

**Database Management System**

**(DBMS – 204)**

Experiment # 07

**Manipulating Data**

|  |
| --- |
| Student Name: Muhammad Faraz Ansar |
| Roll Number: SE-061 |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Marks** | **Performance = 05** | **Viva = 05** | **Total = 10** |
| **Marks Obtained** |  |  |  |
| **Remarks (if any)** |  | | |
|  | | | |
| **Experiment evaluated by** | | | |
| Instructor Name: Engr. Adiba Jafar | | | |
| Signature and Date: | | | |

**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:

1. Add new rows to a table
2. Modify existing rows in a table
3. 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**

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

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

1. Only one row is inserted at a time with this syntax.
2. Insert a new row containing values for each column.
3. List values in the default order of the columns in the table.
4. Optionally, list the columns in the INSERT clause.
5. Enclose character and date values within single

quotation marks.

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

**Inserting Rows with Null Values**

1. Implicit method: Omit the column from the column list.

INSERT INTO dept (deptno,dname ) VALUES (30, 'Purchasing');

1. 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;

**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* ];

1. 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);

**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 );

INSERT INTO

**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

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

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 (290, 'Corporate Tax', NULL, 1700);

•Commit the changes.

COMMIT;

Commit complete.

Example

Remove departments 290 and 300 in the DEPT table, and update a row in the NEW\_EMP table. Make the data change permanent.

DELETE FROM dept WHERE deptno IN (20, 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?**

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.

**Practice 6**

Insert data into the MY\_EMPLOYEE table.

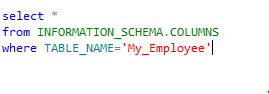
1. Run the statement in the lab7\_1.sql script to build the MY\_EMPLOYEE table to be used for the lab.

* **Creating table my\_employee from emp table (sql server)**



1. Describe the structure of the MY\_EMPLOYEE table to identify the column names.

* **Structure of my\_employee (sql server)**



1. 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

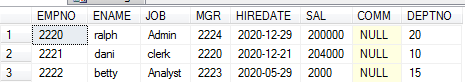
1. 2220,ralph,ADMIN,2224…………
2. 2221,dani,clerk,2220………………
3. 2222,betty,analyst,2223…………..
4. ……………………………………..
5. ………………………………………..



1. 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.



5. Confirm your addition to the table.

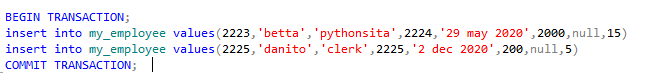


6. Write an INSERT statement in a text file named loademp.sql

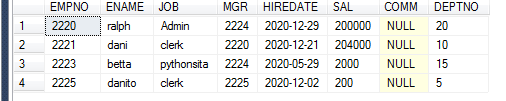
to load rows into the MY\_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID.



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

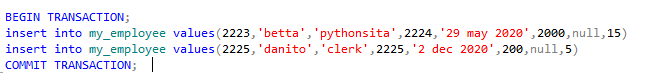


8. Confirm your additions to the table.



9. Make the data additions permanent.

Commit command to permanent



Update and delete data in the MY\_EMPLOYEE table.

10. Change the name of employee 3 to Drexler.

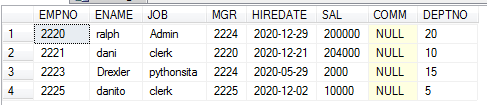




11. Change the salary to 1000 for all employees with a salary less than 900.



12. Verify your changes to the table.



13. Delete Betty and Dani from the MY\_EMPLOYEE table.



14. Confirm your changes to the table.



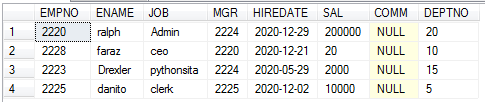
15. Commit all pending changes. Control data transaction to the MY\_EMPLOYEE table.



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.



17. Confirm your addition to the table.



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



19. Empty the entire table.



20. Confirm that the table is empty.

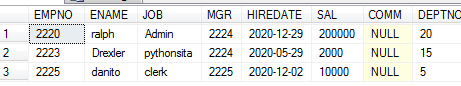


21. Discard the most recent DELETE operation without discarding the earlier



INSERT operation.

22. Confirm that the new row is still intact.



Rows come back

23. Make the data addition permanent.

