SQL THEORATICAL QUESTIONS

What is SQL?

SQL (Structured Query Language) is a programming language used to manage and interact with databases. It helps you:

- **Store** data
- **Retrieve** data (using queries)
- **Update** data
- **Delete** data
- **Create** and **modify** database structures (like tables)

It's used with databases like MySQL, SQL Server, PostgreSQL, and more.

What are the subsets of SQL or types of SQL commands and briefly explain?

SQL commands are categorized into **five main subsets**:

1. **DDL (Data Definition Language):**
    - **Purpose:** Defines and manages database structures like tables, schemas, etc.
    - **Commands:**
        - **CREATE** – Creates new tables or databases
        - **ALTER** – Modifies existing tables
        - **DROP** – Deletes tables or databases
        - **TRUNCATE** – Removes all data from a table without deleting the structure
2. **DML (Data Manipulation Language):**
    - **Purpose:** Manages data within tables.
    - **Commands:**
        - **SELECT** – Retrieves data from tables
        - **INSERT** – Adds new data
        - **UPDATE** – Modifies existing data
        - **DELETE** – Removes data
3. **DCL (Data Control Language):**
    - **Purpose:** Controls access to data in the database.
    - **Commands:**
        - **GRANT** – Gives user permissions
        - **REVOKE** – Removes user permissions

4. **TCL (Transaction Control Language):**
    - **Purpose:** Manages transactions to ensure data integrity.
    - **Commands:**
        - **COMMIT** – Saves changes made by a transaction
        - **ROLLBACK** – Undoes changes if there's an error
        - SAVEPOINT – Sets a point to roll back to if needed
5. **DQL (Data Query Language):**
    - **Purpose:** Focuses only on querying data.
    - **Command:**
        - **SELECT** – Retrieves data from the database

In simple terms:

- **DDL** defines the structure,
- **DML** works with the data,
- **DCL** controls access,
- **TCL** manages transactions, and
- **DQL** retrieves data.

What is the sequence of execution in SQL?

The **sequence of execution in SQL** (also known as the **SQL query execution order**) follows a specific logical order, even if we write queries differently.

Order of Execution:

1. **FROM** – Identifies the tables to retrieve data from.
2. **JOIN** – Combines data from multiple tables (if applicable).
3. **WHERE** – Filters rows based on conditions.
4. **GROUP BY** – Groups rows that have the same values in specified columns.
5. **HAVING** – Filters groups created by GROUP BY.
6. **SELECT** – Chooses the specific columns to display.
7. **DISTINCT** – Removes duplicate records (if used).
8. **ORDER BY** – Sorts the final result.
9. **LIMIT/OFFSET** – Limits the number of records returned (optional).

Advantages & disadvantage of SQL

Advantages of SQL:

1. **Easy to Learn and Use:**
    - SQL uses simple English-like syntax, making it beginner-friendly.
2. **Efficient Data Handling:**
    - Quickly retrieves, updates, and manipulates large amounts of data.
3. **Standardized Language:**
    - SQL is a universal standard supported by databases like MySQL, SQL Server, PostgreSQL, Oracle, etc.
4. **Powerful Query Capabilities:**
    - Supports complex queries with functions like JOIN, GROUP BY, SUBQUERIES, etc.
5. **Data Security:**
    - Provides commands (GRANT, REVOKE) to control user access and permissions.
6. **Scalable:**
    - Handles databases of all sizes, from small to enterprise-level systems.
7. **Integration:**
    - Easily integrates with programming languages like Python, Java, C#, etc.

❌ **Disadvantages of SQL:**

1. **Complexity with Advanced Queries:**
    - Writing highly complex queries (nested subqueries, multiple joins) can be challenging for beginners.
2. **Limited Control Over Database Logic:**
    - SQL focuses on data operations; it's not suitable for complex business logic like traditional programming languages.
3. **Vendor Dependency:**
    - Some SQL features are specific to certain databases (e.g., T-SQL for SQL Server, PL/SQL for Oracle), which affects portability.
4. **Performance Issues with Large Data:**
    - Poorly optimized queries can slow down performance, especially with very large datasets.
5. **Security Risks if Not Handled Properly:**

- Vulnerable to SQL injection attacks if queries are not secured in applications.

Summary:

SQL is powerful, easy to learn, and widely used, but handling complex queries and ensuring security can be challenging if not done correctly.

What is Database? And how to create a database in SQL?

A **database** is an organized collection of data that allows you to **store**, **manage**, and **retrieve** information efficiently. It helps in handling large volumes of data, ensuring easy access, security, and quick updates.

- **Examples:**
  - A **bank database** stores customer accounts, transactions, and balances.
  - An **e-commerce database** manages products, orders, and customer information.

Databases can be:

- **Relational (SQL-based)** – Data stored in tables (e.g., MySQL, PostgreSQL).
- **Non-Relational (NoSQL)** – Data stored in documents or key-value pairs (e.g., MongoDB).

Syntax: ***CREATE DATABASE database_name;***

What is DBMS?

A **DBMS (Database Management System)** is software that enables users to create, manage, and interact with databases. It provides a systematic way of storing, retrieving, and manipulating data. The DBMS ensures that data is stored securely, efficiently, and can be accessed by multiple users while maintaining data integrity and consistency.

Some key features of a DBMS include:

1. **Data Storage**: It allows for the efficient storage of data and provides mechanisms for fast data retrieval.
2. **Data Manipulation**: It provides operations such as inserting, updating, deleting, and querying data.
3. **Data Security**: A DBMS helps secure data through authentication, authorization, and encryption.
4. **Concurrency Control**: It manages concurrent access to data, ensuring that multiple users can interact with the database without conflicting.
5. **Data Integrity**: It enforces constraints to maintain data consistency and accuracy.
6. **Backup and Recovery**: A DBMS provides features for backing up data and recovering it in case of system failures.

Examples of DBMS include MySQL, Oracle, Microsoft SQL Server, and PostgreSQL.

What are Tables and Fields?

In a **database**, **tables** and **fields** are fundamental components used to store and organize data.

Tables:

- A **table** is a collection of data organized into rows and columns.
- Each table represents a specific entity or object (e.g., **Customers**, **Orders**, **Products**) in the database.
- A table is made up of multiple **fields** (columns) and **records** (rows).
- Tables are structured in a way that each row contains a set of related data, and each column stores a specific type of information about the entity.

Fields:

- A **field** (also called a **column**) represents a single type of data within a table. Each field has a name and a specific data type (e.g., text, number, date).
- Fields define the structure of the data stored in a table. Every record in a table has a value for each field.
- The fields hold the attributes or characteristics of the entity represented by the table.

Example: In the **Customers** table:

- **Customer ID** is a field that stores unique identifiers for customers (e.g., 1, 2).
- **Name** is a field that stores the customer's name (e.g., Alice, Bob).
- **Email** is a field that stores the customer's email address (e.g., alice@email.com, bob@email.com).

Each field contains a specific type of data that helps describe the entity represented by the table.

What are Constraints in SQL?

In SQL, **constraints** are rules that are applied to columns or tables to ensure data integrity and consistency. They are used to define the conditions under which data can be inserted, updated, or deleted. Constraints help maintain the correctness and reliability of the data in a database.

NOT NULL: Prevents NULL values in a column.
PRIMARY KEY: Uniquely identifies records, no duplicate or NULL values allowed.
FOREIGN KEY: Ensures referential integrity between tables.
UNIQUE: Ensures all values in a column are distinct.
CHECK: Enforces a condition on column values.
DEFAULT: Provides default values when none are supplied.
INDEX: Improves query performance by indexing frequently searched columns.

What is a primary key and foreign key?

Primary Key:

A **Primary Key** is a column (or a combination of columns) in a table that uniquely identifies each row in the table. It must contain **unique values** and cannot contain **NULL** values.

Key characteristics of a Primary Key:

- Each value in the primary key column must be unique.
- It cannot have a NULL value.
- There can only be one primary key in a table.
- It ensures that each record in the table is identifiable.

Foreign Key:

A **Foreign Key** is a column (or a combination of columns) in one table that refers to the **Primary Key** in another table. The foreign key establishes a **relationship** between the two tables, ensuring referential integrity.

Key characteristics of a Foreign Key:

- It links two tables by referencing the **primary key** of another table.
- It can contain duplicate values and NULL values.
- It ensures that a value in the foreign key column matches a value in the referenced primary key column, or is NULL (if allowed).
- A table can have multiple foreign keys.

Summary of Primary Key vs Foreign Key:

- **Primary Key**: Uniquely identifies each record in the same table and ensures data integrity within the table.
- **Foreign Key**: Links a column in one table to the primary key of another table, ensuring relationships between tables and maintaining referential integrity.

How to create and delete a table in SQL?

- **Creating a Table in SQL**:

To **create** a **table** in **SQL**, you use the **CREATE TABLE** statement, followed by the table name and the definition of each column, including the column name and data type. You can also include any **constraints** like **PRIMARY KEY**, **NOT NULL**, **FOREIGN KEY**, etc.

Syntax:

```
CREATE TABLE table_name
        (column1_name column1_data_type [constraint],
        column2_name column2_data_type [constraint]);
```

- **Deleting a Table in SQL:**

Syntax:

DROP TABLE *table_name;*

How to change a table name in SQL?

To change the name of an existing table in SQL, you use the **ALTER TABLE** statement with the **RENAME** TO clause.

Syntax:

ALTER TABLE *old_table_name* RENAME TO *new_table_name*;

What is join in SQL? List its different types.

In SQL, a **JOIN** is used to combine rows from two or more tables based on a related column.

Types of Joins:

1. **INNER JOIN**:
    - Returns only the rows where there is a match in both tables.
2. **LEFT JOIN (or LEFT OUTER JOIN)**:
    - Returns all rows from the left table and the matched rows from the right table. If no match is found, NULL values are returned for columns from the right table.
3. **RIGHT JOIN (or RIGHT OUTER JOIN)**:
    - Returns all rows from the right table and the matched rows from the left table. If no match is found, NULL values are returned for columns from the left table.
4. **FULL JOIN (or FULL OUTER JOIN)**:
    - Returns all rows when there is a match in either the left or right table. If there is no match, NULL values are returned for missing matches in either table.
5. **CROSS JOIN**:
    - Returns the Cartesian product of both tables, combining each row from the first table with every row from the second table.
6. **SELF JOIN**:
    - A join where a table is joined with itself. It is used to compare rows within the same table.

What is Normalization in SQL?

Normalization in SQL is the process of organizing the data in a database to reduce redundancy and improve data integrity. The goal is to ensure that the database structure is efficient and that it supports data consistency.

Key Points of Normalization:

- **Reduces Data Redundancy**: By organizing data into smaller, related tables, normalization minimizes duplicate data.
- **Improves Data Integrity**: It ensures that the data is consistent and adheres to rules for relationships between tables.

Normal Forms:

Normalization is typically done in stages, called **Normal Forms (NF)**. Each subsequent normal form builds on the previous one.

1. **First Normal Form (1NF)**:
    - Ensures that each column contains atomic (indivisible) values and that there are no repeating groups or arrays in a column.
2. **Second Normal Form (2NF)**:
    - Achieved when a table is in 1NF and all non-key columns are fully functionally dependent on the primary key (i.e., eliminates partial dependency).
3. **Third Normal Form (3NF)**:

    - Achieved when a table is in 2NF and all columns are non-transitively dependent on the primary key (i.e., no transitive dependency).
4. **Boyce-Codd Normal Form (BCNF)**:
    - A stricter version of 3NF where every determinant is a candidate key.
5. **Fourth Normal Form (4NF)**:
    - Achieved when a table is in BCNF and has no multi-valued dependencies (i.e., no column has multiple independent sets of values).
6. **Fifth Normal Form (5NF)**:
    - Achieved when a table is in 4NF and has no join dependencies, meaning it cannot be decomposed further without loss of information.

Benefits of Normalization:

- **Efficient Data Storage**: Reduces the storage space by eliminating redundancy.
- **Improved Query Performance**: By organizing the data into smaller, logical tables, it can make querying more efficient.
- **Consistency**: Ensures that data is consistent and up-to-date across the database.

When to Stop Normalizing:

While normalization helps with consistency and reduces redundancy, sometimes too much normalization can lead to complex queries with multiple joins. In certain cases, some denormalization (reducing the level of normalization) may be used to optimize performance.

How to insert a date in SQL?

Syntax:

```
INSERT INTO table_name (column_name)
                VALUES ('YYYY-MM-DD');
```

What are the TRUNCATE, DELETE and DROP statements?

The **TRUNCATE**, **DELETE**, and **DROP** statements in SQL are used to remove data, but they differ in their functionality and use cases.

1. TRUNCATE

- **Purpose**: Removes all rows from a table, but does not remove the table itself. It is faster than DELETE because it doesn't log individual row deletions.
- **Effect**: Resets any auto-increment values and cannot be rolled back in many databases (depending on transaction settings).
- **Use Case**: When you want to delete all data from a table but keep the structure for future use.

Syntax:

```
TRUNCATE TABLE table_name;
```

---

2. DELETE

- **Purpose**: Removes rows from a table based on a condition. Unlike TRUNCATE, it can delete specific rows and is logged for each row.
- **Effect**: Data can be deleted selectively, and the operation can be rolled back if wrapped in a transaction.
- **Use Case**: When you want to delete specific rows based on a condition or when you need to preserve the table structure with constraints.

Syntax:

```
DELETE FROM table_name WHERE condition;
```

---

3. DROP

- **Purpose**: Completely removes a table, including its structure, data, and associated indexes, constraints, and triggers.
- **Effect**: The table is permanently deleted from the database, and it cannot be rolled back unless the database is backed up.
- **Use Case**: When you want to completely remove a table and all its data from the database.

Syntax:

```
DROP TABLE table_name;
```

Key Differences:

- **TRUNCATE**: Removes all rows quickly and resets auto-increment values.
- **DELETE**: Removes specific rows, can be rolled back, and preserves the table structure.
- **DROP**: Completely deletes the table structure and data permanently.

What are the different types of SQL operators?

Arithmetic: +, -, *, /, %

Comparison: =, !=, >, <, >=, <=

Logical: AND, OR, NOT

Set: IN, BETWEEN

Pattern Matching: LIKE

Null: IS NULL, IS NOT NULL

Existence: EXISTS

Set Operations: UNION, UNION ALL

String: Concatenation (|| or +)

Aggregates: COUNT, SUM, AVG, MIN, MAX

What are Aggregate and Scalar functions?

Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single result. They are typically used with the GROUP BY clause to group rows based on a certain column.

Common Aggregate Functions:

- **COUNT()**: Returns the number of rows in a specified column or table.
- **SUM()**: Returns the sum of a numeric column.
- **AVG()**: Returns the average value of a numeric column.
- **MIN()**: Returns the smallest value in a column.
- **MAX()**: Returns the largest value in a column.
- **GROUP_CONCAT()** (MySQL) or **STRING_AGG()** (PostgreSQL, SQL Server): Returns a concatenated string of values from a group.

Scalar Functions

Scalar functions perform operations on individual values and return a single result for each input value. They are applied to single values or columns in the query.

Common Scalar Functions:

- **LEN() or LENGTH()**: Returns the length of a string.
- **UPPER()**: Converts a string to uppercase.
- **LOWER()**: Converts a string to lowercase.
- **ROUND()**: Rounds a numeric value to a specified number of decimal places.
- **NOW() or GETDATE()**: Returns the current date and time.
- **COALESCE()**: Returns the first non-null value in a list of expressions.
- **CONCAT()**: Concatenates two or more strings.

- **ABS()**: Returns the absolute value of a number.
- **DATEPART()**: Extracts a specific part (like year, month, day) from a date.

Key Differences:

- **Aggregate Functions**: Operate on a set of rows, returning a single result (e.g., COUNT, SUM, AVG).
- **Scalar Functions**: Operate on a single value and return a single result for each row (e.g., UPPER, LEN, ROUND).

What does a window function do in SQL?

A **window function** in SQL performs calculations across a set of rows related to the current row within a query's result set, without collapsing the rows into a single output. This allows you to calculate aggregates, rankings, or other values over a specific window of data while still keeping the individual rows in the result set.

Key Characteristics:

- **Retains Row Detail**: Unlike aggregate functions, which group rows into a single result, window functions return a value for each row.
- **Operates Over a Window**: The "window" is a set of rows defined by a specified range or partition in the result set, based on an ordered or grouped context.
- **Does Not Alter Row Count**: The number of rows in the result set remains the same, but additional calculated columns are added based on the window.

Common Window Functions:

1. **ROW_NUMBER()**: Assigns a unique sequential number to rows within a partition of the result set, starting at 1.
2. **RANK()**: Assigns a rank to each row within a partition of the result set, with gaps between ranks for ties.
3. **DENSE_RANK()**: Similar to RANK(), but does not leave gaps in the ranking when there are ties.
4. **NTILE(n)**: Divides the result set into n buckets and assigns a number to each row indicating the bucket it belongs to.
5. **LEAD()**: Provides access to the value of the next row in the result set (relative to the current row).
6. **LAG()**: Provides access to the value of the previous row in the result set.
7. **SUM(), AVG(), MIN(), MAX()**: These aggregate functions can also be used as window functions to perform calculations over a range of rows.

Window Function Use Cases:

- **Running totals**: Calculate cumulative sums or averages.
- **Ranking**: Rank rows based on values like sales, scores, etc.
- **Comparisons**: Compare a row’s value with the next or previous row (e.g., calculating the difference between consecutive rows).

Difference between rank, dense_rank and row_number in sql.

Key Differences:

Feature	`ROW_NUMBER()`	`RANK()`	`DENSE_RANK()`
Uniqueness of Rank	Unique for each row	Same rank for ties	Same rank for ties, but no gaps
Handling of Ties	No ties allowed (unique number)	Skips ranks after ties	No gap between ranks
Example on Ties	1, 2, 3, 4	1, 1, 3, 4	1, 1, 2, 3

When to Use Each:

- `ROW_NUMBER()` : When you need a unique identifier for each row, regardless of ties.
- `RANK()` : When you want to assign ranks but leave gaps between ranks in case of ties.
- `DENSE_RANK()` : When you want to assign ranks without gaps, even in case of ties.

What are clustered and non-clustered index in SQL?

In SQL, **indexes** are used to speed up the retrieval of rows from a table. There are two main types of indexes: **clustered index** and **non-clustered index**. They both help improve query performance, but they function differently.

1. Clustered Index:

- **Definition**: A clustered index determines the physical order of data rows in a table. The table's data is actually stored in the order of the clustered index. There can only be one clustered index per table, as the data rows themselves can only be sorted one way.
- **Structure**: The data rows of the table are stored in the index itself. So, the clustered index is essentially the table's data organized in a specific order.
- **Performance**: Since the data is sorted according to the index, retrieving data using the clustered index is very fast for range queries (e.g., between two values) and exact matches.

Key Points:

- Only one clustered index is allowed per table.
- The table is physically sorted by the clustered index.
- Used by default for primary keys (if no other index is specified).

Non-Clustered Index:

- **Definition**: A non-clustered index is a separate structure from the data table. It contains pointers to the data rows, and the data itself is not stored in the same order as the index. There can be multiple non-clustered indexes on a table.
- **Structure**: A non-clustered index contains a sorted list of the indexed column(s) and a reference (pointer) to the corresponding data rows in the table. The actual table data is stored independently of the index.
- **Performance**: Non-clustered indexes are useful for lookups and specific column searches but can be slower for range queries because the data is not stored in the index order.

Key Points:

- Multiple non-clustered indexes can exist on a table.
- Does not affect the physical storage of data.
- Commonly used for foreign keys, frequently queried columns, or non-unique columns.

Key Differences Between Clustered and Non-Clustered Indexes:

Feature	Clustered Index	Non-Clustered Index
Storage of Data	Data is stored in the index itself (physical order)	Data is stored separately from the index (logical order)
Number of Indexes per Table	Only one clustered index per table	Multiple non-clustered indexes per table
Impact on Data Organization	Data rows are physically ordered by the clustered index	Data rows are not physically reordered
Performance	Fast for range queries and primary key lookups	Fast for lookups but can be slower for range queries
Use Case	Best for primary key or unique identifiers	Best for frequently queried columns or non-primary keys

When to Use Each:

- **Clustered Index**: When you want the data to be stored in a specific order, such as for primary keys or frequently used range queries (e.g., between two dates).
- **Non-Clustered Index**: When you need to speed up searches for specific columns that are not used as primary keys, especially when the↓re multiple search criteria or non-unique values.