Practice

Using SQL Loader

Practice Target

In this practice, you will implement multiple data load scenarios on using SQL*Loader to load data files into Oracle database.

Practice Overview

In this practice, you will perform the following tasks:

- Load a data file with relatively positioned columns based on delimiters
- Load a data file with fixed positioning data
- Load a data file into multiple tables
- Install Oracle Database Examples

Assumptions

- This practice assumes that `srv1` is up and running from the **CDB** snapshot.

A. Loading a data file with relatively positioned columns

In this section of the practice, you will load a data file with relatively positioned columns based on delimiters.

1. Open Putty and connect to `srv1` as `oracle`.

2. Create a directory to save the practice files in it. Change the current directory to it.

```
mkdir ~/sqlldr
cd ~/sqlldr
```

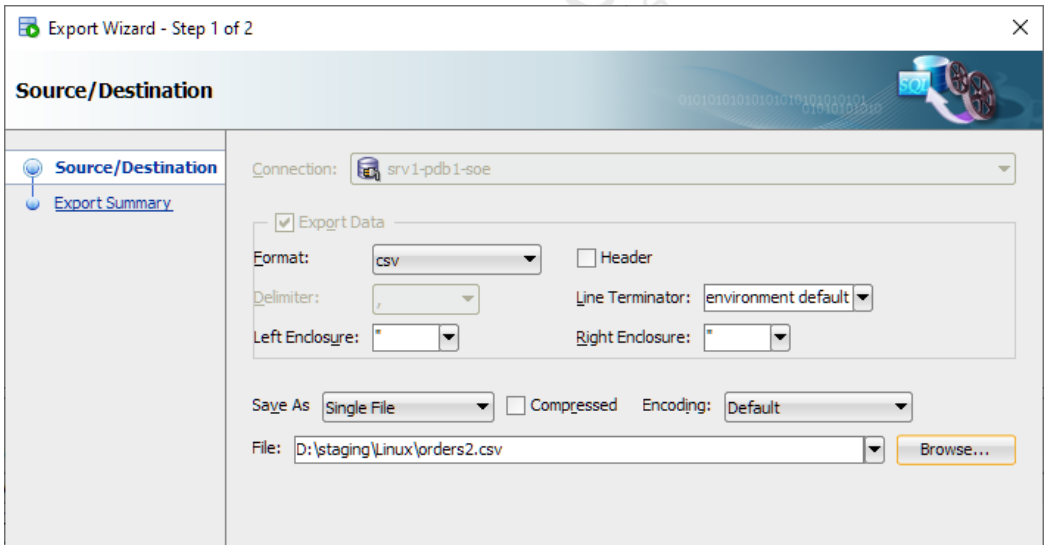
3. Start SQL Developer and login to `PDB1` as `SOE`.

4. Submit the following query to retrieve list of the records to be saved in an external file. It returns a few order records.

```
SELECT ORDER_ID, TO_CHAR(ORDER_DATE,'DD-MM-RRRR HH24:MI:SS') ORDER_DATE, CUSTOMER_ID,
ORDER_STATUS, DELIVERY_TYPE, ORDER_TOTAL FROM ORDERS FETCH FIRST 20 ROWS ONLY;
```

5. Export the output of the query into a CSV file. Save the file in the shared folder.

In SQL Developer, right-click on the data table > **Export** > Fill in the Wizard window as follows to save the CSV file into the staging folder > **Next** > **Finish**



6. Move the CSV file to the practice directory files.

```
mv /media/sf_staging/orders2.csv ~/sqlldr
```

7. Invoke SQL*Plus with connecting to `PDB1` as `SOE`.

```
sqlplus soe/ABcd##1234@pdb1
```

8. Run the following statement to create the destination table.

```
CREATE TABLE ORDERS2 (
 ORDER_ID               NUMBER(12)   NOT NULL,
 ORDER_DATE             DATE ,
 CUSTOMER_ID            NUMBER(12) NOT NULL,
 ORDER_STATUS           NUMBER(2),
 DELIVERY_TYPE          VARCHAR2(15),
 ORDER_TOTAL            NUMBER(8,2)
)
/
```

9. Exit from SQL*Plus

```
quit
```

10. Create the control file as follows:

Observe how we set the format of the Date data type.

Observe that we do not define the datatype of the other columns in this example. In this case, SQL*Loader handles the data in those columns as character and in the database side the data is internally converted into the database column data type.

```
vi ~/sqlldr/orders2.ctl
```

```
LOAD DATA
INFILE 'orders2.csv'
TRUNCATE
INTO TABLE ORDERS2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
( ORDER_ID,
  ORDER_DATE DATE(21) 'DD-MM-YYYY HH24:MI:SS',
  CUSTOMER_ID,
  ORDER_STATUS,
  DELIVERY_TYPE,
  ORDER_TOTAL)
```

11. Using SQL*Loader, try loading `orders2.csv` into the database.

The utility should return the message "`0 Rows successfully loaded.`", which means the load attempt failed.

```
sqlldr soe/ABcd##1234@pdb1 control=orders2.ctl log=orders2.log
```

12. Check if the bad file is created.

Bad files are automatically created even if the bad parameter is not set. However, discard files do not get created unless you explicitly set it.

```
ls ~/sqlldr/
```

13. Check on the contents of the log file to check the root cause of the failure.

You should observe the following error reported for all the records:

```
Record xx: Rejected - Error on table ORDERS2, column ORDER_TOTAL.
ORA-01722: invalid number
```

The data in the ORDER_TOTAL is numeric. However, in CSV files, the lines are terminated by a carriage return and a line feed characters. By default, SQL*Loader in Linux considers the line feed only as a record terminator. It loads the carriage return character with the data. You will fix this issue in the incoming steps.

Note: SQL*Loader has a control file setting to handle CSV files. We did not use it from the beginning because we wanted to demonstrate handling the errors in SQL*Loader.

Note: If you want to see the carriage return characters in the loading data, change the datatype of ORDER_TOTAL to character. The load will be successful. Then check the contents of the ORDER_TOTAL column using the ASCII function.

```
cat orders2.log
```

14. Modify the control file as follows:

\r corresponds to a carriage return character and \n corresponds to the line feed character

```
vi orders2.ctl
```

```
LOAD DATA
INFILE 'orders2.csv' "str '\r\n'"
TRUNCATE
INTO TABLE ORDERS2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
( ORDER_ID,
  ORDER_DATE DATE(21) 'DD-MM-YYYY HH24:MI:SS',
  CUSTOMER_ID,
  ORDER_STATUS,
  DELIVERY_TYPE,
  ORDER_TOTAL)
```

15. Try loading the data file now.

The load should succeed.

```
sqlldr soe/ABcd##1234@pdb1 control=orders2.ctl log=orders2.log
```

16. In the SQL Developer, verify the data successfully loaded into the table.

```
SELECT * FROM ORDERS2;
```

Cleanup

17. Drop the ORDERS2 table.

```
DROP TABLE ORDERS2 PURGE;
```

18. Delete the files in the temporary directory.

```
host rm ~/sqlldr/*
```

B. Loading a data file with fixed positioning data

In this section of the practice, you will load a data file with fixed positioning data.

19. Submit the following query to generate sample data.

Make sure the Putty window is wide enough to display the entire row width.

```
conn soe/ABcd##1234@pdb1

set LINESIZE 150
col CUST_NAME for a30
col EMAIL for a30
SELECT CUSTOMER_ID, INITCAP(CUST_FIRST_NAME) || ' ' || INITCAP(CUST_LAST_NAME) CUST_NAME,
CUST_EMAIL EMAIL, TO_CHAR(ROUND(DBMS_RANDOM.VALUE(100000,2000000),-2),'$999,999,999')
TOTAL_SALES
FROM CUSTOMERS FETCH FIRST 10 ROWS ONLY;
```

20. In Putty window, highlight all the returned rows including the headers. This automatically saves the records in the clipboard.

21. Create a new file and paste the contents of the clipboard in it. Save the file as `customers.dat`. We have now a fixed positioning data in the file.

```
Host vi customers2.dat
```

22. Create the destination table.

```
CREATE TABLE CUSTOMERS2 (
 CUSTOMER_ID NUMBER(12) NOT NULL,
 CUST_NAME   VARCHAR2(81),
 EMAIL       VARCHAR2(100),
 TOTAL_SALES NUMBER(9),
 LOADSEQ NUMBER(4)
)
/
```

23. Exit from SQL*Plus

```
quit
```

24. Create the control file as follows.

Observe the following about this control file:

- It uses the keyword APPEND which means the current data in the table is not touched by the SQL*Loader.
- The customer name is converted into upper case using the UPPER function.
- The `TOTAL_SALES` is converted into `NUMBER` using the `TO_NUMBER` function.
- `LOADSEQ` is assigned to the function `SEQUENCE(MAX,1)`. This means that SQL*Loader will retrieve the current maximum `LOADSEQ` value in the `CUSTOMERS2` table. It then fills the `LOADSEQ` with sequence values starting from the maximum value plus one. The assigned values are incremented by 1.

```
vi customers2.ctl
```

```
LOAD DATA
INFILE 'customers2.dat'
APPEND
INTO TABLE CUSTOMERS2
( CUSTOMER_ID POSITION(01:11) INTEGER EXTERNAL,
  CUST_NAME   POSITION(13:42) CHAR "UPPER(:CUST_NAME)",
  EMAIL       POSITION(44:73) CHAR,
  TOTAL_SALES POSITION(75:88) CHAR "TO_NUMBER(:TOTAL_SALES,'$999,999,999')",
  LOADSEQ     SEQUENCE(MAX,1)
)
```

25. Try loading the data file now. The load should succeed.

Observe that we use SKIP=2 parameter because the first two rows are headers.

```
sqlldr soe/ABcd##1234@pdb1 control=customers2.ctl log=customers2.log skip=2
```

26. In the SQL Developer, verify the data successfully loaded into the table.

```
SELECT * FROM CUSTOMERS2;

exit
```

27. Try loading the same data file again.

The load should succeed.

```
sqlldr soe/ABcd##1234@pdb1 control=customers2.ctl log=customers2.log
```

28. In the SQL Developer, verify the data successfully loaded again into the table. You should see 20 rows.

Some gaps were observed in the sequential numbers.

```
SELECT * FROM CUSTOMERS2;
```

Cleanup

29. Drop the `CUSTOMERS2` table.

```
DROP TABLE CUSTOMERS2 PURGE;
```

30. Delete the temporary directory.

```
host rm ~/sqlldr/*
```

C. Loading a data file into multiple tables

In this section of the practice, you will load a data file into multiple tables in the database.

31. In Putty session, create a sample data file as follows. Copy the data from the attached file **`students.dat`**. Do not copy it from the PDF file. You should find the file in the lecture downloadable resources.

This is a sample data of records delimited by TAB character. These records represent students and the course codes they enrolled into. A student might be enrolled into one, two or three courses. The last three columns are the courses that each student is enrolled into. If a student is enrolled into a less-than-three courses, the remaining columns equals to the characters 'NULL'.

Note: In SQL*Loader, it always better to replace a blank or empty field value with special phrases like 'NULL' or 'EMPTY'.

```
vi students.dat
```

```
# do not copy the data from the PDF file. Copy it from the attached file.
1021      Elissa Brown   C002    NULL    NULL
2011      Charlotte Wilson      C101    C121    C001
1875      Isabella Davies       C102    NULL    NULL
1924      Kumar Sujan    C003    C001    NULL
2115      Shah Mohamed   C005    C003    NULL
2112      Scott Morison  C102    NULL    NULL
1001      Peter Johnson  C001    C002    C004
```

32. Invoke SQL*Plus with connecting to `PDB1` as `SOE`

```
sqlplus soe/ABcd##1234@pdb1
```

33. Create the two destination tables as follows:

```
CREATE TABLE STUDENTS
( STUDENT_ID NUMBER(4) PRIMARY KEY,
  STUDENT_NAME VARCHAR2(30)
)
/

CREATE TABLE STUDENT_COURSES
(STUDENT_ID NUMBER(4) REFERENCES STUDENTS(STUDENT_ID),
 COURSE_CODE VARCHAR2(4) NOT NULL)
/
```

34. Exit from SQL*Plus.

```
quit
```

35. Create the control file as follows:

Observe the following about this control file:

- `INTO TABLE` is used four times in the file. The first time is used to insert students data into the `STUDENTS` table. The other three times are used to insert the courses into the `STUDENT_COURSES` table.
- `POSITION(1)` is used to set the reader pointer back to the first column in the input record.
- In the `INTO TABLE` sections for loading the student courses, we do not need to save the `STUDENT_NAME`, that is why it is marked as `FILLER` in those sections.
- In the second `INTO TABLE STUDENT_COURSES` section, the first course code field is already loaded by the first `INTO TABLE STUDENT_COURSES` section. That is why the first course code field is marked as `FILLER` in that section.
- The hexadecimal of the `TAB` character is 9. That is why `TERMINATED BY X'9'` is used in this control file.
- The `FIELDS TERMINATED BY X'9'` is used at the table level only in the first `INTO TABLE` section. It is used at the column level in the remaining `INTO TABLE` sections.

```
vi students.ctl
```

```
LOAD DATA
INFILE 'students.dat'
REPLACE

INTO TABLE STUDENTS
-- fields terminated by TAB
FIELDS TERMINATED BY X'9'
( STUDENT_ID POSITION(*),
  STUDENT_NAME POSITION(*))

INTO TABLE STUDENT_COURSES
-- WHEN cannot check if a field is null
WHEN COURSE_CODE != 'NULL'
( STUDENT_ID POSITION(1) TERMINATED BY X'9'  ,
  STUDENT_NAME FILLER POSITION(*) TERMINATED BY X'9',
  COURSE_CODE TERMINATED BY X'9')

INTO TABLE STUDENT_COURSES
WHEN COURSE_CODE != 'NULL'
( STUDENT_ID POSITION(1) TERMINATED BY X'9'  ,
  STUDENT_NAME FILLER POSITION(*) TERMINATED BY X'9',
  COURSE1 FILLER POSITION(*) TERMINATED BY X'9',
  COURSE_CODE TERMINATED BY X'9')

INTO TABLE STUDENT_COURSES
WHEN COURSE_CODE != 'NULL'
( STUDENT_ID POSITION(1) TERMINATED BY X'9'  ,
  STUDENT_NAME FILLER POSITION(*) TERMINATED BY X'9',
  COURSE1 FILLER POSITION(*) TERMINATED BY X'9',
  COURSE2 FILLER POSITION(*) TERMINATED BY X'9',
  COURSE_CODE TERMINATED WHITESPACE)
```

36. Load the data into the database tables.

You should observe that 7 students were loaded, all of them enrolled on at least one course, 4 of them enrolled on at least two courses, and 2 of them enrolled into three courses. Just as expected and required.

```
sqlldr soe/ABcd##1234@pdb1 control=students.ctl log=students.log
```

37. In SQL Developer session, check the contents of the tables.

```
SELECT * FROM STUDENTS;
SELECT * FROM STUDENT_COURSES;
```

Cleanup

38. Drop the created tables.

```
DROP TABLE STUDENT_COURSES PURGE;
DROP TABLE STUDENTS PURGE;
```

39. Delete the files in the temporary directory.

```
rm *
cd ..
rmdir sqlldr
```

D. Installing Oracle Database Examples

In this section of the practice, you will install Oracle Database Examples to check on the sample files created by this product for using SQL*Loader (In Oracle documentation, they are called SQL*Loader Case Study files).

Note: If your sole target is to look at the SQL*Loader sample files, then you do not have to download and install Oracle Database Examples file (Its size is nearly 850M). I have attached the SQL*Loader Case Study files to this lecture downloadable resources in the file **ulcases.zip**. You can (if you want to) download the file, examine its contents and skip implementing this section.

40. Download Oracle Database 19c Examples from this link.

41. Copy the file to the staging directory.

42. Change the current directory to the staging directory then extract the file into `oracle` home. The extraction creates a directory called `examples`

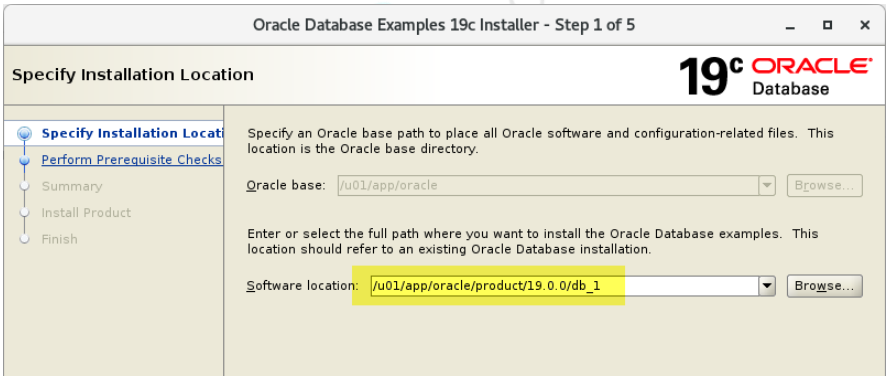
```
cd /media/sf_staging/
unzip LINUX.X64_193000_examples.zip -d ~
```

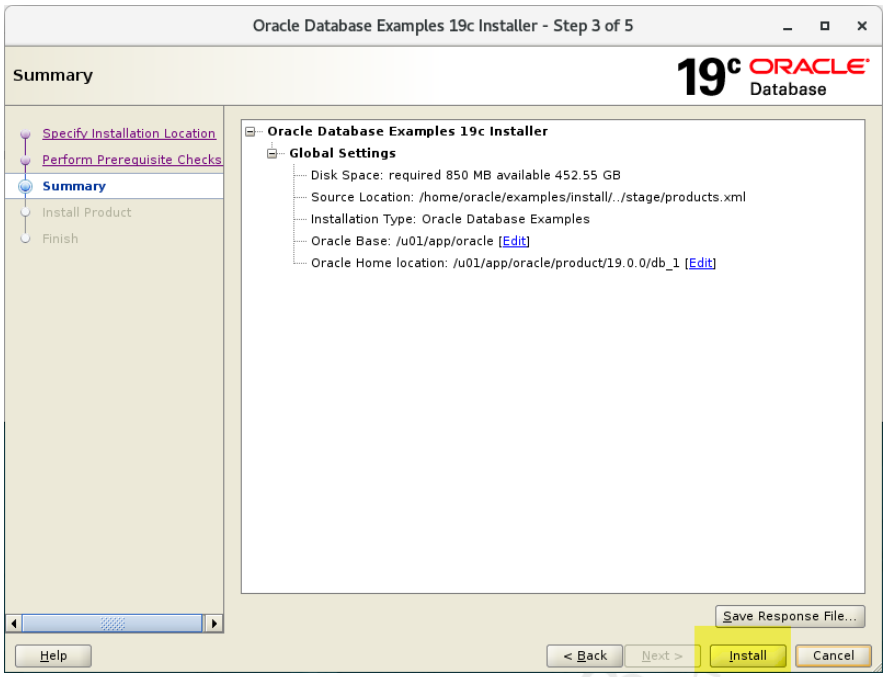
43. In the VirtualBox window of `srv1`, login to the machine as `oracle` and open a terminal window.

44. Change the current directory to the `examples` directory and invoke the installer.

```
cd ~/examples
./runInstaller
```

45. Respond to the Wizard as follows.





46. After the installation is finished, in the Putty session, verify that the SQL*Loader Case Study files are installed. There are 11 sample case studies in the directory.

```
cd $ORACLE_HOME/rdbms/demo
ls ulcase*
```

47. Go through the control file (*.ctl) of each Case Study and examine their contents to learn about what they demonstrate about.

In my opinion, they are the best point to start learning about using SQL*Loader. They demonstrate the common scenarios of using SQL*Loader, like the ones that you learnt in this lecture.

Cleanup

48. Delete the installation files.

```
rm -rf ~/examples
```

49. Exit from sessions connected to `srv1` (Putty, SQL Developer, and VirtualBox).

50. Create a new snapshot for `srv1` and delete the old snapshot.

Summary

- SQL*Loader has the capability of loading the data files with relatively positioned columns based on delimiters or with fixed positioning data.
- SQL*Loader has the capability of loading logical records into different tables with basic transformation and auto-filling features.
- By installing Oracle database examples, we can install SQL*Loader case study files to learn on how to setup SQL*Loader control files for different scenarios.

Note:

Not all the SQL*Loader features have been demonstrated in this lecture. To learn more about SQL*Loader capabilities, refer to the documentation "Oracle Database Utilities".