## MySQL Interview Questions

2.what is the difference between where and having?

1. **WHERE clause**:
    - The **WHERE** clause is used to filter rows before any aggregation is performed.
    - It is used with the **SELECT**, **UPDATE**, **DELETE**, and **MERGE** statements to specify conditions that filter rows to be included in the result set.
    - It is applied to individual rows of the queried tables.
    - It typically contains conditions that compare column values to constants, other column values, or expressions.

Example:

SELECT column1, column2 FROM table_name WHERE condition;

2. **HAVING clause**:
    - The **HAVING** clause is used to filter rows after the aggregation has been performed, specifically with grouped data.
    - It is used in conjunction with the **GROUP BY** clause to filter the rows resulting from the grouping operation.
    - It is applied to groups of rows defined by the **GROUP BY** clause.
    - It typically contains conditions that apply to aggregated values, such as the result of **COUNT**, **SUM**, **AVG**, etc.

Example:

SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING COUNT(*) > 5;

3.what is the use of group by clause?

The **GROUP BY** clause in SQL is used to group rows that have the same values into summary rows, typically to perform aggregate functions (like **COUNT**, **SUM**, **AVG**, **MAX**, **MIN**) on each group. It's commonly used in conjunction with aggregate functions to generate summary reports from a dataset.

Uses : Aggregation, Data Summarization, Eliminating duplicate data, Filtering data.

4.Explain different types of joins in mysql?

**INNER JOIN**:

- Returns only the rows that have matching values in both tables based on the specified join condition.
- Rows from the tables that do not meet the join condition are excluded from the result set.
- It is the most commonly used type of join.

SELECT *FROM table1

INNER JOIN table2 ON table1.column = table2.column;

**LEFT JOIN (or LEFT OUTER JOIN)**:

- Returns all rows from the left table (the first table mentioned in the query) and matching rows from the right table (the second table mentioned in the query).
- If there's no match for a row in the right table, NULL values are returned for columns from the right table.

SELECT * FROM table1

LEFT JOIN table2 ON table1.column = table2.column;

**RIGHT JOIN (or RIGHT OUTER JOIN)**:

- Returns all rows from the right table and matching rows from the left table.
- If there's no match for a row in the left table, NULL values are returned for columns from the left table.

SELECT * FROM table1

RIGHT JOIN table2 ON table1.column = table2.column;

**FULL JOIN (or FULL OUTER JOIN)**:

- Returns all rows from both tables and combines them based on the join condition.
- If there's no match for a row in one table, NULL values are returned for columns from the other table.

SELECT * FROM table1

FULL JOIN table2 ON table1.column = table2.column;

5.what are triggers in mysql?

In MySQL, triggers are special types of stored programs that are automatically executed or fired in response to specific events or actions performed on a table, such as INSERT, UPDATE, or DELETE operations. Triggers are used to enforce business rules, maintain data integrity, or automate tasks without requiring explicit invocation by the user.

Triggers are defined to execute in response to specific events such as INSERT, UPDATE, or DELETE operations on a table.

Triggers can be defined to execute either before or after the triggering event. These are known as **BEFORE** and **AFTER** triggers, respectively.

**CREATE TRIGGER trigger_name**

**{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON table_name**

**FOR EACH ROW**

**BEGIN**

**-- Trigger logic here**

**END;**

**Limitations**: Triggers have certain limitations, such as:

- They cannot be used with views.
- They cannot perform transactions (i.e., you cannot use **COMMIT** or **ROLLBACK** statements within a trigger).

6.What is meant by stored procedure in mysql?

A stored procedure in MySQL is a precompiled collection of SQL statements and procedural logic that is stored in the database catalog. It's similar to a function in programming languages, but it resides and is executed within the database system.

Stored procedures are precompiled and stored in the database catalog. This means that they can be executed repeatedly without recompilation, which can improve performance, especially for complex operations.

Once created, stored procedures can be called and executed multiple times from different parts of the application or within other SQL statements. This promotes code reuse and simplifies maintenance.

Stored procedures can accept parameters, allowing them to be more flexible and adaptable to different scenarios. Parameters can be used to customize the behavior of the procedure at runtime.

**CREATE PROCEDURE procedure_name(param1 datatype, param2 datatype, ...)**

**BEGIN**

**-- Procedure logic here**

**END;**

7.explain the difference between Truncate and delete?

1. **DELETE**:
    - **DELETE** is a DML (Data Manipulation Language) command.
    - It is used to remove individual rows from a table based on a specified condition or criteria.
    - **DELETE** can be used with a **WHERE** clause to specify which rows should be deleted.
    - It generates transaction logs for each deleted row, making it possible to rollback the changes if needed (if used within a transaction).
    - It may take longer to execute, especially for large tables, as it deletes rows one by one and triggers associated triggers, constraints, and cascades for each deleted row.

Example:

DELETE FROM table_name WHERE condition;

2. **TRUNCATE**:
    - **TRUNCATE** is a DDL (Data Definition Language) command.
    - It is used to remove all rows from a table, effectively resetting the table to its initial state.
    - **TRUNCATE** cannot be used with a **WHERE** clause; it removes all rows from the table.
    - It deallocates the data pages used by the table, effectively resetting the table's storage space.

- It resets auto-increment counters to their initial values.
- It is faster than **DELETE**, especially for large tables, as it does not generate transaction logs for each deleted row and does not trigger associated triggers, constraints, or cascades.
- It cannot be rolled back (unlike **DELETE** within a transaction).

Example:

TRUNCATE TABLE table_name;

8.explain DDL,DML,DCL,DQL

1. **DDL (Data Definition Language)**:
    - DDL commands are used to define, modify, and delete database objects such as tables, indexes, views, and schemas.
    - Examples of DDL commands include **CREATE**, **ALTER**, **DROP**, **TRUNCATE**, and **RENAME**.
    - DDL commands affect the structure of the database and its objects but not the data itself.
2. **DML (Data Manipulation Language)**:
    - DML commands are used to manipulate data within database objects such as tables.
    - Examples of DML commands include **INSERT**, **UPDATE**, **DELETE**, and **MERGE**.
    - DML commands are concerned with adding, modifying, and deleting data records within tables.
3. **DCL (Data Control Language)**:
    - DCL commands are used to control access to data within the database, manage user privileges, and define security settings.
    - Examples of DCL commands include **GRANT** and **REVOKE**, which are used to grant or revoke permissions to users and roles.
    - DCL commands are essential for ensuring data security and controlling user access to the database.
4. **DQL (Data Query Language)**:
    - DQL commands are used to retrieve data from the database.

- The primary DQL command is **SELECT**, which is used to retrieve specific data records or information from one or more tables based on specified criteria.
- DQL commands are used to query data and retrieve results for analysis, reporting, or manipulation.

Here's a summary of the four categories:

- **DDL** (Data Definition Language): Used to define, modify, and delete database objects.
- **DML** (Data Manipulation Language): Used to manipulate data within database objects.
- **DCL** (Data Control Language): Used to control access to data and manage user privileges.
- **DQL** (Data Query Language): Used to retrieve data from the database.

9.explain aggregate functions?

Aggregate functions in SQL are functions that perform a calculation on a set of values and return a single value. These functions are commonly used in conjunction with the **SELECT** statement to perform operations such as calculating totals, averages, counts, maximum values, and minimum values on columns of data.

1. **SUM()**: Calculates the sum of all values in a column.

```
SELECT SUM(sales_amount) AS total_sales FROM sales;
```

2. **AVG()**: Calculates the average of all values in a column.

```
SELECT AVG(sales_amount) AS average_sales FROM sales;
```

3. **COUNT()**: Counts the number of rows or non-null values in a column.

```
SELECT COUNT(*) AS total_orders FROM orders;
```

4. **MAX()**: Finds the maximum value in a column.

```
SELECT MAX(sales_amount) AS max_sales FROM sales;
```

5. **MIN()**: Finds the minimum value in a column.

```
SELECT MIN(sales_amount) AS min_sales FROM sales;
```

10.which is faster between CTE and subquery?

The performance of Common Table Expressions (CTEs) and subqueries can vary depending on the specific query, the database engine being used, and the underlying data structures. However, in many cases, there is no significant difference in performance between using a CTE and a subquery.

CTE stands for Common Table Expression. It is a temporary named result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs were introduced in SQL as part of the SQL:1999 standard and are supported by most modern relational database management systems, including MySQL, PostgreSQL, SQL Server, and Oracle.

```
WITH cte_name AS (
   -- CTE definition
   SELECT column1, column2
   FROM table_name
   WHERE condition
)
SELECT *
FROM cte_name;
```

11.what are constraints and explain its types?

In MySQL, constraints are rules that enforce data integrity and define restrictions on the values that can be inserted, updated, or deleted in a table. They are used to maintain the accuracy, consistency, and reliability of the data stored in the database.

**PRIMARY KEY constraint**:

- A primary key constraint uniquely identifies each record in a table and ensures that the values in the specified column(s) are unique and not null.
- Each table can have only one primary key constraint.

```
CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50)
);
```

**FOREIGN KEY constraint**:

- A foreign key constraint establishes a relationship between two tables by enforcing referential integrity.
- It ensures that the values in a column (or a set of columns) in one table match the values in a column (or a set of columns) in another table.
- Example:

```
CREATE TABLE orders
( order_id INT PRIMARY KEY,
customer_id INT,
order_date DATE,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id) );
```

**UNIQUE constraint**:

- A unique constraint ensures that all values in the specified column(s) are unique and do not contain duplicate values.
- Unlike a primary key constraint, a unique constraint allows null values (except for the columns that are part of the constraint).
- Example:

```
CREATE TABLE employees
( employee_id INT PRIMARY KEY,
 employee_email VARCHAR(50) UNIQUE );
```

**CHECK constraint**:

- A check constraint enforces specific conditions on the values allowed in a column.
- It ensures that the values inserted or updated in the column satisfy the specified condition.
- Example:

```
CREATE TABLE products
( product_id INT PRIMARY KEY,
product_name VARCHAR(50),
unit_price DECIMAL(10, 2),
quantity INT,
CHECK (unit_price > 0 AND quantity >= 0) );
```

**NOT NULL constraint**:

- A not null constraint ensures that a column cannot contain null values.
- It requires that all values inserted or updated in the column are not null.
- Example:

CREATE TABLE customers

( customer_id INT PRIMARY KEY,

customer_name VARCHAR(50) NOT NULL );

12.Explain about different types of keys in mysql?

**Primary Key**:

- A primary key uniquely identifies each record in a table.
- It must contain unique values and cannot contain NULL values.
- Each table can have only one primary key.
- Example:

CREATE TABLE students ( student_id INT PRIMARY KEY, student_name VARCHAR(50) );

**Foreign Key**:

- A foreign key establishes a relationship between two tables by referencing the primary key or unique key of another table.
- It enforces referential integrity, ensuring that values in the foreign key column(s) match values in the referenced primary key or unique key column(s) of the related table.
- Example:

CREATE TABLE orders ( order_id INT PRIMARY KEY,

customer_id INT,

order_date DATE,

FOREIGN KEY (customer_id) REFERENCES customers(customer_id) );

**Unique Key**:

- A unique key ensures that all values in the specified column(s) are unique and do not contain duplicate values.

- Unlike a primary key, a unique key can contain NULL values (except for the columns that are part of the unique key constraint).
- Example:

CREATE TABLE employees ( employee_id INT PRIMARY KEY, employee_email VARCHAR(50) UNIQUE );

**Composite Key**:

- A composite key consists of multiple columns that together uniquely identify each record in a table.
- It is used when a single column is not sufficient to uniquely identify records.
- Example:

CREATE TABLE order_details

( order_id INT,

product_id INT,

PRIMARY KEY (order_id, product_id),

FOREIGN KEY (order_id) REFERENCES orders(order_id),

FOREIGN KEY (product_id) REFERENCES products(product_id) );

13.difference between where and group by?

1. **WHERE clause**:
    - The **WHERE** clause is used to filter rows from a table based on specific conditions.
    - It is used to specify which rows should be included in the result set.
    - Conditions in the **WHERE** clause are applied to individual rows before any grouping or aggregation takes place.
    - It is typically used to filter data before grouping or to apply conditions to individual rows.
    - Example:

SELECT column1, column2 FROM table_name WHERE condition;

2. **GROUP BY clause**:
    - The **GROUP BY** clause is used to group rows that have the same values into summary rows, typically to perform aggregate functions on each group.

- It is used in conjunction with aggregate functions such as **SUM**, **AVG**, **COUNT**, etc.
- It divides the rows returned by the **FROM** clause into groups based on the specified column(s).
- It is applied after the **WHERE** clause and before the **SELECT** clause.
- It is used to summarize data and perform calculations on groups of rows.
- Example:

SELECT department, COUNT(*) AS num_employees FROM employees GROUP BY department;

14.what are views?

A view is a virtual table that does not store data itself but instead represents a result set of a query executed against one or more base tables. A view is defined by a SELECT statement, which specifies the columns to include, any filtering or aggregation conditions, and the source tables. Views provide a layer of abstraction that separates the logical presentation of data from its physical storage. Users can interact with views without needing to know the underlying structure of the tables. Views can be used to restrict access to certain columns or rows of a table, providing a way to enforce security policies and control data access.

**CREATE VIEW employee_view AS**

**SELECT employee_id, first_name, last_name, department**

**FROM employees**

**WHERE department = 'Engineering';**

15.difference between varchar and nvarchar?

1. **VARCHAR**:
    - **VARCHAR** stands for "Variable Character".
    - It is used to store non-Unicode (or ASCII) character data.
    - It typically requires 1 byte of storage per character.
    - The maximum number of characters that can be stored depends on the declared length of the column.
    - Example:

CREATE TABLE my_table ( name VARCHAR(50) );

2. **NVARCHAR**:

- **NVARCHAR** stands for "National Variable Character".
- It is used to store Unicode character data.
- It typically requires 2 bytes of storage per character because Unicode characters require more space to represent.
- The maximum number of characters that can be stored depends on the declared length of the column, but keep in mind that since it uses 2 bytes per character, it can store fewer characters compared to **VARCHAR**.
- Example:

CREATE TABLE my_table ( name NVARCHAR(50) );

16.difference between char and nchar?

1. **CHAR**:
    - **CHAR** stands for "Character".
    - It is used to store non-Unicode (or ASCII) character data.
    - It stores strings in a fixed-length format, padding the string with spaces if it is shorter than the specified length.
    - It typically requires 1 byte of storage per character.
    - Example:

sqlCopy code

CREATE TABLE my_table ( name CHAR(10) );

2. **NCHAR**:
    - **NCHAR** stands for "National Character".
    - It is used to store Unicode character data.
    - It also stores strings in a fixed-length format, padding the string with spaces if it is shorter than the specified length.
    - It typically requires 2 bytes of storage per character because Unicode characters require more space to represent.
    - Example:

sqlCopy code

CREATE TABLE my_table ( name NCHAR(10) );

17.what are index and its types?

In MySQL, an index is a data structure that improves the speed of data retrieval operations on a table by providing quick access to rows based on the values of one or more columns. Indexes are created on specific columns of a table, and they facilitate efficient data retrieval by organizing the data in a sorted manner.

Here are the main types of indexes in MySQL:

1. **Single-Column Index**:
    - A single-column index is created on a single column of a table.
    - It speeds up queries that filter or sort data based on the indexed column.
    - Example:

CREATE INDEX index_name ON table_name (column_name);

2. **Composite Index**:
    - A composite index is created on multiple columns of a table.
    - It speeds up queries that filter or sort data based on a combination of the indexed columns.
    - Composite indexes can improve query performance for multi-column WHERE clauses, ORDER BY clauses, and JOIN operations.
    - Example:

CREATE INDEX index_name ON table_name (column1, column2);

3. **Unique Index**:
    - A unique index ensures that all values in the indexed column(s) are unique and do not contain duplicate values.
    - It provides a constraint to enforce data integrity, similar to a PRIMARY KEY constraint.
    - Unique indexes can be created on both single columns and multiple columns.
    - Example:

CREATE UNIQUE INDEX index_name ON table_name (column_name);

4. **Primary Key Index**:
    - A primary key index is created automatically when a PRIMARY KEY constraint is defined on a table.

- It enforces the uniqueness of values in the specified column(s) and provides a fast lookup mechanism for primary key values.
- Each table can have only one primary key index.
- Example:

CREATE TABLE table_name ( column_name INT PRIMARY KEY, ... );

18.explain about different types of relationships in mysql?

relationships between tables are established using foreign keys, which define how data in one table relates to data in another table. There are three main types of relationships that can exist between tables:

**One-to-One (1:1) Relationship**:

- In a one-to-one relationship, each record in one table is related to exactly one record in another table.
- Example: A table for storing employee information (e.g., employee details) linked to a table for storing employee contact information (e.g., email, phone number), where each employee has exactly one corresponding contact record.

**One-to-Many (1:N) Relationship**:

- In a one-to-many relationship, each record in one table can be related to one or more records in another table, but each record in the second table is related to at most one record in the first table.
- Example: A table for storing customers linked to a table for storing orders, where each customer can have multiple orders associated with them, but each order is associated with exactly one customer.

**Many-to-Many (N:M) Relationship**:

- In a many-to-many relationship, each record in one table can be related to one or more records in another table, and vice versa.
- Example: A table for storing students linked to a table for storing courses, where each student can enroll in multiple courses, and each course can have multiple enrolled students. This relationship is implemented using a join table to store student-course pairs.

19.difference between union and union all?

1. **UNION**:
    - The **UNION** operator is used to combine the result sets of two or more **SELECT** queries into a single result set.
    - It removes duplicate rows from the final result set.
    - The columns in the result sets being combined must have the same data types and be in the same order.
    - Example:

SELECT column1, column2 FROM table1 UNION SELECT column1, column2 FROM table2;

2. **UNION ALL**:
    - The **UNION ALL** operator is also used to combine the result sets of two or more **SELECT** queries into a single result set.
    - Unlike **UNION**, **UNION ALL** does not remove duplicate rows from the final result set.
    - It includes all rows from all result sets, including duplicates.
    - **UNION ALL** is generally faster than **UNION**, as it does not need to perform the additional step of removing duplicates.
    - Example:

SELECT column1, column2 FROM table1 UNION ALL SELECT column1, column2 FROM table2;

20.write an SQL query to print the second highest salary of an employee?

**SELECT MAX(salary) AS second_highest_salary**

**FROM employees**

**WHERE salary < (SELECT MAX(salary) FROM employees);**

This query uses a subquery to find the maximum salary (which is the highest salary) in the **employees** table. Then, it uses this value to filter out the maximum salary from the main query using the **<** (less than) operator. Finally, it selects the maximum value among the remaining salaries, which corresponds to the second highest salary.

**SELECT DISTINCT salary AS second_highest_salary**

**FROM employees**

**ORDER BY salary DESC**

**LIMIT 1 OFFSET 1;**

- We order the salaries in descending order using ORDER BY salary DESC.
- We then use LIMIT 1 OFFSET 1 to skip the first row (which corresponds to the highest salary) and select the second row, which gives us the second highest salary.
- We use DISTINCT to ensure that we only get unique values, in case there are multiple employees with the same salary.

21.Explain what is query optimization with an example?

Query optimization is the process of improving the efficiency and performance of a database query. This involves selecting the most efficient execution plan from various possible alternatives to retrieve data from the database. The goal is to minimize the query's resource consumption (such as CPU, memory, and disk I/O) and execution time while producing the same result set.

Let's consider an example to understand query optimization better:

Suppose you have a database table named "Employees" with columns like "EmployeeID," "Name," "DepartmentID," "Salary," and "JoiningDate." Now, imagine you want to retrieve the names of all employees in the "Marketing" department who joined after a certain date and earn a salary greater than a specific amount.

Here's a simplified version of the SQL query to retrieve this information:

**SELECT Name FROM Employees WHERE DepartmentID = 'Marketing' AND JoiningDate > '2023-01-01' AND Salary > 50000;**

Now, let's discuss how query optimization might be applied to this query:

1. **Indexing**: Indexes can be created on columns that are frequently used in WHERE clauses, such as "DepartmentID," "JoiningDate," and "Salary." These indexes help the database system quickly locate the relevant rows, reducing the overall query execution time.
2. **Query Rewriting**: Sometimes, queries can be rewritten to make them more efficient. For instance, combining multiple conditions using boolean logic (AND, OR) strategically can lead to a more optimized execution plan.
3. **Join Optimization**: If your query involves joining multiple tables, the order of joining and the type of join (e.g., INNER JOIN, LEFT JOIN) can significantly impact performance. The query optimizer analyzes different join strategies to determine the most efficient way to retrieve the data.
4. **Statistics Analysis**: The query optimizer relies on statistics about the data distribution in the tables to make informed decisions. Regularly updating these statistics ensures that the optimizer has accurate information to generate efficient execution plans.

5. **Query Caching**: If the same query is executed frequently with the same parameters, the database system may cache the query results. This can eliminate the need for executing the query repeatedly, improving overall performance.

22.Explain normalization ?

https://www.youtube.com/watch?v=ABwD8IYByfk

Normalization in SQL is a process used to organize a database structure efficiently, minimizing redundancy and dependency. It involves breaking down a single large table into smaller tables and defining relationships between them. The primary objectives of normalization are to reduce data redundancy, ensure data integrity, and make the database more flexible and scalable.

There are several levels of normalization, commonly referred to as normal forms. The most commonly used normal forms are:

1. **First Normal Form (1NF)**: This level requires that each column in a table contains atomic values, meaning that each value is indivisible. It eliminates repeating groups within individual rows and ensures that each column has a single value for each row.
2. **Second Normal Form (2NF)**: In addition to meeting the requirements of 1NF, a table in 2NF must ensure that no non-key attributes are functionally dependent on only a portion of a composite primary key. This form helps in removing partial dependencies.
3. **Third Normal Form (3NF)**: A table in 3NF must meet the criteria of 2NF and eliminate transitive dependencies. In other words, non-key attributes should not be dependent on other non-key attributes.