PRACTICAL-6

- ## Displaying Data from Multiple Tables (Join)

1) Write a query to display the details of required customer.

   QUERY:

   select d.act_no, d.c_name, d.amount, d.a_date, c.city, b.b_name, b.city from DEPOSIT d,BRANCH b, CUSTOMER c WHERE d.c_name = c.c_name ANDd.b_name = b.b_name AND d.c_name = '';

```
SQL> select d.act_no, d.c_name, d.amount, d.a_date, c.city, b.b_name, b.city from DEPOSIT d,
  2  BRANCH b, CUSTOMER c WHERE d.c_name = c.c_name AND
  3  d.b_name = b.b_name AND d.c_name = '';

no rows selected
```

2) Write a query to find out the customers' name who are depositor as well as borrower andliving in city '**Vadodara**'.

   QUERY:

   select c.c_name from CUSTOMER c, DEPOSIT d, BORROW b WHERE c.c_name = d.c_nameAND c.c_name = b.c_name AND c.city = 'Vadodara';

```
SQL> select c.c_name from CUSTOMER c, DEPOSIT d, BORROW b WHERE c.c_name = d.c_name
  2  AND c.c_name = b.c_name AND c.city = 'Vadodara';

no rows selected
```

3) Write a query to find out that city in which customer is living is same as city of branch.QUERY:

   select c.city from CUSTOMER c, BRANCH b WHERE c.city = b.city;

```
SQL> select c.city from CUSTOMER c, BRANCH b WHERE c.city = b.city;

CITY
----------
MUMBAI
JAIPUR
DELHI
BAKROL
VATAO
```

4) Write a query to display the employee name, department number, department name for all employees.
   QUERY:
   select e.e_name, e.d_no, d.d_name from EMPLOYEE e, DEPARTMENT d WHERE e.d_no =d.d_no;

```
SQL> select e.e_name, e.d_no, d.d_name from EMPLOYEE e, DEPARTMENT d WHERE e.d_no = d.d_no;

E_NAME                    D_NO D_NAME
-------------------- ---------- --------------------
Juned                       101 Administration
Mitesh                      101 Administration
Hitesh                      102 HR
Pooja                       104 Finance
```

5) Write a query to create unique listing of all jobs that are in department number '**102**'. Include the location of department in output.
   QUERY:
   select j.j_id, j.j_title, e.d_no, d.d_location from JOBS j, EMPLOYEE e, DEPARTMENT d WHEREj.j_id = e.j_id AND e.d_no = d.d_no AND e.d_no = 102;

```
SQL> select j.j_id, j.j_title, e.d_no, d.d_location from JOBS j, EMPLOYEE e, DEPARTMENT d
WHERE j.j_id = e.j_id AND e.d_no = d.d_no AND e.d_no = 102;

      J_ID J_TITLE                   D_NO D_LOCATION
---------- -------------------- ---------- --------------------
      1001 Lecturer                    102 Anand
```

6) Write a query to display the employee name, department number, department name for all employees who work in '**New York**'.
   QUERY:
   select e.e_name, e.d_no, d.d_name from EMPLOYEE e, DEPARTMENT d WHERE e.d_no =d.d_no AND d_location = 'New York';

```
SQL> select e.e_name, e.d_no, d.d_name from EMPLOYEE e, DEPARTMENT d WHERE
e.d_no = d.d_no AND d_location = 'New York';

E_NAME                    D_NO D_NAME
-------------------- ---------- --------------------
Juned                       101 Administration
Mitesh                      101 Administration
```

PRACTICAL-7

- **Using Commit and Rollback show Transaction ACID Property**

A transaction is a unit of work that is performed against a database. Transactions are units sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

➢ Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym **ACID.**

1) **ATOMICITY** − ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.

2) **CONSISTENCY** − ensures that the database properly changes states upon a successfully committed transaction.

3) **ISOLATION** − enables transactions to operate independently of and transparent to each other.

4) **DURABILITY** − ensures that the result or effect of a committed transaction persists in case of a system failure.

➢ Transaction Control

The following commands are used to control transactions.

1) **COMMIT** − to save the changes.

2) **ROLLBACK** − to roll back the changes.

3) **SAVEPOINT** − creates points within the groups of transactions in which to ROLLBACK.

4) **TRANSACTION** − places a name on a transaction.

➢ Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

➢ The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows:

COMMIT;

❖ **Example**

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	HUNAID	20	MUMBAI	100000.00
2	HITENDRA	19	JAIPUR	150000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

Following is an example which would delete those records from the table which have age = 18 and then COMMIT the changes in the database.

SQL> DELETE FROM CUSTOMERS WHERE AGE = 18;

SQL> COMMIT;

Thus, two rows from the table would be deleted and the SELECT statement would produce as below:

ID	NAME	AGE	ADDRESS	SALARY
1	HUNAID	20	MUMBAI	100000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

➢ The ROLLBACK Command

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued. The syntax for a ROLLBACK command is as follows:

ROLLBACK;

❖ **Example**

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	HUNAID	20	MUMBAI	100000.00
2	HITENDRA	19	JAIPUR	150000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

Following is an example, which would delete those records from the table which have the age = 18 and then ROLLBACK the changes in the database.

SQL> DELETE FROM CUSTOMERS WHERE AGE = 18;

SQL> ROLLBACK;

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	HUNAID	20	MUMBAI	100000.00
2	HITENDRA	19	JAIPUR	150000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

➢ The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
The syntax for a SAVEPOINT command is as shown below:

SAVEPOINT SAVEPOINT_NAME;

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions. The syntax for rolling back to a SAVEPOINT is as shown below:

ROLLBACK TO SAVEPOINT_NAME;

Following is an example where you plan to delete the three different records from the CUSTOMERS table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

❖ **Example**

Consider the CUSTOMERS table having the following records.

The following code block contains the series of operations.

ID	NAME	AGE	ADDRESS	SALARY
1	HUNAID	20	MUMBAI	100000.00
2	HITENDRA	19	JAIPUR	150000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

SQL> SAVEPOINT SP1;
Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.

SQL> SAVEPOINT SP2;
Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.

SQL> SAVEPOINT SP3;
Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:

SQL> ROLLBACK TO SP2;
Rollback complete.

Notice that only the first deletion took place since you rolled back to SP2.

SQL> SELECT * FROM CUSTOMERS

ID	NAME	AGE	ADDRESS	SALARY
2	HITENDRA	19	JAIPUR	150000.00
3	HARSHAD	18	DELHI	175000.00
4	AESHWARY	19	BAKROL	155000.00

➢ The RELEASE SAVEPOINT Command

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows:

RELEASE SAVEPOINT SAVEPOINT_NAME;

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

➢ The SET TRANSACTION Command

The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows. For example, you can specify a transaction to be read only or read write.
SQL language is divided into four types of primary language statements: DML, DDL, DCL and TCL. Using these statements, we can define the structure of a database by creating and altering database objects, and we can manipulate data in a table through updates or deletions. We also can control which user can read/write data or manage transactions to create a single unit of work.

➢ TCL (Transaction Control Language)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

1) **BEGIN Transaction** – opens a transaction.

2) **COMMIT Transaction** – commits a transaction.

3) **ROLLBACK Transaction** – ROLLBACK a transaction in case of any error.

- **Difference between ROLLBACK and COMMIT commands**:

ROLLBACK	COMMIT
ROLLBACK command is used to undo the changes made by the DML commands.	The COMMIT command is used to save the modification done to the database values by the DML commands.
It rollbacks all the changes of the current transaction.	It will make all the changes permanent that cannot be rolled back.
Syntax: DELETE FROM table_name ROLLBACK;	**Syntax:** COMMIT;

PRACTICAL-8

- **Securing data using Views and Controlling User Access (DCL)**

➢ Introduction to Privileges

A privilege is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to:

- Connect to the database (create a session).
- Create a table.
- Select rows from another user's table.
- Execute another user's stored procedure.

➢ DCL (Data Control Language)

- DCL statements control the level of access that users have on database objects.
- **GRANT** – allows users to read/write on certain database objects.
- **REVOKE** – keeps users from read/write permission on database objects.

You grant privileges to users so these users can accomplish tasks required for their job. You should grant a privilege only to a user who absolutely requires the privilege to accomplish necessary work. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly. For example, you can explicitly grant the privilege to insert records into the employees table to the user SCOTT.
- You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the employees table to the role named clerk, which in turn you can grant to the users scott and brian.

Because roles allow for easier and better management of privileges, you should normally grant privileges to roles and not to specific users.
There are two types of different privileges:

1. System privileges
2. Schema object privileges

➢ System privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create table spaces and to delete the rows of any table in a database are system privileges. There are over 60 distinct system privileges.

➢ Grant and Revoke System Privileges

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, then you can use the roles to manage system privileges. For example, roles permit privileges to be made selectively available.
Use either of the following to grant or revoke system privileges to users and roles:

Oracle Enterprise Manager Console
The SQL statements GRANT and REVOKE

❖ Who Can Grant or Revoke System Privileges?

Only users who have been granted a specific system privilege with the ADMIN OPTION or users with the system privileges GRANT ANY PRIVILEGE or GRANT ANY OBJECT
PRIVILEGE can grant or revoke system privileges to other users.

➢ Schema Object Privileges

A schema object privilege is a privilege or right to perform a particular action on a specific schema object:

- Table
- View
- Sequence
- Procedure
- Function
- Package

Different object privileges are available for different types of schema objects. For example, the privilege to delete rows from the departments table is an object privilege.

Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege. A schema object and its synonym are equivalent with respect to privileges. That is, the object privileges granted for a table, view, sequence, procedure, function, or package apply whether referencing the base object by name or using a synonym.
For example, assume there is a table jward.emp with a synonym named jward.employee and the user jward issues the following statement:

GRANT SELECT ON emp TO swilliams;

The user swilliams can query jward.emp by referencing the table by name or using the synonym jward.employee:

SELECT * FROM jward.emp;

SELECT * FROM jward.employee;

If you grant object privileges on a table, view, sequence, procedure, function, or package to a **synonym** for the object, the effect is the same as if no synonym were used. For example, if jward wanted to grant the SELECT privilege for the emp table to swilliams, jward could issue either of the following statements:

GRANT SELECT ON emp TO swilliams;

GRANT SELECT ON employee TO swilliams;

If a synonym is dropped, all grants for the underlying schema object remain in effect, even if the privileges were granted by specifying the dropped synonym.

➢ Grant and Revoke Schema Object Privileges

Schema object privileges can be granted to and revoked from users and roles. If you grant object privileges to roles, you can make the privileges selectively available. Object privileges for users and roles can be granted or revoked using the following:

- The SQL statements GRANT and REVOKE, respectively
- The Add Privilege to Role/User dialog box and the Revoke Privilege from Role/User dialog box of Oracle Enterprise Manager.

❖ Who Can Grant Schema Object Privileges?

A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. A user with the GRANT ANY OBJECT PRIVILEGE can grant or revoke any specified object privilege to another user with or without the GRANT OPTION of the GRANT statement.
Otherwise, the grantee can use the privilege, but cannot grant it to other users.

For example, assume user SCOTT has a table named t2:

SQL>GRANT any object privilege TO U1;SQL>

connect u1/u1

Connected.

SQL> GRANT select on scott.t2 \TO U2;

SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS

WHERE TABLE_NAME = 'employees';

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
U2	SCOTT	SCOTT	SELECT	NO

➢ Privileges Required to Create Views

To create a view, you must meet the following requirements:
You must have been granted one of the following system privileges, either explicitly or through a role:

- The CREATE VIEW system privilege (to create a view in your schema).
- The CREATE ANY VIEW system privilege (to create a view in another user's schema).

You must have been explicitly granted one of the following privileges:

- The SELECT, INSERT, UPDATE, or DELETE object privileges on all base objects underlying the view.
- The SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, or DELETE ANY TABLE system privileges.

Additionally, in order to grant other user's access to your view, you must have received object privileges to the base objects with the GRANT OPTION clause or appropriate system privileges with the ADMIN OPTION clause. If you have not, grantees cannot access your view.

➢ Increase Table Security with Views

To use a view, you require appropriate privileges only for the view itself. You do not require privileges on base objects underlying the view.

Views add two more levels of security for tables, column-level security and value-based security:

- A view can provide access to selected columns of base tables. For example, you can define a view on the employees table to show only the employee_id, last_name, and manager_id columns:

  CREATE VIEW employees_manager AS SELECT last_name, employee_id, manager_id FROM employees;

- A view can provide value-based security for the information in a table. A WHERE clause in the definition of a view displays only selected rows of base tables. Consider the following two examples:

  CREATE VIEW lowsal AS SELECT * FROM employees WHERE salary < 10000;

- The LOWSAL view allows access to all rows of the employees table that have a salary value less than 10000. Notice that all columns of the employees table are accessible in the LOWSAL view.

  CREATE VIEW own_salary AS SELECT last_name, salary FROM employees WHERE last_name = USER;

  In the own_salary view, only the rows with an last_name that matches the current user of the view are accessible. The own_salary view uses the user pseudocolumn, whose values always refer to the current user. This view combines both column-level security and value-based security.