SQL (Structured Query Language) ::

SQL (Structured Query Language) is a standardized query language used to manage and manipulate relational databases. It is designed for querying, updating, inserting, and deleting data in a database. SQL allows users to interact with the database to perform various tasks, such as:

1. **Querying Data**: Using `SELECT` statements to retrieve data from one or more tables.
2. **Inserting Data**: Using `INSERT INTO` statements to add new records to a table.
3. **Updating Data**: Using `UPDATE` statements to modify existing records.
4. **Deleting Data**: Using `DELETE` statements to remove records from a table.
5. **Database Structure**: Defining and modifying tables, indexes, and relationships using `CREATE`, `ALTER`, and `DROP` statements.
6. **Data Security**: Controlling access to data using commands like `GRANT`, `REVOKE` and `DENY`

SQL is used in many popular database systems, including **MySQL**, **PostgreSQL**, **Microsoft SQL Server**, and **SQLite**.

SQL allows us (authenticated users only) to execute complex operations (transactions) on database objects.

In SQL we have 5 Sub-Languages. They are listed here :

1. Data Definition Language (**DDL**)
2. Data Manipulation Language (**DML**)
3. Data Control Language (**DCL**)
4. Transaction Control Language (**TCL**)
5. Data Query Language (**DQL**)

Data Definition Language (DDL) : AUTO-COMMIT

- DDL is used to define and manage database structures, such as tables, schemas, indexes, and views.
- It includes commands to create, modify, and delete database objects. The primary DDL commands are:
  CREATE, ALTER, RENAME, TRUNCATE, DROP
- By default, all DDL commands are auto-commit. It means that Whatever transactions are performed using DDL those transactions will be executed in database immediately.

Data Definition Language :

CREATE :

By using ‘create’ command we can create objects in database such as database, table, index, view, trigger, function, procedure, sequence …

CREATE TABLE -------------------- Create a new table.
CREATE DATABASE ------------- Create a new database.
CREATE INDEX ------------------- Create an index on one or more columns.
CREATE VIEW --------------------- Create a view.
CREATE PROCEDURE ------------ Create a stored procedure.
CREATE TRIGGER ----------------- Create a trigger.
CREATE SCHEMA ----------------- Create a schema to organize objects.
CREATE USER ---------------------- Create a new user (in some DBMS).
CREATE SEQUENCE -------------- Create a sequence to generate unique values.

Table Creation Using Create Command :

Creating a table (Normal Table) :

```
CREATE    TABLE   <TABLE_NAME>   (
        COLUMN_1 <DATATYPE> [CONSTRAINT] [DEFAULT <VALUE>],
        COLUMN_2  <DATATYPE> [CONSTRAINT] [DEFAULT <VALUE>],
        COLUMN_3  <DATATYPE> [CONSTRAINT] [DEFAULT <VALUE>],
        *
);
```

NOTE :   [***]   INDICATES THAT – FIELD IS OPTIONAL
         < *** > INDICATES THAT – FIELD IS MANDATORY

Creating Table using foreign key:

```
 CREATE TABLE departments (
             id INT PRIMARY KEY,           -- Primary key for the parent table
             dept VARCHAR(100) NOT NULL
 );
CREATE TABLE employees (
             id INT PRIMARY KEY,                  -- Primary key for the child table
             name VARCHAR(100) NOT NULL,
             dept INT,                     -- FKreferring to departments
       FOREIGN KEY (dept) REFERENCES departments(id)
);
```

Creating table from existing table (with custom columns):

```
CREATE TABLE <new_table>  AS
SELECT * FROM <existing_table>;
```

CREATING TEMP TABLE (SESSION BASED):

```
CREATE TEMPORARY TABLE <table_name> (
   column1 datatype,
   column2 datatype,
   ...
);
```

Creating table using select into statement:

```
SELECT column1, column2, ...
INTO new_table
FROM existing_table
[ WHERE condition ];
```

Creating table using IF NOT EXISTS:

```
CREATE TABLE IF NOT EXISTS table_name (
   column1 datatype,
   column2 datatype,
   ...
);
```

Creating table with Partitioning: (**Advanced**):

```
CREATE TABLE sales (
   sale_id      INT,
   sale_date    DATE,
   amount       DECIMAL(10, 2)
)
PARTITION BY RANGE (YEAR(sale_date)) (
   PARTITION   p0   VALUES LESS THAN (2020),
   PARTITION   p1   VALUES LESS THAN (2025)
);
```

Note: ** [we have types in partitioning]**

ALTER:

- The ALTER command in SQL is used to modify the structure of an existing database object, such as a table, column, index, or constraint.
- This command allows you to make changes without having to recreate the entire object.
- You can use ALTER to add, drop, or modify columns, rename tables, or change other aspects of database objects.

Add a Column to a Table:

```
ALTER TABLE table_name
ADD column_name datatype [DEFAULT default_value];
```

Note:

Impact: Adding a new column to a table with existing data typically does not affect the existing data in the table. However, the new column will contain NULL values (unless you specify a DEFAULT value).

Modifying a Column:

```
ALTER TABLE table_name
MODIFY column_name  new_datatype;
```

Note:

Impact: When modifying a column's data type or constraints, it's important to consider the existing data because the operation could fail if the existing data does not match the new column constraints or data type.

Renaming a Column:

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```

Drop a column

```
ALTER TABLE table_name
DROP COLUMN column_name [CASCADE CONSTRAINTS];
```

Other operations:

Renaming table:

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```

Adding pk:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (column_name);
```

Adding FK:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name FOREIGN KEY (column_name) REFERENCES
parent_table (parent_column);
```

Droping PK:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

RENAME:

Renaming the existing table:

```
RENAME old_table_name TO new_table_name;
```

Renaming the column of a table:

```
ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;
```

Renaming the Index:

```
ALTER INDEX old_index_name RENAME TO new_index_name;
```

Renaming the Constraint:

```
ALTER TABLE table_name RENAME CONSTRAINT old_constraint_name TO
new_constraint_name;
```

TRUNCATE :

The TRUNCATE command removes all rows from a table, effectively resetting the table to an empty state. Unlike DELETE, TRUNCATE is not transactional and cannot be rolled back once executed, unless the session is under a transaction.

Truncate existing table:

```
TRUNCATE TABLE table_name;
```

DROP :

DROP command is a **Data Definition Language (DDL)** operation used to completely remove a database object, such as a table, index, view, or other schema objects, from the database. Once an object is dropped, it cannot be recovered unless you have a backup.

Characteristics of DROP :

- **Permanent Deletion**: The object is permanently deleted from the database, including all its data and structure.
- **No Rollback**: DROP is a DDL operation, and like other DDL commands, it cannot be rolled back once executed (unless inside a transaction, but even then, it is only temporarily undone within that session).
- **Affects Dependencies**: Dropping an object may affect other objects that depend on it, such as foreign key constraints, triggers, views, or other references.

Syntax:

```
DROP <Object>  <Name>;
```

Object may be a [view, database, table, sequence, procedure, function, trigger] etc…

DATA MANIPULATION LANGUAGE

DML stands for **Data Manipulation Language**.

- It is a subset of SQL (Structured Query Language) used for managing data within relational databases.
- DML is primarily concerned with the manipulation of data, rather than the structure of the database itself.
- The common DML commands are:
  1) INSERT
  2) INSERT ALL
  3) UPDATE
  4) DELETE

INSERT:

WAY 01 :

```
INSERT INTO
<TABLE_NAME>   (column1, column2, column3, ...)
VALUES         (value1, value2, value3, ...);

[ columns count == values count ]
```

WAY 02 :

```
INSERT ALL
        INTO table1 (column1, column2, ...)
                VALUES (value1, value2, ...)
        INTO table2 (column1, column2, ...)
                VALUES (value1, value2, ...)
 ...
SELECT * FROM dual;
```

NOTE:

dual: A special dummy table in Oracle used when no actual data is needed in the SELECT statement (usually for operations like this).

WAY 03 : ***** imp ****

Table is having 5 columns but I want to insert data into first 2 columns only

INSERT INTO
<TABLE_NAME> (column1, column2)
VALUES (value1, value2);
[columns count == values count]

Note: Mentioning the columns details after table name is optional]

INSERT INTO
<TABLE_NAME> VALUES (value1, value2);

Note: Dynamic data binding

```
INSERT INTO employees (first_name, last_name, salary)
VALUES ('&first_name', '&last_name', &salary);
```

UPDATE :

UPDATE statement is used to modify existing data in a table. You can update one or more columns for specific rows based on a condition.

```
UPDATE    table_name
SET   column1 = value1,   column2 = value2,   ...
WHERE condition;
```

[If we do not use where condition on that time all records will be updated with provided value]

DELETE :

In Oracle, DML (Data Manipulation Language) is a subset of SQL used to manage data within tables. The DELETE operation is a part of DML and is used to remove one or more rows from a table.

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

DELETE vs. TRUNCATE:

- The DELETE operation can delete specific rows and allows the use of conditions, and it is slower because it logs each row deletion.
- The TRUNCATE operation removes **all rows** in a table, but it is faster and does not log individual row deletions. However, TRUNCATE cannot be rolled back in the same way that DELETE can.

TCL (Transaction Control Language)

Transaction Control Language :

- **TCL** is a subset of SQL used to manage the changes made by DML (Data Manipulation Language) statements like INSERT, UPDATE, and DELETE.
- TCL allows you to control the transaction boundaries, ensuring that a series of operations can be treated as a single unit of work.
- The primary purpose of TCL is to ensure the **ACID properties** (Atomicity, Consistency, Isolation, Durability) of transactions.

COMMIT:

- The COMMIT command is used to permanently save all changes made during the current transaction to the database. After a COMMIT, the changes become visible to other users and the transaction is complete.
- Syntax: **COMMIT;**

ROLLBACK:

- The ROLLBACK command is used to undo any changes made during the current transaction. It reverts all DML operations (like INSERT, UPDATE, DELETE) to their previous state.
- Syntax: **ROLLBACK**

SAVEPOINT:

- A SAVEPOINT is used to set a point within a transaction to which you can later roll back. It allows partial rollback within a larger transaction.
- Syntax: **SAVEPOINT savepoint_name;**

SET TRANSACTION:

- The SET TRANSACTION command is used to configure the properties of the current transaction, such as setting the isolation level or setting the transaction mode.
- Syntax: SET TRANSACTION ISOLATION LEVEL ;

LEVELS:

READ COMMITTED: This is the default isolation level in Oracle. A transaction can only see committed data. If another transaction modifies data, the current transaction sees the updated value only after it is committed.
SERIALIZABLE: This isolation level provides the highest level of isolation. It ensures that the current transaction sees a consistent snapshot of the data and no other transactions can modify data that it reads until it is completed.
READ ONLY: Ensures that the transaction can only read data but cannot modify it. Useful for long-running reports or querying.
READ WRITE: Allows both reading and writing of data. This is typically the default unless explicitly set.

A savepoint is a named point within a transaction that allows you to **roll back** only a portion of the work done in the transaction, instead of undoing the entire transaction. This can be useful if you want to revert to a known state within a transaction without losing all the progress made up to that point.

Sytax:

ROLLBACK TO SAVEPOINT savepoint_name;

DCL (Data Control Language) :

- DCL in Oracle SQL refers to a subset of SQL commands used to manage access to data and control permissions within a database.
- DCL primarily consists of the following two commands:
  1) GRANT
  2) REVOKE

GRANT:

The GRANT statement is used to give users or roles permission to perform certain operations (such as SELECT, INSERT, UPDATE, DELETE) on database objects (tables, views, procedures, etc.). It can be applied to individual users or roles, allowing for specific privileges to be granted.

Syntax:

GRANT privilege_name ON object_name TO user_or_role;
Example: GRANT SELECT, INSERT ON employees TO john;

GRANT WITH OPTION:

John can grant those select and insert to other users too;
GRANT SELECT, INSERT ON employees TO john WITH GRANT OPTION;

REVOKE

The REVOKE statement is used to remove previously granted permissions from a user or role. It allows administrators to revoke specific privileges from users.

REVOKE privilege_name ON object_name FROM user_or_role;

Example:
REVOKE SELECT, INSERT ON employees FROM john;

In case of assigning permissions to ROLE;

GRANT SELECT, INSERT, UPDATE ON employees TO <ROLE_NAME>;

OBJECT PRIVILEGES

Object privileges allow users to perform operations on specific database objects (e.g., tables, views, sequences, procedures). Here are the key object privileges:

SELECT: Allows the user to query data from a table, view, or synonym.

- **INSERT**: Allows the user to insert data into a table or view.
- **UPDATE**: Allows the user to modify data in a table or view.
- **DELETE**: Allows the user to delete data from a table or view.
- **ALTER**: Allows the user to modify the structure of a table or view (e.g., adding or removing columns).
- **DROP**: Allows the user to delete a table, view, or other object.
- **INDEX**: Allows the user to create or drop indexes on a table.
- **REFERENCES**: Allows the user to create foreign key constraints that reference a specific table or column.
- **EXECUTE**: Allows the user to execute a stored procedure or function.

Role-Based Privileges

Roles are a collection of privileges that can be granted to users or other roles. Common predefined roles in Oracle include:

- **DBA**: Provides a user with all system privileges in the database (full administrative rights).
- **RESOURCE**: Provides privileges to create and manage database objects (e.g., tables, views).
- **CONNECT**: Provides basic privileges to connect to the database and create objects (like tables and views).
- **SELECT_CATALOG_ROLE**: Provides read access to Oracle data dictionary views.
- **EXECUTE_CATALOG_ROLE**: Provides execute privileges on procedures and functions in the data dictionary.

System Privileges

System privileges allow users to perform operations at the system level (such as creating and managing database objects, modifying users, etc.). Some common system privileges include:

- CREATE SESSION: Allows the user to connect to the database.
- CREATE TABLE: Allows the user to create a new table in the database.
- CREATE VIEW: Allows the user to create a view.
- CREATE PROCEDURE: Allows the user to create a stored procedure or function.
- ALTER SESSION: Allows the user to change session-level settings.
- CREATE USER: Allows the user to create new database users.
- DROP USER: Allows the user to drop a database user.
- GRANT ANY PRIVILEGE: Allows the user to grant any privilege to any other user.
- CREATE ROLE: Allows the user to create roles.

System Privileges

- DROP ROLE: Allows the user to drop roles.
- SELECT ANY TABLE: Allows the user to query any table in the database, regardless of ownership.
- DELETE ANY TABLE: Allows the user to delete from any table.
- INSERT ANY TABLE: Allows the user to insert data into any table.
- UPDATE ANY TABLE: Allows the user to update data in any table.

DQL (Data Query Language)

- DQL (Data Query Language) in Oracle SQL is used for querying data from a database.
- It includes the SELECT statement, which is the primary command used to retrieve data from one or more tables.
- Here's an overview of key concepts and operations in DQL :

SELECTION Statement :

Selection :

- Selection refers to the operation of filtering rows based on a certain condition.
- This operation is analogous to the WHERE clause in SQL, which selects rows that meet a specified condition.
- Example

```
SELECT *
FROM employees
WHERE department = 'Sales';
```

Projection:

- Projection refers to the operation of selecting certain columns from a table, ignoring the others.
- This operation is analogous to the SELECT clause in SQL, which specifies which columns to retrieve.
- Example

```
SELECT first_name,  last_name
FROM employees;
```

SQL USERS:

How to Create USER:

Creating new User

- We can create new user in Database using following command :
- **Syntax:**
  CREATE USER <username> IDENTIFIED BY <password>;
  username: The name of the user you want to create.
  password: The password for the user.

Granting Privileges

- After creating the user, you typically need to assign privileges. The following example grants the user john basic login privileges:
- Syntax:
  GRANT CONNECT, RESOURCE TO <username>;
  CONNECT: Allows the user to log in to the database.
  RESOURCE: Allows the user to create and manage schema objects like tables, views etc…

LOCKING and UNLOCKING USERS

- To lock a user account, preventing the user from logging in, use the following syntax:
- LOCKING: ALTER USER username ACCOUNT LOCK;
- UNLOKING: ALTER USER username ACCOUNT UNLOCK;

Disable/Enable Login for a User Account

- In addition to locking and unlocking the account, we can also disable or enable login access by modifying specific user account settings like password expiration or invalid attempts.

- Here's how to manage some of these settings:
  Expire a User's Password: `ALTER USER username PASSWORD EXPIRE;`

  Set Account Locking after Failed Login Attempts:

```
ALTER  PROFILE   DEFAULT  LIMIT  FAILED_LOGIN_ATTEMPTS 3;
```

Assign a Profile to a User :

To manage resource limits, password policies, and failed login attempts, you can assign a **profile** to a user. Here's how to assign a profile:

```
ALTER USER username PROFILE profile_name;
```

Creating a new Profile

- profiles are used to manage user resource limits and security settings, such as password complexity and account lockout rules.
- We can create and modify profiles to define rules that apply to multiple users.
- Here's how to create a profile and assign it to a user:

```
CREATE PROFILE profile_name
LIMIT
 SESSIONS_PER_USER      number_of_sessions
 CPU_PER_SESSION        cpu_limit
 CONNECT_TIME           connect_time_limit
 IDLE_TIME              idle_time_limit
 PASSWORD_LIFE_TIME     password_lifetime
 FAILED_LOGIN_ATTEMPTS  failed_attempts
 PASSWORD_LOCK_TIME     lock_time
 PASSWORD_REUSE_TIME    reuse_time;
```

Example:

```
CREATE PROFILE user_profile
LIMIT
   SESSIONS_PER_USER       5
   CPU_PER_SESSION        1000
   CONNECT_TIME           120
   IDLE_TIME              30
   PASSWORD_LIFE_TIME     90
   FAILED_LOGIN_ATTEMPTS  3
   PASSWORD_LOCK_TIME     30
   PASSWORD_REUSE_TIME    180;
```

Explanation:

SESSIONS_PER_USER: ====== Maximum of 5 sessions per user. (session: connection to Database)

CPU_PER_SESSION: ====== Maximum CPU usage of 1000 seconds per session.

CONNECT_TIME: ====== Maximum of 120 minutes of continuous connection time.

IDLE_TIME: ====== Session will be automatically logged off after 30 minutes of inactivity.

PASSWORD_LIFE_TIME: ==== Password expires after 90 days.

FAILED_LOGIN_ATTEMPTS:== The account locks after 3 failed login attempts.

PASSWORD_LOCK_TIME:==== Account is locked for 30 minutes after reaching the failed login limit.

PASSWORD_REUSE_TIME: === A user cannot reuse the same password for 180 days.

Assign a Profile to a User

ALTER USER username PROFILE profile_name;

(NOTE: IN CASE OF REMOVING PROFILE FOR A SPECIFIC USER ON THAT TIME WE CAN ASSIGN DEFAULT PROFILE)

Syntax:

ALTER USER john PROFILE DEFAULT;