Kabeer Ahmed
SE-19028

Database Management System (DBMS – 204)

Experiment # 07

Manipulating Data

Student Name: Kabeer Ahmed

Roll Number: SE-19028

Maximum Marks	**Performance = 05**	**Viva = 05**	**Total = 10**
Marks Obtained			
Remarks (if any)			

Experiment evaluated by

Instructor Name: Engr. Adiba Jafar

Signature and Date:

Kabeer Ahmed
SE-19028

Outcome

After completing this lesson, you should be able to do the following:

1. Describe each DML statement
2. Insert rows into a table
3. Update rows in a table
4. Delete rows from a table
5. Merge rows in a table
6. Control transactions

Data Manipulation Language

A DML statement is executed when you:

a. Add new rows to a table
b. Modify existing rows in a table
c. Remove existing rows from a table

A transaction consists of a collection of DML statements that form a logical unit of work.

Adding a New Row to a Table

Insert a new row into the DEPT table.

The INSERT Statement Syntax

a. Add new rows to a table by using the INSERT statement.

INSERT INTO table[(column[, column...])] VALUES(value[, value...]);

b. Only one row is inserted at a time with this syntax.
c. Insert a new row containing values for each column.
d. List values in the default order of the columns in the table.
e. Optionally, list the columns in the INSERT clause.
f. Enclose character and date values within single quotation marks.

INSERT INTO dept(deptno, dname,,loc)   VALUES (70, 'Public ', 'NEWYORK');

Inserting Rows with Null Values

a. Implicit method: Omit the column from the column list.
   INSERT INTO dept (deptno,dname   ) VALUES (30, 'Purchasing');

b. Explicit method: Specify the NULL keyword in the VALUES clause.
   INSERT INTO dept VALUES (10, 'Finance', NULL);

Inserting Special Values

The SYSDATE function records the current date and time.

INSERT INTO emp (empno,ename, hiredate, job, sal,comm, mgr,deptno)
VALUES (113, 'LouisPopp' , SYSDATE,'ACCOUNT', 6900, NULL, 205, 10);

Confirming Additions to the Table

SELECT empno, ename, job, hiredate, comm FROM   emp
WHERE empno = 113;

Kabeer Ahmed
SE-19028

Inserting Specific Date Values

•Add a new employee.
INSERT INTO emp VALUES(114,'Den','ACCOUNT',1100,
TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),2000, NULL, 30);
INSERT INTO dept(deptno, dname, loc) VALUES (&deptno, ‘&dname’,’&loc’);

Copying Rows from Another Table

Write your INSERT statement with a subquery.
Create table new_emp as(select * from emp);
INSERT INTO new_emp
(SELECT empno, ename, job,mgr,hiredate,sal, comm,deptno FROM emp WHERE job LIKE '%MAN%');
•Do not use the VALUES clause.
• Match the number of columns in the INSERT clause to those in the subquery.

INSERT INTO new_emp SELECT * FROM emp;

The UPDATE Statement Syntax

1. Modify existing rows with the UPDATE statement.

UPDATE *table* SET *column* = *value* [, *column* = *value, ...*]
[WHERE *condition*];

2. Update more than one row at a time, if required.

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMP table by name is dangerous, because more than one employee may have the same name.

Updating Rows in a Table

Specific row or rows are modified if you specifyThe WHERE clause.
UPDATE emp
SET deptno = 70 WHERE empno = 7499;

• All rows in the table are modified if you omit the WHERE clause.
UPDATE new_emp SET dept = 20;

Updating Two Columns with a Subquery

UPDATE new_emp SET job = (SELECT job FROM emp WHERE empno = 7499),
sal = (SELECT sal FROM emp WHERE empno = 7521)
WHERE empno = 114;

Updating Rows Based on Another Table

Use subqueries in UPDATE statements to update
rows in a table based on values from another table.
UPDATE new_emp SET deptno = (SELECT deptno
FROM emp WHERE empno= 100) WHERE job = (SELECT job
FROM emp WHERE empno = 7486);

Kabeer Ahmed
SE-19028

Updating Rows:

Integrity Constraint Error

```
UPDATE emp SET    deptno = 55 WHERE deptno= 110;
UPDATE emp
*
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

Integrity Constraint Error

Removing a Row from a Table

The DELETE Statement

You can remove existing rows from a table by using The DELETE statement.

```
DELETE [FROM] table [WHERE condition ];
```

Note:If no rows are deleted, a message “ 0 rows deleted .” is returned:

Deleting Rows from a Table

•Specific rows are deleted if you specify the WHERE clause.

```
DELETE FROM dept
WHERE dname = 'Finance';
1 row deleted.
```

•All rows in the table are deleted if you omit the WHERE clause.

```
DELETE FROM  new_emp;
22 rows deleted.
```

Example

Remove rows identified in the WHERE clause.

```
DELETE FROM  emp WHERE       empno = 113;
DELETE FROM  dept WHERE      deptno IN (30, 40);
```

Deleting Rows Based on Another Table

Use subqueries in DELETE statements to remove rows from a table based on values from another table.

```
DELETE FROM  emp  WHERE   deptno = (SELECT deptno FROM    dept WHERE dname
LIKE '%Public%');
1 row deleted.
```

Deleting Rows:

```
DELETE FROM dept WHERE      deptno = 60;


DELETE FROM  dept WHERE      deptno = 70;
1 row deleted.
```

Using a Subquery in an

INSERT Statement

```
INSERT INTO(SELECT empno, ename,hiredate, job, sal,deptno
FROM  emp
WHERE deptno= 50)
VALUES (99999, 'Taylor', TO_DATE('07-JUN-99', 'DD-MON-RR'),
```

'ST_CLERK', 5000, 50);
1 row created.

Using a Subquery in an INSERT Statement

•Verify the results
SELECT empno, ename, hiredate, Job, sal, dept
FROM emp WHERE dept = 50;

Using the WITH CHECK OPTION Keyword on DML Statements

•A subquery is used to identify the table and columns of the DML statement.
•The WITH CHECK OPTION keyword prohibits you from changing rows that are not in the subquery.
INSERT INTO (SELECT empno, ename,job,mgr,hiredate, sal,comm,deptno
FROM new_emp
WHERE deptno = 55 WITH CHECK OPTION) VALUES (999, 'SMITH','MANAGER',7839, TO_DATE('09-JUN-81', 'DD-MON-RR'),2450,NULL,55);

The MERGE Statement

•Provides the ability to conditionally update or insert data into a database table
•Performs an
UPDATE
if the row exists and an INSERT if it is a new row:
–Avoids separate updates
–Increases performance and ease of use
–Is useful in data warehousing applications

MERGE Statement Syntax

You can conditionally insert or update rows in a table by using the MERGE statement.
MERGE INTO table_name AS table_alias
USING (table|view|sub_query) AS alias
ON (join condition)
WHEN MATCHED THEN
UPDATE SET
col1 = col_val1,
col2 = col2_val
WHEN NOT MATCHED THEN
INSERT (column_list)
VALUES (column_values);

Merging Rows

Insert or update rows in the NEW_EMP table to match the EMP table.
MERGE INTO new_emp c USING emp e ON (c.empno = e.empno)
WHEN MATCHED THEN UPDATE SET
c.ename   = e.ename,
c.deptno = e.deptno
WHEN NOT MATCHED THEN
INSERT VALUES(e.empno, e.ename, e.job,e.mgr,e.hiredate,e.sal, e.comm,e.deptno);

Merging Rows: Example

MERGE INTO new_emp c USING emp e
ON (c.empno = e.empno) WHEN MATCHED THEN
UPDATE SET

Kabeer Ahmed
SE-19028

```
c.ename     = e.ename,
c.hiredate      = e.hiredate,
c.job       = e.job,
c.sal       = e.sal,
c.comm= e.comm,
c.mgr     = e.mgr,
c.deptno = e.deptno
WHEN NOT MATCHED THEN
INSERT VALUES(e.ename, e, e.hiredate, e.job,e.sal, e.comm, e.mgr,e.deptno);

SELECT * FROM NEW_EMP;

MERGE INTO new_emp c USING emp e
ON (c.empno = e.empno)
WHEN MATCHED THEN
UPDATE SET
...
WHEN NOT MATCHED THEN
INSERT VALUES...;
SELECT *
FROM NEW_EMP;
20 rows selected.
```

Database Transactions

A database transaction consists of one of the following:

- DML statements which constitute one consistent change to the data
- One DDL statement
- One DCL statement

Database Transactions

- Begin when the first DML SQL statement is executed
- End with one of the following events:
  - A COMMIT or ROLLBACK statement is issued
  - A DDL or DCL statement executes (automatic commit)
  - The user exits iSQL*Plus
  - The system crashes

Advantages of COMMIT and ROLLBACK Statements With COMMIT and ROLLBACK

Statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

Controlling Transactions
COMMIT Time
Transaction
DELETE
SAVEPOINT A
INSERT
UPDATE
SAVEPOINT B

Kabeer Ahmed
SE-19028

INSERT
ROLLBACK
ROLLBACK
ROLLBACK
to SAVEPOINT A
to SAVEPOINT B
Note:SAVEPOINT is not ANSI standard SQL.

Rolling Back Changes to a Marker

- Create a marker in a current transaction by using The SAVEPOINT statement.
- Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.
  UPDATE... SAVEPOINT update_done;
- Savepoint created.
    INSERT... ROLLBACK TO update_done;
  Rollback complete.

Implicit Transaction Processing

• An automatic commit occurs under the following circumstances:
  – DDL statement is issued
  – DCL statement is issued
  – Normal exit from iSQL*Plus, without explicitly
    Issuing COMMIT or ROLLBACK statements

•An automatic rollback occurs under an abnormal termination of iSQL*Plus or a system failure.
Implicit Transaction Processing
Status Circumstances

State of the Data Before COMMIT Or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the SELECT statement.
- Other users cannot view the results of the DML statements by the current user.
- The affected rows are locked; other users cannot change the data within the affected rows.

State of the Data After COMMIT

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

Committing Data

•Make the changes.
DELETE FROM new_emp WHERE empno = 7499;

INSERT INTO dept
VALUES (29, 'Corporate Tax', ‘LONDON’);

•Commit the changes.
COMMIT;

Kabeer Ahmed
SE-19028

Commit complete.
Example
Remove departments 29 and 30 in the DEPT table, and update a row in the NEW_EMP table.
Make the data change permanent.
DELETE FROM dept WHERE deptno IN (29, 30);
2 rows deleted.
UPDATE new_emp SET deptno = 80 WHERE empno = 7499;
COMMIT;
Commit Complete.

State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

DELETE FROM new_emp;
ROLLBACK;
Rollback complete.

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle Server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly
  by executing a COMMIT or ROLLBACK statement.

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with changes made by another user.
- Read consistency ensures that on the same data:
    - Readers do not wait for writers
    - Writers do not wait for readers

Implementation of Read Consistency

User A Data
UPDATE emp blocks SET sal = 2000
WHERE ename = 'Goyal';
Undo segments changed and
SELECT *
Read unchanged
FROM userA.emp;
Data consistent before image change: “old” data
User B

Locking

In an Oracle database, locks:

- Prevent destructive interaction between concurrent transactions
- Require no user action
- Use the lowest level of restrictiveness
- Are held for the duration of the transaction
- Are of two types: explicit locking and implicit locking

What Are Locks?

Kabeer Ahmed
SE-19028

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource, either a user object (such as tables or rows) or a system object not visible to users (such as shared data structures and data dictionary rows).

How the Oracle Database Locks Data

Locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Implicit locking occurs for all SQL statements except SELECT.
The users can also lock data manually, which is called explicit locking.

Implicit Locking

•Two lock modes:
- Exclusive: Locks out other users
- Share: Allows other users to access the server

•High level of data concurrency:
- DML: Table share, row exclusive
- Queries: No locks required
- DDL: Protects object definitions

• Locks held until commit or rollback

DML Locking

When performing data manipulation language (DML) operations, the Oracle Server provides data concurrency through DML locking. DML locks occur at two levels:
• A share lock is automatically obtained at the table level during DML operations. With share lock mode, several transactions can acquire share locks on the same resource.
• An exclusive lock is acquired automatically for each row modified by a DML statement. Exclusive locks prevent the row from being changed by other transactions until the transaction is committed or rolled back. This lock ensures that no other user can modify the same row at the same time and
overwrite changes not yet committed by another user.
Note: DDL locks occur when you modify a database object such as a table.

Kabeer Ahmed
SE-19028

Kabeer Ahmed
SE-19028

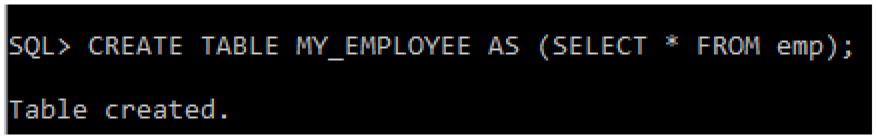
<u>LAB # 07</u>

Manipulating Data

Practice 6

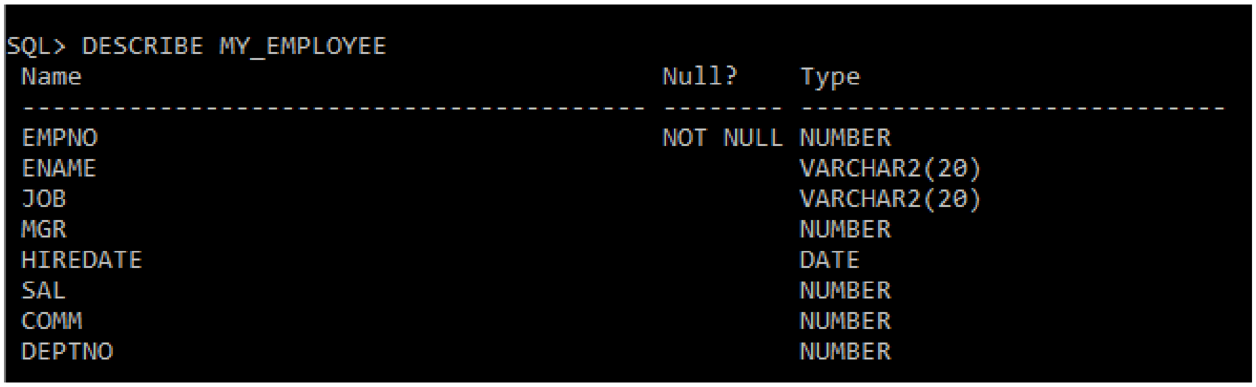
Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab7_1.sql script to build the MY_EMPLOYEE table to be used for the lab.



```
SQL> CREATE TABLE MY_EMPLOYEE AS (SELECT * FROM emp);

Table created.
```

2. Describe the structure of the MY_EMPLOYEE table to identify the column names.

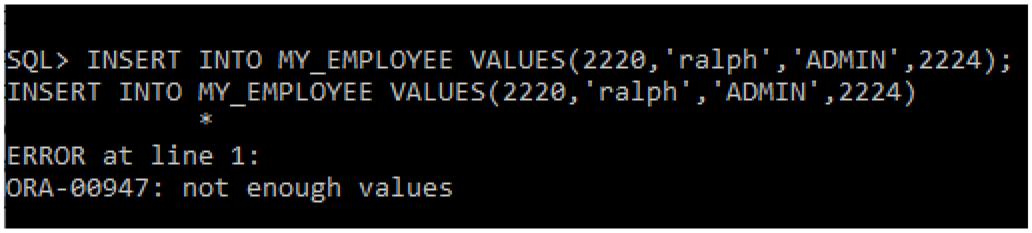


```
SQL> DESCRIBE MY_EMPLOYEE
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER
 ENAME                                              VARCHAR2(20)
 JOB                                                VARCHAR2(20)
 MGR                                                NUMBER
 HIREDATE                                           DATE
 SAL                                                NUMBER
 COMM                                               NUMBER
 DEPTNO                                             NUMBER
```

3. Add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO

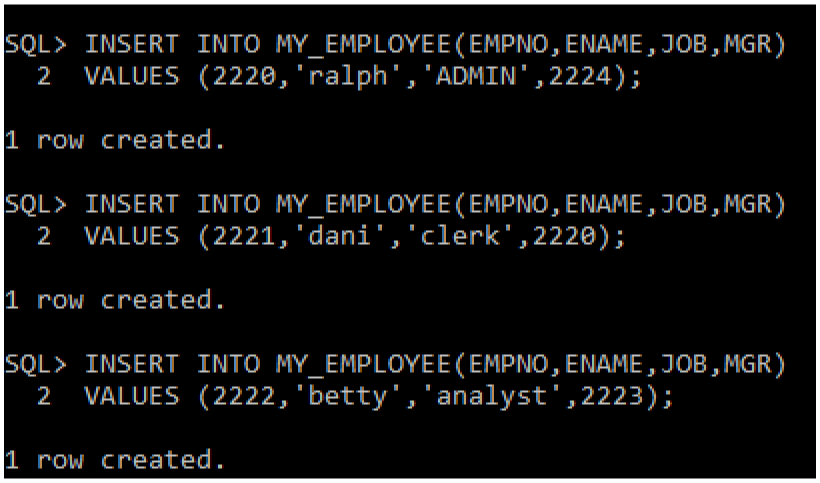
a. **2220,ralph,ADMIN,2224…………**
b. **2221,dani,clerk,2220………………**
c. **2222,betty,analyst,2223…………..**
d. **……………………………………..**
e. **………………………………………..**



```
SQL> INSERT INTO MY_EMPLOYEE VALUES(2220,'ralph','ADMIN',2224);
INSERT INTO MY_EMPLOYEE VALUES(2220,'ralph','ADMIN',2224)
            *
ERROR at line 1:
ORA-00947: not enough values
```

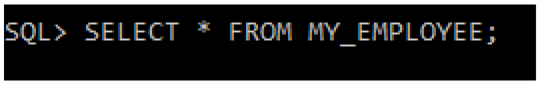
Kabeer Ahmed
SE-19028

4. Populate the MY_EMPLOYEE table with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

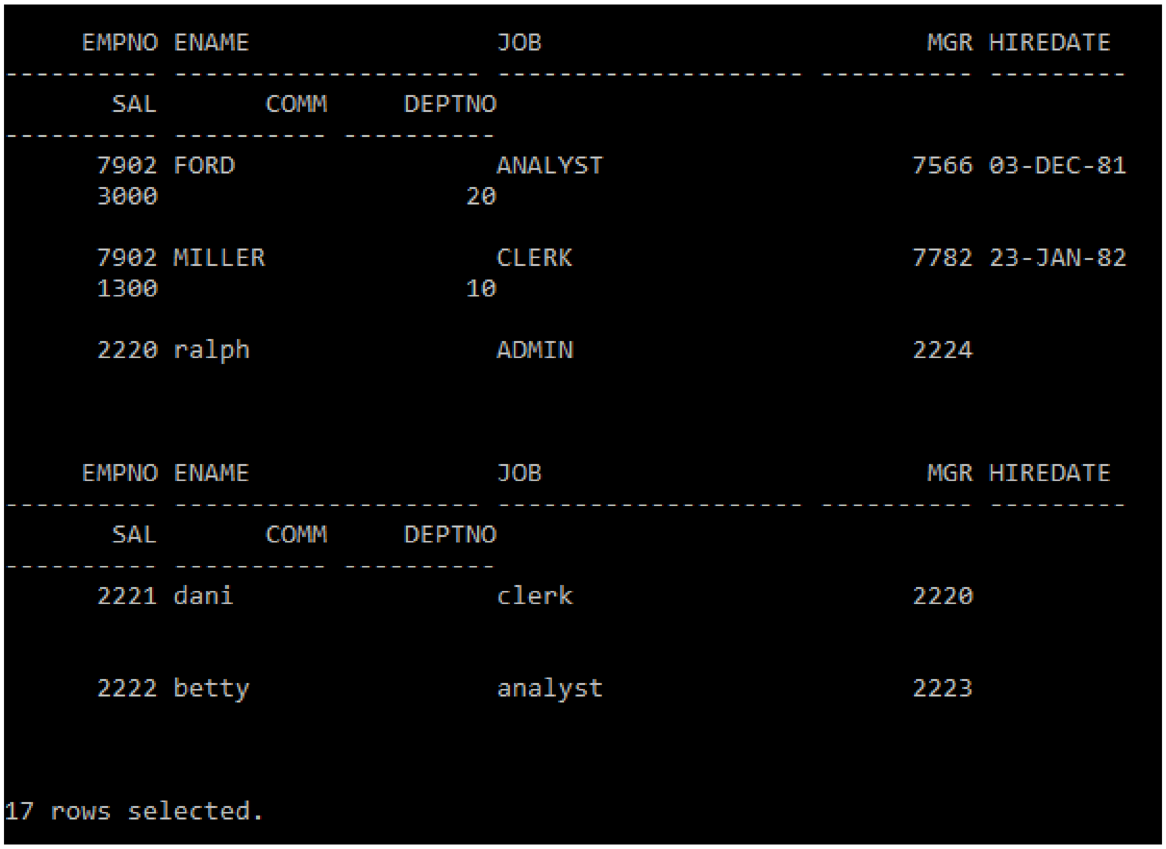


```
SQL> INSERT INTO MY_EMPLOYEE(EMPNO,ENAME,JOB,MGR)
  2  VALUES (2220,'ralph','ADMIN',2224);

1 row created.

SQL> INSERT INTO MY_EMPLOYEE(EMPNO,ENAME,JOB,MGR)
  2  VALUES (2221,'dani','clerk',2220);

1 row created.

SQL> INSERT INTO MY_EMPLOYEE(EMPNO,ENAME,JOB,MGR)
  2  VALUES (2222,'betty','analyst',2223);

1 row created.
```

5. Confirm your addition to the table.



```
SQL> SELECT * FROM MY_EMPLOYEE;
```

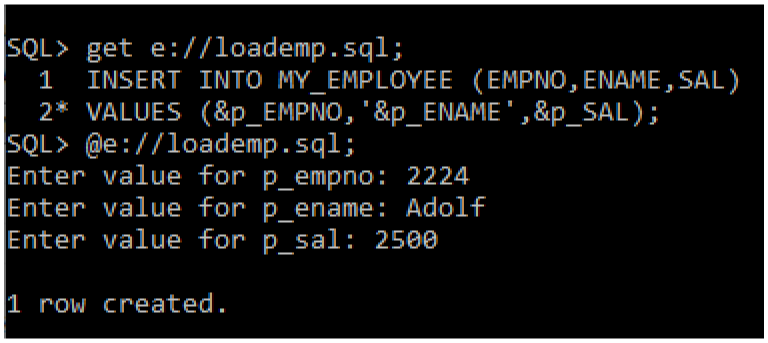


```
     EMPNO ENAME                JOB                         MGR HIRDATE
---------- -------------------- -------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      7902 FORD                 ANALYST                    7566 03-DEC-81
      3000                    20

      7902 MILLER               CLERK                      7782 23-JAN-82
      1300                    10

      2220 ralph                ADMIN                      2224


     EMPNO ENAME                JOB                         MGR HIREDATE
---------- -------------------- -------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      2221 dani                 clerk                      2220


      2222 betty                analyst                    2223



17 rows selected.
```

Kabeer Ahmed
SE-19028

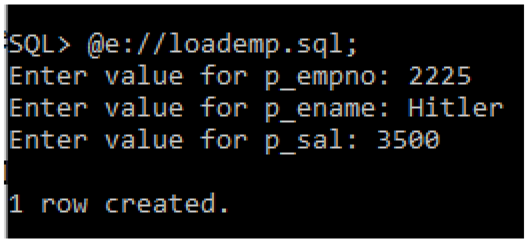
6. Write an INSERT statement in a text file named loademp.sql

to load rows into the MY_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID.

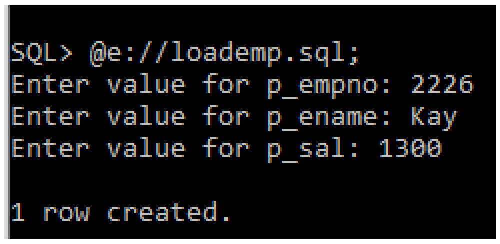


```
SQL> get e://loademp.sql;
  1  INSERT INTO MY_EMPLOYEE (EMPNO,ENAME,SAL)
  2* VALUES (&p_EMPNO,'&p_ENAME',&p_SAL);
SQL> @e://loademp.sql;
Enter value for p_empno: 2224
Enter value for p_ename: Adolf
Enter value for p_sal: 2500

1 row created.
```

7. Populate the table with the next two rows of sample data by running the INSERT statement in the script that you created.

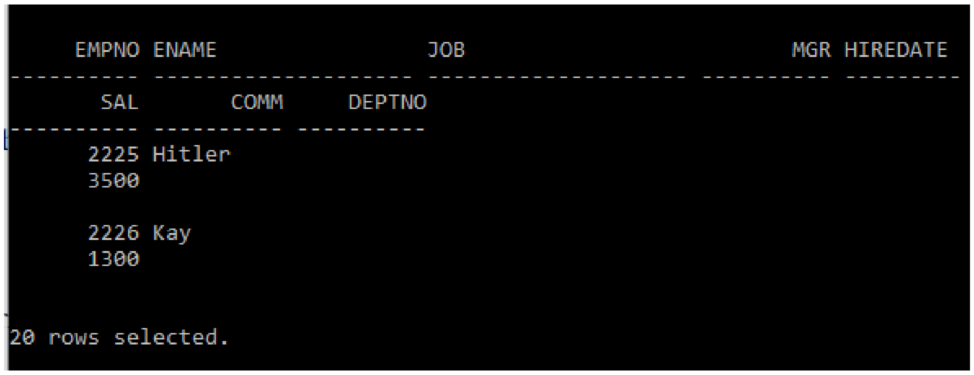


```
SQL> @e://loademp.sql;
Enter value for p_empno: 2225
Enter value for p_ename: Hitler
Enter value for p_sal: 3500

1 row created.
```



```
SQL> @e://loademp.sql;
Enter value for p_empno: 2226
Enter value for p_ename: Kay
Enter value for p_sal: 1300

1 row created.
```

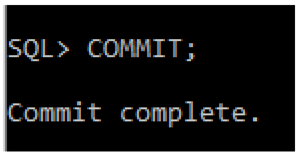
8. Confirm your additions to the table.



```
     EMPNO ENAME                JOB                         MGR HIREDATE
---------- -------------------- -------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      2225 Hitler
      3500

      2226 Kay
      1300


20 rows selected.
```

Kabeer Ahmed
SE-19028

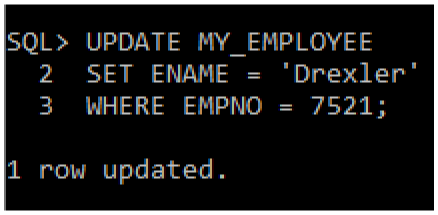
9. Make the data additions permanent.



```
SQL> COMMIT;

Commit complete.
```

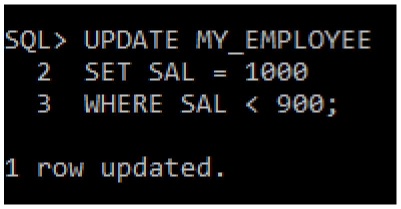
Update and delete data in the MY_EMPLOYEE table.

10. Change the name of employee 3 to Drexler.



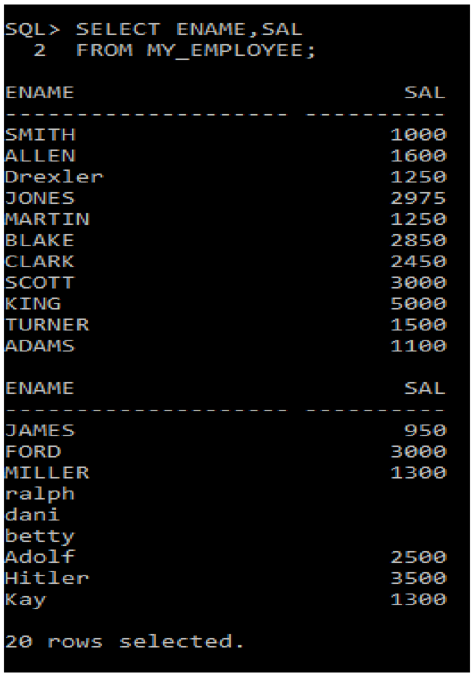
```
SQL> UPDATE MY_EMPLOYEE
  2  SET ENAME = 'Drexler'
  3  WHERE EMPNO = 7521;

1 row updated.
```

11. Change the salary to 1000 for all employees with a salary less than 900.



```
SQL> UPDATE MY_EMPLOYEE
  2  SET SAL = 1000
  3  WHERE SAL < 900;

1 row updated.
```

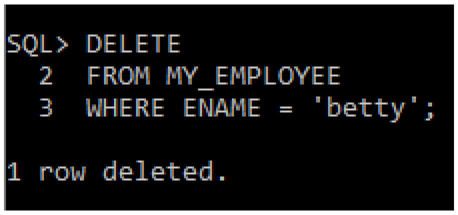
12. Verify your changes to the table.



```
SQL> SELECT ENAME,SAL
  2  FROM MY_EMPLOYEE;

ENAME                       SAL
-------------------- ----------
SMITH                      1000
ALLEN                      1600
Drexler                    1250
JONES                      2975
MARTIN                     1250
BLAKE                      2850
CLARK                      2450
SCOTT                      3000
KING                       5000
TURNER                     1500
ADAMS                      1100

ENAME                       SAL
-------------------- ----------
JAMES                       950
FORD                       3000
MILLER                     1300
ralph
dani
betty
Adolf                      2500
Hitler                     3500
Kay                        1300

20 rows selected.
```

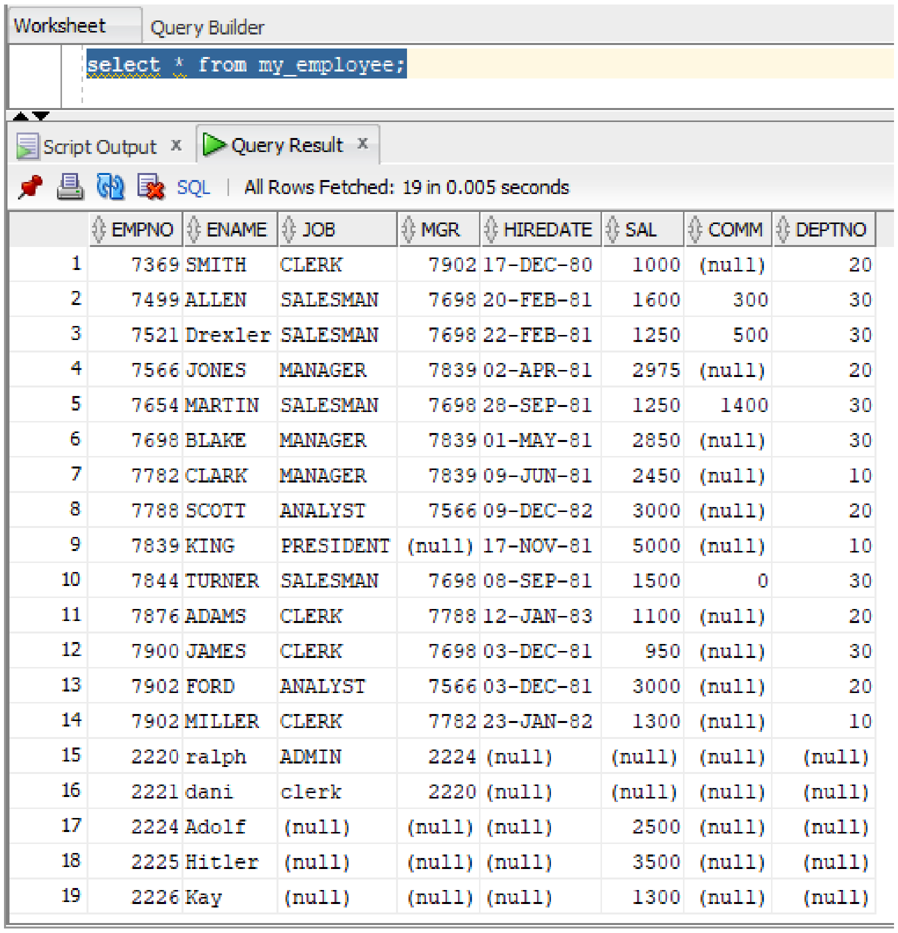
Kabeer Ahmed
SE-19028

13. Delete Betty Dancs from the MY_EMPLOYEE table.



```
SQL> DELETE
  2  FROM MY_EMPLOYEE
  3  WHERE ENAME = 'betty';

1 row deleted.
```

14. Confirm your changes to the table.

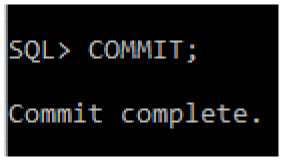


```
select * from my_employee;
```

All Rows Fetched: 19 in 0.005 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	1000	(null)	20
2	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
3	7521	Drexler	SALESMAN	7698	22-FEB-81	1250	500	30
4	7566	JONES	MANAGER	7839	02-APR-81	2975	(null)	20
5	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
6	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	(null)	30
7	7782	CLARK	MANAGER	7839	09-JUN-81	2450	(null)	10
8	7788	SCOTT	ANALYST	7566	09-DEC-82	3000	(null)	20
9	7839	KING	PRESIDENT	(null)	17-NOV-81	5000	(null)	10
10	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
11	7876	ADAMS	CLERK	7788	12-JAN-83	1100	(null)	20
12	7900	JAMES	CLERK	7698	03-DEC-81	950	(null)	30
13	7902	FORD	ANALYST	7566	03-DEC-81	3000	(null)	20
14	7902	MILLER	CLERK	7782	23-JAN-82	1300	(null)	10
15	2220	ralph	ADMIN	2224	(null)	(null)	(null)	(null)
16	2221	dani	clerk	2220	(null)	(null)	(null)	(null)
17	2224	Adolf	(null)	(null)	(null)	2500	(null)	(null)
18	2225	Hitler	(null)	(null)	(null)	3500	(null)	(null)
19	2226	Kay	(null)	(null)	(null)	1300	(null)	(null)

15. Commit all pending changes.

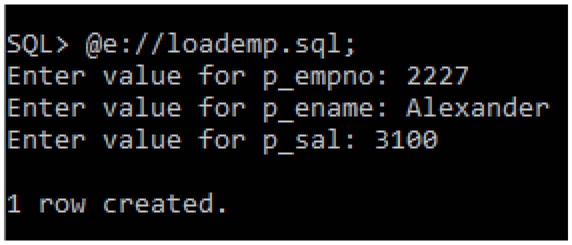


```
SQL> COMMIT;

Commit complete.
```

Kabeer Ahmed
SE-19028

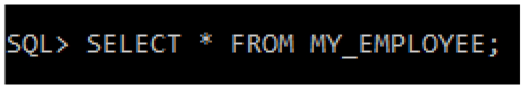
Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

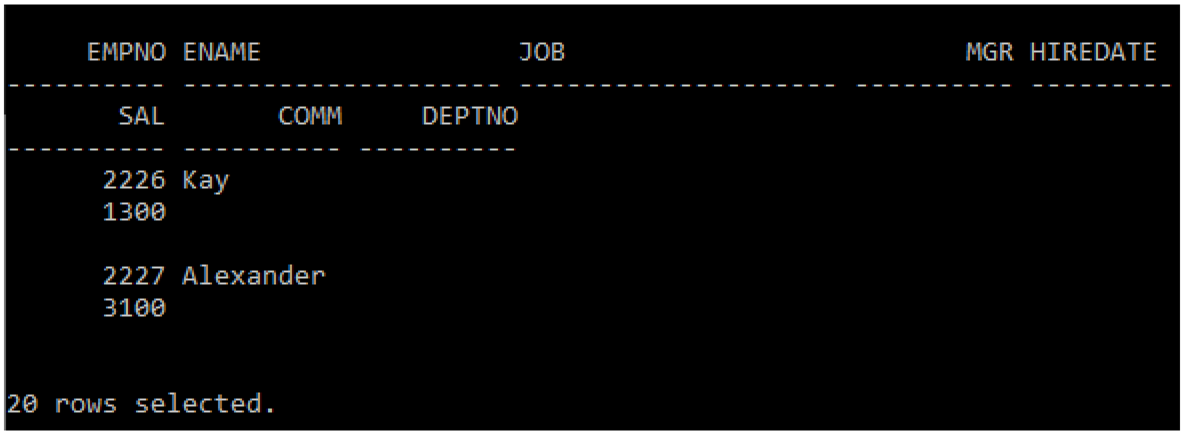


```
SQL> @e://loademp.sql;
Enter value for p_empno: 2227
Enter value for p_ename: Alexander
Enter value for p_sal: 3100

1 row created.
```

17. Confirm your addition to the table.

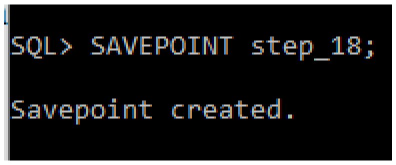


```
SQL> SELECT * FROM MY_EMPLOYEE;
```



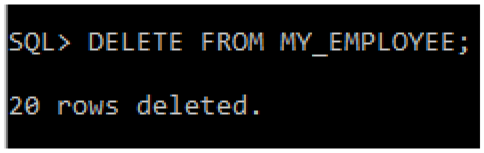
```
     EMPNO ENAME                JOB                         MGR HIREDATE
---------- -------------------- -------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      2226 Kay
      1300

      2227 Alexander
      3100


20 rows selected.
```

18. Mark an intermediate point in the processing of the transaction.



```
SQL> SAVEPOINT step_18;

Savepoint created.
```

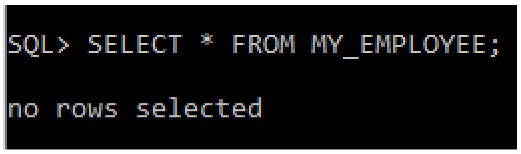
19. Empty the entire table.



```
SQL> DELETE FROM MY_EMPLOYEE;

20 rows deleted.
```

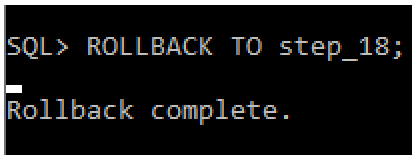
Kabeer Ahmed
SE-19028

20. Confirm that the table is empty.



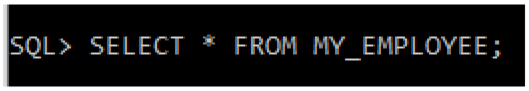
```
SQL> SELECT * FROM MY_EMPLOYEE;

no rows selected
```

21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.

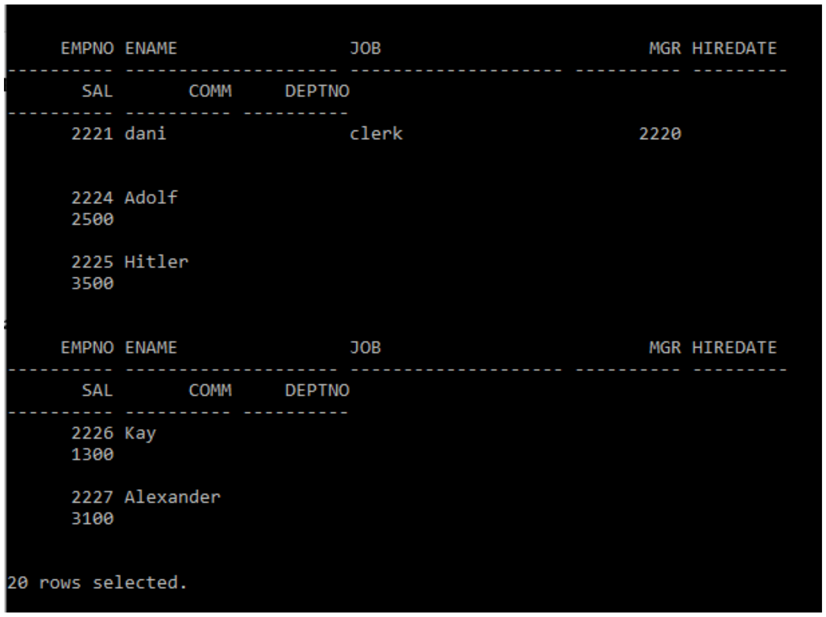


```
SQL> ROLLBACK TO step_18;

Rollback complete.
```

22. Confirm that the new row is still intact.

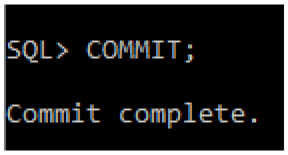


```
SQL> SELECT * FROM MY_EMPLOYEE;
```



```
     EMPNO ENAME              JOB                       MGR HIREDATE
---------- ------------------ ------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      2221 dani               clerk                    2220


      2224 Adolf
      2500

      2225 Hitler
      3500


     EMPNO ENAME              JOB                       MGR HIREDATE
---------- ------------------ ------------------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      2226 Kay
      1300

      2227 Alexander
      3100


20 rows selected.
```

23. Make the data addition permanent.



```
SQL> COMMIT;

Commit complete.
```