Database Management System

EXPERIMENT 2 MANIPULATING DATA

NAME: Madhan RV

ROLL: 241801142

1. Create MY_EMPLOYEE table with the given structure

```
CREATE TABLE my_employee (
    id NUMBER(4) NOT NULL,
    last_name VARCHAR2(25),
    first_name VARCHAR2(25),
    userid VARCHAR2(25),
    salary NUMBER(9,2)
);
```

Expected Output:

```
Table created.
```

Verify table structure:

```
DESC my_employee;
```

Expected Output:

```
Name       Null? Type
---------- -------- -----------
ID         NOT NULL NUMBER(4)
```

```
LAST_NAME          VARCHAR2(25)
FIRST_NAME         VARCHAR2(25)
USERID             VARCHAR2(25)
SALARY             NUMBER(9,2)
```

Inserting Data:

2. Add the first two rows to MY_EMPLOYEE table

```
INSERT INTO my_employee (id, last_name, first_name, userid,
salary)
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);

INSERT INTO my_employee (id, last_name, first_name, userid,
salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

Expected Output:

```
1 row created.
1 row created.
```

3. Display the table with values

```
SELECT * FROM my_employee;
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                        SALARY
---------- ------------------------- -----------------------
-- ------------------------- ----------
         1 Patel                     Ralph
```

```
rpatel                            895
         2 Dancs                           Betty
bdancs                            860
```

4. Populate the next two rows using concatenation for Userid

```
INSERT INTO my_employee (id, last_name, first_name, userid,
salary)
VALUES (3, 'Biri', 'Ben', UPPER(SUBSTR('Ben', 1, 1) ||
SUBSTR('Biri', 1, 7)), 1100);

INSERT INTO my_employee (id, last_name, first_name, userid,
salary)
VALUES (4, 'Newman', 'Chad', UPPER(SUBSTR('Chad', 1, 1) ||
SUBSTR('Newman', 1, 7)), 750);
```

Expected Output:

```
1 row created.
1 row created.
```

Alternative method using substitution variables:

```
INSERT INTO my_employee
VALUES (&id, '&last_name', '&first_name',
        UPPER(SUBSTR('&first_name', 1, 1) ||
SUBSTR('&last_name', 1, 7)),
        &salary);
```

When prompted:

- For first record: id=3, last_name=Biri, first_name=Ben, salary=1100

- For second record: id=4, last_name=Newman, first_name=Chad, salary=750

Verify the inserts:

```
SELECT * FROM my_employee;
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                       SALARY
---------- ------------------------- ------------------------
-- ------------------------- ----------
         1 Patel                     Ralph
rpatel                          895
         2 Dancs                     Betty
bdancs                          860
         3 Biri                      Ben
BBIRI                          1100
         4 Newman                    Chad
CNEWMAN                         750
```

5. Make the data additions permanent

```
COMMIT;
```

Expected Output:

```
Commit complete.
```

Updating Data:

6. Change the last name of employee 3 to Drexler

```
UPDATE my_employee
SET last_name = 'Drexler'
WHERE id = 3;
```

Expected Output:

```
1 row updated.
```

Verify the update:

```
SELECT * FROM my_employee WHERE id = 3;
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                       SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         3 Drexler                   Ben
BBIRI                          1100
```

7. Change salary to 1000 for all employees with salary less than 900

```
UPDATE my_employee
SET salary = 1000
WHERE salary < 900;
```

Expected Output:

```
2 rows updated.
```

Verify the update:

```
SELECT * FROM my_employee;
```

Expected Output:

```
       ID LAST_NAME                         FIRST_NAME
USERID                            SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         1 Patel                          Ralph
rpatel                              1000
         2 Dancs                          Betty
bdancs                              1000
         3 Drexler                        Ben
BBIRI                               1100
         4 Newman                         Chad
CNEWMAN                             1000
```

Deleting Data:

8. Delete Betty Dancs from MY_EMPLOYEE table

```
DELETE FROM my_employee
WHERE first_name = 'Betty' AND last_name = 'Dancs';
```

Expected Output:

```
1 row deleted.
```

Verify the deletion:

```
SELECT * FROM my_employee;
```

Expected Output:

```
      ID LAST_NAME                 FIRST_NAME
USERID                       SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         1 Patel                     Ralph
rpatel                         1000
         3 Drexler                   Ben
BBIRI                          1100
         4 Newman                    Chad
CNEWMAN                        1000
```

Transaction Control:

9. Create a savepoint

```
SAVEPOINT before_insert;
```

Expected Output:

```
Savepoint created.
```

10. Insert the fifth row

```
INSERT INTO my_employee (id, last_name, first_name, userid,
salary)
VALUES (5, 'Ropebur', 'Audrey', UPPER(SUBSTR('Audrey', 1, 1)
|| SUBSTR('Ropebur', 1, 7)), 1550);
```

Expected Output:

```
1 row created.
```

Verify the insert:

```
SELECT * FROM my_employee;
```

Expected Output:

```
       ID LAST_NAME                         FIRST_NAME
USERID                         SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         1 Patel                          Ralph
rpatel                            1000
         3 Drexler                        Ben
BBIRI                             1100
         4 Newman                         Chad
CNEWMAN                           1000
         5 Ropebur                        Audrey
AROPEBUR                          1550
```

11. Empty all rows from the table (for demonstration)

```
DELETE FROM my_employee;
```

Expected Output:

```
4 rows deleted.
```

Verify the table is empty:

```
SELECT * FROM my_employee;
```

Expected Output:

```
no rows selected
```

12. Rollback to the savepoint

```
ROLLBACK TO before_insert;
```

Expected Output:

```
Rollback complete.
```

Verify data is restored:

```
SELECT * FROM my_employee;
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                       SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         1 Patel                     Ralph
rpatel                         1000
         3 Drexler                   Ben
BBIRI                          1100
         4 Newman                    Chad
CNEWMAN                        1000
         5 Ropebur                   Audrey
AROPEBUR                       1550
```

Advanced DML Operations:

13. Using MERGE statement (UPSERT operation)

First, create a backup table:

```
CREATE TABLE my_employee_backup AS SELECT * FROM
my_employee;
```

Expected Output:

```
Table created.
```

Merge example:

```
MERGE INTO my_employee_backup target
USING (SELECT * FROM my_employee) source
ON (target.id = source.id)
WHEN MATCHED THEN
    UPDATE SET
        target.last_name = source.last_name,
        target.first_name = source.first_name,
        target.userid = source.userid,
        target.salary = source.salary
WHEN NOT MATCHED THEN
    INSERT (id, last_name, first_name, userid, salary)
    VALUES (source.id, source.last_name, source.first_name,
source.userid, source.salary);
```

Expected Output:

```
4 rows merged.
```

14. Insert with NULL values demonstration

Implicit method (omitting the column):

```
INSERT INTO my_employee (id, last_name, first_name, userid)
VALUES (6, 'Smith', 'John', 'jsmith');
```

Expected Output:

```
1 row created.
```

Explicit method (specifying NULL):

```
INSERT INTO my_employee
VALUES (7, 'Brown', 'Lisa', NULL, NULL);
```

Expected Output:

```
1 row created.
```

Verify NULL values:

```
SELECT * FROM my_employee WHERE id IN (6, 7);
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                      SALARY
---------- ------------------------- ----------------------
-- ------------------------- ----------
         6 Smith                     John
jsmith
```

```
        7 Brown                      Lisa
```

15. Insert with SYSDATE and TO_DATE

Create a table with date column for demonstration:

```
CREATE TABLE employee_audit (
    employee_id NUMBER,
    action VARCHAR2(10),
    action_date DATE
);
```

Expected Output:

```
Table created.
 INSERT INTO employee_audit VALUES (100, 'HIRED', SYSDATE);
INSERT INTO employee_audit VALUES (101, 'HIRED',
TO_DATE('FEB 3,1999','MON DD,YYYY'));
```

Expected Output:

```
1 row created.
1 row created.
```

Verify date inserts:

```
SELECT * FROM employee_audit;
```

Expected Output:

```
EMPLOYEE_ID ACTION     ACTION_DA
----------- ---------- ---------
        100 HIRED      23-JAN-24  -- Current date
```

```
       101 HIRED      03-FEB-99
```

Final Cleanup:

Make final commit:

```
COMMIT;
```

Expected Output:

```
Commit complete.
```

Final verification of MY_EMPLOYEE table:

```
SELECT * FROM my_employee ORDER BY id;
```

Expected Output:

```
       ID LAST_NAME                 FIRST_NAME
USERID                        SALARY
---------- ------------------------- -----------------------
-- ------------------------- ----------
         1 Patel                     Ralph
rpatel                          1000
         3 Drexler                   Ben
BBIRI                           1100
         4 Newman                    Chad
CNEWMAN                         1000
         5 Ropebur                   Audrey
AROPEBUR                        1550
         6 Smith                     John
jsmith
```

```
      7 Brown                          Lisa
```

Summary of DML Operations Demonstrated:

1. **INSERT** - Adding new rows with various methods
2. **UPDATE** - Modifying existing data
3. **DELETE** - Removing rows
4. **MERGE** - Upsert operations (Update/Insert)
5. **COMMIT** - Making changes permanent
6. **ROLLBACK** - Undoing changes
7. **SAVEPOINT** - Marking transaction points
8. **Substitution variables** - Dynamic value input
9. **NULL handling** - Both implicit and explicit methods
10. **Date operations** - Using SYSDATE and TO_DATE