**MySQL Interview Based**

**Questions**

**1. What is DBMS?**

DBMS stands for Database Management System. It is used to store, retrieve, manage, and organize the data in the database. It defines the structure of the database and acts as an intermediary between the users and the data.

**2. What is RDBMS?**

RDBMS stands for Relational Database Management System. In RDBMS, the data is structured in table format. Each table represents an entity, a relationship between entities, or a relationship between entities in the real-world domain.

### 3. What is a database?

A database is a collection of structured data stored in a system. It is used to store, retrieve, and manage huge amounts of data. The data is organized in table format. Each table will have rows and columns to represent the structure of the data.

### ****4. What are the different types of database systems?****

There are two main types of database systems: relational and non-relational databases.

* **Relational databases** are the most common type of database system. They use a relational model to store data, which means that the data is stored in tables that are related to each other by common keys.
* **Non-relational databases** are a newer type of database system that can store data in a variety of formats, including graphs, documents, and JSON objects.

### ****5. What is the ER diagram in DBMS?****

### **An Entity-Relationship (ER) diagram is a visual representation of the entities (objects or concepts) and the relationships between them in a database management system (DBMS). It is a modeling technique used to design and represent the structure of a database.**

### ****6. Define a relation schema.****

### **Relation schemas are often depicted using entity-relationship diagrams (ER diagrams) that visually represent the relationships between entities. They help database designers and developers understand and define the structure of the database, allowing for effective data organization, retrieval, and manipulation.**

### ****7. What is normalization in DBMS?****

### **Normalization is the process of designing a database so that the data is stored in a way that is efficient and easy to maintain.**

### ****8. What is denormalization?****

**Denormalization is the process of adding redundancy to a database in order to improve performance.**

### ****9. What is a database constraint?****

**A database constraint is a rule that is enforced on a database table. Constraints can be used to enforce data integrity, such as ensuring that a column can only contain a certain type of data or that a column cannot be left blank.**

### ****10. What is a database view?****

**A database view is a virtual table that is created from one or more tables. Views can be used to simplify complex queries, hide sensitive data, or provide a different perspective on the data.**

### ****11. What is a database trigger?****

A database trigger is a piece of code that is executed when an event occurs in a database. Events that can cause a database trigger include insert, update, and delete operations.

### ****12. What is a database stored procedure?****

A database stored procedure is a collection of SQL statements that are stored in the database. Stored procedures can be used to perform complex tasks such as inserting, updating, and deleting data or running reports.

**13. What is a surrogate key? When and why is it used in a DBMS?**

A surrogate key is a unique identifier that is not related to the meaning of the data in a database. It is often used when a natural key (such as a customer name or product ID) is not unique. Surrogate keys are used to improve performance and data integrity.

### ****14. What is the concept of data warehousing in a DBMS?****

Data warehousing is the process of collecting data from a variety of sources and storing it in a central location for analysis. This data can be used to make decisions about business operations, such as which products to sell, where to open new stores, and how to market products.

### 15. What is MySQL?

MySQL is an open-source relational database management system (RDBMS) based on SQL (Structured Query Language).

**16.** **What is the difference between MySQL and SQL?**

SQL is a standard language for accessing and manipulating databases, while MySQL is an RDBMS that uses SQL to manage databases.

**17. What are some features of MySQL?**

Some features include:

* + Open-source
  + High performance
  + High flexibility
  + Cross-platform support
  + Data security
  + Scalability and reliability

### ****18. What is the default port for MySQL Server?****

The default port for MySQL Server is **3306**. Another standard default port is 1433 in [TCP/IP](https://intellipaat.com/blog/what-is-tcp-ip-model/) for SQL Server.

### ****19. How can a user get the current SQL version?****

The syntax for getting the current version of MySQL:

|  |  |
| --- | --- |
|  | **SELECT** VERSION (); |

#### **20.What are the different data types in MySQL?**

MySQL supports several data types, including **numeric types** (INT, FLOAT, DOUBLE), **date and time types** (DATE, DATETIME, TIMESTAMP), **string types** (CHAR, VARCHAR, TEXT), and more.

### ****21. What is the difference between CHAR and VARCHAR?****

When a table is created, CHAR is used to define the fixed length of the table and columns. The length value could be in the range of 1–255. The VARCHAR command is used to adjust the column and table lengths as required.

**22. What is DDL?**

DDL Stands for **Data Definition Language** in MySQL, and it is used in database schemas and description to determine how data should be stored in the database.

DDL Queries:

CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME.

**23. What is DML?**

**DML Stands for Data Manipulation Language and is used to manipulate data in databases. It largely consists of standard SQL commands for storing, modifying, retrieving, deleting, and updating data.**

**DML Queries:**

**SELECT, INSERT, UPDATE, DELETE, MERGE, CALL, EXPLAIN PLAN, LOCK TABLE.**

**24. What is DCL?**

**DCL stands for Data Control Language and encompasses instructions that deal with user rights, permissions, and other database system controls.**

**List of queries for DCL:**

**GRANT, REVOKE.**

### ****25. What are the advantages of using MySQL?****

There are various advantages of using MySQL. Some of them are given below:

**Advantages**

* MySQL helps in the secure management of databases. By using it, we can securely execute database transactions.
* It is fast and efficient in comparison to other database management systems as it supports varieties of storage engines.
* As its transaction processing is high, MySQL can execute millions of queries.

**26. What are the disadvantage of using MySQL?**

There are various disadvantages of using MySQL. Some of them are given below:

**Disadvantages**

* Scalability in MySQL is a redundant task.
* MySQL serves good for large databases mostly.
* There are issues of the instability of software.

### ****27. What are the different types of strings used in database columns in MySQL?****

In MySQL, the different types of strings that can be used for database columns are SET, BLOB, VARCHAR, TEXT, ENUM, and CHAR.

### ****28. How to create a user-defined function in MySQL?****

User-defined functions are created using the ‘CREATE FUNCTION’ syntax. It allows you to define your function that can be used in SQL statements. They come in handy when you perform calculations or operations frequently.

### ****29. What are subqueries in MySQL?****

When we write queries inside a query or nested queries are called subqueries. They allow complex operations and can be used in [SELECT](https://intellipaat.com/blog/tutorial/sql-tutorial/select-query-in-sql/), [INSERT](https://intellipaat.com/blog/tutorial/sql-tutorial/insert-query-in-sql/), [UPDATE](https://intellipaat.com/blog/tutorial/sql-tutorial/update-query/), or [DELETE](https://intellipaat.com/blog/tutorial/sql-tutorial/delete-query/) statements. It can return single values, row sets, or tables as well as an output. They are often used for comparison and aggregation.

**30. What are indexes in MySQL?**

Indexes are special data structures that act like an organised filling system for tables. They speed up retrieved of specific data by referencing the indexed columns.

**31. What are the benefits of using indexes in MySQL?**

* **Faster SELECT queries, especially for filtering or sorting data.**
* **Improved JOIN performed by efficiently locating matching rows.**

**32. What are the different types of indexes in MySQL?**

* **Primary key: Enforce uniqueness, acts as a cluster index (data stored in primary key order).**
* **Unique: Similar to primary key, but allows for multiple columns.**
* **Regular Index: Most common type, used to speed up searches on single or multiple columns.**
* **Full-text Index: Designed for searching text content within a column.**

**33. When should you consider to using indexes in MySQL?**

* **Columns frequently used in WHERE clause conditions (equality, range-based queries).**
* **Columns involved in JOIN operations (ON clause).**
* **Ordering or grouping data (ORDER BY, GROUP BY).**

**34. What are some drawbacks of using indexes?**

* **Overhead: Creating and maintaining indexes consumes storage space and processing power.**
* **Slower inserts/updates: Due to index updates happening alongside data modifications.**
* **Not ideal for all queries: Only beneficial for specific search patterns.**

**35. What is Index selectivity?**

**Index selectivity refers to how effectively an index narrows down search results. A highly selective index is more beneficial (e.g., indexing a gender column with only male /female values).**

**36. How can you identify if an index is useful for a query?**

**Use the MySQL EXPLAIN command to analyse how the optimizer plans to use indexes for a specific query.**

**37.Is it okay to have multiple indexes on a single table?**

**Yes, but use them strategically. Too many indexes can slow down inserts/updates.**

**38. What happen if the data in an indexed column is frequently modified?**

**Frequently updates can impact performance as the index needs to be updated as well. Consider if the benefit of faster searches outweighs the overhead of maintaining the index.**

**39. How can you decide which columns to index?**

**Analyse your queries to identity frequently used columns in WHERE clauses, JOIN, and filtering operations. Choose columns with high selectivity for optimal performance gains.**

### ****40.What is known as the covering index in MySQL?****

A covering index is an index that includes all the columns needed to answer a query, allowing the database to retrieve query results directly from the index without accessing the table data.

### ****41. How to handle large datasets in MySQL?****

Handling large datasets in MySQL efficiently requires partitioning tables, using indexing strategically, optimizing queries, and possibly leveraging MySQL’s advanced storage engines like InnoDB for better performance and reliability.

### ****42. Explain the GRANT command in MySQL.****

The GRANT command is used to give permission or certain privileges to perform various database operations whenever a new user is created.

### ****43. What is the main difference between MySQL and PostgreSQL?****

**The basic difference between**[**MySQL and PostgreSQL**](https://intellipaat.com/blog/postgresql-vs-mysql/)**are:**

My SQL is purely a relational database whereas PostgreSQL is an object-relational database. PostgreSQL is more complex and slower than MySQL. In MYSQL, troubleshooting is easy but it is difficult to troubleshoot PostgreSQL. MySQL does not support materialized view whereas PostgreSQL support materialized view.

### ****44. What is the use of the DELIMITER command in MySQL?****

The DELIMITER command is used to change the default delimiter used by MySQL, which is a semicolon (;). DELEIMITER is used while writing TRIGGER and STORED PROCEDURES in MySQL.

### ****45. What's the difference between CHAR\_LENGTH and LENGTH in MySQL?****

CHAR\_LENGTH counts how many characters are in a string, seeing all as single characters. LENGTH counts how many bytes the string is, which changes with different characters.

### ****46. How do you change a table's name?****

Change a table’s name with RENAME TABLE old\_table\_name TO new\_table\_name;. This updates the table’s name in your database.

### ****47. What are operators in MySQL?****

Operators in MySQL are symbols that specify the type of operation to be performed on operands. They include arithmetic, comparison, logical, and bitwise operators, among others.

### ****48**. **Can you explain the different types of operators available in MySQL?****

MySQL operators can be categorized as follows:

**Arithmetic Operators:** +, -, \*, /, % (modulus)

**Comparison Operators:** =, !=, <>, <, <=, >, >=, BETWEEN, IN, IS NULL, IS NOT NULL, LIKE

**Logical Operators:** AND, OR, NOT, XOR

**Bitwise Operators:** &, |, ^, ~, <<, >>

**Assignment Operators:** =, :=

### ****49.** **How does the** BETWEEN **operator work in MySQL?****

The **BETWEEN** operator is used to filter the result set within a certain range. It is inclusive, meaning it includes both the start and end values.

### ****50**. **What is the difference between** = **and** LIKE **operators?****

The = operator is used for exact matches in a comparison, while the LIKE operator is used for pattern matching. Example

(1) **SELECT \* FROM users WHERE username = 'john\_doe';**

This query selects users with the exact username 'john\_doe'.

(2) **SELECT \* FROM users WHERE username LIKE 'john%';**

This query selects users whose usernames start with 'john'.

### ****51**. **How does the ‘IN’ operator work, and how is it different from ‘=’?****

The IN operator allows you to specify multiple values in a WHERE clause. It is functionally similar to multiple OR conditions. Example:

**SELECT \* FROM employees WHERE department IN ('HR', 'Finance', 'IT');**

This query selects employees who work in the HR, Finance, or IT departments.

### ****52.** **Explain the use of the IS NULL operator.****

The **IS NULL** operator is used to check for NULL values. Example

**SELECT \* FROM employees WHERE manager\_id IS NULL;**

This query selects employees who do not have a manager (i.e., their manager\_id is NULL).

**53. What is the purpose of the AND operator?**

The AND operator is used to combine multiple conditions in a WHERE clause. All conditions must be true for the row to be included in the result set. Example:

**SELECT \* FROM employees WHERE salary > 50000 AND department = 'IT';**

This query selects employees who have a salary greater than 50,000 and work in the IT department.

**54. How do the AND and OR operators differ in MySQL?**

The AND operator requires all combined conditions to be true, while the OR operator requires at least one of the conditions to be true. Example:

**SELECT \* FROM employees WHERE salary > 50000 AND department = 'IT';**

**vs.**

**SELECT \* FROM employees WHERE salary > 50000 OR department = 'IT';**

The first query selects employees in the IT department with salaries over 50,000, while the second query selects employees with salaries over 50,000 or employees in the IT department.

**55. What is the NOT operator used for in MySQL?**

The NOT operator negates a condition. It is used to exclude rows that meet a certain condition. Example:

**SELECT \* FROM employees WHERE NOT department = 'HR';**

This query selects employees who do not work in the HR department.

**56. How does the % (modulus) operator work in MySQL?**

The % operator calculates the remainder of a division operation. Example:

**SELECT 10 % 3;**

This query returns 1, as 10 divided by 3 leaves a remainder of 1.

**57. What are Joins in MySQL?**

Joins in MySQL are used to combine rows from two or more tables based on a related column between them. They allow for querying data across multiple tables.

**58. What are the different types of Joins in MySQL?**

The different types of joins in MySQL are:

* INNER JOIN
* LEFT JOIN (or LEFT OUTER JOIN)
* RIGHT JOIN (or RIGHT OUTER JOIN)
* FULL JOIN (or FULL OUTER JOIN) - not directly supported in MySQL, but can be simulated using UNION
* CROSS JOIN
* SELF JOIN

**59. Explain INNER JOIN with an example.**

INNER JOIN returns rows that have matching values in both tables. Example:

**SELECT employees.name, departments.name**

**FROM employees**

**INNER JOIN departments ON employees.department\_id = departments.id;**

This query returns the names of employees along with the names of their corresponding departments where there is a match.

**60. What is a LEFT JOIN?**

A LEFT 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. Example:

**SELECT employees.name, departments.name**

**FROM employees**

**LEFT JOIN departments ON employees.department\_id = departments.id;**

This query returns all employees, and the department names where a match exists; otherwise, it returns NULL for department names.

**61. Describe a RIGHT JOIN with an example.**

A RIGHT 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. Example:

**SELECT employees.name, departments.name**

**FROM employees**

**RIGHT JOIN departments ON employees.department\_id = departments.id;**

This query returns all departments, and the employee names where a match exists; otherwise, it returns NULL for employee names.

**62. What is a FULL JOIN, and how can it be achieved in MySQL?**

A FULL JOIN returns rows when there is a match in one of the tables. Since MySQL does not support FULL JOIN directly, it can be simulated using UNION:

**SELECT employees.name, departments.name**

**FROM employees**

**LEFT JOIN departments ON employees.department\_id = departments.id**

**UNION**

**SELECT employees.name, departments.name**

**FROM employees**

**RIGHT JOIN departments ON employees.department\_id = departments.id;**

This query combines the results of LEFT JOIN and RIGHT JOIN to include all rows from both tables.

**63. What is a CROSS JOIN?**

A CROSS JOIN returns the Cartesian product of the two tables, meaning it returns all possible combinations of rows from the two tables. Example:

**SELECT employees.name, departments.name**

**FROM employees**

**CROSS JOIN departments;**

This query returns all combinations of employees and departments.

**64. Explain a SELF JOIN with an example.**

A SELF JOIN is a regular join, but the table is joined with itself. Example:

**SELECT e1.name AS employee, e2.name AS manager**

**FROM employees e1**

**INNER JOIN employees e2 ON e1.manager\_id = e2.id;**

This query returns a list of employees and their corresponding managers from the same employees table.

**65. What is the difference between INNER JOIN and OUTER JOIN?**

INNER JOIN returns only the rows with matching values in both tables. OUTER JOIN (LEFT JOIN, RIGHT JOIN, FULL JOIN) returns all rows from one or both tables, including rows that do not have matching values in the other table, with NULL values for missing matches.

**66. When should you use an INNER JOIN over a LEFT JOIN?**

Use an INNER JOIN when you need only the rows with matching values in both tables. Use a LEFT JOIN when you need all rows from the left table, regardless of whether there are matching rows in the right table.

### 67. How can you join three or more tables in MySQL?

You can join three or more tables by using multiple JOIN statements. Example:

**SELECT e.name, d.name, p.name**

**FROM employees e**

**INNER JOIN departments d ON e.department\_id = d.id**

**INNER JOIN projects p ON e.project\_id = p.id;**

This query joins the employees, departments, and projects tables.

### 68. What is a NATURAL JOIN?

A NATURAL JOIN automatically joins tables based on columns with the same name and compatible data types in both tables. Example:

**SELECT \***

**FROM employees**

**NATURAL JOIN departments;**

This query joins the employees and departments tables on columns with the same name.

### 69. What are the performance implications of using joins?

Joins, especially on large tables, can be resource-intensive and slow if not optimized properly. Indexing the columns used in join conditions can significantly improve performance. Properly designing database schemas and writing efficient queries are also important for performance.

### 70. How can you optimize join queries?

### To optimize join queries, you can:

* Use indexes on columns used in join conditions.
* Avoid joining unnecessary tables.
* Use appropriate join types (INNER, LEFT, etc.).
* Write queries to minimize the result set early (e.g., using WHERE clauses).
* Analyze and use query execution plans to understand and optimize query performance.

### 71. What is the purpose of using aliases in joins?

Aliases provide temporary names for tables or columns, making the SQL queries easier to read and write, especially when joining multiple tables or using self joins. Example:

**SELECT e.name AS employee, d.name AS department**

**FROM employees AS e**

**INNER JOIN departments AS d ON e.department\_id = d.id;**

This query uses aliases e for employees and d for departments to simplify the query.

### 72. What is the CASE clause, and how is it used in MySQL?

The CASE clause is used to create conditional logic in SQL queries. It allows you to return different values based on different conditions. Example:

**SELECT name,**

**CASE**

**WHEN salary > 50000 THEN 'High'**

**WHEN salary BETWEEN 30000 AND 50000 THEN 'Medium'**

**ELSE 'Low'**

**END AS salary\_range**

**FROM employees;**

This query categorizes employees' salaries into 'High', 'Medium', and 'Low' ranges.

### 73. How does the DISTINCT clause work in MySQL?

The DISTINCT clause is used to remove duplicate rows from the result set. Example:

**SELECT DISTINCT department\_id FROM employees;**

This query retrieves unique department IDs from the employees table.

### 74. How do you use the OFFSET clause with LIMIT?

The OFFSET clause is used to skip a specified number of rows before starting to return rows. It is often used with the LIMIT clause for pagination.

### 75. Explain the GROUP BY clause with an example.

The GROUP BY clause is used to arrange identical data into groups. It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to perform operations on each group of data. Example:

**SELECT department\_id, COUNT(\*) AS num\_employees**

**FROM employees**

**GROUP BY department\_id;**

This query counts the number of employees in each department.

### 76. How does the ORDER BY clause work in MySQL?

The ORDER BY clause is used to sort the result set in either ascending (ASC) or descending (DESC) order based on one or more columns. Example:

**SELECT \* FROM employees ORDER BY salary DESC;**

This query retrieves all employees sorted by their salary in descending order.

### 77. What is the purpose of the WHERE clause in MySQL?

The WHERE clause is used to filter records that meet a certain condition. It is used to extract only those records that fulfill a specified criterion. Example:

**SELECT \* FROM employees WHERE department\_id = 1;**

This query retrieves all employees in the department with department\_id 1.

### 78. What is the purpose of the UNION operator in MySQL?

The UNION operator is used to combine the result sets of two or more SELECT statements. It removes duplicate rows from the result set.

### 79. What is the difference between UNION and UNION ALL?

The UNION operator removes duplicate rows from the combined result set, while UNION ALL includes all rows, allowing duplicates. UNION performs a distinct operation by default, which can add overhead in terms of performance, whereas UNION ALL is faster because it does not perform this additional step.

### 80. Can ORDER BY be used with UNION and UNION ALL? If so, how?

Yes, ORDER BY can be used with UNION and UNION ALL, but it must be placed at the end of the combined result set.

### 81. Can you use UNION with subqueries?

### Yes, UNION can be used with subqueries.

### 82. What is a primary key in MySQL?

A primary key is a column, or a set of columns, that uniquely identifies each row in a table. It enforces uniqueness and ensures that no duplicate values are stored in the primary key column(s). Additionally, a primary key column cannot contain NULL values.

### 83. Can a table have more than one primary key?

No, a table can have only one primary key. However, a primary key can consist of multiple columns (composite primary key).

### 84. Can a primary key be NULL in MySQL?

No, a primary key cannot be NULL. It must have a unique, non-null value for each row.

### 85. What is a composite primary key?

A composite primary key is a primary key that consists of two or more columns. It is used when a single column is not sufficient to uniquely identify each row.

### 86. What is a foreign key in MySQL?

A foreign key is a column, or a set of columns, in one table that uniquely identifies a row of another table. It creates a link between the two tables, enforcing referential integrity.

### 87. What happens when a referenced row is deleted in MySQL?

The behavior depends on the foreign key constraints. Common actions are:

* **CASCADE**: Deletes the rows in the child table.
* **SET NULL**: Sets the foreign key column in the child table to NULL.
* **RESTRICT**: Prevents the deletion of the referenced row.
* **NO ACTION**: Similar to RESTRICT, but enforcement happens at the end of the transaction.
* **SET DEFAULT**: Sets the foreign key column in the child table to its default value.

### 88. What is the purpose of using foreign keys?

Foreign keys enforce referential integrity by ensuring that a value in the child table corresponds to a valid value in the parent table. This helps maintain consistency and prevent orphaned records.

### 89. Can a foreign key reference a non-primary key column?

Yes, a foreign key can reference a column with a unique constraint. However, it is more common and best practice to reference the primary key of the parent table.

### 90. What is a unique key in MySQL?

A unique key is a column, or a set of columns, that must contain unique values. Unlike primary keys, unique keys can contain NULL values, but the combination of columns must be unique for each row.

### 91. Can a table have multiple unique keys?

Yes, a table can have multiple unique keys. Each unique key ensures that the combination of values in the specified columns is unique across all rows in the table.

### 92. Can a unique key contain NULL values?

Yes, a unique key can contain NULL values. However, any combination of non-NULL values must be unique.

### 93. What is the difference between a primary key and a unique key?

The primary differences are:

* A primary key enforces both uniqueness and non-null constraints, while a unique key enforces only uniqueness and can contain NULL values.
* A table can have only one primary key, but multiple unique keys.
* The primary key uniquely identifies each row in a table, whereas a unique key ensures that a set of columns contains unique values.

### 94. Can a unique key be a composite key?

Yes, a unique key can be a composite key, ensuring that the combination of values in the specified columns is unique.

### 95. What is partitioning in MySQL?

Partitioning in MySQL is a database organization technique that divides a large table into smaller, more manageable pieces called partitions. Each partition is a subset of the table's data, and together they comprise the whole table.

### 96. What are the types of partitioning available in MySQL?

MySQL supports several types of partitioning:

* **Range Partitioning:** Divides data based on ranges of values.
* **List Partitioning:** Divides data based on a list of values.
* **Hash Partitioning:** Divides data using a hash function.
* **Key Partitioning:** Similar to hash partitioning but based on MySQL’s internal function.
* **Subpartitioning:** Allows further division of partitions into subpartitions.

### 97. How does range partitioning work in MySQL?

Range partitioning assigns rows to partitions based on column values that fall within a specified range. Example:

**CREATE TABLE sales (**

**id INT,**

**amount DECIMAL(10, 2),**

**sale\_date DATE**

**)**

**PARTITION BY RANGE (YEAR(sale\_date)) (**

**PARTITION p0 VALUES LESS THAN (2010),**

**PARTITION p1 VALUES LESS THAN (2015),**

**PARTITION p2 VALUES LESS THAN (2020),**

**PARTITION p3 VALUES LESS THAN MAXVALUE**

**);**

This table partitions sales data into ranges based on the year of the sale\_date.

### 98. What is list partitioning in MySQL?

List partitioning assigns rows to partitions based on column values that match a predefined list. Example:

**CREATE TABLE employees (**

**id INT,**

**name VARCHAR(100),**

**country\_code CHAR(2)**

**)**

**PARTITION BY LIST (country\_code) (**

**PARTITION pUSA VALUES IN ('US'),**

**PARTITION pCAN VALUES IN ('CA'),**

**PARTITION pMEX VALUES IN ('MX')**

**);**

This table partitions employee data based on country codes.

### 99. Explain hash partitioning in MySQL.

Hash partitioning assigns rows to partitions based on a hash function applied to a column value. Example:

**CREATE TABLE orders (**

**id INT,**

**customer\_id INT,**

**order\_date DATE**

**)**

**PARTITION BY HASH (customer\_id) PARTITIONS 4;**

This table partitions order data into 4 partitions based on a hash of the customer\_id.

### 100. What is key partitioning in MySQL?

Key partitioning is similar to hash partitioning but uses MySQL’s internal function to determine the partition. Example:

**CREATE TABLE products (**

**id INT,**

**name VARCHAR(100),**

**category\_id INT**

**)**

**PARTITION BY KEY (category\_id) PARTITIONS 3;**

This table partitions product data into 3 partitions based on MySQL’s internal key function applied to category\_id.

### 101. What are subpartitions in MySQL?

Subpartitions allow further division of each partition into smaller subpartitions. This is useful for more granular data management. Example:

**CREATE TABLE transactions (**

**id INT,**

**amount DECIMAL(10, 2),**

**transaction\_date DATE**

**)**

**PARTITION BY RANGE (YEAR(transaction\_date))**

**SUBPARTITION BY HASH (id)**

**SUBPARTITIONS 2 (**

**PARTITION p0 VALUES LESS THAN (2015),**

**PARTITION p1 VALUES LESS THAN MAXVALUE**

**);**

This table first partitions data by year and then subpartitions each range using a hash function on id.

### 102. What are the benefits of partitioning in MySQL?

### Benefits include:

* Improved query performance by scanning only relevant partitions.
* Easier data management and maintenance.
* Efficient data archiving and purging.
* Better load distribution for parallel processing.

### 103. Can you explain the MAXVALUE keyword in range partitioning?

The MAXVALUE keyword is used in range partitioning to define an upper bound for the last partition that captures any value greater than the specified ranges. Example:

**PARTITION BY RANGE (year) (**

**PARTITION p0 VALUES LESS THAN (2000),**

**PARTITION p1 VALUES LESS THAN (2010),**

**PARTITION p2 VALUES LESS THAN MAXVALUE**

**);**

The p2 partition includes any year value from 2010 onwards.

### 104. How do you check the partitions of a table in MySQL?

You can use the SHOW CREATE TABLE statement to view the partitioning scheme:

**SHOW CREATE TABLE table\_name;**

Or, use the INFORMATION\_SCHEMA.PARTITIONS table to query partition details:

**SELECT \* FROM INFORMATION\_SCHEMA.PARTITIONS WHERE TABLE\_NAME = 'table\_name';**

### 105. What is the difference between partitioning and sharding?

Partitioning divides a table within a single database instance into smaller, manageable pieces, whereas sharding distributes data across multiple database instances or servers. Partitioning is about breaking down a table internally, while sharding is about distributing the data across multiple nodes.

### 106. Can a table be partitioned by multiple columns in MySQL?

MySQL does not directly support partitioning by multiple columns. However, you can achieve similar results using subpartitioning. Example:

**CREATE TABLE logs (**

**id INT,**

**log\_date DATE,**

**server\_id INT**

**)**

**PARTITION BY RANGE (YEAR(log\_date))**

**SUBPARTITION BY HASH (server\_id)**

**SUBPARTITIONS 4 (**

**PARTITION p0 VALUES LESS THAN (2015),**

**PARTITION p1 VALUES LESS THAN MAXVALUE**

**);**

This table partitions by log\_date and subpartitions by server\_id.

### 107. What are some limitations of partitioning in MySQL?

Limitations include:

* Primary keys must include all partitioning columns.
* Foreign keys are not supported with partitioned tables.
* Some complex queries and joins might not benefit from partitioning.
* Partition maintenance can add overhead.

### 108. How do you add a partition to an existing partitioned table in MySQL?

You can use the ALTER TABLE statement to add a partition. Example:

**ALTER TABLE sales ADD PARTITION (**

**PARTITION p4 VALUES LESS THAN (2025)**

**);**

This adds a new partition to the sales table for values less than 2025.

### 109. How do you remove a partition from an existing partitioned table in MySQL?

Use the ALTER TABLE statement to drop a partition. Example:

**ALTER TABLE sales DROP PARTITION p3;**

This removes the p3 partition from the sales table.

### 110. What is partition pruning in MySQL?

Partition pruning is a query optimization technique where the query optimizer only scans the partitions that contain relevant data, reducing the amount of data to be scanned and improving query performance.

**111. How do you merge partitions in MySQL?**

Use the ALTER TABLE statement to merge partitions. Example:

**ALTER TABLE sales REORGANIZE PARTITION p1, p2 INTO (**

**PARTITION p1\_p2 VALUES LESS THAN (2020)**

**);**

This merges the p1 and p2 partitions into a new p1\_p2 partition.

### 112. Can you create indexes on partitioned tables in MySQL?

Yes, indexes can be created on partitioned tables, but the partition key must be part of the primary key or unique key. Example:

**CREATE TABLE employees (**

**id INT,**

**name VARCHAR(100),**

**department\_id INT,**

**PRIMARY KEY (id, department\_id)**

**)**

**PARTITION BY HASH (department\_id) PARTITIONS 4;**

This table creates a primary key that includes the partition key department\_id.

### 113. What is the impact of partitioning on query performance?

Partitioning can significantly improve query performance by reducing the amount of data scanned through partition pruning. It can also enhance load balancing and parallel processing. However, inappropriate partitioning or poor query design can negate these benefits.

### 114. Can you use ALTER TABLE to change the partitioning scheme of an existing table in MySQL?

Yes, but changing the partitioning scheme often involves significant data reorganization and can be resource-intensive. Example:

**ALTER TABLE employees PARTITION BY RANGE (id) (**

**PARTITION p0 VALUES LESS THAN (1000),**

**PARTITION p1 VALUES LESS THAN (2000),**

**PARTITION p2 VALUES LESS THAN MAXVALUE**

**);**

This changes the partitioning scheme of the employees table to range partitioning based on id.

### 116. What are aggregate functions in MySQL?

Aggregate functions in MySQL perform calculations on a set of values and return a single value, often used with the GROUP BY clause. Common aggregate functions include COUNT(), SUM(), AVG(), MAX(), and MIN().

**117. What does the COUNT() function do?**

The COUNT() function returns the number of rows that match a specified condition.

### 118. How does the SUM() function work?

The SUM () function calculates the total sum of a numeric column.

### 119. Explain the AVG() function with an example.

The AVG () function calculates the average value of a numeric column.

### 120. What is the purpose of the MAX() function?

The MAX () function returns the maximum value in a specified column.

### 121. How is the MIN() function used in MySQL?

The MIN () function returns the minimum value in a specified column.

### 122. Can you use aggregate functions with the GROUP BY clause?

Yes, aggregate functions are often used with the GROUP BY clause to perform calculations on each group of rows.

### 123. What is the difference between COUNT(\*) and COUNT(column\_name)?

COUNT(\*) counts all rows in a table, including rows with NULL values. COUNT(column\_name) counts non-NULL values in the specified column.

### 124. How can you find the total number of unique values in a column using aggregate functions?

You can use the COUNT(DISTINCT column\_name) function to count the number of unique values in a column.

### 125. What is the GROUP BY clause used for?

The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows, often used with aggregate functions.

### 126. How does the HAVING clause work with aggregate functions?

The HAVING clause is used to filter groups based on a condition involving aggregate functions, similar to the WHERE clause but for grouped rows.

### 127. Can you use multiple aggregate functions in a single query?

Yes, you can use multiple aggregate functions in a single query to perform various calculations on the data.

### 128. What is the WITH ROLLUP modifier in MySQL?

The WITH ROLLUP modifier is used with the GROUP BY clause to create summary rows for each group and an overall total row.

### 129. How do you calculate the percentage of a column's total using aggregate functions?

You can calculate the percentage by dividing the column value by the total sum and multiplying by 100.

### 130. What is the VARIANCE () function used for?

The VARIANCE () function calculates the statistical variance of a numeric column.

### 131. How do you use the STDDEV () function in MySQL?

The STDDEV () function calculates the standard deviation of a numeric column.

### 132. Can you use aggregate functions in a WHERE clause?

No, aggregate functions cannot be used directly in a WHERE clause. Instead, use the HAVING clause to filter based on aggregate functions.

### 133. What is the BIT\_AND () function used for?

The BIT\_AND () function calculates the bitwise AND of all values in a column.

### 134. How does the BIT\_OR () function work?

The BIT\_OR () function calculates the bitwise OR of all values in a column.

### 135. Explain the GROUP\_CONCAT () function with an example.

The GROUP\_CONCAT () function concatenates non-NULL values from a group into a single string, allowing you to see combined values from a group in one result.

Top of Form

Bottom of Form