SQL

SQL Commands:

1.DDL (Data Definition Language):

Purpose: Define or modify the structure of the database (e.g., schemas, tables, indexes).

1. **CREATE** (to create databases, tables, indexes, etc.)

SYNTAX: CREATE DATABASE database_name;

```
CREATE TABLE students (
   id INT,
   name VARCHAR(50),
   grade INT
);
```

2. **ALTER** (to modify an existing table - add column, rename a table, to drop a column)

SYNTAX: ALTER TABLE table_name

ADD column_name datatype;

```
ALTER TABLE students
ADD age INT;
```

3. **DROP** (to delete a table, schema, database) #Deletes the entire table/schema

SYNTAX : DROP TABLE table_name;

```
DROP TABLE students;
```

4. **TRUNCATE** (to remove all records(row) from a table #Deletes all data, keeps table
   Faster than DELETE

SYNTAX : TRUNCATE TABLE table_name;

```
TRUNCATE TABLE students;
```

5. **RENAME** (to rename an table or column)

SYNTAX: RENAME TABLE old_table TO new_table;

```
RENAME TABLE students TO learners;
```

A **schema** is like a **container or namespace** that holds all the **database objects** — such as:

- Tables
- Views
- Indexes
- Functions
- Triggers

A **schema** is like a **folder** inside a database.

Imagine this:

Database = A **whole school**
Schema = A **classroom**

Tables = Students' **registers**

So, a database may have:

- Schema **class10**
  - Table: `students`
  - Table: `marks`
- Schema **class11**
  - Table: `students`
  - Table: `marks`

Each **schema** keeps its own set of tables.

2. **DML (Data Manipulation Language):**

Purpose: Manipulate the data stored in the database. **DML** is used to **work with data** inside tables — not the structure.

1. **SELECT** (Retrieve data from a table)

SYNTAX: SELECT * FROM table_name;

```
SELECT * FROM students;
```

With condition:

SYNTAX:

SELECT column1, column2, ...

FROM table_name

WHERE condition;

SELECT name FROM students WHERE grade = 10;

2. **INSERT** (to add new records(row)into a table)

SYNTAX:

INSERT INTO table_name (column1, column2, ...)

VALUES (value1, value2, ...);

INSERT INTO students (id, name, grade)

VALUES (1, 'Aarav', 10);

3. **UPDATE** (to modify existing records in the table)

SYNTAX:

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

UPDATE students

SET grade = 11

WHERE id = 1;

4. **DELETE** (to remove records from a table)

SYNTAX:

DELETE FROM table_name

WHERE condition;

DELETE FROM students WHERE id = 1;

3. **DCL (Data Control Language):**

Purpose: Control access to data and database objects.

1. **GRANT** (to give user access privileges) Allow users to access/modify data

SYNTAX: GRANT privilege_type ON object_name TO user_name;

Privilege_type = `SELECT, INSERT, UPDATE, etc, .....`

GRANT SELECT, INSERT ON students TO user1;

2. **REVOKE** (to take away privileges) Remove access from users

SYNTAX: REVOKE privilege_type ON object_name FROM user_name;

REVOKE INSERT ON students FROM user1;

4. TCL (Transaction Control Language):

Purpose: Manage transactions within the database.

1. **COMMIT** (to save transactions permanently)

```
BEGIN;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 1000 WHERE account_id = 2;
COMMIT;  # Both updates are saved permanently in the database.
```

2. **ROLLBACK** (to undo transactions(if error occurs))

```
BEGIN;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;
-- Suppose something goes wrong now...
ROLLBACK;  # The change is undone, so ₹1000 is not deducted.
```

3. **SAVEPOINT** (to set a point within a transaction to which you can roll back)

```
BEGIN;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;
SAVEPOINT deduct_done;
UPDATE accounts SET balance = balance + 1000 WHERE account_id = 2;
-- Suppose error here
```

```
ROLLBACK TO deduct_done;

-- Only undo the second update

COMMIT;  # Only the second step is undone, first step stays.
```

Comments Line

1. Single-Line Comments

Use -- at the beginning of a line to comment out one line.

2. Multi-Line Comments

Use /* ... */ to comment out **multiple lines** or block of SQL code.

USE

```
USE classroom;
```

Purpose: This tells the SQL engine to **select the `classroom` database** as the **active database** for all upcoming SQL operations (like creating tables, inserting data, etc.).

DESC

```
DESC emp; or DESCRIBE emp;
```

Purpose : Used in **MySQL** to **display the structure of a table**

Data Modeling

1.Normalization :

Normalization in SQL is a systematic process of organizing the columns (attributes) and tables (relations) of a relational database to minimize data redundancy and improve data integrity. In simpler terms, normalization helps ensure that each piece of data is stored in the best possible place, reducing duplication and making it easier to maintain and update the database.

Why Normalize?

Reduce Data Redundancy: Storing the same data multiple times can waste space and cause inconsistencies if that data needs to change.

Improve Data Integrity: Ensures that your data follows certain rules, preventing errors.

Make Maintenance Easier: Updates, deletions, and insertions become simpler and more efficient when data is well-organized.

Common Normal Forms

The most commonly taught normal forms are:

First Normal Form (1NF).

Second Normal Form (2NF)

Third Normal Form (3NF)

(There are higher normal forms like 4NF and 5NF, but they are more specialized and not always required for most practical Typically, going up to 3NF or BCNF is sufficient.)

1. **First Normal Form (1NF)**

Definition & Purpose (1NF)

1NF requires that every column holds only atomic (indivisible) values and that there are no repeating. groups.

Purpose: To ensure each field contains a single piece of data.

2. **Second Normal Form (2NF)**

Definition & Purpose (2NF)|

2NF requires the table to be in 1NF and that all non-key columns(non -primary key) are fully functionally dependent on the entire primary key.

Purpose: To remove partial dependencies (where a column depends on only part of a composite key).

Example : We have a 5 columns(2 primary key , 3 non-primary key) , partial dependencies refers to

If 3 non-primary key only deponds on either one of the primary key , all the 3 non-primary key must fully dependent on all primary key not paritally.

3. **Third Normal Form (3NF)**

Definition & Purpose (3NF)

3NF requires that the table is in 2NF and that all the columns are directly dependent on the primary key

(i.e., no transitive dependencies).

Purpose: To remove transitive dependencies where a non-key column depends on another non-key column.

2. SCD (Slowly Changing Dimension) is a method used in databases to track and preserve historical changes in data over time.

Why We Use SCD

Track Historical Data: We want to retain older versions of our data for accurate historical analysis and trend reporting (e.g., how many sales happened when the customer had a different address).

Data Accuracy: Ensures that updates to dimension data (like a last name change) won't override previous valid data in a way that confuses analysis.

Simplify Reporting and Analytics: Makes it easier for analysts to query the data, knowing they can trust the historical records.

Main Types of SCD

Type 1: Overwrite (If new updates comes, delete old record by overwriting)

Type 2: Add New Record (historical data + current update data – add this new record in new row)

Type 3: Add New Attribute(add new record by column in same row eg: previous address , new address)

We are not creating new address as a separate row

ACID

Four Pillars of RDBMS

1.Atomicity - A transaction must be **all-or-nothing** — either all steps succeed, or none are applied.If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its previous state.

Eg :

1. In savings, 199 debt(monthly subscription)
   Here debt is Success
2. Credit to Netflix app
   Here credit is Fail

So, return 199

2.Consistency – Consistency ensures that a database starts and ends a transaction in a valid state, preserving all rules, constraints, and relationships. A transaction must transform the database from one valid state to another. If any integrity rule or constraint is violated, the transaction is rolled back.

Eg:

Before the transaction:

trade account: $500

mutual funds: $300

Total: $800

Transaction: Transfer $100 from trade account to mutual funds .

After the transaction:

trade account: $400

mutual funds: $400

Total: $800

3.Isolation: Isolation ensures that concurrent transactions do not interfere with each other. Transactions run independently, even when executed concurrently

Eg:

Person A is transferring $ 100 to Person C

Person B is transferring $ 200 to Person C

Both transactions happen at the same time.

4.Durability: Once a transaction is committed, its changes are permanent,
—even if there's a power failure, crash, or system shutdown right after.

Eg:

Even if the database crashes immediately , after when it comes back online, the transaction will still be reflected in your account balances.

Committed transactions are never lost, even in case of a crash

You transfer ₹1000 from Account A to B. The transaction commits successfully.

Immediately after that, the database crashes. When it restarts, the ₹1000 transfer still reflects in both accounts.