data

Infile 'friends.txt' "str '\n'"

truncate

continueif this (14)='\*'

Into table friends

(frnd\_name position(1:10) char,

frnd\_age position(11:12) integer external(1),

spouse\_name position(15:24) char,

spouse\_age position(25:26) integer external(1))

sqlldr userid=scott@xe/abc control=c92.ctl

To create multiple logical records from a single physical record make use of multiple **into table** clauses. This is similar to Multi-table inserts

Consider the following table :

create table scores(player varchar2(10), score number(5));

Consider the following input data file ‘scores.txt’ which contains scores of each player in three attempts. The first 10 positions contain player name, then position(11-15) for score1, position(17-21) for score 2 and position(23-27) for score 3.

Saloni 775 12345 78

Jairaj 89 56 6754

Mahi 5467 4325 6788

Now, write a control file C10.ctl as follows :

Load Data

Infile 'scores.txt'

truncate

Into table scores

(player position(1:10) char,

score position(11:15) integer external)

Into table scores

(player position(1:10) char,

score position(17:21) integer external)

Into table scores

(player position(1:10) char,

score position(23:27) integer external)

Also, write a control file C10.ctl as follows to see how only first 2 digits are considered :

Load Data

Infile 'scores.txt'

truncate

Into table scores

(player position(1:10) char,

score position(11:15) integer external(2))

Into table scores

(player position(1:10) char,

score position(17:21) integer external(2))

Into table scores

(player position(1:10) char,

score position(23:27) integer external(2))

sqlldr userid=scott/tiger control=c10.ctl

Consider the following table :

create table emp30(first\_name varchar2(10), last\_name varchar2(10), age number(3));

Consider the following input data file ‘emps.txt’ :

Hemant,Pangam,'32'

Ramesh,Surve,'29'

Saloni,Shah,'24'

Hetal,Varma,'31'

Ram,Venugopalan,23

Subramania,Iyer,17

Consider the following control file c11.ctl:

Load Data

Infile 'emps.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

truncate

Into table emp30

when age!='32'

fields terminated by ","

(first\_name,last\_name,age integer external(3) enclosed by "'")

sqlldr userid=scott/tiger control=c11.ctl

Out of the 6 logical records considered, 3 get added to the table. The record(Hemant,Pangam,'32') gets added to the **discard file ‘dis.txt’**. Records which do not meet the selection criterion get added to the discard file. The records[(Subramania,Iyer,17),(Ram,Venugopalan,23)] get added to the **Bad file ‘bad.txt’,** because they do not conform to the format expected by the control file(that is the age is not enclosed in single quotes) and the length of the column respectively. The **log file C11.log** gives you the complete details of the operation

**RECNUM keyword**

Consider the following table :

create table emps(empno number, first\_name varchar2(10), last\_name varchar2(10));

Consider the following input data file ‘empnames.txt’ :

Hemant,Pangam

Ramesh,Surve

Saloni,Shah

Hetal,Varma

Subramanian,Iyer

Ram,Venugopalan

Manoj,Pangam

Heena,Saraf

Consider the following control file c12.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

truncate

Into table emps

fields terminated by ","

(empno recnum, first\_name,last\_name)

sqlldr userid=scott/tiger control=c12.ctl

The records[(Subramanian,Iyer,17),(Ram,Venugopalan,23)] get added to the **Bad file ‘bad.txt’**.

The RECNUM keyword assigns a record number to each logical record as it is read from the datafile, and that value will be assigned into the assigned column of the table.

select \* from emps;

EMPNO FIRST\_NAME LAST\_NAME

------------ -------------------- ------------------

1 Hemant Pangam

2 Ramesh Surve

3 Saloni Shah

4 Hetal Varma

7 Manoj Pangam

8 Heena Saraf

Notice the empno values, which show that the 5th and 6th physical records from the input file were read, but for some reason could not be inserted into the table.

**CONSTANT keyword**

Consider the following table:

create table empdept(empno number,first\_name varchar2(10), last\_name varchar2(10), deptno number(2));

Consider the following input data file ‘empnames.txt’ :

Hemant,Pangam

Ramesh,Surve

Saloni,Shah

Hetal,Varma

Manoj,Pangam

Heena,Saraf

Consider the following control file c13.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

truncate

Into table empdept

fields terminated by ","

(empno recnum, first\_name,last\_name, deptno constant 50)

sqlldr userid=scott/tiger control=c13.ctl

select \* from empdept;

EMPNO FIRST\_NAME LAST\_NAME DEPTNO

----------- -------------------- ------------------- ----------

1 Hemant Pangam 50

2 Ramesh Surve 50

3 Saloni Shah 50

4 Hetal Varma 50

5 Manoj Pangam 50

6 Heena Saraf 50

The **Constant** keyword allows you to assign a constant value to a column during the load. For character columns, enclose the constant value with single quotes. If you use the **SYSDATE** keyword, the selected column will be populated with the current system date and time

Example : (empno, first\_name,last\_name, hiredate sysdate)

create table emphire(empno number,first\_name varchar2(10), last\_name varchar2(10), hiredate date);

Consider the following control file c131.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

truncate

Into table emphire

fields terminated by ","

(empno recnum, first\_name,last\_name, hiredate sysdate)

sqlldr userid=scott/tiger control=c131.ctl

**SEQUENCE keyword**

Consider the following table:

create table empseq(empno number, first\_name varchar2(10), last\_name varchar2(10));

Consider the following input data file ‘empnames.txt’ :

Hemant,Pangam

Ramesh,Surve

Saloni,Shah

Hetal,Varma

Manoj,Pangam

Heena,Saraf

Consider the following control file c14.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

truncate

Into table empseq

fields terminated by ","

(empno sequence, first\_name,last\_name)

sqlldr userid=scott/tiger control=c14.ctl

If you use the **sequence** keyword , SQL \* Loader will maintain a sequence of values during the load. As records are processed, the sequence value will be increased by the increment you specify. If the rows fail during insert(and are sent to the bad file), those sequence values will not be reused.

Following are various ways of using SEQUENCE keyword

SEQUENCE

Start with 1 increment by 1

SEQUENCE(100)

Start with 100 increment by 1

SEQUENCE(100,5)

Start with 100 increment by 5

Also, consider the following control file c15.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

append

Into table empseq

fields terminated by ","

(empno sequence(max), first\_name,last\_name)

sqlldr userid=scott/tiger control=c15.ctl

If you use the max keyword within the sequence option, the sequence values will use the current maximum value from the column as the starting point for the sequence, and then increment by 1. You may also user sequence(max,10) which means the sequence values will use the current maximum value from the column as the starting point for the sequence, and then increment by 10

Also try the following control file :

C151.ctl

Load Data

Infile 'empnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

append

Into table empseq

fields terminated by ","

(empno sequence(max,5), first\_name,last\_name)

Note : If you use numbers in VARCHAR2 columns, avoid the use of SEQUENCE option for such columns. For example, if you table already contains the values 1 through 10 in a varchar2 column, then the maximum value within the column is considered as 9. Using that as the basis for a sequence option will cause SQL \* LOADER to attempt to insert a record using 10 as the newly created value and that will conflict the existing record with value 10 for that column. This behavior illustrates why storing numbers in a character columns is a poor practice in general.

Consider the following table :

create table empcseq(empno varchar2(10), first\_name varchar2(10), last\_name varchar2(10));

Consider the input data file ‘empcnames.txt’ as follows :

Hemant,Pangam

Ramesh,Surve

Saloni,Shah

Hetal,Varma

Manoj,Pangam

Heena,Saraf

Mamta,Varma

Dhaval,Varma

Heena,Saraf

Jiby,Dsouza

Consider the following control file c16.ctl

Load Data

Infile 'empcnames.txt' "str'\n'"

badfile 'bad.txt'

discardfile 'dis.txt'

append

Into table empcseq

fields terminated by ","

(empno sequence(max,10), first\_name,last\_name)

sqlldr userid=scott/tiger control=c16.ctl

The 10 physical records are added with number 10,20,30…100.

SQL> select empno, ascii(empno) from empcseq;

EMPNO ASCII(EMPNO)

---------- ------------

10 49

20 50

30 51

40 52

50 53

60 54

70 55

80 56

90 57

100 49

Thus, the largest ascii value is that of 90(since ascii returns the actual ascii value of the first character), hence sequence(max,10) will assume the largest value to be 90, for the next run

Now, run the above command as follows :

sqlldr userid=scott/tiger control=c16.ctl

The 10 physical records are now added with numbers 100,110,120,..190 to the existing 10 records. If the empno had primary key constraint then the first row from the data file is sent to the bad file, and the remaining 9 rows are added.

Now, run the above command as follows :

sqlldr userid=scott/abc control=c16.ctl

The 10 physical records are now added with numbers 100,110,120,..190 to the existing records.