QC-1-SQL

1. Three Models in Database Design (RDBMS):

- **Conceptual Model:**
  - **Definition:** The conceptual model is a high-level representation of organizational data. It defines what the system contains and the rules governing it, often in the form of ER diagrams (Entity-Relationship diagrams).
  - **Example:**
    - **Real-life:** Think of a blueprint for building a house, which shows rooms, doors, and windows but not how the wiring or plumbing is done.
    - **Technical:** An ER diagram showing entities like `Customer`, `Order`, and `Product`, with their relationships (e.g., a customer places orders).
- **Logical Model:**
  - **Definition:** The logical model takes the conceptual model and translates it into a logical structure that can be implemented in a database. It focuses on the structure, data types, relationships, and constraints without considering physical storage.
  - **Example:**
    - **Real-life:** Designing a room layout with furniture placement but without worrying about where the electrical outlets are.
    - **Technical:** A relational schema with tables such as `Customer`, `Order`, and `Product`, showing primary keys, foreign keys, and data types.
- **Physical Model:**
  - **Definition:** The physical model is the actual implementation of the logical model into a database system. It deals with the specifics of storage, indexing, and retrieval.

- **Example:**
  - **Real-life:** The actual construction of the house, including the placement of plumbing, electrical systems, and materials used.
  - **Technical:** Implementing the database schema in SQL, optimizing tables, choosing indexes, and defining storage parameters.

2. What is Dialect in SQL?

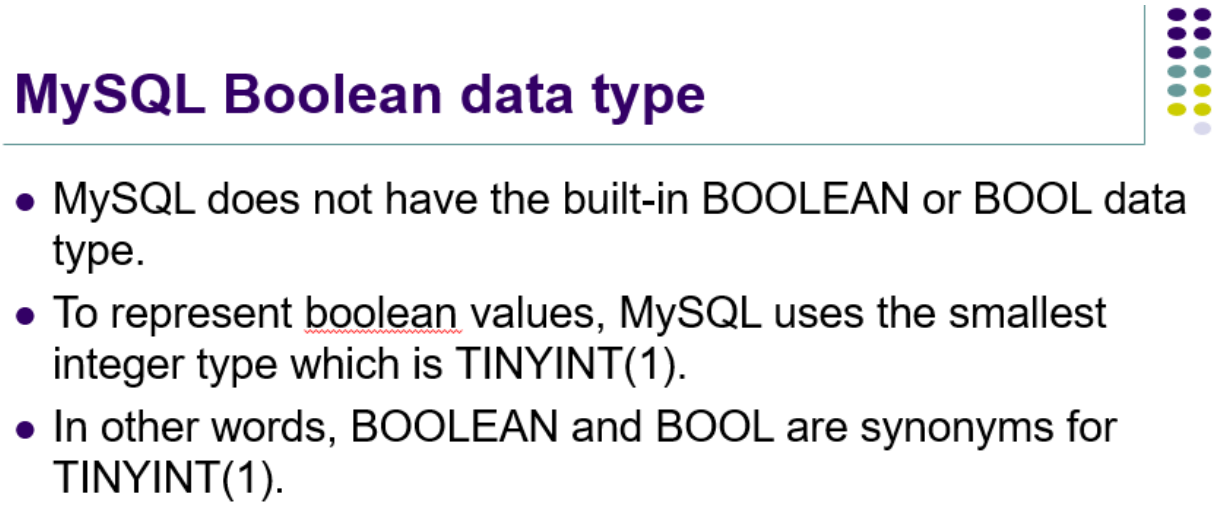
- **Definition:** A dialect in SQL refers to the specific version or variant of SQL used by a particular RDBMS. Different database systems like MySQL, PostgreSQL, Oracle, etc., may have slight variations in syntax and features.
- **Example:**
  - **Real-life:** Different accents or slang in the same language (e.g., American English vs. British English).
  - **Technical:** MySQL uses `LIMIT` for pagination, while Oracle uses `ROWNUM`.

3. Sublanguages in SQL:

- **DDL (Data Definition Language):**
  - **Definition:** Commands that define the structure of the database, such as creating, altering, and dropping tables.
  - **Example:** `CREATE TABLE`, `ALTER TABLE`.
- **DML (Data Manipulation Language):**
  - **Definition:** Commands that manipulate the data within the database, like inserting, updating, and deleting records.
  - **Example:** `INSERT INTO`, `UPDATE`, `DELETE`.
- **TCL (Transaction Control Language):**
  - **Definition:** Commands that manage transactions within the database, ensuring that they are completed successfully.

    - **Example:** `COMMIT`, `ROLLBACK`, `SAVEPOINT`.
- **DQL (Data Query Language):**
    - **Definition:** Commands that query and retrieve data from the database.
    - **Example:** `SELECT`.
- **DCL (Data Control Language):**
    - **Definition:** Commands that control access to the data within the database.
    - **Example:** `GRANT`, `REVOKE`

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)



4.

What is Integrity in RDBMS?

Set of rules followed by table or entity to make sure that data is consistent and no any sort of data inconsistency. Integrity refers to the correctness, consistency, and accuracy of the data stored in a database

Types of Integrity:

- **Entity Integrity:** Ensures that each row in a table is uniquely identified, typically through primary keys.
- **Referential Integrity:** Ensures that foreign keys in a table correctly reference primary keys in another table.
- **Domain Integrity:** Ensures that data values in a column are valid based on their data type, format, or range.

5. What is Data Multiplicity?

- **Definition:** Data multiplicity refers to the relationship between tables and how data in one table relates to data in another table, commonly defined as one-to-one, one-to-many, or many-to-many relationships.
- **Example:**
  - **Real-life:** A teacher can teach multiple classes (one-to-many), but each class can only have one teacher assigned (one-to-one).
  - **Technical:** An `Authors` table related to a `Books` table where an author can write multiple books (one-to-many).

What is Delimiter?

A delimiter is **a special character in MySQL that signals the end of a SQL statement or separates blocks of code**. The default delimiter in MySQL is a semicolon (;), but you can change it to another character using the DELIMITER command.

7. Normalization:

Normalization is the process of organizing data to reduce redundancy and improve data integrity by dividing a database into smaller tables and defining relationships between them.

- **1NF (First Normal Form):**
  - **Definition:** A table is in 1NF if all columns contain atomic values, meaning each column contains only one value per row.
  - **Example:**
    - **Real-life:** A contact list where each contact's phone numbers are stored in separate rows rather than a single field.
    - **Technical:** Splitting a `Contacts` table to ensure each phone number is stored in a different row rather than a comma-separated list.

- 2NF

TEACHER_ID	SUBJECT	TEACHER_AGE

Definition: A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key.

For example, in the "Orders" table, if the combination of OrderID and ProductID determines the value of CustomerID, then CustomerID is fully dependent on the primary key. However, if only OrderID decides the value of CustomerID, then CustomerID has a partial dependency on the primary key.

Student Id Student Name Course Id Course Name

Look student id can get student name while course id can get course name so here is partial dependency this need to be resolved by converting them in new table

3NF (Third Normal Form):

- **Definition:** A table is in 3NF if it is in 2NF and all columns are only dependent on the primary key and not on any other non-primary key attribute.
- **Example:**
  - **Real-life:** An employee database where employee names are associated only with their IDs and not redundantly with other information.
  - **Technical:** Splitting the `Employee` table into separate tables for `EmployeeDetails` (e.g., `EMPID`, `EMPNAME`) and `AddressDetails` (e.g., `PINCODE`, `CITY`).

BCNF

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.

- For BCNF, the table should be in 3NF, and for every FD, LHS is super key

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

STU_ID	COURSE	HOBBY

Fifth normal form (5NF)

• A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

What is SQL? What are some SQL databases?

- **SQL (Structured Query Language):** SQL is a standard programming language used to manage and manipulate relational databases. It allows you to query, update, insert, delete, and manage data.
- **Some SQL Databases:**
  - **MySQL**
  - **PostgreSQL**
  - **Oracle Database**
  - **Microsoft SQL Server**
  - **SQLite**

2. What is ERD?

- **ERD (Entity-Relationship Diagram):** An ERD is a graphical representation of entities and their relationships to each other within a database. It is used during the conceptual design phase to model the data and its relationships visually.
- **Example:** In an ERD, you might have entities like `Customer`, `Order`, and `Product`, with relationships such as a customer can place many orders.

3. What are the 5 Sublanguages of SQL? Which Commands Correspond to Them?

- **DDL (Data Definition Language):**
  - **Commands:** `CREATE`, `ALTER`, `DROP`, `TRUNCATE`
  - **Purpose:** Defines the structure of the database (e.g., tables, schemas).
- **DML (Data Manipulation Language):**
  - **Commands:** `INSERT`, `UPDATE`, `DELETE`
  - **Purpose:** Manipulates data within the database.
- **TCL (Transaction Control Language):**
  - **Commands:** `COMMIT`, `ROLLBACK`, `SAVEPOINT`
  - **Purpose:** Manages transactions in the database.
- **DQL (Data Query Language):**
  - **Commands:** `SELECT`
  - **Purpose:** Retrieves data from the database.
- **DCL (Data Control Language):**
  - **Commands:** `GRANT`, `REVOKE`
  - **Purpose:** Controls access to data within the database.

4. What is the Difference Between DELETE, DROP, and TRUNCATE Commands?

- **DELETE:**
  - **Purpose:** Deletes specific rows from a table based on a condition.
  - **Effect:** Rows can be deleted conditionally. The operation can be rolled back.
- **DROP:**
  - **Purpose:** Completely removes a table, index, or database.
  - **Effect:** Entire structure is removed, and the operation cannot be rolled back.
- **TRUNCATE:**
  - **Purpose:** Removes all rows from a table.
  - **Effect:** All data is deleted, but the table structure remains. The operation cannot be rolled back.

5. What are Some SQL Clauses You Can Use with SELECT Statements?

- **Clauses:**
  - `WHERE`: Filters rows based on a condition.
  - `ORDER BY`: Sorts the result set.
  - `GROUP BY`: Groups rows that have the same values in specified columns.
  - `HAVING`: Filters groups based on a condition.
  - `LIMIT`: Limits the number of rows returned.
  - `JOIN`: Combines rows from two or more tables based on a related column.

6. What is the Difference Between WHERE and HAVING?

- **WHERE:**
  - **Purpose:** Filters rows before grouping is applied.
  - **Use Case:** Used with SELECT, UPDATE, DELETE.
- **HAVING:**
  - **Purpose:** Filters groups after the GROUP BY clause has been applied.
  - **Use Case:** Used with GROUP BY to filter aggregated data.

7. Explain What the ORDER BY and GROUP BY Clauses Do

- **ORDER BY:**
  - **Purpose:** Sorts the result set by one or more columns, either in ascending (`ASC`) or descending (`DESC`) order.
  - **Example:** `SELECT * FROM Employees ORDER BY Salary DESC;`
- **GROUP BY:**
  - **Purpose:** Groups rows that have the same values in specified columns, often used with aggregate functions (e.g., `SUM`, `COUNT`).
  - **Example:** `SELECT Department, COUNT(*) FROM Employees GROUP BY Department;`

8. Explain the Concept of Relational Integrity

- **Relational Integrity:** Ensures the accuracy and consistency of data within a relational database. It prevents data anomalies through rules like foreign key constraints, ensuring that relationships between tables remain valid.
- **Example:** Ensuring that every order in an `Orders` table has a valid customer in the `Customers` table through a foreign key constraint.

9. List the Integrity Constraints

- **Primary Key Constraint:** Ensures that each row in a table is unique.

- **Foreign Key Constraint:** Ensures that a value in one table matches a value in another, maintaining referential integrity.
- **Unique Constraint:** Ensures that all values in a column are unique.
- **Not Null Constraint:** Ensures that a column cannot have NULL values.
- **Check Constraint:** Ensures that all values in a column satisfy a specific condition

10. Define the Word “Schema”

- **Schema:** A schema is a logical container for database objects such as tables, views, indexes, and procedures. It defines the structure of the database.
- **Example:** In a school database, you might have a `SchoolSchema` containing tables like `Students`, `Teachers`, and `Classes`.

Candidate Key:

- **Definition:** A column, or a set of columns, that can uniquely identify a row in a table. A table can have multiple candidate keys.
- **Example:** Both `EmployeeID` and `NationalIDNumber` can be candidate keys in an `Employees` table.

12. What Conditions Lead to Orphan Records?

- **Orphan Records:** These occur when a record in a child table references a record in the parent table that no longer exists. This typically happens when a referenced record is deleted from the parent table without updating or removing the related records in the child table.
- **Example:** Deleting a `Customer` from the `Customers` table while their orders still exist in the `Orders` table.

16. Explain the Different Isolation Levels. What Read Phenomena Do Each Prevent?

- **Isolation Levels:**
  - **Read Uncommitted:** No isolation; dirty reads are possible.
  - **Read Committed:** Prevents dirty reads but allows non-repeatable reads.
  - **Repeatable Read:** Prevents dirty and non-repeatable reads but allows phantom reads.
  - **Serializable:** Prevents dirty, non-repeatable, and phantom reads; highest isolation level.
- **Read Phenomena:**
  - **Dirty Read:** Reading uncommitted data from another transaction.
  - **Non-Repeatable Read:** Data is read twice during a transaction, but the data changes in between reads.
  - **Phantom Read:** New rows are added by another transaction after the initial read, leading to inconsistent results.

17. What is the Difference Between Joins and Set Operators?

- **Joins:**
  - **Purpose:** Combines columns from two or more tables based on related columns.
  - **Example:** `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL OUTER JOIN`.
- **Set Operators:**
  - **Purpose:** Combines result sets from two or more SELECT queries.
  - **Example:** `UNION`, `UNION ALL`, `INTERSECT`, `EXCEPT`.

Isolation in SQL Transactions

Isolation is one of the four key properties of a database transaction, as part of the ACID (Atomicity, Consistency, Isolation, Durability) principles. It determines how the operations within one transaction are isolated from the operations in other concurrent transactions.

- **Purpose of Isolation:**
  - Isolation ensures that transactions are executed in such a way that they do not interfere with each other, maintaining data integrity and consistency.
  - Depending on the isolation level, transactions can either see changes made by other concurrent transactions or be shielded from them until they are complete.

Real-Life Example of Isolation Levels:

Imagine you're at a bookstore:

- **Read Uncommitted:** You look at a book and decide to buy it. The cashier starts processing your purchase, but before the payment is confirmed, someone else (another transaction) comes in and changes the price of the book. You saw the "dirty" or unconfirmed price.
- **Read Committed:** You wait for the cashier to finish processing before checking the price. This way, you only see the final, confirmed price. However, if you check the price again later, someone might have changed it in the meantime.
- **Repeatable Read:** Once you check the price, no one else can change it while you're still deciding. But, if new books arrive (phantom rows), you won't see them unless you start your check again.
- **Serializable:** The cashier closes the section of the store where the book is located so that no one else can buy, move, or change anything until you finish your transaction. You have a fully isolated experience.

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
Read Uncommitted	Allowed	Allowed	Allowed

Read Committed	Prevented	Allowed	Allowed
Repeatable Read	Prevented	Prevented	Allowed
Serializable	Prevented	Prevented	Prevented

1. What is the Difference Between Joins and Set Operators?

- **Joins:**
  - **Purpose:** Joins combine columns from two or more tables based on a related column between them.
  - **Operation:** Rows from the joined tables are combined into a single result based on a specified condition (like matching keys).
  - **Example:**
    - **Technical:** If you have two tables, `Employees` and `Departments`, you can use a join to combine them based on a common column like `DepartmentID`, to get employee details along with their department names.
    - **Real-Life:** Imagine merging two lists, one of students and one of their courses, based on a common student ID to see which students are in which courses.
- **Set Operators:**
  - **Purpose:** Set operators combine the results of two or more `SELECT` statements into a single result set.
  - **Operation:** They work on the entire result sets of queries rather than on individual rows or columns.
  - **Types:** UNION, UNION ALL, INTERSECT, and EXCEPT.
  - **Example:**
    - **Technical:** Use `UNION` to combine two lists of employees from different branches into a single list.

    - **Real-Life:** Merging two guest lists from different events to create one master list.

2. What Are the Types of Joins? Explain the Differences.

- **Inner Join:**
    - **Description:** Returns only the rows that have matching values in both tables.
    - **Example:** Finding all employees who have a matching department in the `Departments` table.
- **Left (Outer) Join:**
    - **Description:** Returns all rows from the left table and the matched rows from the right table. If no match is found, `NULL` is returned for columns from the right table.
    - **Example:** Getting a list of all employees, including those who are not assigned to any department.
- **Right (Outer) Join:**
    - **Description:** Returns all rows from the right table and the matched rows from the left table. If no match is found, `NULL` is returned for columns from the left table.
    - **Example:** Getting a list of all departments, including those with no employees assigned.
- **Full (Outer) Join:**
    - **Description:** Returns all rows when there is a match in either the left or right table. If there is no match, `NULL` is returned for the columns without a match in both tables.
    - **Example:** Listing all employees and all departments, showing relationships where they exist and `NULL` where they do not.
- **Cross Join:**

    - **Description:** Returns the Cartesian product of both tables, meaning all possible combinations of rows.
    - **Example:** Pairing every employee with every department, regardless of whether they work there.
- **Self Join:**
    - **Description:** A join where a table is joined with itself, useful for hierarchical data.
    - **Example:** Finding all employees and their managers if both are stored in the same table.

3. Explain the Difference Between UNION, UNION ALL, and INTERSECT

- **UNION:**
    - **Description:** Combines the results of two `SELECT` statements and removes duplicate rows.
    - **Example:** Merging two lists of customers from different regions, showing each customer only once.
- **UNION ALL:**
    - **Description:** Combines the results of two `SELECT` statements but includes all duplicates.
    - **Example:** Merging two lists of customers from different regions, showing all customers, even if they appear in both lists.
- **INTERSECT:**
    - **Description:** Returns only the rows that are present in both `SELECT` statements.
    - **Example:** Finding customers who are in both the online and in-store customer lists.

4. What is a Cascade Delete?

- **Description:** A database feature where deleting a row in a parent table automatically deletes related rows in a child table.
- **Example:**
  - **Technical:** If you delete a department from the `Departments` table, all employees in that department are automatically deleted from the `Employees` table.
  - **Real-Life:** Deleting a folder on your computer that also deletes all the files within it.

5. What is the Purpose of a View? What About an Index?

- **View:**
  - **Purpose:** A virtual table that is based on the result of a `SELECT` query. It provides a way to simplify complex queries or present data in a specific format without storing it physically.
  - **Example:** Creating a view that shows only active employees, which simplifies queries that need this data.
  - Data is Fetched At Runtime.

Index:

- **Purpose:** A database object that improves the speed of data retrieval operations on a table at the cost of slower write operations.
- **Example:** Indexing a `CustomerName` column so that searches for customers by name are faster.

6. What's the Difference Between a Clustered and Non-Clustered Index?

- **Clustered Index:**

    - **Description:** Physically sorts the data rows in the table according to the index key. There can be only one clustered index per table because the data can be sorted in only one way.
    - **Example:** Indexing a `PrimaryKey` column, such as `CustomerID`, so that the table is physically sorted by `CustomerID`.
- **Non-Clustered Index:**
    - **Description:** Creates a separate structure to hold the index, with pointers to the physical data. Multiple non-clustered indexes can exist on a table.
    - **Example:** Indexing a `LastName` column separately, allowing faster searches by `LastName` without altering the physical order of the table.

In SQL, a trigger is **a database object that automatically executes SQL code when a specific database event occurs**. Triggers are usually associated with a particular table and can be invoked before or after the following events:

- INSERT: A new row is inserted in the table
- UPDATE: An existing row of the table gets updated
- DELETE: A row in the table gets deleted

Triggers can be used for a variety of purposes, such as:

- Logging: Automatically writing to another table when a record is inserted, updated, or deleted from a table
- Data validation: Ensuring data is a certain type and correct values can be set when needed
- Data synchronization: Keeping related tables updated

```
CREATE TRIGGER trigger_name
ON table_name
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
-- SQL code to execute
END;
```

8. How Would You Set Up a Primary Key That Automatically Increments with Every INSERT Statement?

```
CREATE TABLE Employees (
EmployeeID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(100)
);
```

9. What is the Difference Between Scalar and Aggregate Functions? Give Examples of Each.

- **Scalar Functions:**
  - **Description:** Operate on a single value and return a single value.
  - **Examples:**
    - `UPPER()`: Converts a string to uppercase.
    - `LEN()`: Returns the length of a string.
  - **Real-Life:** Taking a single product and checking its price.
- **Aggregate Functions:**
  - **Description:** Operate on a set of values and return a single summary value.
  - **Examples:**
    - `SUM()`: Calculates the total of a numeric column.
    - `COUNT()`: Counts the number of rows.
  - **Real-Life:** Summing up the total sales in a day from all transactions.

10. What are Stored Procedures?

- **Description:** A stored procedure is a group of SQL statements that can be executed as a single unit. Stored procedures are used to encapsulate repetitive tasks, enforce business rules, or manage transactions.
- **Example:**
  - **Technical:** Creating a stored procedure to calculate and apply discounts to all products in a category.
  - **Real-Life:** A recipe (stored procedure) that you follow every time you bake a cake (run a process).

11. Explain the Parameter Types in Stored Procedures (IN, OUT, INOUT)

- **IN Parameter:**
  - **Description:** Passes a value into the procedure. The procedure can read but not modify this value.
  - **Example:** A stored procedure that takes a customer ID as input and retrieves the customer's details.
- **OUT Parameter:**
  - **Description:** Returns a value from the procedure. The procedure writes to this parameter, but it cannot be used as input.
  - **Example:** A stored procedure that calculates and returns a total invoice amount.
- **INOUT Parameter:(price,discount)**
  - **Description:** Passes a value into the procedure and returns a possibly modified value from the procedure.
  - **Example:** A stored procedure that takes an initial discount rate, modifies it based on certain rules, and returns the updated rate.

```
DELIMITER //
CREATE PROCEDURE procedure_name(IN param1 datatype, OUT param2
```

```
datatype)
BEGIN
-- SQL code to execute
-- Example: SELECT SUM(some_column) INTO param2 FROM some_table WHERE some_condition;
END//
DELIMITER ;
```

12. Difference Between Stored Procedure and Stored Function

- **Stored Procedure:**
  - **Purpose:** Performs an action such as modifying data, handling business logic, or managing transactions. It may or may not return a value.
  - **Example:** A procedure that updates inventory levels after a sale.
- **Stored Function:**
  - **Purpose:** Primarily used to calculate and return a value. Functions must return a value and are often used in queries.
  - **Example:** A function that calculates the total price after tax and returns it for use in a query.
- **Key Differences:**
  - **Return Value:** Functions must return a value, whereas stored procedures may return zero or more values.
  - **Usage in Queries:** Functions can be used in SQL queries like any other expression, while procedures cannot.
  - **Side Effects:** Procedures can perform actions like modifying data, while functions should not modify the database state.

```
DELIMITER //
CREATE FUNCTION GetTotalSalesByRegion(regionName VARCHAR(50))
RETURNS DECIMAL(10,2)
BEGIN
```

```
DECLARE totalSales DECIMAL(10,2);
SELECT SUM(SalesAmount) INTO totalSales FROM Sales WHERE Region =
regionName;
RETURN totalSales;
END//
DELIMITER ;
```

Both `BETWEEN` and `IN` are SQL operators used to filter query results based on specified conditions, but they serve different purposes and are used in different contexts.

1. `BETWEEN` Operator

Purpose:

- `BETWEEN` is used to filter results within a specific range. It is commonly used with numeric values, dates, or other ordered types.

```
SELECT column_name
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

2. `IN` Operator

Purpose:

- `IN` is used to filter results based on a list of specific values. It allows you to specify multiple discrete values for a column.

Characteristics:

- **Exact Match:** The `IN` operator checks if the column value matches any of the values in the list.
- **Non-Range:** It does not work with ranges but with a set of distinct values

```
SELECT column_name
FROM table_name
WHERE column_name IN (value1, value2, value3, ...);
```

1. Anomalies

In database management, anomalies refer to inconsistencies or errors that arise due to poor database design, particularly in non-normalized databases. These anomalies can lead to problems with data integrity, redundancy, and efficiency. There are several types of anomalies:

1. Update Anomaly

Definition:
An update anomaly occurs when data is duplicated in multiple places, and a change in one place is not reflected everywhere, leading to inconsistencies.

2. Insertion Anomaly

Definition:
An insertion anomaly occurs when certain data cannot be inserted into the database without the presence of other data. This often happens in a poorly designed schema where some attributes are dependent on other attributes.

Example:
Consider a table where new department information cannot be entered without assigning an employee to that department:

EmployeeID	EmployeeName	DepartmentName	DepartmentLocation
1	John Smith	HR	Building A
2	Jane Doe	IT	Building B

If you want to add a new department, such as "Marketing," you cannot do so without assigning at least one employee to it:

- **Desired Insertion:**
  - Department Name: Marketing
  - Location: Building D

This insertion anomaly occurs because the table structure requires that at least one employee be linked to the department.

3. Deletion Anomaly

Definition:
A deletion anomaly occurs when the deletion of a record causes unintended loss of other valuable information.

Example:
Consider a table that stores employee and department information:

EmployeeID	EmployeeName	DepartmentName	DepartmentLocation
1	John Smith	HR	Building A
2	Jane Doe	IT	Building B
3	Alice Johnson	HR	Building A

If you delete all employees from the HR department, you also lose information about the HR department itself:

- **Action:** Delete all employees from the HR department.
- **Resulting Table:**

EmployeeID	EmployeeName	DepartmentName	DepartmentLocation
2	Jane Doe	IT	Building B

The HR department data is now lost, even though the department information may be valuable independently.

Deterministic Functions:

- Always return the same result given the same input parameters.
- **Example:** `ABS()` (absolute value function), `UPPER()` (convert text to uppercase).

Non-Deterministic Functions:

- May return different results even if given the same input parameters, often because they rely on external factors.
- **Example:** `NOW()` (returns the current date and time), `RAND()` (generates a random number)

Super Key

- **Definition:** A set of one or more attributes that can uniquely identify a record in a table.
- **Example:** In a table with columns `EmployeeID`, `Email`, and `PhoneNumber`, a super key might be `{EmployeeID}`, `{Email}`, or `{EmployeeID, Email}`.
- **Key Difference:** Super key is a broader term that includes all possible keys.

Primary Key

- **Definition:** A specific super key chosen to uniquely identify each record in a table. It must be unique and not null.
- **Example:** `{EmployeeID}` in an `Employees` table.
- **Key Difference:** It is a unique and non-null key selected from among the super keys.

Candidate Key

- **Definition:** Any set of attributes that can uniquely identify a record and can be a potential primary key. There may be multiple candidate keys.
- **Example:** `{EmployeeID}`, `{Email}` in an `Employees` table.
- **Key Difference:** Candidate keys are the possible choices for the primary key.

Alternate Key

- **Definition:** A candidate key that is not chosen as the primary key.
- **Example:** If `{EmployeeID}` is chosen as the primary key, `{Email}` would be an alternate key.
- **Key Difference:** Alternate keys are candidate keys not used as the primary key.

Foreign Key

- **Definition:** An attribute or set of attributes in one table that refers to the primary key of another table, used to establish relationships between tables.
- **Example:** In an `Orders` table, `CustomerID` may be a foreign key referring to `CustomerID` in a `Customers` table.
- **Key Difference:** It establishes a relationship between tables and ensures referential integrity.

Compound Key: A compound key in SQL is **a type of composite key that's created when two or more primary keys from different tables are used as foreign keys within an entity**

Composite Key: A composite key in SQL is **a combination of two or more columns in a database table that uniquely identify each row**

In SQL, **a candidate key uniquely identifies a table's rows, using either a single key or a combination of keys. A composite key is a candidate key that uses two or more attributes (columns) to uniquely identify a row**.

Surrogate Key: An artificial key created to uniquely identify a row in a table. It has no business meaning and is usually an auto-incremented number.

1. Stored Procedure

Stored Procedure is a precompiled set of one or more SQL statements stored on the database server. They are used to perform repetitive tasks, enforce business logic, or ensure data integrity. Stored procedures can be invoked with a single call, and they can accept input parameters, return output parameters, and manage transactions.

Benefits of Stored Procedures:

- **Performance**: As the SQL code is precompiled and stored in the database, it can execute faster.
- **Security**: Users can be given permission to execute the procedure without giving them direct access to the underlying tables.
- **Maintainability**: Business logic can be centralized in the database, making it easier to maintain and update.
- **Reusability**: They can be reused across different applications or parts of an application

2. Sequencing

Sequencing in databases refers to the generation of unique numerical values in a sequential order. Sequences are often used to generate unique identifiers, like primary keys, for rows in a table.

Difference Between Stored Proceudre and function?

Usage in SQL Statements

- **Stored Function**: Can be used directly in SQL expressions, such as in the `SELECT`, `WHERE`, or `JOIN` clauses. For example, a stored function can be called in a query like this:

```
sqlCopy code
SELECT EmployeeID, Salary, dbo.CalculateTax(Salary) AS Tax
Amount
FROM Employees;
```

- **Stored Procedure**: Cannot be used in SQL expressions. It must be invoked with the `EXEC` or `CALL` statement. Stored procedures are typically used to execute a series of operations that might include multiple SQL statements, such as `INSERT`, `UPDATE`, `DELETE`, or complex business logic.

Clustered and Non-Clustered Indexing (TextBook Example)

Indexes in a database are used to speed up the retrieval of data by providing quick access paths to the rows in a table. There are two main types of indexes: **clustered** and **non-clustered**.

1. Clustered Index

A **clustered index** determines the physical order of data in a table. Because of this, there can be only one clustered index per table. The data rows are stored on the disk in the order specified by the clustered index.

Characteristics:

- **Physical Order**: The rows in the table are physically stored in the order of the clustered index. This is why a table can have only one clustered index.
- **Faster Retrieval**: Since the data is stored in the same order as the index, retrieval of ranges of data is faster.

- **Primary Key**: By default, when you create a primary key on a table, a clustered index is automatically created on that column.

Example:

Consider a table `Employees`:

```
sqlCopy code
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(100),
    LastName VARCHAR(100),
    DepartmentID INT
);
```

When the table is created, the `EmployeeID` column is the primary key, and a clustered index is automatically created on this column. This means that the rows in the `Employees` table are physically stored on the disk in the order of `EmployeeID`.

If you query for a range of `EmployeeID` values, the database can quickly locate the rows because they are stored in sorted order.

2. Non-Clustered Index

A **non-clustered index** does not alter the physical order of the table. Instead, it creates a separate structure (a copy of selected columns) that points to the actual data rows in the table. A table can have multiple non-clustered indexes.

Characteristics:

- **Logical Order**: The non-clustered index creates a logical order of the data, but the data itself is stored in the table in an unrelated order.
- **Pointers**: Each entry in a non-clustered index contains a pointer to the corresponding data row in the table. This pointer is typically a reference to the clustered index key or the physical row identifier (RID).
- **Multiple Indexes**: A table can have many non-clustered indexes, allowing for faster searches on different columns.

Example:

Consider the same `Employees` table. Suppose you frequently query the table by `LastName`. You can create a non-clustered index on the `LastName` column:

```
sqlCopy code
CREATE NONCLUSTERED INDEX IDX_LastName
ON Employees (LastName);
```

This creates a non-clustered index on the `LastName` column. The index contains a sorted list of `LastName` values along with pointers to the corresponding rows in the `Employees` table. Even though the rows in the table are stored by `EmployeeID`, the non-clustered index allows for fast lookups by `LastName`.

Order of execution in Mysql?

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT
- DISTINCT
- ORDER BY
- LIMIT

what is data modelling in sql?

Data modeling in SQL refers to the process of designing and organizing the data structure for a database. It involves defining how data is stored, accessed, and related to other data. This process typically includes:

1. **Entity-Relationship (ER) Modeling**:
    - **Entities**: These are objects or concepts, often corresponding to tables in SQL, that have data stored about them (e.g., `Customers`, `Orders`).
    - **Attributes**: These are the properties or characteristics of an entity, corresponding to columns in SQL tables (e.g., `CustomerID`, `OrderDate`).
    - **Relationships**: These define how entities are related to each other, such as a `one-to-many` or `many-to-many` relationship between `Customers` and `Orders`.
2. **Normalization**: This is the process of organizing the data in the database to reduce redundancy and improve data integrity. It typically involves dividing large tables into smaller, related tables and defining relationships between them.
3. **Primary Keys**: These are unique identifiers for records in a table, ensuring that each record can be uniquely identified (e.g., `CustomerID` in a `Customers` table).
4. **Foreign Keys**: These are fields in a table that link to the primary key of another table, establishing a relationship between the two tables (e.g., `CustomerID` in an `Orders` table linking to `CustomerID` in a `Customers` table).
5. **Schema Design**: This involves defining the overall structure of the database, including tables, columns, data types, constraints, and indexes.

Importance of Data Modeling

- **Efficiency**: Proper data modeling ensures that the database is efficient, minimizing data duplication and optimizing query performance.
- **Data Integrity**: It ensures that the data remains accurate and consistent across the database.
- **Scalability**: A well-designed data model can be more easily scaled as the amount of data grows.
- **Clarity**: It helps in understanding the data, making it easier for developers, analysts, and stakeholders to work with the database.

The maximum length of a `VARCHAR` field in MySQL in terms of characters depends on the character set you are using:

1. **Single-byte character sets (e.g., `latin1`)**:
    - You can store up to **65,533 characters** in a `VARCHAR` field, since each character uses 1 byte and there's a 2-byte overhead.
2. **Multi-byte character sets (e.g., `utf8mb4`)**:
    - You can store up to **16,383 characters** if each character uses the maximum 4 bytes (`16,383 characters * 4 bytes = 65,532 bytes`).

In terms of **words**, assuming an average word length of 5 characters:

- **Single-byte (e.g., `latin1`)**: Approximately 13,106 words.
- **Multi-byte (e.g., `utf8mb4`)**: Approximately 3,276 words.

Difference between structured and unstructured data?

The main difference between structured and unstructured data lies in how they are organized and managed:

1. **Structured Data:**
    - **Organization:** Highly organized and easily searchable in a predefined format. It typically resides in relational databases or spreadsheets.
    - **Format:** Data is arranged in rows and columns, making it easy to query and analyze.
    - **Examples:** SQL databases, spreadsheets (e.g., Excel), customer information databases, inventory records.
2. **Unstructured Data:**
    - **Organization:** Lacks a predefined structure or format, making it harder to search and analyze directly.
    - **Format:** Can come in various forms such as text, images, videos, audio, social media posts, or emails.
    - **Examples:** Emails, social media content, video files, web pages, text documents.

Key Points:

- **Structured Data:** Easier to process with traditional data management tools and query languages like SQL.
- **Unstructured Data:** Requires advanced techniques for analysis, such as natural language processing (NLP) or machine learning.

What is Cascading in sql?

In SQL, "cascading" refers to the automatic propagation of changes or actions through related data in a database. It is commonly associated with foreign key constraints and can influence how updates and deletions are handled between parent and child tables.

Types of Cascading

1. **Cascading Updates:**
    - When a primary key value in a parent table is updated, cascading updates ensure that corresponding foreign key values in child tables are automatically updated to reflect this change.
    - **Example:** If you have a `Customers` table with a primary key `customer_id` and an `Orders` table with a foreign key `customer_id`, a cascading update will automatically update the `customer_id` in the `Orders` table if it is changed in the `Customers` table.
2. **Cascading Deletes:**
    - When a row in a parent table is deleted, cascading deletes ensure that related rows in child tables are automatically removed.
    - **Example:** If you delete a record from the `Customers` table, a cascading delete will automatically remove related records in the `Orders` table that reference the deleted customer.

CREATE TABLE Orders (
order_id INT PRIMARY KEY,
customer_id INT,

```
FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
ON UPDATE CASCADE
ON DELETE CASCADE
);
```

Transaction Properties -ACID Properties

What is Aggregate Functions?List out some

Aggregate functions in SQL perform a calculation on a set of values and return a single value. They are often used in conjunction with the `GROUP BY` clause to summarize data.

Common Aggregate Functions

1. `COUNT()`
   - **Purpose:** Returns the number of rows or non-null values in a column.
   - **Example:** `SELECT COUNT(*) FROM Employees;`
2. `SUM()`
   - **Purpose:** Returns the sum of values in a numeric column.
   - **Example:** `SELECT SUM(salary) FROM Employees;`
3. `AVG()`
   - **Purpose:** Returns the average value of a numeric column.
   - **Example:** `SELECT AVG(salary) FROM Employees;`
4. `MIN()`
   - **Purpose:** Returns the minimum value in a column.
   - **Example:** `SELECT MIN(salary) FROM Employees;`
5. `MAX()`

- **Purpose:** Returns the maximum value in a column.
- **Example:** `SELECT MAX(salary) FROM Employees;`

6. `GROUP_CONCAT()` (MySQL specific)
   - **Purpose:** Concatenates values from multiple rows into a single string.
   - **Example:** `SELECT GROUP_CONCAT(employee_name) FROM Employees;`
7. `COUNT(DISTINCT)`
   - **Purpose:** Returns the number of distinct values in a column.
   - **Example:** `SELECT COUNT(DISTINCT department_id) FROM Employees;`

Explain select statement?

The
`SELECT` statement is one of the most fundamental and commonly used commands in SQL. It retrieves data from one or more tables and displays it as a result se

What is Theta Join?

A Theta Join in SQL is a type of join that allows you to combine rows from two tables based on a condition that is more general than equality. Unlike the more common Inner Join, which specifically uses equality (
`=`) for joining tables, a Theta Join can use any comparison operator (e.g., `<`, `>`, `<=`, `>=`, `<>`).

SELECT Employees.employee_name, Departments.department_name
FROM Employees
JOIN Departments
ON Employees.employee_id < Departments.department_id;

Comparison with Other Joins

- **Inner Join:** Specifically uses `=` to match rows between tables. Example: `ON table1.column = table2.column`
- **Outer Join:** Includes rows that do not have matching rows in the other table. Can be combined with any join condition.
- **Cross Join:** Combines every row from the first table with every row from the second table, creating a Cartesian product. It does not use a join condition.

Differences between primary key and unique key now

ChatGPT

Certainly! Here are the key differences between a **Primary Key** and a **Unique Key** in the context of relational databases like MySQL:

1. Purpose:

- **Primary Key**:
  - Serves as the unique identifier for each record in a table.
  - It ensures that each record can be uniquely identified.
- **Unique Key**:
  - Ensures that all the values in a column or a group of columns are unique across the table.
  - It is used to enforce uniqueness in columns other than the primary key.

2. Uniqueness:

- **Primary Key**:
  - A primary key enforces uniqueness and automatically creates a unique index.
  - Only one primary key is allowed per table.
- **Unique Key**:

- A unique key also enforces uniqueness but does not serve as the main identifier.
- Multiple unique keys can be defined in a table.

3. NULL Values:

- **Primary Key**:
  - Does not allow `NULL` values. Every field defined as part of the primary key must contain a non-`NULL` value.
- **Unique Key**:
  - Allows `NULL` values. In MySQL, multiple `NULL` values are allowed in a column with a unique constraint.

4. Indexing:

- **Primary Key**:
  - Automatically creates a clustered index on the column(s) that constitute the primary key. If a clustered index already exists, a non-clustered index is created instead.
- **Unique Key**:
  - Automatically creates a non-clustered index to enforce uniqueness.

5. Usage:

- **Primary Key**:
  - Primarily used to uniquely identify a record and establish relationships between tables in the form of foreign keys.
- **Unique Key**:
  - Used when you need to ensure uniqueness in columns other than the primary key but without setting them as the main identifier of the table.

6. Number of Keys per Table:

- **Primary Key**:

- Only one primary key is allowed per table.

- **Unique Key**:
  - A table can have multiple unique keys.

7. Default Indexing Type:

- **Primary Key**:
  - Usually results in a clustered index by default, depending on the database system.
- **Unique Key**:
  - Results in a non-clustered index.

Difference Between Unique And Distinct?

Unique and **Distinct** are terms used in SQL, but they serve different purposes and are applied in different contexts. Here are the key differences between them:

1. Purpose:

- **Unique**:
  - Refers to a constraint or condition applied to a column (or a combination of columns) in a table to ensure that all values in that column(s) are distinct and not repeated.
  - Ensures data integrity at the database schema level by preventing duplicate entries.
- **Distinct**:
  - Is a keyword used in a `SELECT` statement to retrieve only unique (non-duplicate) rows from the result set.
  - It is used in query operations to filter out duplicates from the output.

2. Usage Context:

- **Unique**:

- Used when defining table schemas, particularly when creating or altering tables.
- Example:

```
sqlCopy code
CREATE TABLE Employees (
    EmployeeID INT NOT NULL UNIQUE,
    Email VARCHAR(255) UNIQUE
);
```

- **Distinct**:
  - Used in `SELECT` queries to return only distinct values from the queried columns.
  - Example:

```
sqlCopy code
SELECT DISTINCT Country FROM Customers;
```

3. Application:

- **Unique**:
  - Applied as a constraint on a column or set of columns to ensure no duplicate values are inserted.
  - Affects data entry and database integrity.
- **Distinct**:
  - Applied during data retrieval in a query to filter the result set.
  - Does not affect how data is stored, only how it is displayed or processed in the query results.

4. Effect on Data:

- **Unique**:
  - Enforces a rule that affects how data can be stored in the table. It prevents the insertion of duplicate values in the constrained column(s).
- **Distinct**:
  - Has no effect on how data is stored in the database. It only removes duplicates from the result set of a query.

5. Scope:

- **Unique**:
  - Operates at the database schema level and affects the entire table where the constraint is applied.
- **Distinct**:
  - Operates at the query level and affects only the output of the specific `SELECT` statement.

6. Indexing:

- **Unique**:
  - Often creates an index to enforce the uniqueness constraint, which can improve query performance when searching for unique values.
- **Distinct**:
  - Does not create an index; it simply filters the query result at runtime