LAB Session 27 Jan
Swaraj Wattamwar
150096724157

a. What is DML?

DML (Data Manipulation Language) is a subset of SQL (Structured Query Language) used to **manipulate data** stored in a database. DML commands are used to retrieve, insert, update, and delete data in tables. Key operation are -

INSERT: Adds new records to a table.

UPDATE: Modifies existing records in a table.

DELETE: Removes records from a table.

SELECT: Retrieves data from a table (read-only operation).

Example

Crate table Student
INSERT INTO Students (student_id, name, age)
VALUES (1, 'John Doe', 20);

B. What are the commands in DML?
DML commands are used to manipulate the data stored in a database.

1. SELECT: Used to retrieve data.
   Example:
   SELECT * FROM Students
2. INSERT: Adds new records to a table.
   Example:
   INSERT INTO Students (StudentID, Name) VALUES (2, 'Jane Smith');
3. UPDATE: Modifies existing data.
   Example:
   UPDATE Students SET Age = 21 WHERE StudentID = 1;

4. DELETE: Removes specific data.
   Example:
   DELETE FROM Students WHERE StudentID = 2;

C. What is a Primary Key?

1. Define Primary Key.

A **Primary Key** is a field or a combination of fields in a database table that uniquely identifies each record in that table. The primary key ensures that no two rows have the same value for this field, providing uniqueness and enabling efficient retrieval of data. It is often used to link data between related tables.

2. Two Important Rules for Primary Key:

1. **Uniqueness**:

The value of the primary key must be unique for each record in the table. No two rows can have the same primary key value.

- **Example**: If id is the primary key, no two records can have the same id.

2. **Non-nullability**:

A primary key cannot have a NULL value. Every record must have a valid (non-null) primary key value.

- **Example**: A student's id cannot be NULL, and every student must have a unique id.

SQL Example:

```
CREATE TABLE Students (
    id INT NOT NULL,
    name VARCHAR(50),
    age INT,
    PRIMARY KEY (id)
);
```

D. What are the types of databases?

1. Relational Databases (RDBMS)

- **Definition**: Relational databases store data in tables (relations) where each table consists of rows and columns. Data in these tables is organized and structured using keys (primary and foreign).
- **Examples**:
    - **MySQL**
    - **PostgreSQL**
    - **Oracle**
    - **SQL Server**

2. NoSQL Databases

- **Definition**: NoSQL databases are used for storing unstructured or semi-structured data. They are non-relational and often used in applications requiring high scalability and flexibility.
- **Examples**:
    - **MongoDB**
    - **Cassandra**
    - **Redis**
    - **Couchbase**

3. Graph Databases

- **Definition**: These databases use graph structures to store data. They represent data as nodes (entities), edges (relationships), and properties, making them ideal for analyzing relationships between complex data.
- **Examples**:
    - **Neo4j**

- **ArangoDB**
- **Amazon Neptune**

E. **Types of Relationships in Databases**

In relational databases, relationships define how tables are linked to each other

1. One-to-One Relationship

- **Definition**: In a one-to-one relationship, each record in the first table corresponds to exactly one record in the second table, and vice versa.
- **Real-world Example**:

A **person** can have only **one passport**, and each passport is assigned to only **one person**.

2. One-to-Many Relationship

- **Definition**: In a one-to-many relationship, each record in the first table can relate to multiple records in the second table, but each record in the second table relates to only one record in the first table.
- **Real-world Example**:
    - A **teacher** can teach **many students**, but each student has only one teacher.

3. Many-to-Many Relationship

- **Definition**: In a many-to-many relationship, multiple records in the first table can relate to multiple records in the second table. This often requires a **junction table** to manage the relationships.
- **Real-world Example**:
    - A **student** can enroll in **many courses**, and a **course** can have many **students** enrolled.

F. What is an Entity?
An **Entity** is a real-world object or concept that can be distinctly identified and has data stored about it in the database. It is typically represented as a **table** in a relational database. An entity could be something tangible like a **student**, **car**, or **employee**, or something abstract like a **project** or **course**. Each entity represents an object that we want to keep track of, and it often has many attributes describing it.

Attribute:
An **Attribute** is a specific property or characteristic of an entity. It provides more detailed information about that entity. For example, for the entity **Student**, the attributes could be **Name**, **Age**, **ID**, **Course**, etc. Attributes describe the qualities of the entity and give meaning to the entity's data.
Real-Life Example:

- **Entity**: **Student**
    - This represents a person who is enrolled in a school or university.
- **Attributes**: **Name**, **Roll Number**, **Age**, **Course**
    - These attributes describe various characteristics of the student.

Difference Between Primary Key and Unique Key:

1. **Uniqueness**:
    - **Primary Key**: Ensures that each record in the table is unique, and it **cannot have null values**.
    - **Unique Key**: Ensures that the values in a column are unique, but it **can accept null values** (although only one null value is allowed).
2. **Number of Keys**:

- **Primary Key**: A table can have **only one** primary key, and it can consist of one or more columns (composite primary key).
- **Unique Key**: A table can have **multiple unique keys** to enforce uniqueness across different columns.

3. **Indexing**:
   - **Primary Key**: Automatically creates a **unique index** on the column(s) and ensures faster query processing.
   - **Unique Key**: Also creates a **unique index**, but it does not automatically enforce a clustered index (depends on the DBMS).

SQL Example for Primary Key:

```
CREATE TABLE Students (
    Student_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT
);
```

SQL Example for Unique Key:

```
CREATE TABLE Employees (
    Employee_ID INT,
    Email VARCHAR(100) UNIQUE,
    Name VARCHAR(100),
    Age INT
);
```

9. What is a Relationship in a Database Model?

In a **Database Management System (DBMS)**, a **relationship** refers to the **association** between two or more tables based on a common attribute. These relationships help to define how data in one table is related to data in another table. Relationships in databases are used to maintain data integrity and support complex queries involving multiple tables.

A **Foreign Key** is a column (or a set of columns) in a table that is used to establish a link between the data in two tables. It refers to the **Primary Key** in another table, and this reference helps to establish a **relationship** between the two tables.

One-to-Many Relationship Example (Customer and Orders):

```
CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY,
    Order_Date DATE,
    Customer_ID INT,
    FOREIGN KEY (Customer_ID) REFERENCES
Customers(Customer_ID)
);
```

A **Foreign Key** is a **column or a set of columns** in a table that establishes a **link** between data in two tables. It is a key used to enforce a **relationship** between the data in the **parent table** (the table with the **primary key**) and the **child table** (the table with the **foreign key**).

Purpose:

1. **Establish Relationships**: A foreign key is used to create relationships between tables, such as **One-to-Many** or **One-to-One** relationships.

2. **Maintain Referential Integrity**: It ensures that data is consistent across related tables. For example, you can't have an order with a customer ID that doesn't exist in the **Customers** table.

3. **Prevent Orphan Records**: By using foreign keys, you ensure that data in the child table cannot exist without a corresponding record in the parent table.

SQL Example:

```
-- Creating the Customers table with a Primary Key
CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    Name VARCHAR(100)
);

-- Creating the Orders table with a Foreign Key referencing the
Customers table
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY,
    Order_Date DATE,
    Customer_ID INT,
    FOREIGN KEY (Customer_ID) REFERENCES
Customers(Customer_ID)
);
```