ORAtips
On Database

Simplify Your Life with Oracle® SQL*Loader

By Natalka Roshak

*Editor's Note: Oracle by design comes with the utility tools to help manage your database operations. SQL*Loader is one such utility and in terms of data loading/unloading programs, it is one of the best. Regular ORAtips contributor Natalka Roshak presents several case studies to highlight the power of using SQL*Loader. As Natalka states, "once you've used SQL*Loader a few times, you'll wonder how you ever lived without it."*

Introduction

One common Oracle DBA duty is getting heterogeneous data from any number of sources in and out of the database. There are a lot of expensive data loading and unloading programs available, but every Oracle install includes a utility that will load any data you can extract to the file system, at no extra cost. Yet many DBAs and developers are unpracticed with, or even unaware of, this powerful tool.

This tool is Oracle's SQL*Loader. In this article, we'll look at using SQL*Loader to load data from a variety of heterogeneous sources. First, though, we'll cover the basics.

SQL*Loader Basics

SQL*Loader is a command-line utility, run by invoking the sqlldr executable in the $ORACLE_HOME/bin directory. Because it's a command-line utility, it's easily scriptable; however, because it's also very flexible, the plethora of available options can make sqlldr syntax daunting at first approach.

In a nutshell, each SQL*Loader session requires three types of input:

- data;
- a control file, which tells SQL*Loader how the data is formatted and where to put it;
- and a set of parameters, such as the database connect string, the location of the control file and data, etc.

A single SQL*Loader session can write data to up to four separate places:

- the database;
- a log of the session is written to the log file;
- any rows that don't fit the data format specified in the control file get written to the bad file;
- and any rows that don't fit the optional row-loading criteria you may specify in the control file get written to the discard file.

The SQL*Loader inputs and outputs are summarized in Figure 1.

The simplest way to see how SQL*Loader uses these inputs is to work through a case study. Our first case study will cover the SQL*Loader basics.

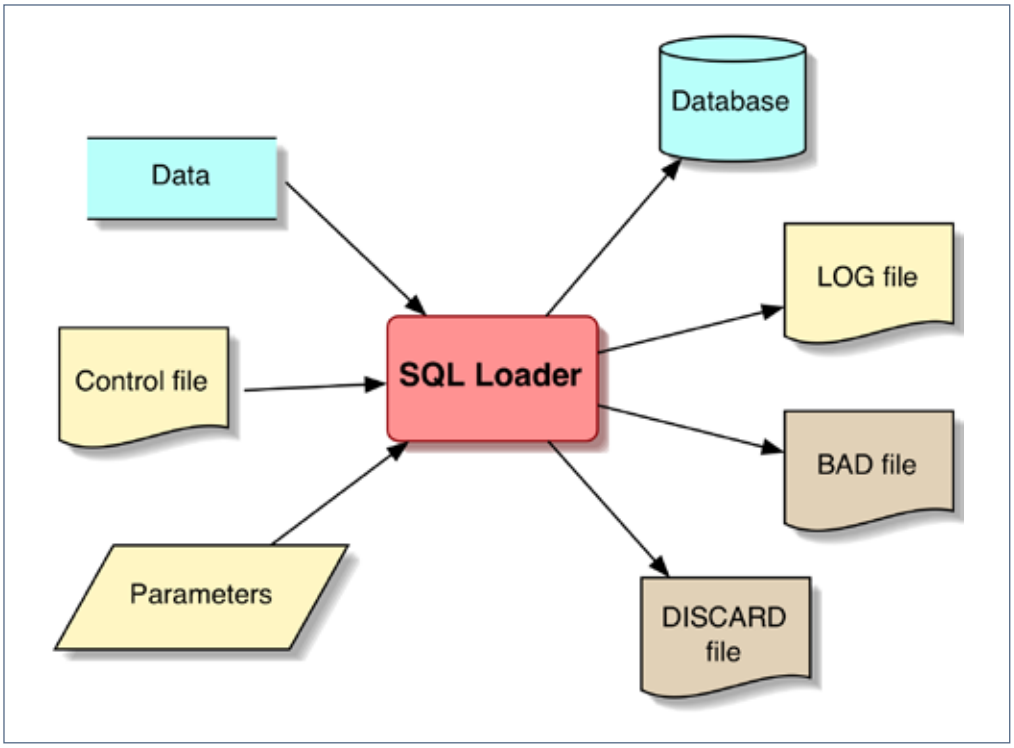

Figure 1: SQL*Loader Inputs and Outputs

Case Study 1: The Basics

Suppose we have a file named DEPARTMENTS.TXT, which contains the following data about departments:

```
SAL   SALES DEPARTMENT
HRD   HUMAN RESOURCES
ACT   ACCOUNTING
OPS   BUILDING OPERATIONS
FCY   FACTORY FLOOR
MGT   MANAGEMENT
```

and that we need to load this data into an Oracle database.

The first step is to make sure that the database contains a target table; SQL*Loader never creates tables, but only loads into existing ones. For this example, assume the following target table:

```
CREATE TABLE REF_DEPT_ID
( DEPT_ID CHAR(3),
 DEPT_NAME VARCHAR2(100) );
```

The next step is to create the control file. This file must tell SQL*Loader:

- where to find the data (DEPARTMENTS.TXT);
- where to put the data (REF_DEPT_ID);
- whether to append or replace the new data to any data already in REF_DEPT_ID;
- how to parse the data into fields; and
- which fields in the datafile go to which fields in REF_DEPT_ID.

With a fixed-length data file like DEPARTMENTS.TXT, where the records can be parsed simply by counting characters, the last two requirements are simple: the control file only needs to specify which column positions in the datafile hold the data destined for each field in the target table. In our example, characters 1-3 on each line hold the department ID, characters 4-5 are blank, and characters 6 and higher hold the department name.

SQL*Loader never creates tables, but only loads into existing ones.

In the following control file, the comments in red show how each line satisfies one of the requirements above. (In SQL*Loader control files, as in SQL and PL/SQL, all text after a double hyphen is treated as a comment.)

```
LOAD DATA
INFILE 'DEPARTMENTS.TXT' --Where to find the
data;
APPEND --Whether to append to or replace the
data in the target tbl;
INTO TABLE REF_DEPT_ID --Where to put the
data; and
--Which col. pos. in the datafile hold the data
-- for each field in the target table:
( DEPT_ID POSITION (01:03) char(3),
 DEPT_NAME POSITION (06:105) char(100) )
```

Save this control file as departments.ctl . Now that we've got the control file set up, invoking SQL*Loader is simple. The only parameters we need to supply are the userid and the location of the control file. Here's the command:

```
[oracle@mysrv tmp]$ sqlldr userid=scott/tiger
control=departments.ctl

SQL*Loader: Release 10.2.0.2.0 - Production on
Sun Aug 13 20:15:21 2006

Copyright (c) 1982, 2005, Oracle. All rights
reserved.

Commit point reached - logical record count 6
```

That's all there is to it! The data in DEPARTMENTS.TXT has been loaded. If you look in the working directory, you'll see that SQL*Loader automatically created a log file with the default name of departments.log (based on the name of the control file):

```
[oracle@ mysrv tmp]$ ls
DEPARTMENTS.TXT departments.ctl departments.
log
```

The log file contains a good deal of useful information about the SQL*Loader session:

```
[oracle@ mysrv tmp]$ cat departments.log

SQL*Loader: Release 10.2.0.2.0 - Production on Sun Aug 13 20:15:21 2006

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Control File:  departments.ctl
Data File:   DEPARTMENTS.TXT
 Bad File:   DEPARTMENTS.bad
 Discard File: none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:   64 rows, maximum of 256000 bytes
Continuation:  none specified
Path used:   Conventional

Table REF_DEPT_ID, loaded from every logical record.
Insert option in effect for this table: APPEND

 Column Name    Position  Len Term Encl Datatype
---------------------- ---------- ----- ---- ---- ---------------------
DEPT_ID          1:3  3     CHARACTER
DEPT_NAME        6:105  100     CHARACTER


Table REF_DEPT_ID:
 6 Rows successfully loaded.
 0 Rows not loaded due to data errors.
 0 Rows not loaded because all WHEN clauses were failed.
 0 Rows not loaded because all fields were null.

Space allocated for bind array:        6912 bytes(64 rows)
Read  buffer bytes: 1048576

Total logical records skipped:    0
Total logical records read:      6
Total logical records rejected:    0
Total logical records discarded:   0

Run began on Sun Aug 13 20:15:21 2006
Run ended on Sun Aug 13 20:15:21 2006

Elapsed time was:   00:00:00.07
CPU time was:     00:00:00.02
```

We can double-check that the rows were loaded into the target table in the database by querying it:

```
SQL> select * from ref_dept_id ;

DEP DEPT_NAME
--- ------------------------------
SAL SALES DEPARTMENT
HRD HUMAN RESOURCES
ACT ACCOUNTING
OPS BUILDING OPERATIONS
FCY FACTORY FLOOR
MGT MANAGEMENT

6 rows selected.
```

This load ran without errors, so SQL*Loader did not write anything to the bad or discard files. If there had been any bad rows, i.e., rows that SQL*Loader was unable to parse according to our instructions, SQL*Loader would have created a default-named file DEPARTMENTS.bad; if there had been any discards, and if we had specified a discard file name to collect them, SQL*Loader would have copied the discarded rows to the discard file. (We'll see how discards are generated later in this article.)

We can specify the names of the log, bad, and discard files on the command line instead of using the defaults. For example:

```
[oracle@mysrv tmp]$ sqlldr userid=scott/tiger
control=departments.ctl bad=depts.bad
log=DEPTLOG.TXT discard=DUMP.DSC
```

We can also wrap these command-line parameters up in a parfile, instead of typing them all on the command line.

```
[oracle@mysrv tmp]$ cat departments.parfile
userid=scott/tiger
control=departments.ctl
log=dept_load.log
bad=dept_bad.txt
oracle@mysrv tmp]$ sqlldr parfile=departments.
parfile
```

In fact, SQL*Loader is quite flexible about how it gets its data, control file, and parameter inputs. The parameters can be on the command line, in a parfile, or even wrapped into the control file; the data can be in a datafile, as we've seen, but it can also be contained right in the control file, as I'll demonstrate below. The one constant is that there must be a control file.

The syntax for putting the data right in the control file is simple. Specify the INFILE as *, and put the data at the end of the control file, preceded by the keyword "begindata". For example:

```
[oracle@mysrv tmp]$ cat departments.ctl
LOAD DATA
INFILE *
REPLACE
INTO TABLE REF_DEPT_ID
( DEPT_ID POSITION (01:03) char(3),
 DEPT_NAME POSITION (06:105) char(100) )
BEGINDATA
SAL SALES DEPARTMENT
HRD HUMAN RESOURCES
ACT ACCOUNTING
OPS BUILDING OPERATIONS
FCY FACTORY FLOOR
MGT MANAGEMENT
```

Note that I used REPLACE instead of APPEND in this control file; this means that the data we'd already loaded into REF_DEPT_ID will be deleted before the new data is inserted. Another option available here is TRUNCATE, which truncates the table instead of simply deleting all its rows.

Now that we've covered some SQL*Loader basics, we're ready to move on to real-world uses.

Loading an Excel File into the Database

One request that comes up time and time again is to load data from an Excel spreadsheet into the database. It's such a common need that there are even third-party tools dedicated to handling this task. Such tools are convenient, but not necessary. In this example, we'll step through loading an Excel worksheet into the database.

	A	B
1	Department ID	Department Name
2	SAL	SALES DEPARTMENT
3	HRD	HUMAN RESOURCES
4	ACT	ACCOUNTING
5	OPS	BUILDING OPERATIONS
6	FCY	FACTORY FLOOR
7	MGT	MANAGEMENT
8		

Figure 2: Excel Worksheet Used in This Case Study

Case Study 2: Loading an Excel Worksheet

For this example, let's load an Excel worksheet that just contains the data we used in the previous case study.

SQL*Loader needs this data in a text file, so save this worksheet as a CSV (comma-separated values) file. From the File menu, choose Save As. In the Save As dialog, choose "CSV (Comma delimited)" from the Format drop-down:

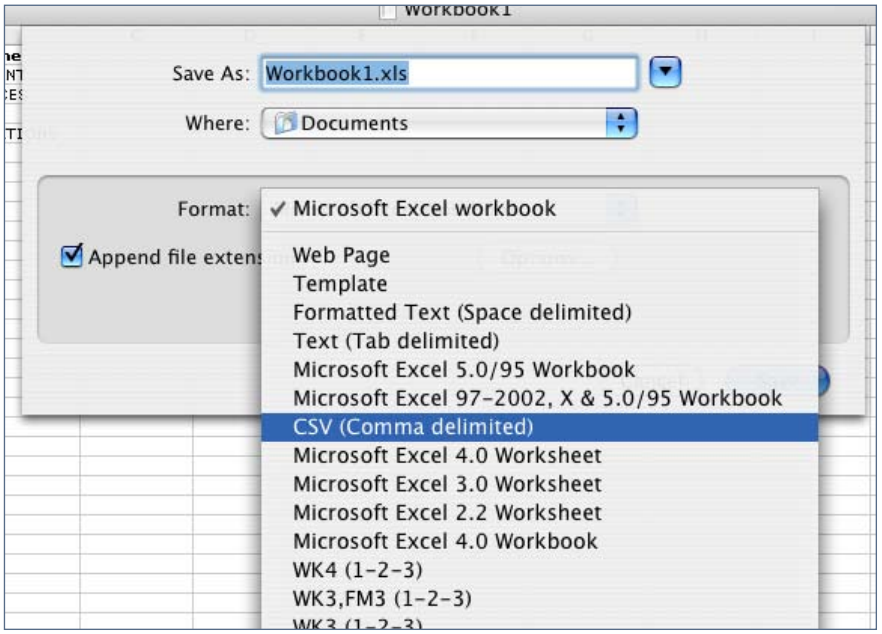

Figure 3: Save As CSV

This will generate a text file containing the spreadsheet data:

```
Department ID,Department Name
SAL,SALES DEPARTMENT
HRD,HUMAN RESOURCES
ACT,ACCOUNTING
OPS,BUILDING OPERATIONS
FCY,FACTORY FLOOR
MGT,MANAGEMENT
```

The next step is to create a control file to load this data. The main

difference from the previous case study is that instead of using character position to parse the data file, we'll tell SQL*Loader that the data fields are separated by the delimiter character",". To specify the correspondence between data-file fields and target-table fields, instead of saying which field in REF_DEPT_ID corresponds to which stretch of characters in the data file, we simply list the target columns in REF_DEPT_ID in the order in which their corresponding fields appear in the data file. In this case, the fields in the data file are in the same order as the fields in the table.

	A	B	C
1	Department ID	Staffing total Oct 05	Department Name
2	SAL	134	SALES DEPARTMENT
3	HRD	21	HUMAN RESOURCES
4	ACT	10	ACCOUNTING
5	OPS	5	BUILDING OPERATIONS
6	FCY	521	FACTORY FLOOR
7	MGT	57	MANAGEMENT

Figure 4: Sample Worksheet with Extraneous Data

	A	B	C	D
1	**Department ID**	**Department Name**	**Staff Census - May 2006**	**Total Dept Staff Payroll - May 2006**
2	SAL	SALES DEPARTMENT	135	6,075,017.65
3	HRD	HUMAN RESOURCES	20	1,304,675.11
4	ACT	ACCOUNTING	15	931,265.92
5	OPS	BUILDING OPERATIONS	7	489,786.12
6	FCY	FACTORY FLOOR	700	21,576,271.43
7	MGM	MANAGEMENT	65	5,782,398.29
8	SNX	SNAKES	0	0.00

Figure 5: Sample Worksheet with Commas in Column Values

```
LOAD DATA
INFILE deptdata.csv
TRUNCATE
INTO TABLE REF_DEPT_ID
FIELDS TERMINATED BY ','
( dept_id,
 dept_name
   )
```

Also, the first row of the CSV file holds the column titles from the Excel spreadsheet; we don't want to load those. Fortunately, there's a SQL*Loader parameter that lets us skip the first n records at the head of a datafile, SKIP=n.

```
[oracle@mysrv tmp]$ sqlldr scott/tiger
control=departments.ctl skip=1

SQL*Loader: Release 10.2.0.2.0 - Production on
Sun Aug 13 23:46:15 2006

Copyright (c) 1982, 2005, Oracle. All rights
reserved.

Commit point reached - logical record count 6
```

Case Study 3: Spreadsheets with Extraneous Data

Often, spreadsheets contain columns of extraneous data that we don't want to load. For example:

You can delete this data by manipulating the spreadsheet. Or, you can simply tell SQL*Loader to ignore it by labeling it as FILLER in the control file:

```
LOAD DATA
INFILE deptdata.csv
TRUNCATE
INTO TABLE REF_DEPT_ID
FIELDS TERMINATED BY ','
( dept_id,
 col2 FILLER,
 dept_name
)
```

After running this load, the log file shows that SQL*Loader understood our instructions to ignore the data in the second column:

It's also common for spreadsheet data to include commas in the column data. For example:

Fortunately, Excel realizes that this could lead to rather confusing CSV files; when you save such a file as CSV, any column values with commas in them are automatically enclosed in double quotes:

```
Department ID,Department Name
SAL,"Sales, Department of"
HRD,"Human Res., Dept. of"
ACT,"Accounting, Department of"
OPS,Operations Department
FCY,Factory Floor Umbrella Dept.
MGT,Amalgamated Managerial Dept.
```

In the SQL*Loader control file, we simply need to tell SQL*Loader that the data fields may be enclosed by quotes:

```
LOAD DATA
INFILE deptdata.csv
TRUNCATE
INTO TABLE REF_DEPT_ID
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '""'
( dept_id,
 dept_name
)
```

After loading the data with the control file above, checking the target table shows that SQL*Loader correctly parsed the data:

```
SQL> select * from ref_dept_id ;

DEP DEPT_NAME
--- ----------------------------------------
SAL Sales, Department of
HRD Human Res., Dept. of
ACT Accounting, Department of
OPS Operations Department
FCY Factory Floor Umbrella Dept.
MGT Amalgamated Managerial Dept.
```

Unfortunately, not all data comes in convenient, less-than-4000-character fields. In our next examples, we'll look at loading longer fields – LOB (Large OBject) -sized fields – into the database.

Loading LOB Data

LOB, or Large OBject data, can be stored in several formats. Character data that's too long to be stored in VARCHAR2 columns can be stored in CLOBs; binary data like images and mp3s can be stored in BLOBs or BFILEs. We'll start with CLOB data.

Case Study 5: Loading CLOB Data

The maximum amount of data that a VARCHAR2 column can hold is 4000 bytes. But real-world data fields are often longer. A spreadsheet, for example, might have more than 4000 bytes of character data stored in some cells:

We'll begin by adding a column to the target table to hold the CLOB data:

```
SQL> alter table ref_dept_id add (dept_long_
info clob);

Table altered.
```

To load this particular dataset, the first change we'll need to make to the control file has nothing to do with LOBs. However, it's necessary for the data in the worksheet displayed in Figure 6. Note that some of the rows do not have a value in the "Additional Information" column. In the CSV file, these rows will seem to have trailing commas:

```
SAL,SALES DEPARTMENT,
HRD,HUMAN RESOURCES,
```

The ',' at the end of the record confuses SQL*Loader; it expects a third column. If we tried to load such rows with one of our previous control files, these rows would go to the badfile and the log file would say:

```
Column not found before end of logical record (use
TRAILING NULLCOLS)
```

So, we'll need to specify TRAILING NULLCOLS in the control file so that SQL*Loader knows not to expect every row to have data in the last column.

The second change is simply to let SQL*Loader know that the maximum data length for the last field is longer than its default. In our sample spreadsheet, the longest column value is about 6000 characters; to be on the safe side, we'll specify 8000 in the control file.

```
LOAD DATA
INFILE deptdata.csv
TRUNCATE
INTO TABLE REF_DEPT_ID
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( dept_id,
 dept_name,
 dept_long_info char(8000)
)
```

	A	B	C
1	**Department ID**	**Department Name**	**Additional Information**
2	SAL	SALES DEPARTMENT	
3	HRD	HUMAN RESOURCES	
4	ACT	ACCOUNTING	elit. Sed euismod lacinia nisi. Pellentesque orci rutrum eu, semper vitae, euismod quis, elit. Cu tincidunt, ante a vulputate auctor, mi odio dictur lorem, sed ullamcorper neque est commodo pur Aenean dui. Vestibulum viverra arcu at arcu. Sed Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Sed
5	OPS	BUILDING OPERATIONS	
6	FCY	FACTORY FLOOR	Ut pulvinar nonummy felis. Quisque et massa. lectus. Curabitur mauris. Nunc non urna. Praese cursus. Morbi a velit vel quam eleifend tincidunt Curabitur at eros. Lorem ipsum dolor sit amet,
7	MGT	MANAGEMENT	elitr, sed diam nonumy eirmod tempor invidunt labore et dolore magna aliquyam erat, sed dian voluptua. At vero eos et accusam et justo duo d et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit ame Lorem ipsum dolor sit amet, consetetur sadipsci
8			

Figure 6: Sample Worksheet with Some Cells Holding Data Longer Than 4000 Bytes

Querying REF_DEPT_ID shows that the longer-than-4000-byte CLOB data loaded successfully:

```
SQL> select dept_id, dept_name, length(dept_
long_info) from ref_dept_id

DEP DEPT_NAME   LENGTH(DEPT_LONG_INFO)
--- ------------------------------ --------------------
SAL SALES DEPARTMENT                  0
HRD HUMAN RESOURCES                   0
ACT ACCOUNTING                      719
OPS BUILDING OPERATIONS               0
FCY FACTORY FLOOR                   281
MGT MANAGEMENT                     6086
```

Of course, you can use SQL*Loader to load more than just CLOB data into the database, and the LOBs do not have to be in-line in the data records. In our next case study, we'll look at loading BLOBs that are found in separate files.

Case Study 6: LOB Data in Files

For this case study, suppose that we want to load in department logos for some departments. The department logos came to us separately from each department, as one JPG image file from each department:

```
[oracle@mysrv tmp]$ ls -lh deptlogos/
total 32K
-rw-r--r-- 1 oracle oinstall 2.8K Aug 14 01:13
acctg.jpg
-rw-r--r-- 1 oracle oinstall 2.3K Aug 14 01:17
mgmt.jpg
-rw-r--r-- 1 oracle oinstall 12K Aug 14 01:14 sales.
jpg
```

Again, the first step is to make sure we have a table in the target database to hold the data:

```
SQL> create table dept_logos
 2 (dept_id varchar2(3), dept_logo_jpg blob);

Table created.
```

In order to establish the department ID - imagefile correspondence, we will create a simple record set that lists the department IDs and the corresponding datafiles, like so:

```
ACT,deptlogos/acctg.jpg
SAL,deptlogos/sales.jpg
MGT,deptlogos/mgmt.jpg
```

and place this data inline in the control file. Here is the control file we will use to load the data:

```
LOAD DATA
INFILE *
TRUNCATE
INTO TABLE dept_logos
FIELDS TERMINATED BY ','
(
 dept_id  char(3),
 file_name filler,
 dept_logo_jpg LOBFILE (file_name) TERMI-
NATED BY EOF
)
BEGINDATA
ACT,deptlogos/acctg.jpg
SAL,deptlogos/sales.jpg
MGT,deptlogos/mgmt.jpg
```

LOB, or Large OBject data, can be stored in several formats.

After running the load, we can confirm that the images were loaded by querying the target table:

```
SQL> select dept_id, dbms_lob.getlength(dept_
logo_jpg)
 2 from dept_logos ;

DEP DBMS_LOB.GETLENGTH(DEPT_LOGO_JPG)
--- ---------------------------------
ACT                              2810
SAL                             11426
MGT                              2253
```

What is going on in this control file? The "file_name" field in the data is treated like a variable in the expression LOBFILE (file_name). SQL*Loader determines the contents of the DEPT_LOGO_JPG field dynamically, based on the contents of the FILE_NAME filler field. In the log file for this load, the DEPT_LOGO_JPG field shows as DERIVED:

```
Column Name     Position  Len Term Encl Datatype
---------------------- ---------- ----- ---- ---- ------------
DEPT_ID         FIRST  3 ,   CHARACTER
FILE_NAME       NEXT  * ,   CHARACTER
 (FILLER FIELD)
DEPT_LOGO_JPG     DERIVED  * EOF  CHARACTER
  Dynamic LOBFILE. Filename in field FILE_NAME
```

The ability to derive field values dynamically is one aspect of the powerful set of complex operations that SQL*Loader can perform on the data on its way into the database. We'll look at some of these operations in the next section.

Manipulating Data with SQL*Loader

In addition to just plain loading, SQL*Loader can manipulate the data on its way in. SQL*Loader can dynamically populate derived fields, as we saw in the previous case study; it can dynamically merge fields from more than one data file into the target table, or load data into more than one target table in the same session;

it can selectively load records from the data using a WHERE clause; and more.

Case Study 7: Manipulating Data Field Contents and Selectively Loading Records

SQL*Loader can manipulate or derive the data field contents using any SQL expression that would be admissible in the VALUES clause of an INSERT statement. In this case study, we'll load the data in this spreadsheet:

There are a couple of "errors" in this spreadsheet – the MANAGEMENT department has dept_id "MGM" instead of "MGT", and the "SNX" department is a mistake. In this case study, we'll use the SQL*Loader control file to fix these mistakes, round the payroll numbers, and add a derived field.

We'll export this data as a CSV file called deptstaffing.csv, create a table called DEPT_STAFFING_05 to hold it, and load it into the database, as before. Briefly, here is the CSV file:

```
[oracle@mysrv tmp]$ cat deptstaffing.csv
Department ID,Department Name,Staff Census - May 2006,Total Dept Staff Payroll - May 2006
SAL,SALES DEPARTMENT,135,"6,075,017.65"
HRD,HUMAN RESOURCES,20,"1,304,675.11"
ACT,ACCOUNTING,15,"931,265.92"
OPS,BUILDING OPERATIONS,7,"489,786.12"
FCY,FACTORY FLOOR,700,"21,576,271.43"
MGM,MANAGEMENT,65,"5,782,398.29"
SNX,SNAKES,0,0.00
```

Note that Excel has exported the "Total Dept Staff Payroll" values as numbers with thousands separators (commas), just as they displayed in the spreadsheet. We could go back, change the spreadsheet's number format to suppress thousands separators, and re-export - or we could simply clean this up in the control file. We'll take the latter approach.

Here is the target table:

```
create table dept_staffing_05
( dept_id char(3), dept_staff_count number(10,0),
 dept_total_payroll number, dept_average_salary number );
```

Note that we've added an extra field to the target table, DEPT_AVERAGE_SALARY, which does not appear in the data file. We'll use SQL*Loader to derive this field from the staff census and total payroll fields.

So, our control file will need to instruct SQL*Loader to manipulate the data in the following ways:

- Use a SQL DECODE statement on the Department ID to substitute 'MGT' for 'MGM'.
- The DEPT_TOTAL_PAYROLL field shows up in the CSV as a CHAR string with internal commas, not as a string that Oracle would recognize as a number; SQL*Loader needs to strip the internal commas from each value and cast it as a number.
- Calculate DEPT_AVERAGE_SALARY.

Here is a control file that does all of the above:

```
LOAD DATA
INFILE 'deptstaffing.csv'
TRUNCATE
INTO TABLE dept_staffing_05
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( dept_id "decode (:dept_id, 'MGM','MGT', :dept_id )",
 dept_name filler,
 dept_staff_count,
 dept_total_payroll "to_number(replace( :dept_total_payroll, ',', '') )" ,
 dept_average_salary "decode(:dept_staff_count, 0, 0, round ( to_number(replace( :dept_total_payroll,',','') ) / :dept_staff_count, 2))"
)
```

	A	B	C	D
1	Department ID	Department Name	Staff Census - May 2006	Total Dept Staff Payroll - May 2006
2	SAL	SALES DEPARTMENT	135	6,075,017.65
3	HRD	HUMAN RESOURCES	20	1,304,675.11
4	ACT	ACCOUNTING	15	931,265.92
5	OPS	BUILDING OPERATIONS	7	489,786.12
6	FCY	FACTORY FLOOR	700	21,576,271.43
7	MGM	MANAGEMENT	65	5,782,398.29
8	SNX	SNAKES	0	0.00

Figure 7: Worksheet of Data for Case Study 7.

Examining the table post-load shows that our fields were successfully transformed:

```
SQL> select * from dept_staffing_05
2 ;

DEP DEPT_STAFF_COUNT DEPT_TOTAL_PAY-
ROLL DEPT_AVERAGE_SALARY
--- ---------------- ------------------ ---------------
----
SAL              135       6075017.65        45000.13
HRD               20       1304675.11        65233.76
ACT               15        931265.92        62084.39
OPS                7        489786.12        69969.45
FCY              700       21576271.4        30823.24
MGT               65       5782398.29        88959.97
SNX                0                0               0
```

We've almost achieved our goal of cleaning up the data with SQL*Loader. However, we are still loading that annoying SNX row. It's not at the beginning of the data file, so we can't simply use the SKIP parameter to skip it. Fortunately, SQL*Loader lets us selectively load records. We simply have to add a line to the control file, known as a WHEN clause, to exclude rows where dept_id = 'SNX':

```
LOAD DATA
INFILE 'deptstaffing.csv'
TRUNCATE
INTO TABLE dept_staffing_05
WHEN DEPT_ID != 'SNX'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( dept_id "decode (:dept_id, 'MGM','MGT', :
dept_id )",
 dept_name filler,
 dept_staff_count,
 dept_total_payroll "to_number(replace( :
dept_total_payroll, ',','') )" ,
 dept_average_salary "decode(:dept_staff_count,
0, 0, round ( to_number(replace( :dept_total_
payroll, ',','') ) / :dept_staff_count, 2))"
)
```

Make sure to specify a discard file for this load, to hold a copy of any records that fail the WHEN clause:

```
[oracle@mysrv tmp]$ sqlldr scott/tiger
control=departments.ctl skip=1
discard=deptstaffing.dsc
SQL*Loader: Release 10.2.0.2.0 - Production on
Mon Aug 14 22:12:57 2006
Copyright (c) 1982, 2005, Oracle. All rights
reserved.
Commit point reached - logical record count 7
```

After running the load with this control file, the log file shows that SQL*Loader skipped the 'SNX' row:

```
[oracle@mysrv tmp]$ cat departments.log
SQL*Loader: Release 10.2.0.2.0 - Production on
Mon Aug 14 22:07:55 2006
Copyright (c) 1982, 2005, Oracle. All rights
reserved.
Control File:  departments.ctl
Data File:   deptstaffing.csv
 Bad File:   deptstaffing.bad
 Discard File: deptstaffing.dsc
[...]
Table DEPT_STAFFING_05, loaded when
DEPT_ID != 0X534e58(character 'SNX')
[...]
Record 7: Discarded - failed all WHEN clauses.
Table DEPT_STAFFING_05:
 6 Rows successfully loaded.
 0 Rows not loaded due to data errors.
 1 Row not loaded because all WHEN clauses were
failed.
 0 Rows not loaded because all fields were null.
```

As the log file says, the 'SNX' row was "discarded" - not loaded, and copied to the discard file:

```
[oracle@mysrv mysrv]$ cat deptstaffing.dsc
SNX,SNAKES,0,0.00
```

Loading Data From Non-Oracle Databases

So far, we've concentrated on data already found in the file system, such as spreadsheets, text files, and JPGs. But it's worth noting that SQL*Loader can also be used to load moderate amounts of data from non-Oracle databases. This may seem obvious, but it often doesn't occur to developers and DBAs.

People often ask whether Oracle provides an "unloading" utility, one that creates delimited or fixed-length data files like the files SQL*Loader loves to load. The answer is no, and yes. Every SQL database can generate a plain text file of data simply by spooling query results to a file. For example, to extract the contents of the table in the preceding case study, DEPT_STAFFING_05, to a CSV file, simply spool the results of this query to file:

```
select '"' || dept_id || '","' || dept_staff_
count || '","'
 || dept_total_payroll || '","' || dept_aver-
age_salary || '"'
from dept_staffing_05;
```

The generated file:

```
[oracle@mysrv tmp]$ cat dept_staffing.csv
"SAL","135","6075017.65","45000.13"

"HRD","20","1304675.11","65233.76"

"ACT","15","931265.92","62084.39"
"OPS","7","489786.12","69969.45"
"FCY","700","21576271.43","30823.24"

"MGT","65","5782398.29","88959.97"
```

Note: There's a little more to it than this, if you don't want to have to edit the spool file to remove a few extraneous lines; you must first suppress headers, echo, etc., using SQL*Plus "SET" commands. However, SQL*Plus commands are beyond the scope of this article. See the SQL*Plus documentation for more information.

So, it's very simple to use SQL*Loader to load data from a non-Oracle database into an Oracle database. First, issue SQL commands against the non-Oracle database to generate a plain text data file, like the

one above. Then, use SQL*Loader to move the data from the plain-text data file into your Oracle database.

This is a cost-effective, easy, and satisfactory way to move moderate amounts of data from a non-Oracle database to an Oracle database. Because SQL*Loader is a command line utility, recurring heterogeneous data loads can easily be scripted. Such a script might look like this:

```
# Pseudo-code illustrating a simple heterogeneous extract-load script
# from Sybase to Oracle
isql -U scott -P tiger -i generate_dept_csv.sql -o dept_output.csv
sqlldr scott/tiger control=dept.ctl
```

Of course, SQL*Loader is an inadequate solution in situations where the Oracle database needs real-time information from a heterogeneous database. And for very large loads, the overhead of unloading the data to an OS file, then reloading it, would be prohibitive. A third-party solution, Oracle Gateways, or custom-made applications would be better solutions than SQL*Loader. However, for moderate amounts of data, or for recurring moderately sized loads, SQL*Loader can provide a satisfactory, easy-to-implement solution.

Performance

This introduction to SQL*Loader wouldn't be complete without a note on performance. SQL*Loader has two distinct ways of loading data: conventional and direct path. In a conventional load, SQL*Loader executes a SQL INSERT statement for each row in the datafile. For example, in our first case study, a conventional load would execute the following statements:

```
INSERT INTO TABLE REF_DEPT_ID (DEPT_ID, DEPT_NAME)
VALUES ('SAL','SALES DEPARTMENT');
INSERT INTO TABLE REF_DEPT_ID (DEPT_ID, DEPT_NAME)
VALUES ('HRD','HUMAN RESOURCES');
....
INSERT INTO TABLE REF_DEPT_ID (DEPT_ID, DEPT_NAME)
VALUES ('MGT','MANAGEMENT');
```

A direct path load, on the other hand, bypasses Oracle's SQL processing layer. SQL*Loader pulls the data from the file, formats it into data blocks, and writes it directly to the data files. Because it bypasses the SQL processing layer, a direct path load has much less overhead and thus is much faster. However, because it bypasses the SQL layer, there are several restrictions on direct path load. These and other considerations are discussed in the documentation; see chapter 11 of the 10gR2 Oracle Database Utilities book.

To use direct path load, supply the DIRECT=TRUE parameter:

```
[oracle@mysrv tmp]$ sqlldr scott/tiger control=dept_logos.ctl direct=true
```

For an even faster load, consider making the load unrecoverable by prepending the UNRECOVERABLE keyword to the control file:

```
UNRECOVERABLE
LOAD DATA
INFILE *
....
```

No redo information will be generated for an UNRECOVERABLE load. This will have the same effect on the recoverability of the data you've loaded as doing an INSERT /*+ APPEND */ on a NOLOGGING table: if the instance crashes, the data is gone. It's a good idea to take a backup immediately after doing an UNRECOVERABLE load.

Direct loads are not the only way to achieve speed. SQL*Loader also provides parameters to optimize conventional path loading, and while you cannot execute conventional path loads in parallel, you can execute multiple conventional loads to the same table concurrently.

Conclusion

In this article, we've seen how to use SQL*Loader to transform and load many kinds of data, from many different sources, into Oracle tables.

While an Oracle Gateway license may make sense for heterogeneous systems exchanging real-time data, and third-party solutions may make sense in some contexts for massive recurring data loads, there's no question that SQL*Loader is the best tool for one-time or recurring small- to moderately-sized heterogeneous data loads. I encourage anyone who's not familiar with SQL*Loader to step through a few of the case studies in this article. Once you've used SQL*Loader a few times, you'll wonder how you ever lived without it.

Natalka Roshak – Natalka is a Senior Oracle Database Administrator and an Oracle Certified Professional in Database Administration. She provides expert database consulting solutions across North America from her base in Southern Ontario. Natalka may be contacted at Natalka.Roshak@ERPtips.com.

ORAtips Journal