Unit 3: Managing Users and Security

- Profiling and Managing users
- managing user privileges and roles
- managing and querying role information
- Database Security and Auditing
- Creating and managing DB objects: Tables, indexes, triggers, views, stored procedures, etc.

Managing Oracle Users:

Access to Oracle database is provided using database accounts known as usernames (users). A user account is identified by a user name and defines the user's attributes, including the following:

- Password for database authentication
- Profile
- Default tablespace for database objects
- Default temporary tablespace for query processing work space

When you create a user, you are also implicitly creating a schema for that user. A **schema** is a logical container for the database objects (such as tables, views, triggers, and so on) that the user creates.
When you drop (delete) a user, you must either first drop all the user's schema objects, or use the cascade feature of the drop operation, which simultaneously drops a user and all of his schema objects.

Viewing Users

You can view database user by using a SQL command:
select * from all_users;
After viewing a list of users, you can then select an individual users to alter or drop (delete).

Creating Users

Before creating a user, determine the following:

- Whether or not you want to permit the user to create database objects in his own schema.

If so, grant the RESOURCE role or grant individual create object system privileges

- Whether or not you want to grant the user DBA privileges.

If so, grant the DBA role. Because DBA privileges include the ability to create database objects in any schema, if you grant the DBA role, you do not need to grant the RESOURCE role or individual create object system privileges.
Caution: Granting the DBA role to a user has security implications, because the user can modify objects in other users' schemas.

- Whether or not to create the user with an expired password.

When you do this, the password that you assign the user is used only for the user's first login. Upon first login, the user is prompted to select a new password.
Example:

```
SQL> CREATE USER RAMAN IDENTIFIED BY RAMAN123 PROFILE CSIT
DEFAULT TABLESPACE USERS PASSWORD EXPIRE ACCOUNT UNLOCK;

User created.
```

Altering User:

Altering a user means changing some of his user attributes. You can change all user attributes except the user name, default tablespace, and temporary tablespace. If you want to change the user name, you must drop the user and re-create him with a different name.
One of the attributes that you can alter is the user password. To do this you can either assign new password to the user, or request the new password from the user and then enter it. An easier and more secure way to cause a password change is to expire the password.
When you **expire** a password, the user is prompted to change his password the next time that he logs in.
Example:

```
SQL> ALTER USER RAMAN IDENTIFIED BY RAAMAAN;
User altered.
```

Locking and Unlocking User

To temporarily deny access to the database for a particular user, you can lock the user account. If the user then attempts to connect, the database displays an error message and disallows the connection. You can unlock the user account when you want to allow database access again for that user.

Note:

Many internal user accounts are locked (or both expired and locked). You should not attempt to log in with these locked user accounts.

The HR user account, which contains a sample schema, is initially expired and locked. You must log in as SYSTEM, unlock the account, and assign a password before you can log in as HR.

```
SQL> ALTER USER HR ACCOUNT UNLOCK;
User altered.
```

Dropping User:

Dropping a user removes the user from the database. Before you can drop a user, you must first drop all the user's schema objects. Or, you can use the cascade feature of the drop operation, which simultaneously drops a user and all his schema objects.
The following are two alternatives to dropping a user and losing all the user's schema objects:

- To temporarily deny access to the database for a particular user while preserving the user's schema objects, you can lock the user account.
- To drop a user but retain the data from the user's tables, export the tables first.

Caution
Under no circumstances should you attempt to drop the SYS or SYSTEM users, or any other internal user account. Doing so could cause Oracle Database to malfunction.

Profiles:

A profile is a collection of parameters that sets limits on database resources. When you change the profile to a user, you assign a profile but you apply also a set of parameters.
If you assign the profile to a user, then that user cannot exceed these limits parameters.
You can use profiles to configure database settings such as:

- sessions per user,
- logging and tracing features,
- Controlling user passwords, and so on.

Creating a profile:

You can create many profiles in the database that specify both resource management parameters and password management parameters.
However, you can assign a user only one profile at any given time. To create a profile, use CREATE PROFILE command by assigning a name of profile and then specifying the parameter names and their values separated by space(s).
Profiles only take effect when resource limits are "turned on" for the database as a whole.
To see status of resource limit:

```
SQL> show parameter resource_limit
NAME                                 TYPE
VALUE
------------------------------------ ----------- ---
------
resource_limit                       boolean
FALSE
```

To enable resource limit, use the ALTER SYSTEM statement and set RESOURCE_LIMIT=TRUE.

```
SQL> ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;
System altered.
SQL> show parameter resource_limit
NAME                                                        TYPE
VALUE
---------------------------------- -----------
---------
resource_limit                                              boolean
TRUE
```

Example:
SQL> CREATE PROFILE csit LIMIT SESSIONS_PER_USER 2
IDLE_TIME 5
CONNECT_TIME 10;
Profile created.

Assigning Profiles:

Profile can be assign in two ways either during USER creation or by using ALTER statement.
SQL> ALTER USER shraddha PROFILE csit;
User altered.

Altering Profiles:

To alter profile use ALTER PROFILE command. A DBA must have the ALTER PROFILE system privilege to use this command. You can change any parameter in profile by using this command. The changes takes effect next time the user connects to the database.
SQL> ALTER PROFILE csit LIMIT COMPOSITE_LIMIT 1500
PASSWORD_LOCK_TIME 5;
Profile altered.

Dropping Profile:

Profiles that are no longer needed can be dropped using DROP PROFILE command. If the profile you want to delete is assigned to a user, trying to delete that profile will return error. For such profile use CASCADE command which will delete the profile and assigns default profile to the respective user.
SQL> DROP PROFILE CSIT;
*ERROR at line 1:ORA-02382: profile CSIT has users assigned, cannot drop without CASCADE
SQL> DROP PROFILE csit CASCADE;
Profile dropped.

Managing User Privileges and Roles:

In Oracle database, privileges control access to the data and restricts the action user can

perform. With proper privileges, user can create, drop, or modify objects in their own schema or in another user's schema. Privileges also determines the data to which a user should have access.
There are two main types of user 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 tables and to delete the rows of any table in a database are system privileges.
- Object privileges—An object privilege is a right to perform a particular action on a specific schema object. Different object privileges are available for different types of schema objects. The privilege to delete rows from the DEPARTMENTS table is an example of an object privilege.

You can grant privileges to a user by two methods:
1. Assign directly to user
2. Assign privilege to role and then assign role to the user.

Object Privilege:

Object privileges are granted on a specific object. The owner of the object has all privileges on the object. The owner can grant privileges on that object to any other users of the database or can also authorize another user in the database to grant privileges on the object to other users.
eg:
SQL> GRANT SELECT, UPDATE ON CUSTOMER TO SUSHANT;
Here SUSHANT can only query and update rows in the CUSTOMER table but cannot insert or delete, or grant privilege to another user in the database.
SQL> GRANT SELECT, UPDATE ON CUSTOMER TO SUSHANT WITH GRANT OPTION;
Now, SUSHANT can grant the privilege SELECT and UPDATE on CUSTOMER to other users.
Some of the object privileges that can be granted to users are: SELECT, UPDATE, DELETE, INSERT, EXECUTE, ALTER, etc.

System Privilege:

System privileges are the privileges that enable the user to perform an action; that are not specified on any particular object. Like object privilege, system privilege can also be granted to a user, a role, or PUBLIC. When a privilege is specified with ANY, the user is authorized to perform the actions on any schema in the database.
The use of WITH ADMIN OPTION clause gives privilege to a user to the grant privilege to another user.
eg: GRANT SELECT ANY TABLE TO SUSHANT WITH ADMIN OPTION;
Some of the system privileges that can be granted to users are: SELECT ANY TABLE, ALTER DATABASE, CREATE SESSION, BACKUP ANY TABLE, etc.

Revoking Privileges:

User's object privileges and system privileges can be revoked by using REVOKE statement.
eg:To revoke UPDATE privilege granted to SUSHANT on CUSTOMER: SQL> REVOKE UPDATE ON CUSTOMER FROM SUSHANT;
To revoke multiple privileges, use comma to separate multiple privileges: SQL> REVOKE CREATE SESSION, SELECT ANY TABLE FROM SUSHANT;
To revoke REFERENCES privilege, specify CASCADE CONSTRAINTS clause, which will drop the referential integrity constraints created using the privileges. You must use this clause if any constraints exists.
SQL> REVOKE REFERENCES ON CUSTOMER FROM SUSHANT CASCADE CONSTRAINTS;
To revoke all privileges:
SQL> REVOKE ALL ON CUSTOMER FROM SUSHANT;

Roles:

Role is a named set of related privileges that are granted to users or to other roles, which eases the management of privileges and hence improve security.

Role Characteristics

- Privileges are granted to and revoked from roles as if the role were a user.
- Roles can be granted to and revoked from users or other roles as if they were system privileges.
- A role can consist of both system and object privileges.
- A role can be enabled or disabled for each user who is granted the role.
- A role can require a password to be enabled.
- Roles are not owned by anyone; and they are not in any schema.

Benefits of Roles

- Easier privilege management.
- Dynamic privilege management.
- Selective availability of privileges.
- Can be granted through the operating system.

Easier Privilege Management

Use roles to simplify privilege management. Rather than granting the same set of privileges to several users, you can grant the privileges to a role, and then grant that role to each user.

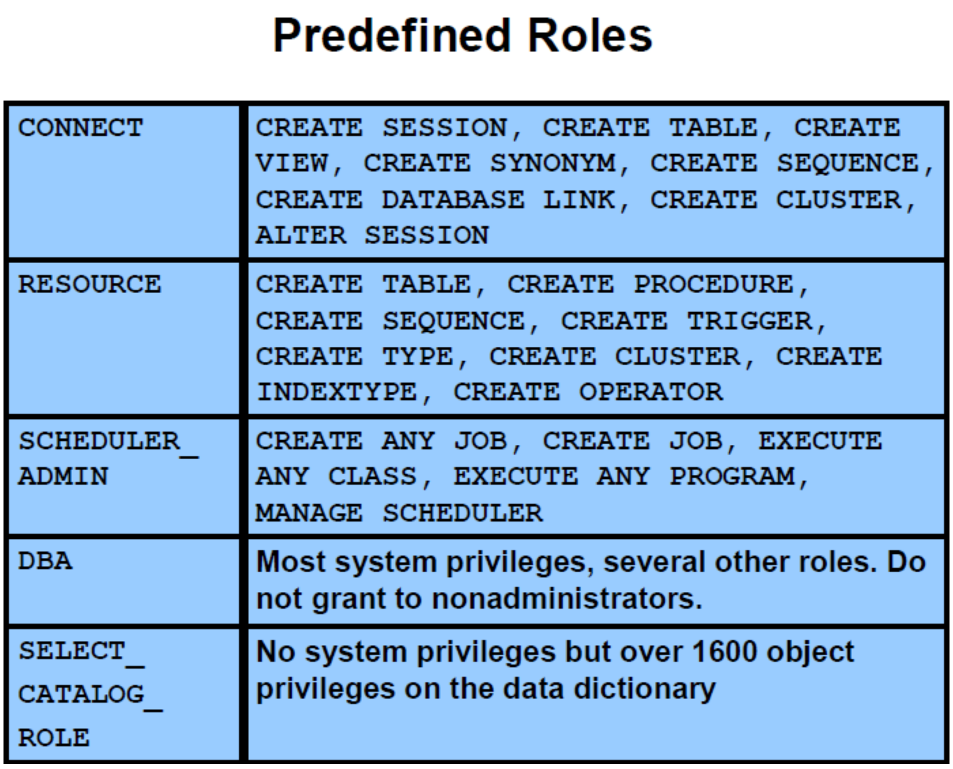
Dynamic Privilege Management

If the privileges associated with a role are modified, all the users who are granted the role acquire the modified privileges automatically and immediately.

Selective Availability of Privileges

Roles can be enabled and disabled to turn privileges on and off temporarily. Enabling a role can also be used to verify that a user has been granted that role.

Granting Through the Operating System
Operating system commands or utilities can be used to assign roles to users in the database.



Predefined Roles

CONNECT	CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE SYNONYM, CREATE SEQUENCE, CREATE DATABASE LINK, CREATE CLUSTER, ALTER SESSION
RESOURCE	CREATE TABLE, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TRIGGER, CREATE TYPE, CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR
SCHEDULER_ ADMIN	CREATE ANY JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
DBA	Most system privileges, several other roles. Do not grant to nonadministrators.
SELECT_ CATALOG_ ROLE	No system privileges but over 1600 object privileges on the data dictionary

Creating Role:
Using CREATE ROLE command creates the role. No user owns the role; it is owned by the database. When roles are created, no privileges are associated with it. Privileges will be granted to roles later.
eg: SQL> CREATE ROLE MANAGER;
SQL> GRANT SELECT ON CUSTOMER TO MANAGER;
SQL> GRANT INSERT, UPDATE ON CUSTOMER TO MANAGER;
Similar to user, role can also be authenticated. The default is NOT IDENTIFIED which means no authorization is required.

Two authorization methods are available:

1. Database:
Database authorizes role by using a password associated with the role. Whenever such roles are enabled, user is prompted for a password. Eg.
SQL> CREATE ROLE MANAGER IDENTIFIED BY MNGR123;

2. Operating System:
Role is authorized by Operating System when the OS can associate its privileges with the application privileges, and information about each user is configured in operating system files. To enable OS role authorization, set the parameter OS_ROLES=TRUE;

Eg. SQL> CREATE ROLE APPLICATION_USER IDENTIFIED EXTERNALLY;

Removing Roles:

Use DROP ROLES statement to remove role from database. Dropping a role will cause all privileges, that users had through the role, to get lost.
eg: SQL> DROP ROLE admin;

Querying Role Information:

- The data dictionary view DBA_ROLES lists the roles defined inthe database. The column PASSWORD specifies the authorization method.

```
SQL> SELECT * FROM DBA_ROLES;
ROLE PASSWORD
CONNECT       NO
RESOURCE      NO
DBA    NO
LOGSTDBY_ADMINISTRATOR NO
```

- The view SESSION_ROLES lists the roles that are enabled inthe current session

```
SQL> SELECT * FROM SESSION_ROLES;
```

- The view DBA_ROLE_PRIVS (or USER_ROLE_PRIVS) lists all theroles granted to users and roles.

```
SQL> SELECT * FROM DBA_ROLE_PRIVS;
```

- The view ROLE_ROLE_PRIVS lists the roles granted to theroles, ROLE_SYS_PRIVS lists the system privileges granted to role and ROLE_TAB_PRIVS shows information on the object privileges granted to role.

```
SQL> SELECT * FROM ROLE_ROLE_PRIVS WHERE ROLE='DBA';
ROLE   GRANTED_ROLE        ADM

   DBA            SCHEDULER_ADMIN                        YES
   DBA            DELETE_CATALOG_ROLE                    YES
   DBA            EXP_FULL_DATABASE                      NO

SQL> SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE='CONNECT';
ROLE   PRIVILEGE      ADM
CONNECT       CREATE SESSION        NO

SQL> SELECT * FROM ROLE_TAB_PRIVS WHERE TABLE_NAME='EMPLOYEE';
```

Database Security and Auditing

Database Security:

Database administrator should follow best practices and continuously monitor database activity for a secure system.
A secure system ensures the confidentiality of the data it contains. There are several aspects of security:

- Restricting access to data and services.
- Authenticating users.
- Monitoring for suspicious activity.

Restricting Access to Data and Services:

All users should not have access to all data. Depending on what is stored in your database, restricted access can be mandated by business requirements, customer expectations, and increasingly by legal restrictions. Credit card information, health care data, identity information, and more must be protected from unauthorized access. Oracle provides extremely fined-grained authorization controls to limit database access.
Restricting access should include applying the principal of least privilege.

Authenticating User:

The most basic form of user authentication is by challenging the user to provide something they know such as a password. Ensuring that passwords by following simple rules can greatly increase the security of your system.
Stronger authentication methods include requiring the user to provide something they have, such as a token or Public Key Infrastructure (PKI) certificate. An even stronger form of authentication is to identify the user through a unique biometric characteristic such as a fingerprint, iris scan, bone structure patterns, and so on. Oracle supports advanced authentication techniques such as token-, biometric-, and certificate-based identification through the Advanced Security Option. User accounts that are not in use should be locked to prevent attempts to compromise authentication.

Monitoring for Suspicious Activity:

Even authorized, authenticated users can sometimes compromise your system. Identifying unusual database activity such as an employee who suddenly begins querying large amounts of credit card information, research results, or other sensitive information, can be the first step to detecting information theft. Oracle provides a rich set of auditing tools to track user activity and identify suspicious trends.
Monitoring or auditing should be an integral part of your security procedures. Oracle's built-in audit tools include:

- Standard Database auditing
- Value-based auditing
- Fine-grained auditing (FGA)

Standard database auditing captures several pieces of information about an audited event including that the event occurred, when it occurred, the user who caused the audited event, and which client machine the user was on when the event happened.
Use value-based auditing to audit changes to data (inserts, updates, deletes). Value-based auditing extends standard database auditing, capturing not only that the audited event occurred, but the actual values that were inserted, updated, or deleted. Value-based auditing is implemented through database triggers.
Use fine-grained auditing (FGA) to audit SQL statements. FGA extends standard database auditing, capturing the actual SQL statement that was issued rather than only that the action occurred.

Auditing Database:

Auditing is storing information about database activity. You can use auditing to monitor suspicious database activity and to collect statistics on database usage. On creation of database, Oracle creates a SYS.AUD$ table, known as the audit trail, which stores the audited records.
To enable auditing, set the initialization parameter AUDIT_TRAIL to TRUE or DB. When this parameter is set to OS, Oracle writes the audited records to an OS file instead of inserting them to SYS.AUD$ table.Use the AUDIT command to specify the audit actions.

Oracle has three types of auditing capabilities:

- **Statement auditing:** Audits SQL statements. (AUDIT SELECT BY SUSHANT audits all SELECT statements performed bySUSHANT)
- **Privilege auditing:** Audits privileges (AUDIT CREATE TRIGGER audits all user who exercises their CREATE TRIGGER privileges)
- **Object auditing:** Audits the use of specific object. (AUDIT SELECTON HR.EMPLOYEES monitors the SELECT statement performed on the EMPLOYEES table)

BY clause: restricts the auditing scope by specifying the user list in it.
WHENEVER SUCCESSFUL clause: to specify that only successful statements are to be audited.
WHENEVER NOT SUCCESSFUL clause: to limit auditing to failed statements.
BY SESSION clause: specifies that one audit record is inserted for one session, regardless of number of times statement is executed.
BY ACCESS clause: specifies that one audit record is inserted each time the statement is executed.
Example:
To audit connections and disconnections to the database: AUDIT SESSION; To audit only successful logins: AUDIT SESSION WHENEVER SUCCESSFUL; To audit only failed logins: AUDIT SESSION WHENEVER NOT SUCCESSFUL;
To audit successful logins for specific users: AUDIT SESSION BY SUSHANT, RAM WHENEVER SUCCESSFUL;
To audit successful updates and deletes on the EMPLOYEES table: AUDIT UPDATE, DELETE ON HR.EMPLOYEES BY ACCESS WHENEVER SUCCESSFUL;

To **turn off** all auditing, use NOAUDIT command. You can use all commands available in AUDIT statement except BY SESSION and BY ACCESS.
NO AUDIT UPDATE, DELETE ON HR.EMPLOYEES;

Creating and managing DB objects (Tables, indexes, triggers, views, stored procedures, etc.)

Creating Tables:

Use CREATE TABLE command to create table. You can create a table under the username used to connect to the database or with proper privileges; you can create a table under another username.
CREATE TABLE STUDENT(SSN NUMBER,
NAME VARCHAR2(20),
GRADE VARCHAR(10), AGE NUMBER);
Some of the data types used for creating table are: CHAR(<size>[BYTE | CHAR]), VARCHAR(<size>[BYTE | CHAR]), NUMBER(<precision>,<scale>), DATE, etc.

Altering Tables:

Use ALTER TABLE command to change the table's storage settings, add or drop columns, or modify the columns characteristics such as default value, datatypes, and length.

Dropping table:

If a table is no longer used, you can drop it to free up space. Once dropped, it cannot be undone. The syntax is :
DROP TABLE [schema.] table_name [CASCADE CONSTRAINTS]

Truncating a table:

It is similar to DROP, but it doesn't remove the structure of table, so none of the indexes, constraints, triggers, and privileges on the table are dropped. You cannot roll back a truncate operation. The syntax is:
TRUNCATE {TABLE | CLUSTER} [<schema>.] table_name [{DROP | REUSE} STORAGE].
You cannot truncate the parent table of an enabled referential integrity constraint. You must first disable constraint and then truncate thetable even if the child table has no rows.

Querying table information:

DBA_TABLES/USER_TABLES/ALL_TABLES: use this view to query information about tables (TABS is a synonym for USER_TABLES)
DBA_TAB_COLUMNS: use this view to display information about the columns in a table.

Indexes:

Indexes are used to access data more quickly than reading the whole table, and they reduce

disk I/O considerably when the queries use the available indexes. You can create any number of indexes on a table. A column can be a part of many indexes and you can specify as many as 30 columns in an index. When you specify more than one column, the index is known as a composite index.
Following types of indexes can be created:
Bitmap: A bitmap index doesn't repeatedly store the index column values. Each value is treated as a key, and a bit is set for the corresponding ROWIDs. Bitmap indexes are suitable for columns with low cardinality, such as the SEX column in an EMPLOYEE table, in which possible values are M and F.
B-tree: This type of index is the default. You can create the index by using the b-tree algorithm. The b-tree includes nodes with the index column values and the ROWID of the row. The ROWIDs identify the rows in the table.
Types of b-tree indexes:

- **Non-unique**
- **Unique**
- **Reverse Key**
- **Function-based Creating Index:**

The CREATE INDEX statement creates a non-unique b-tree index on the column specified. You must specify a name for the index and the table name on which the index should be built.
Eg:
To create index on ORDER_DATE column of the ORDERS table:
CREATE INDEX IND1_ORDERS ON ORDERS (ORDER_DATE);
To create a unique index: CREATE UNIQUE INDEX IND2_ORDERS ON ORDERS (ORDER_DATE);
To create a bitmap index: CREATE BITMAP INDEX IND3_ORDERS ON ORDERS (STATUS);
To create a reverse key index on ORDER_NUM and ORDER_DATE columns:
CREATE UNIQUE INDEX IND4_ORDERS ON ORDERS (ORDER_DATE, ORDER_NUM) TABLESPACE USER_INDEX REVERSE;
To create a function-based index on SUBSTR(PRODUCT_ID,1,2): CREATE INDEX IND5_ORDERS ON ORDERS (SUBSTR(PRODUCT_ID,1,2)) TABLESPACE USER_INDEX;

Altering Indexes:

Using the ALTER INDEX command, you can make following changes on an Index:

- Change it's STORAGE clause, except for the parameter INITIAL and MINEXTENTS
    - Deallocate unused blocks
    - Rebuild the index
    - Coalesce leaf nodes
    - Manually allocate extents
    - Change the PARALLEL/NOPARALLEL, LOGGING/NOLOGGING clauses
- Modify partition storage parameters, rename partitions, drop partitions and so on.
    - Rename the index
- Specify the ENABLE/DISABLE clause to enable or disable function- based indexes

Eg:
To allocate an extent:
ALTER INDEX STUDENT.IND1_ORDERS ALLOCATE EXTENT SIZE 200K;

Dropping Indexes:

Use DROP INDEX command to drop index. Oracle frees up all the space used by the index when it is dropped.
DROP INDEX STUDENT.IND4_ORDERS;

Managing Constraints:

Constraints are created in the database to enforce a business rule and to specify relationships between various tables. You can also enforce business rules by using database triggers and application code.
There are five types of integrity constraints that prevents bad data from entering database:
NOT NULL: Prevents NULL values from being entered into the column.
CHECK: Checks whether the condition specified in the constraints is satisfied.
UNIQUE: Ensures that there are no duplicate values for the column(s) specified. Every value or set of values is unique within the table.
PRIMARY KEY: Uniquely identifies each row of the table. Prevents NULL values. A table can have only one PRIMARY KEY constraint.
FOREIGN KEY: Establishes a parent-child relationship between tables by using common columns.

Creating Constraints:

Use CREATE TABLE or ALTER TABLE statements to create constraints. You can specify the constraint definition at the column level if the constraint is defined on a single column. You define multiple column constraints at the table level; specifying the columns in parenthesis separated by comma. To provide name for constraint use keyword CONSTRAINT followed by constraint_name.

NOT NULL:

Characteristics:

- Is defined at column level
- Use CREATE TABLE to define constraint when creating a table CREATE TABLE STAFF(

SSN NUMBER NOT NULL, NAME VARCHAR(19), JOIN_DATE DATE
CONSTRAINT SS_JOIN_DATE NOT NULL
);

- Use ALTER TABLE MODIFY to add or remove a NOT NULL constraint on the column of

existing table.
ALTER TABLE STAFF MODIFY JOIN_DATE NULL;

CHECK:

Characteristics:

- Can be defined at column level or table level.
- The condition specified in the CHECK clause should evaluate to a Boolean result and can refer to values in other columns of the same row; the condition cannot use queries.
- One column can have more than one CHECK constraint defined. The column can have a NULL value.
- CREATE TABLE BONUS(EMP_ID VARCHAR2(20) NOT NULL, SALARY NUMBER(9,2),

BONUS NUMBER(9,2),
CONSTRAINT CK_BONUS CHECK(BONUS > 0));

Unique:

Characteristics:

- Can be defined at column level for single-column unique keys. For multiple-column unique key(maximum number of columns can be 32), the constraint should be defined at table level.
    - Unique constraints allow NULL values in the constraint columns.
    - ALTER TABLE BONUS

ADD CONSTRAINT UQ_EMP_ID UNIQUE (DEPT, EMP_ID);

Primary key:

Characteristics:

- All characteristics of UNIQUE key are applicable except that NULL values are not allowed in the primary key columns.
    - A table can have only one PRIMARY KEY.
- Oracle creates a unique index and NOT NULL constraints for each column in the key.
- CREATE TABLE EMPLOYEE(DEPT_NO VARCHAR2(2), EMP_ID NUMBER(4), NAME VARCHAR2(20),

CONSTRAINT PK_EMPLOYEE PRIMARY KEY (DEPT_NO, EMP_ID));

Foreign Key:

The foreign key is the column or columns in the table (child table) in which the constraint is created; the referenced key is the primary key column(s) in the table (parent table) that is referenced by the constraint.

Characteristics:

- Can be defined at the column level or table level. Define multiple- column foreign keys at the table level.

- The foreign key column(s) and referenced key column(s) can be in the same table
- NULL values are allowed in the foreign key columns.
- The ON DELETE clause specifies the action to be taken when a rowin the parent table is deleted and child rows exist with the deleted parent primary key. You can delete the child rows (CASCADE) or set the foreign key column values to NULL.

ALTER TABLE CITY ADD CONSTRAINT FK_STATE FOREIGN KEY (COUNTRY_CODE, STATE_CODE)
REFERENCES STATE (COUNTRY_CODE, STATE_CODE)
ON DELETE CASCADE;

Creating Disabled Constraints:
Newly created constraint is automatically enabled. To create disabled constraint use DISABLE keyword after constraint definition.

Dropping Constraints:
To drop constraint, use ALTER TABLE and specify the constraint name. ALTER TABLE BONUS DROP CONSTRAINT CK_BONUS2;
To drop unique key constraints with referenced foreign keys, specify the CASCADE clause to drop the foreign key constraints and the unique constraint. Specify the unique key columns.
ALTER TABLE EMPLOYEE DROP UNIQUE (EMP_ID) CASCADE;

Views:
A view is a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.
A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.
Views, which are a type of virtual tables allow users to do the following −
Structure data in a way that users or classes of users find natural or intuitive.

- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used togenerate reports.

Creating Views:
Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.
To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows − CREATE VIEW view_name AS
SELECT column1, column2..... FROM table_name
WHERE [condition];
You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

The WITH CHECK OPTION
The WITH CHECK OPTION is a CREATE VIEW statement option whose purpose is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.
If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.
The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
```

```
SELECT name, age
FROM   CUSTOMERS
```

```
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View
A view can be updated under certain conditions which are given below:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the ageof Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35
WHERE name = 'Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+


   |  1  |  Ramesh    |  35  |  Ahmedabad   |  2000.00   |
   |  2  |  Khilan    |  25  |  Delhi       |  1500.00   |
   |  3  |  kaushik   |  23  |  Kota        |  2000.00   |
   |  4  |  Chaitali  |  25  |  Mumbai      |  6500.00   |
   |  5  |  Hardik    |  27  |  Bhopal      |  8500.00   |
+-|  6  |  Komal     |  22  |  MP          |  4500.00   |
   |  7  |  Muffy     |  24  |  Indore      |  10000.00  |
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.
Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
 WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
|    1  |  Ramesh    |  35  |  Ahmedabad  |  2000.00   |
+-|  2  |  Khilan    |  25  |  Delhi      |  1500.00   |
  |  3  |  kaushik   |  23  |  Kota       |  2000.00   |
  |  4  |  Chaitali  |  25  |  Mumbai     |  6500.00   |
  |  5  |  Hardik    |  27  |  Bhopal     |  8500.00   |
  |  7  |  Muffy     |  24  |  Indore     |  10000.00  |


+----+----------+-----+-----------+----------+
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

Importance of View:

- Views are created for the security purposes for the database. It is always helpful for us to restrict the people from the outside to see the information like columns and data in the columns for the sake of the security of the data access.
- We can use view for abstraction of the data in the table which is not known by the end user.
- The data accessible through view is not stored anywhere in the database as a distinct object.
- A view can join two or more tables and show it as one object to user.

Triggers:

A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run *instead of* or *after* DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.
Triggers help the database designer ensure certain actions are completed regardless of which program or user makes changes to the data.
The programs are called triggers since an event, such as adding a record to a table, fires their execution.

Events

The triggers can occur AFTER or INSTEAD OF a DML action. Triggers are associated with the database DML actions INSERT, UPDATE, and DELETE. Triggers are defined to run when these actions are executed on a specific table.

AFTER triggers

Once the DML actions, such as an INSERT completes, the AFTER trigger executes. Here are some key characteristics of AFTER triggers:

- After triggers are run after a DML action, such as an INSERT statement and any ensuing referential cascade actions and constraint checks have run.
- You can't cancel the database action using an AFTER trigger. This is because the action has already completed.
- One or more AFTER triggers per action can be defined on a table, but to keep things simple I recommend only defining one.
- You can't define AFTER triggers on views.

INSTEAD OF triggers

INSTEAD OF triggers, as their name implies, run in place of the DML action which caused them to fire. Items to consider when using INSTEAD OF triggers include:

- An INSTEAD OF trigger overrides the triggering action. If an INSTEAD OF trigger is defined to execute on an INSERT statement, then once the INSERT statement attempt to run, control is immediately passed to the INSTEAD OF trigger.
- At most, one INSTEAD OF trigger can be defined per action for a table.
- Triggers use two special database objects, INSERTED and DELETED, to access rows affected by the database actions. Within the scope of a trigger the INSERTED and DELETE objects have the same columns as the trigger's table.
- The INSERTED table contains all the new values; whereas, the DELETED table contains old values. Here is how the tables are used:
- INSERT – Use the INSERTED table to determine which rows were added to the table.
- DELETE – Use the DELETED table to see which rows were removed from the table.
- UPDATE – Use the INSERTED table to inspect the new or updated values and the DELETED table to see the values prior to update.

Creating a trigger:

The syntax for creating a trigger is:

```
CREATE OR REPLACE TRIGGER trigger_name
{BEFORE | AFTER |INSTEAD OF}
{INSERT [OR]|UPDATE[OR]|DELETE}
[OF col_name] ON table_name
[REFERENCING OLD AS o NEW AS n] [FOR EACH ROW]
WHEN(condition) DECLARE
declaration_statements BEGIN
executable_statements EXCEPTION
```

exception-handling-statements END;
Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing
- trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF}: This specifies when the trigger will be executed.
- The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that will be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row- level triggers.

EXAMPLE:

```
Select * from customers;


+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |

|  2 | Khilan   |  25 | Delhi     |  1500.00 |

|  3 | kaushik  |  23 | Kota      |  2000.00 |

|  4 | Chaitali |  25 | Mumbai    |  6500.00 |

|  5 | Hardik   |  27 | Bhopal    |  8500.00 |

|  6 | Komal    |  22 | MP        |  4500.00 |

+----+----------+-----+-----------+----------+
```

This trigger will display the salary difference between the old values and new values:
CREATE OR REPLACE TRIGGER display_salary_changes BEFORE DELETE OR INSERT OR UPDATE ON customers FOR EACH ROW
WHEN (NEW.ID > 0) DECLARE
sal_diff number; BEGIN
sal_diff := :NEW.salary - :OLD.salary; dbms_output.put_line('Old salary: ' ||
:OLD.salary);
dbms_output.put_line('New salary: ' ||
:NEW.salary);

```
dbms_output.put_line('Salary difference: ' || sal_diff);
END;    /
```

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, display_salary_changes will be fired and it will display the following result:

```
Old salary:
New salary: 7500
Salary difference:
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table:

```
UPDATE customers
SET salary = salary + 500 WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, display_salary_changes will be fired and it will display the following result:

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

Benefits of Triggers

Triggers can be written for the following purposes:

- Generating some derived column values automatically.
- Enforcing referential integrity.
- Event logging and storing information on table access.
- Auditing.
- Synchronous replication of tables.
- Imposing security authorizations.
- Preventing invalid transactions.