**Task 1**

1. **What is normalization?**

Normalization in a database is a design technique used to organize data efficiently by reducing redundancy and improving data integrity. Here's a breakdown of what it involves:

**🧠 What Is Normalization?**

* **Definition**: Normalization is the process of structuring a relational database to minimize data duplication and ensure logical data storage.
* **Goal**: To eliminate anomalies (insertion, update, deletion) and ensure consistency across the database.

**🔄 Why Normalize?**

* **Avoid Redundancy**: Prevents storing the same data in multiple places.
* **Improve Integrity**: Ensures that updates, deletions, and insertions don’t lead to inconsistent data.
* **Optimize Storage**: Reduces unnecessary data, making the database leaner and more efficient.
* **Enhance Scalability**: Makes it easier to adapt the database to changing business needs.

1. **Explain primary vs foreign key.**

The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values . A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields). For example.

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar (255) NOT NULL,  
    FirstName varchar (255),  
    Age int,  
    PRIMARY KEY (ID)  
);

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables. A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/mySQl/mysql_primarykey.asp) in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table. For example .

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

1. **What are constraints?**

**Constraints in SQL** are rules applied to columns or tables in a relational database to ensure the accuracy, validity, and consistency of the data. They help enforce business logic and database integrity by restricting the type of data that can be inserted, updated, or deleted.

Here are the **common types of SQL constraints**:

1. **NOT NULL**: Ensures that a column cannot have a NULL value.
   * Example: A column for "email" must always have a value.
2. **UNIQUE**: Ensures all values in a column are unique, preventing duplicate entries.
   * Example: A column for "username" must have unique values.
3. **PRIMARY KEY**: Combines NOT NULL and UNIQUE. It uniquely identifies each row in a table.
   * Example: A column like "ID" serves as the primary key.
4. **FOREIGN KEY**: Ensures the value in a column matches a value in another table, maintaining referential integrity.
   * Example: A "customer\_id" in an orders table must exist in the customers table.
5. **CHECK**: Ensures that all values in a column satisfy a specific condition.
   * Example: A column for "age" must have values greater than 18.
6. **DEFAULT**: Assigns a default value to a column if no value is provided during insertion.
   * Example: A column for "status" defaults to "active" if no value is specified.
7. **INDEX** (not technically a constraint but often grouped with them): Improves the speed of data retrieval operations on a table.

These constraints can be applied at two levels:

* **Column Level**: Applied directly to a single column.
* **Table Level**: Applied to multiple columns in a table.

1. **what is surrogate key**

A **surrogate key** is a unique identifier for a record in a database table that is **not derived from application data**. It's typically an auto-generated value, like an integer or UUID, used solely to identify rows.

🔑 Key Characteristics of Surrogate Keys

* **Artificial**: Not based on real-world data (e.g., not a Social Security Number or email).
* **System-generated**: Often created using auto-increment fields or sequences.
* **Immutable**: Once assigned, it doesn’t change—even if other data in the row does.
* **Invisible to users**: Usually not shown in user interfaces or reports.

1. **How do you avoid data redundancy?**

Avoiding data redundancy is essential for maintaining a clean, efficient, and reliable database. Here are the most effective strategies to prevent it:

### 🧹 1. ****Normalization****

* Break data into multiple related tables.
* Apply normal forms (1NF, 2NF, 3NF, etc.) to eliminate duplicate and repetitive data.
* Example: Instead of storing customer address in every order record, store it once in a Customers table.

### 🔑 2. ****Use of Primary and Foreign Keys****

* **Primary Key**: Uniquely identifies each record.
* **Foreign Key**: Links related records across tables.
* This ensures that data is referenced rather than duplicated.

### 📋 3. ****Constraints****

* Enforce **UNIQUE** and **NOT NULL** constraints to prevent duplicate or incomplete entries.
* Example: A UNIQUE constraint on email ensures no two users share the same email.

### 🧠 4. ****Centralized Data Storage****

* Store shared data in one place and reference it elsewhere.
* Example: A Products table stores product details, and an Orders table references products by ID.

### 🔄 5. ****Avoid Manual Duplication****

* Use forms and automation to input data consistently.
* Prevent users from entering the same data in multiple places.

### 🛠️ 6. ****Use Triggers and Stored Procedures****

* Automatically check for duplicates before inserting or updating data.
* Helps enforce business rules programmatically.

### 📊 7. ****Regular Audits and Data Cleaning****

* Periodically review and clean the database to remove duplicates.
* Use scripts or tools to identify and merge redundant records.

1. **What is ER diagram?**

An **ER diagram** (Entity-Relationship diagram) is a visual representation of the structure of a database. It shows how data is organized and how different entities (tables) relate to each other.

**🧩 Key Components of an ER Diagram**

| **Component** | **Description** |
| --- | --- |
| **Entity** | A real-world object or concept, represented as a rectangle (e.g., Student, Course). |
| **Attribute** | A property or characteristic of an entity, shown as an oval (e.g., Name, Age). |
| **Relationship** | A connection between entities, represented as a diamond (e.g., Enrolls, Teaches). |
| **Primary Key** | A unique identifier for an entity, often underlined in the diagram. |
| **Cardinality** | Indicates how many instances of one entity relate to another (e.g., one-to- many, many-to-many). |

**📌 Example**

Imagine a university database:

* **Entities**: Student, Course
* **Attributes**: Student has StudentID, Name; Course has CourseID, Title
* **Relationship**: Student *enrolls in* Course

This would be shown as:

Code

[Student] ———<Enrolls>——— [Course]

**🎯 Why Use ER Diagrams?**

* **Design Blueprint**: Helps plan and visualize database structure before implementation.
* **Clarifies Relationships**: Makes it easier to understand how data connects.
* **Improves Communication**: Useful for discussions between developers, analysts, and stakeholders.

1. **What are the types of relationships in DBMS?**

In a **Database Management System (DBMS)**, relationships define how tables (entities) are connected to each other. These relationships are crucial for organizing data efficiently and maintaining integrity across the database.

### 🔗 Types of Relationships in DBMS

#### 1. **One-to-One (1:1)**

* **Definition**: One record in Table A is related to one record in Table B.
* **Example**: Each person has one passport, and each passport belongs to one person.
* **Usage**: Often used to split data for security or performance reasons.

#### 2. **One-to-Many (1:N)**

* **Definition**: One record in Table A can be related to many records in Table B.
* **Example**: One customer can place many orders, but each order belongs to one customer.
* **Most common relationship type** in relational databases.

#### 3. **Many-to-One (N:1)**

* **Definition**: Many records in Table A relate to one record in Table B.
* **Example**: Many employees work in one department.
* **Technically the inverse of One-to-Many**, but often modeled the same way.

#### 4. **Many-to-Many (M:N)**

* **Definition**: Many records in Table A can relate to many records in Table B.
* **Example**: Students enroll in multiple courses, and each course has multiple students.
* **Implementation**: Requires a **junction table** (also called a bridge or associative table) to manage the relationship.

1. **Explain the purpose of AUTO\_INCREMENT.**

The AUTO\_INCREMENT attribute in SQL is used to automatically generate a unique numeric value for a column whenever a new record is inserted into a table. It's most commonly applied to **primary key** columns to ensure each row has a distinct identifier.

### 🔧 Purpose of AUTO\_INCREMENT

* **Automatic ID Generation**: Saves you from manually assigning unique IDs.
* **Ensures Uniqueness**: Prevents duplication of key values.
* **Simplifies Inserts**: You don’t need to specify the value for the column—it’s handled by the database.
* **Supports Sequential Order**: Values typically increase by 1 with each new row, making it easy to track the order of entries.

1. **What is the default storage engine in MySQL?**

The **default storage engine in MySQL** is **InnoDB**.

### ⚙️ Why InnoDB Is the Default

* **ACID Compliance**: Supports transactions with full **Atomicity, Consistency, Isolation, and Durability**.
* **Row-Level Locking**: Improves performance in high-concurrency environments.
* **Foreign Key Support**: Enforces referential integrity between tables.
* **Crash Recovery**: Uses logs to recover data in case of system failure.
* **Better Performance**: Optimized for read/write operations and scalable workloads.

1. **What is a composite key?**

A **composite key** is a type of primary key that consists of **two or more columns** used together to uniquely identify a record in a table. It's used when no single column is sufficient to guarantee uniqueness on its own.

### 🔑 Key Features of Composite Keys

* **Multi-column uniqueness**: The combination of values across multiple columns must be unique.
* **Used in relationships**: Often found in junction tables that manage many-to-many relationships.
* **Acts as a primary key**: Enforces entity integrity just like a single-column primary key.