SQL Interview Questions

1. Explain the different types of Constraints in SQL.

* Constraints in SQL are rules or Conditions that are applied to the columns or fields in the table. They helps ensure accuracy, integrity and consistency of the data in the database. Following are the types of constraints

1.**Not** **NULL** – Ensures that the column can not have a NULL value.

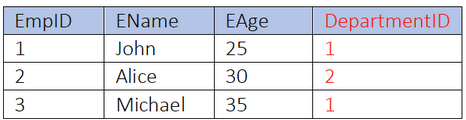
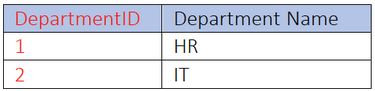
2.**Check** – Verify that all values in the field satisfy the specific condition. For example: Age column have the constraint CHECK age >= 18 means age should be greater than equal to 18 for the Age column.

3.**Default** – This constraint automatically assigns the default value to the field if no value has been specified. For example: salary column’s default value should be 0 if not specified explicitly.

4.**UNIQUE** – Ensures that each value in the field is unique means no duplicates allowed. For example : email column should have the unique values for each and every record.

5.**PRIMARY** **KEY** – Uniquely identifies each record in the table. It is the combination of NOT NULL and UNIQUE constraint. For example: employee id is the primary key which should have unique value for each and every employee and also it is not null.

6.**FOREIGN** **KEY** – Ensures the referential integrity between the tables. It establishes the relationship between the primary key in one table and the foreign key in another table.

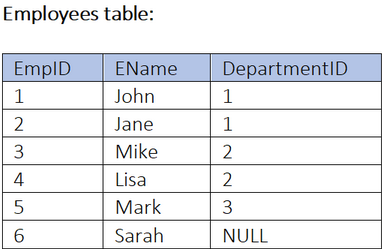
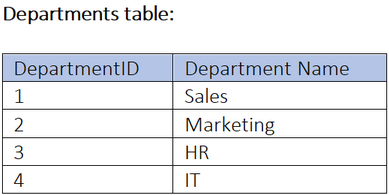
 

In the above example FOREIGN KEY constraint ensures that the referential integrity between the tables with the DepartmentID column.

1. What are the different types of JOINS in SQL.

* Joins are used to combine the rows from two or more tables by means of related columns. There are several types of joins INNER , RIGHT, LEFT, FULL OUTER, CROSS join.

Inner join returns only the matching rows between the tables from the specified conditions.

SELECT \* FROM Employees AS E JOIN Departments AS D ON E.DepartmentID=D.DepartmentID;

Left join return all rows from the left table and only the matching rows from the right table based on the specified conditions. If there are no matches NULL values are returned for the right table.

SELECT \* FROM Employees AS E LEFT JOIN Departments AS D ON E.DepartmentID=D.DepartmentID;

Right join returns all rows from the right table and only the matching rows from the left table based on the specified conditions. If there are no matches NULL values are returned for the left table.

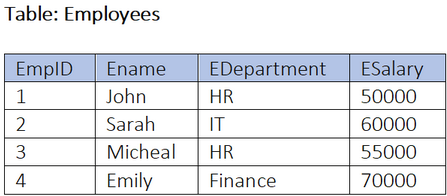
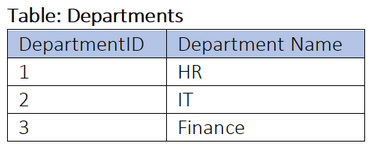
SELECT \* FROM Employees AS E LEFT JOIN Departments AS D ON E.DepartmentID=D.DepartmentID;

Full outer join returns all the rows from both the tables including unmatched rows. If there are no values NULL values are returned for the non-matching tables.

SELECT \* FROM Employees AS E FULL OUTER JOIN Departments AS D ON E.DepartmentID=D.DepartmentID;

1. What is subquery in SQL? Explain it with an example.

Subquery is a query within another query. It allows you to retrieve data from one table based on the result of the another query. Subqueries can be used at the different parts of the SQL statement, such as SELECT , FROM, WHERE and HAVING.



1.Using the subquery in WHERE clause retrieve the employees who worked in the HR department.

SELECT Ename FROM Employees

WHERE EDepartment = (SELECT Departement Name from Departements WHERE Departement Name =’HR’);

2.Using subquery in select clause retrieve avg salary of employees worked in the HR department.

SELECT EDepartement, (SELECT AVG(ESalary) FROM Employees WHERE EDepartement=’HR’) AS ‘AvgSalary’ FROM Departements

WHERE Departments=’HR’;

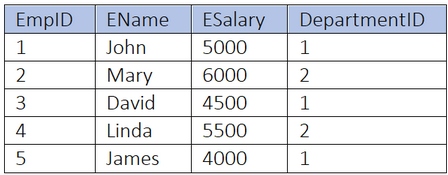
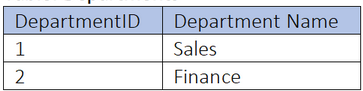
3.Using the subquery in the FROM clause retrieve the employees and their respective departements.

SELECT e.Ename, d.Departement Name FROM (SELECT \* FROM Employees) AS e

JOIN Departments AS d ON e.EDepartement = d.Department Name;

1. Explain GROUP BY in SQL with the example.

In SQL GROUP BY clause is used to group the rows based on One or more columns. It is commonly used with aggregation function like count, sum, avg to perform the calculation of each group of the rows.

We are taking the two tables Employees and Departments:

1.Group employees by department and calculate their total salary for each department.

SELECT EmpID, EName, ESalary FROM Employees

GROUP BY DepartemntID ;

2.Group employees by their departments and calculate their AVG salary for each of the departments.

SELECT e.EName , AVG(e.ESalary) , d.Department Name

FROM Employees AS e JOIN Departments AS d

ON e.DepartemntID = d.DepartementID GROUP BY d.Department Name;

3.Group the employees by their departments and count the number of employees in each of the Departments.

SELECT DepartmentID , COUNT(EmpID)

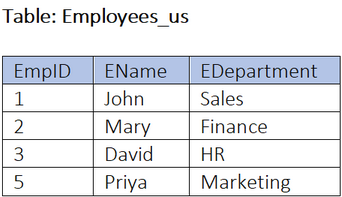
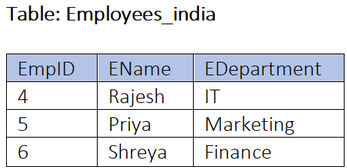
FROM Employees

GROUP BY DepartmentID;

These examples demonstrates how GROUP BY clause is used for grouping the rows in a tables based on the specific columns and perform calculations and aggregations on the specific columns.

1. Explain the SET operators in SQL.

SET operators in MySQL allows you to combine the two or more select statements. The commonly used set operators are UNIION , UNION ALL, INTERSECT and EXCEPT.

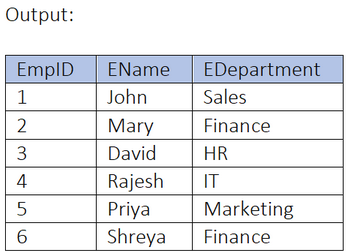
 

1.**UNION** operator : The UNION operator returns the combination of the two or more select statements to one table by removing the duplicates.

SELECT \* FROM Employees\_US

UNION

SELECT \* FROM Employees\_india;

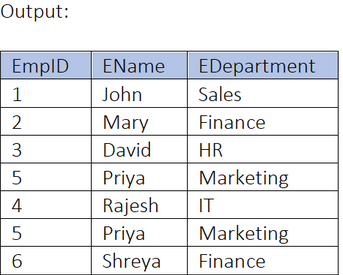


2.**UNION** **ALL** operator: The UNION ALL operator returns the combination of the tow or more select statements to one table by containing also the duplicate values.

SELECT \* FROM Employees\_US

UNION ALL

SELECT \* FROM Employees\_india;

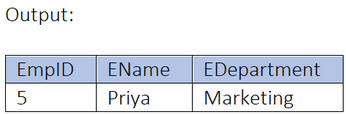


3.**INTERSECT** : INTERSECT returns the rows which are common in both the select statements.

SELECT \* FROM Employees\_US

INTERSECT

SELECT \* FROM Employees\_india;

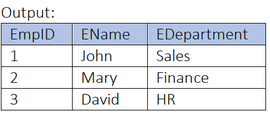


4.**EXCEPT**(**MINUS**) : Returns only the distinct rows from LEFT SELECT that are not in the result set of the RIGHT SELECT statement.

SELECT \* FROM Employees\_US

EXCEPT

SELECT \* FROM Employees\_india;

  
note: The EXCEPT operator is not supported in MySQL instead we can use LEFT JOIN or NOT IN operator.

1. What are the views in SQL give example of simple and complex views. What are the differences between the simple and complex views? Give queries as well as different types of views. And also show the outputs of those queries in the table.

VIEW is the virtual table which is derived from one or more tables. It is the saved SQL query that can be treated as table itself. Views allow users to simplify complex queries, restrict access to certain columns or rows, and provide an additional layer of security.

Simple view: Simple view is the view that is derived from a single table. It selects all the columns or the specific columns from the table and can include WHERE clause to filter out the rows.

CREATE VIEW high\_salary\_employee AS

SELECT \* FROM emoployees

WHERE Salary > 5000;

Query on sample view: SELECT \* FROM high\_salary\_employee;

Complex view: Complex view is the view i.e. derived from the multiple tables or other views. It can include JOINS, subqueries and aggregate functions to combine data from different sources.

CREATE VIEW recent\_orders AS

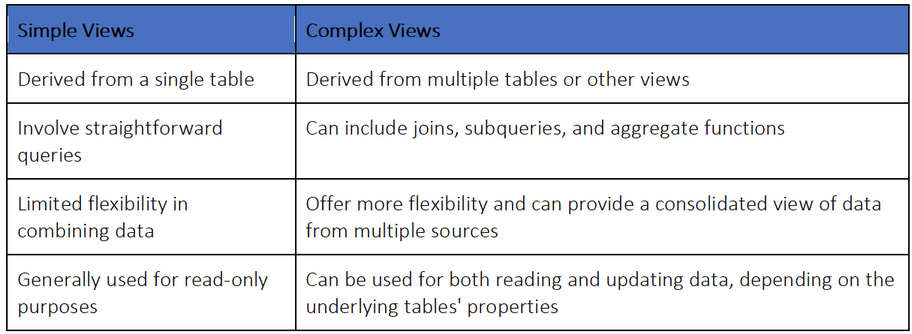
SELECT o.OrderID, o.Order\_Name, c.customer\_name FROM Orders AS o

JOIN Customer AS c ON o.customer\_id = c.customer\_id

WHERE o.order\_date >= CURDATE() – INTERVAL 30 Day;

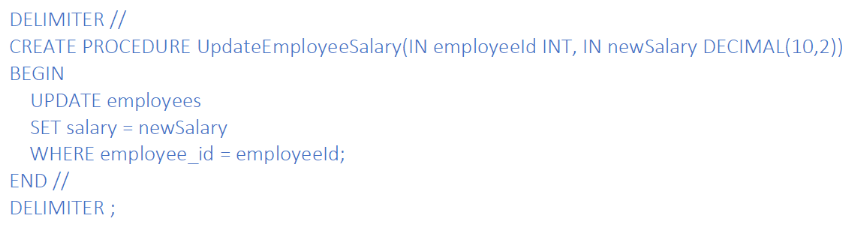
Query on complex view:

SELECT \* ON recent\_orders;



1. What are stored procedures in MySQL give example of the same.

Stored procedure in MySQL are the set of the SQL statements that are stored in the database and can be executed later. They allow the encapsulation and the reusability of the code, making it easier to manage complex database operations.

Here are some examples of stored procedures:

They provide the encapsulation of the complex logic and improve efficiency and maintainability of database operation.

1. What is the normalization in data warehousing? Why is that needed? Give examples for all the normalization forms.

Normalization in data warehousing is the process of organizing data in the database to eliminate redundancy and improve the data integrity. It involves breaking down the large tables into the smaller tables.

More manageable tables and maintaining relationship between them. The main goal of normalization is to reduce the data duplication and to ensure that each piece of information is stored at only one place.

Normalization in data warehousing is needed for the several reasons:

1.Eliminate data redundancy : by breaking down the large tables into the smaller one, normalization helps to eliminate the duplication of the data. This ensures that each datapoint is stored only once, which reduce the storage space and improves the data consistency.

2.Improve data integrity: Normalization establishes the relationship between the table using primary and foreign keys. This helps maintain the data integrity by ensuring that data dependencies are properly maintained and that no orphan record exist.

3.Enhance query performance: Normalized tables are designed to minimized data duplication and maximize the data consistency. This improves the query performance as it reduce the amount of data that needs to be processed and retrieved.

Here are the different normalization forms explained in a layman-friendly manner:

1. First Normal Form: This form ensures that each column in a table contains only atomic values (individual values). For example, instead of storing multiple phone numbers in a single column, each phone number should be stored in a separate column
2. Second Normal Form: In addition to meeting the criteria to 1NF , this form requires that all non-key attributes depend on the entire primary key. For instance , if a table has a composite primary key consisting of two columns, any non-key attribute should depend on both of these columns.
3. Third Normal form: Building on the previous forms, 3NF ensures that no non-key attribute is transitively dependent on the primary key. In simpler terms, it means that no attribute should depend on another attribute that is dependent on the primary key. For example, if a table has a column for a customer’s city and another column for the customer’s state, the state column should not depend on the city column.
4. Fourth Normal Form: This form deals with multi-valued dependencies. It ensures that there are non-key attributes that depend on other non-key attributes. In other words, it eliminates redundant data by separating multi-valued attributes into separate tables.
5. Fifth Normal Form: Also known as Project-Join Normal Form(PJNF), this form handles join dependencies. It ensures that a table is free from any redundancies that arise from combining multiple tables.It separates out the data that can be derived from other tables and stores it separately.

By following these normalization forms, databases can be structures in a way that promotes efficient data storage, retrieval and maintainance, ultimately leading to better data quality and system performance.

1. Explain the case statement in SQL with the example.

In MySQL, the case statement is used to perform conditional logic and control flow within a query. It allows you to specify multiple conditions and their corresponding actions, similar to an if-else statement.

1. Retrieve the employee names and their departments names, but if the department name is NULL, display “No Department” instead:

SELECT e.employee\_name,

CASE

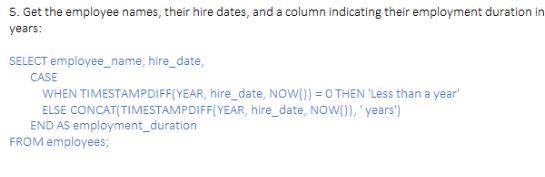
WHEN d.department\_name IS NULL THEN “no departement”

ELSE d.departement\_name

END AS department\_name

FROM employees AS e

LEFT JOIN departments d ON e.department\_id = d.department\_id;



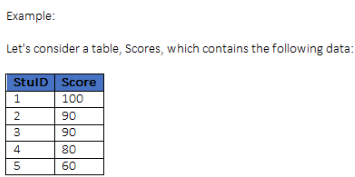
1. Explain the RANK(), DENSE\_RANK() and ROW\_NUMBER() functions in SQL with examples. Also, what are the difference between these functions?

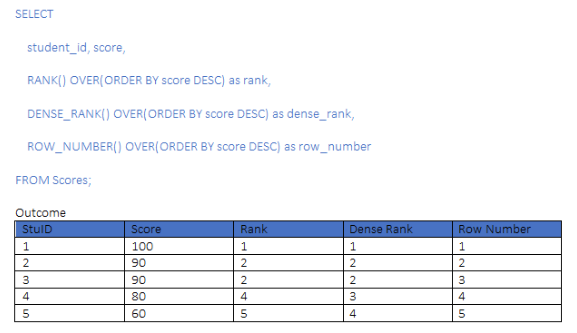
In SQL RANK(), DENSE\_RANK() and ROW\_NUMBER() are window functions used to assign a unique integer to rows within a result set based on the values in one or more columns.

RANK() : function assigns a unique rank to each distinct row within a result set sorted by specific columns ,with ties receiving the same rank. When two (or more) rows have the same rank, a gap appears in the sequence for the subsequent ranks.

DENSE\_RANK() : function also assigns a unique rank to each row within a result set sorted by specified columns, with ties recieveing the same rank. However unlike RANK(), it does not leaves the gap between the rank values when there are tied rows.

ROW\_NUMBER() : function assigns the unique integer to each row within a result set sorted by specified columns, regardless of the values in the sorted columns. Even if the sorted column values are identical, ROW\_NUMBER() will assign a different number to each row.





1. What is the difference between Stored Procedure and Function in MySQL?

SP and Function in MySQL both are the database objects that can be used to perform the specific tasks. However , there are some key differences between them.

1. **Return Value** : A stored procedure does not have a return value, whereas function always have a return value.
2. **Usage**: A SP is typically used to perform a series of actions or operations, whereas a function is used to compute and return a single value.
3. **Call**: A SP is called using the CALL statement, whereas a function is called as part of an SQL statement.
4. **Transaction**: A SP can contain transaction control statements like COMMIT and ROLLBACK , whereas function cannot.
5. **Variables**: A SP can have both input and output parameters, whereas function can only have the input parameters.
6. **Execution**: A SP is precompiled and stored in the database, whereas a function is compiled and executed each time it called.
7. **Security**: A SP can be granted or revoked privileges, whereas a function cannot.

In summary, stored procedure are used for performing a series of actions and do not have a return value, while functions are used for computation and always return a value.

1. How can you optimize subqueries to improve query performance?

There are several techniques to optimize subqueries and improve the query performance:

1.Use **EXISTS** or **NOT** **EXISTS** instead of IN or NOT IN: EXISTS and NOT EXISTS operators are generally faster than IN and NOT IN operators because they stop evaluating as soon as match is found.

2.Use **correlated subqueries** judiciously: correlated subqueries can be inefficient as they execute once for each row in the outer query. Try to rewrite the queries using **JOINs** or other technique to avoid correlated subqueries.

3.Use the **appropriate type of subquery**: There are different types of subqueries such as : scalar subqueries, single subqueries, multi-row subqueries. Choose the appropriate type based on the specific use case to optimize the performance.

4.**Optimize the subquery itself**: Apply general query optimization technique to subquery, such as **indexing** the relevant columns , using **appropriate joins** conditions, and **avoiding unnecessary calculation** and **aggregations**.

5.use **temporary table** or **derived tables**: Instead of using subqueries multiple times within a query, consider using temporary tables or derived tables to store the results of the subquery and then join them with the main query. This can improve the performance by reducing the number of times the subquery needs to be executed.

6.Use **appropriate indexing**: Ensure that the relevant columns in the subquery are properly indexed to speed up the execution. Analyze the execution plan and consider adding indexes on the join columns or the columns used in subquery’s WHERE clause.

7.**Limit the result set**: If possible, limit the result set of the subquery by adding appropriate conditions to reduce the number of rows processed and improve performance.

8.**Review** and **optimize** the main query: Sometimes, the performance issue may not be directly related to the subquery itself but to the overall query structure. Review and optimize the main query to ensure the efficient execution.

9.Consider using **alternative approaches**: In some cases it may be beneficial to rewrite the query, using alternative approaches like using temporary tables, common Table Expressions (CTE) , or window functions, depending on the specific requirements and database capabilities.

It’s important to note that the effectiveness of these optimization techniques may vary depending on the specific database system, schema design and the complexity of the query. It’s recommended to analyze the query execution plan and perform benchmarking to determine the most effective optimization strategy.

13. What are temporary tables in MySQL? Give examples.

Temporary tables in MySQL are tables that created and exist only for the duration of the database session. They are useful for storing intermediate results or temporary data that is needed for the specific query or set of queries.

Here are a few examples of using temporary tables in MySQL:

**1.Creating a temporary table from a SELECT statement.**

CREATE TEMPORARY TABLE temp\_table

SELECT column1, column2

FROM original\_table

WHERE condition;

This will create a temporary table called “temp table” with the columns “column1’ and “column2”, populated with the selected data from “original table” based on the specified condition.

**2.Joining temporary tables**

SELECT \* FROM temp\_table1

JOIN temp\_table2 ON temp\_table1.column1 = temp\_table2.column3;

**3. Updating data using a temporary table.**

UPDATE original\_table

JOIN temp\_table ON original\_table.id = temp\_table.id

SET original\_table.column1 = temp\_table.column1, original\_table.column2 = temp\_table.column2;

NOTE: temporary table automatically deleted and dropped when the database session ends or when they are explicitly dropped using the DROP TABLE command.

14. What is a CTE, and why might you use it in SQL. Give some examples.

A CTE (Common Table Expression) is a temporary named result set that can be used within a SQL query. It allows you to define a query block and use it as it were a regular table or view within the same query.

There are several reasons you might use a CTE in SQL:

1.**Code readability**: CTE’s help to improve the readability of complex queries by breaking them down into smaller, manageable parts , more manageable parts. This makes it easier to understand and maintain the code.

2.**Code reusability**: CTEs allow you to define a query block once and reuse it multiple times within the same query or in different queries. This can help reduce code duplication and improve code maintainability.

3.**Recursive Queries**: CTEs are commonly used for writing recursive queries, where a query refers to its own output. This is particularly useful for querying hierarchical or tree-like data structures, such as organizational charts or bills of materials.

4.**Subquery simplifications**: CTEs can simplify complex subqueries by separating them into separate query blocks. This can make the main query more readable and easier to understand.

Example:

CTE in SQL to retrieve employee hierarchy in an organization.

WITH RecursiveCTE AS (

SELECT employee\_id, employee\_name, manager\_id, 0 AS level

FROM employees

WHERE manager\_id IS NULL – starting point of the hierarchy

UNION ALL

SELECT e.employee\_id, e.employee\_name, e.manager\_id, c.level+1

FROM employees e

INNER JOIN RecursiveCTE c ON e.manager\_id = c.employee\_id )

SELECT employee\_id, employee\_name, level

FROM RecuriveCTE

ORDER BY level, employee\_id;

In this example, the CTE named “RecursiveCTE” is used to recursively retrieve the employee hierarchy. The initial query selects the top-level employees (those without the manager ), and then the recursive part joins the CTE with the employees table to retrieve the next level of employees. The final query selects the employee\_id, employee\_name, and level from the CTE, ordering the results by the hierarchy level and employee\_id.

15. What are the window functions, and why are they useful?

Window functions are a type of function in SQL that allows the calculation of values across a set of rows that are related to current row. They provide a way to perform calculations or aggregations on a specific subset of rows within a larger result set.

Window functions are useful because they provide more flexibility and power in data analysis and querying. They allow for the calculation of running totals, moving averages , rank or row number assignments and other complex calculations that would otherwise be difficult to achieve using standard SQL functions.

Some key benefits of window functions are:

1.**Partitioning**: Window functions can be partitioned based on the specific criteria, allowing for calculations to be performed within distinct groups or subsets of data.

2.**Ordering**: Data can be ordered within the window function, enabling calculations based on specific sequence or patterns.

3.**Aggregation**: Window functions can be used to calculate aggregations or summaries of the data within a window without the need of subqueries or additional joins.

4.**Efficiency**: Window functions can often be more efficient than alternative approaches, as they can avoid repetitive calculations and optimize resource usage.

Overall, window functions provide a powerful and efficient way to perform complex calculations and analysis on subsets of data within SQL queries, making them a valuable tool for data manipulation and reporting.

16.What are some best practices for writing efficient and maintainable SQL queries?

1.**Use appropriate index** : Ensure that the tables being queried have appropriate indexes set up. Indexes can significantly improve performance by allowing the database to quickly locate the required data.

2.**Avoid using SELECT\***: Instead of selecting all columns from a table , explicitly specify the required columns. This reduces unnecessary data transfer and improves query performance.

3.**Use joins wisely**: Avoid using unnecessary or excessive JOIN operations. Only JOIN the tables that are required for the query and use appropriate join type (INNER JOIN, LEFT JOIN, etc) based on the data relationship.

4.**Use subqueries and derived tables**: break down the complex queries into smaller, manageable parts using the subqueries by reducing the amount of data being processed at once.

5.**Avoid using correlated subqueries**: correlated subqueries can be efficient as they are executed for each row of the outer query. Instead, try to rewrite them as JOINs or use another optimization techniques.