12

SQL*Loader Case Studies

The case studies in this chapter illustrate some of the features of SQL*Loader. These case studies start simply and progress in complexity.

> **Note:** The commands used in this chapter, such as `sqlldr`, are UNIX-specific invocations. Refer to your Oracle operating system-specific documentation for information about the correct commands to use on your operating system.

This chapter contains the following sections:

The Case Studies

This chapter contains the following case studies:

- Case Study 1: Loading Variable-Length Data on page 12-5: Loads stream format records in which the fields are terminated by commas and may be enclosed by quotation marks. The data is found at the end of the control file.
- Case Study 2: Loading Fixed-Format Fields on page 12-8: Loads data from a separate datafile.
- Case Study 3: Loading a Delimited, Free-Format File on page 12-11: Loads data from stream format records with delimited fields and sequence numbers. The data is found at the end of the control file.
- Case Study 4: Loading Combined Physical Records on page 12-14: Combines multiple physical records into one logical record corresponding to one database row.
- Case Study 5: Loading Data into Multiple Tables on page 12-18: Loads data into multiple tables in one run.
- Case Study 6: Loading Data Using the Direct Path Load Method on page 12-24: Loads data using the direct path load method.
- Case Study 7: Extracting Data from a Formatted Report on page 12-28: Extracts data from a formatted report.
- Case Study 8: Loading Partitioned Tables on page 12-34: Loads partitioned tables.
- Case Study 9: Loading LOBFILEs (CLOBs) on page 12-38: Adds a `CLOB` column called `resume` to the table `emp`, uses a `FILLER` field (`res_file`), and loads multiple LOBFILEs into the `emp` table.
- Case Study 10: Loading REF Fields and VARRAYs on page 12-43: Loads a customer table that has a primary key as its OID and stores order items in a VARRAY. Loads an order table that has a reference to the customer table and the order items in a VARRAY.

- Case Study 11: Loading Data in the Unicode Character Set on page 12-47: Loads data in the Unicode character set, UTF16, in little-endian byte order. This case study uses character-length semantics.

Case Study Files

The distribution media for SQL*Loader contains files for each case:

- Control files (for example, `ulcase5.ctl`)
- Datafiles (for example, `ulcase5.dat`)
- Setup files (for example, `ulcase5.sql`)

If the sample data for the case study is contained in the control file, then there will be no `.dat` file for that case.

If there are no special setup steps for a case study, there may be no `.sql` file for that case. Starting (setup) and ending (cleanup) scripts are denoted by an S or E after the case number.

Table 12–1 lists the files associated with each case.

Table 12–1 Case Studies and Their Related Files

Case	`.ctl`	`.dat`	`.sql`
1	Yes	No	Yes
2	Yes	Yes	No
3	Yes	No	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	Yes	Yes	Yes
7	Yes	Yes	Yes (S, E)
8	Yes	Yes	Yes
9	Yes	Yes	Yes
10	Yes	No	Yes
11	Yes	Yes	Yes

Tables Used in the Case Studies

The case studies are based upon the standard Oracle demonstration database tables, `emp` and `dept`, owned by `scott/tiger`. (In some case studies, additional columns have been added.)

Contents of Table emp

```
(empno          NUMBER(4) NOT NULL,
 ename          VARCHAR2(10),
 job            VARCHAR2(9),
 mgr            NUMBER(4),
 hiredate       DATE,
 sal            NUMBER(7,2),
 comm           NUMBER(7,2),
 deptno         NUMBER(2))
```

Contents of Table dept

```
(deptno         NUMBER(2) NOT NULL,
 dname          VARCHAR2(14),
 loc            VARCHAR2(13))
```

Checking the Results of a Load

To check the results of a load, start SQL*Plus and perform a select operation from the table that was loaded in the case study. This is done, as follows:

1. Start SQL*Plus as `scott/tiger` by entering the following at the system prompt:

   ```
   sqlplus scott/tiger
   ```

   The SQL prompt is displayed.

2. At the SQL prompt, use the `SELECT` statement to select all rows from the table that the case study loaded. For example, if the table `emp` was loaded, enter:

   ```
   SQL> SELECT * FROM emp;
   ```

   The contents of each row in the `emp` table will be displayed.

References and Notes

The summary at the beginning of each case study directs you to the sections of this guide that discuss the SQL*Loader feature being demonstrated.

In the control file fragment and log file listing shown for each case study, the numbers that appear to the left are not actually in the file; they are keyed to the numbered notes following the listing. Do not use these numbers when you write your control files.

Case Study 1: Loading Variable-Length Data

Case 1 demonstrates:

- A simple control file identifying one table and three columns to be loaded.
- Including data to be loaded from the control file itself, so there is no separate datafile. See Identifying Data in the Control File with BEGINDATA on page 8-11.
- Loading data in stream format, with both types of delimited fields: terminated and enclosed. See Specifying Field Length for Datatypes for Which Whitespace Can Be Trimmed on page 9-47.

Control File for Case Study 1

The control file is `ulcase1.ctl`:

```
1)   LOAD DATA
2)   INFILE *
3)   INTO TABLE dept
4)   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
5)   (deptno, dname, loc)
6)   BEGINDATA
   12,RESEARCH,"SARATOGA"
   10,"ACCOUNTING",CLEVELAND
   11,"ART",SALEM
   13,FINANCE,"BOSTON"
   21,"SALES",PHILA.
   22,"SALES",ROCHESTER
   42,"INT'L","SAN FRAN"
```

Notes:

1. The LOAD DATA statement is required at the beginning of the control file.

2. INFILE * specifies that the data is found in the control file and not in an external file.
3. The INTO TABLE statement is required to identify the table to be loaded (dept) into. By default, SQL*Loader requires the table to be empty before it inserts any records.
4. FIELDS TERMINATED BY specifies that the data is terminated by commas, but may also be enclosed by quotation marks. Datatypes for all fields default to CHAR.
5. The names of columns to load are enclosed in parentheses. Because no datatype or length is specified, the default is type CHAR with a maximum length of 255.
6. BEGINDATA specifies the beginning of the data.

Running Case Study 1

Take the following steps to run the case study.

1. Start SQL*Plus as scott/tiger by entering the following at the system prompt:

   ```
   sqlplus scott/tiger
   ```

   The SQL prompt is displayed.

2. At the SQL prompt, execute the SQL script for this case study, as follows:

   ```
   SQL> @ulcase1
   ```

   This prepares and populates tables for the case study and then returns you to the system prompt.

3. At the system prompt, invoke SQL*Loader and run the case study, as follows:

   ```
   sqlldr USERID=scott/tiger CONTROL=ulcase1.ctl LOG=ulcase1.log
   ```

   SQL*Loader loads the dept table, creates the log file, and returns you to the system prompt. You can check the log file to see the results of running the case study.

Log File for Case Study 1

The following shows a portion of the log file:

```
Control File:   ulcase1.ctl
```

```
Data File:      ulcase1.ctl
  Bad File:     ulcase1.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 DEPT, loaded from every logical record.
Insert option in effect for this table: INSERT

   Column Name                  Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------------------
1) DEPTNO                            FIRST     *   ,  O(") CHARACTER
   DNAME                              NEXT     *   ,  O(") CHARACTER
2) LOC                                NEXT     *   ,  O(") CHARACTER


Table DEPT:
  7 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:                  49536 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0
.
.
.
Elapsed time was:     00:00:01.53
CPU time was:         00:00:00.20
```

Notes:

1. Position and length for each field are determined for each record, based on delimiters in the input file.
2. The notation O(') signifies optional enclosure by quotation marks.

Case Study 2: Loading Fixed-Format Fields

Case 2 demonstrates:

- A separate datafile. See Specifying Datafiles on page 8-8.
- Data conversions. See Datatype Conversions on page 9-24.

In this case, the field positions and datatypes are specified explicitly.

Control File for Case Study 2

The control file is `ulcase2.ctl`.

```
1)   LOAD DATA
2)   INFILE 'ulcase2.dat'
3)   INTO TABLE emp
4)   (empno         POSITION(01:04)   INTEGER EXTERNAL,
       ename          POSITION(06:15)   CHAR,
       job            POSITION(17:25)   CHAR,
       mgr            POSITION(27:30)   INTEGER EXTERNAL,
       sal            POSITION(32:39)   DECIMAL EXTERNAL,
       comm           POSITION(41:48)   DECIMAL EXTERNAL,
5)   deptno         POSITION(50:51)   INTEGER EXTERNAL)
```

Notes:

1. The `LOAD DATA` statement is required at the beginning of the control file.
2. The name of the file containing data follows the `INFILE` parameter.
3. The `INTO TABLE` statement is required to identify the table to be loaded into.
4. Lines 4 and 5 identify a column name and the location of the data in the datafile to be loaded into that column. `empno`, `ename`, `job`, and so on are names of columns in table `emp`. The datatypes (`INTEGER EXTERNAL`, `CHAR`, `DECIMAL EXTERNAL`) identify the datatype of data fields in the file, not of corresponding columns in the `emp` table.
5. Note that the set of column specifications is enclosed in parentheses.

Datafile for Case Study 2

The following are a few sample data lines from the file `ulcase2.dat`. Blank fields are set to null automatically.

```
7782 CLARK      MANAGER   7839  2572.50          10
7839 KING       PRESIDENT       5500.00          10
7934 MILLER     CLERK     7782   920.00          10
7566 JONES      MANAGER   7839  3123.75          20
7499 ALLEN      SALESMAN  7698  1600.00   300.00 30
7654 MARTIN     SALESMAN  7698  1312.50  1400.00 30
7658 CHAN       ANALYST   7566  3450.00          20
7654 MARTIN     SALESMAN  7698  1312.50  1400.00 30
```

Running Case Study 2

Take the following steps to run the case study. If you have already run case study 1, you can skip to Step 3 because the `ulcase1.sql` script handles both case 1 and case 2.

1. Start SQL*Plus as `scott/tiger` by entering the following at the system prompt:

   ```
   sqlplus scott/tiger
   ```

   The SQL prompt is displayed.

2. At the SQL prompt, execute the SQL script for this case study, as follows:

   ```
   SQL> @ulcase1
   ```

   This prepares and populates tables for the case study and then returns you to the system prompt.

3. At the system prompt, invoke SQL*Loader and run the case study, as follows:

   ```
   sqlldr USERID=scott/tiger CONTROL=ulcase2.ctl LOG=ulcase2.log
   ```

   SQL*Loader loads the table, creates the log file, and returns you to the system prompt. You can check the log file to see the results of running the case study.

   Records loaded in this example from the `emp` table contain department numbers. Unless the `dept` table is loaded first, referential integrity checking rejects these records (if referential integrity constraints are enabled for the `emp` table).

Log File for Case Study 2

The following shows a portion of the log file:

```
Control File:   ulcase2.ctl
Data File:      ulcase2.dat
  Bad File:     ulcase2.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 EMP, loaded from every logical record.
Insert option in effect for this table: INSERT

   Column Name                  Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------------------
EMPNO                              1:4     4           CHARACTER
ENAME                             6:15    10           CHARACTER
JOB                              17:25     9           CHARACTER
MGR                              27:30     4           CHARACTER
SAL                              32:39     8           CHARACTER
COMM                             41:48     8           CHARACTER
DEPTNO                           50:51     2           CHARACTER


Table EMP:
  7 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:                  3840 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0
```

```
.
.
.
Elapsed time was:     00:00:00.81
CPU time was:         00:00:00.15
```

Case Study 3: Loading a Delimited, Free-Format File

Case 3 demonstrates:

- Loading data (enclosed and terminated) in stream format. See Delimited Fields on page 9-30.
- Loading dates using the datatype DATE. See Datetime and Interval Datatypes on page 9-16.
- Using SEQUENCE numbers to generate unique keys for loaded data. See Setting a Column to a Unique Sequence Number on page 9-60.
- Using APPEND to indicate that the table need not be empty before inserting new records. See Table-Specific Loading Method on page 8-34.
- Using Comments in the control file set off by two hyphens. See Comments in the Control File on page 8-4.

Control File for Case Study 3

This control file loads the same table as in case 2, but it loads three additional columns (`hiredate`, `projno`, and `loadseq`). The demonstration table `emp` does not have columns `projno` and `loadseq`. To test this control file, add these columns to the `emp` table with the command:

```
ALTER TABLE emp ADD (projno NUMBER, loadseq NUMBER);
```

The data is in a different format than in case 2. Some data is enclosed in quotation marks, some is set off by commas, and the values for `deptno` and `projno` are separated by a colon.

```
1)   -- Variable-length, delimited, and enclosed data format
     LOAD DATA
2)   INFILE *
3)   APPEND
     INTO TABLE emp
4)   FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
     (empno, ename, job, mgr,
```

```
5)   hiredate DATE(20) "DD-Month-YYYY",
   sal, comm, deptno CHAR TERMINATED BY ':',
   projno,
6)   loadseq  SEQUENCE(MAX,1))
7)   BEGINDATA
8)   7782, "Clark", "Manager", 7839, 09-June-1981, 2572.50,,  10:101
   7839, "King", "President", , 17-November-1981,5500.00,,10:102
   7934, "Miller", "Clerk", 7782, 23-January-1982, 920.00,, 10:102
   7566, "Jones", "Manager", 7839, 02-April-1981, 3123.75,, 20:101
   7499, "Allen", "Salesman", 7698, 20-February-1981, 1600.00,
   (same line continued)                 300.00, 30:103
   7654, "Martin", "Salesman", 7698, 28-September-1981, 1312.50,
   (same line continued)               1400.00, 3:103
   7658, "Chan", "Analyst", 7566, 03-May-1982, 3450,,  20:101
```

Notes:

1. Comments may appear anywhere in the command lines of the file, but they should not appear in data. They are preceded with two hyphens that may appear anywhere on a line.
2. `INFILE *` specifies that the data is found at the end of the control file.
3. `APPEND` specifies that the data can be loaded even if the table already contains rows. That is, the table need not be empty.
4. The default terminator for the data fields is a comma, and some fields may be enclosed by double quotation marks (").
5. The data to be loaded into column `hiredate` appears in the format DD-Month-YYYY. The length of the date field is specified to have a maximum of 20. The maximum length is in bytes, with default byte-length semantics. If character-length semantics were used instead, the length would be in characters. If a length is not specified, then the length depends on the length of the date mask.
6. The `SEQUENCE` function generates a unique value in the column `loadseq`. This function finds the current maximum value in column `loadseq` and adds the increment (1) to it to obtain the value for `loadseq` for each row inserted.
7. `BEGINDATA` specifies the end of the control information and the beginning of the data.
8. Although each physical record equals one logical record, the fields vary in length, so that some records are longer than others. Note also that several rows have null values for `comm`.