SQL Questions:

1. What is RDBMS?

* A relational database is the most commonly used database. It contains several tables, and each table has its primary key.
* SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL, and Microsoft Access are based on RDBMS.
* Due to collection of organized set of tables and rows it can be easily accessed .
* A table is the simplest example of data stored in RDBMS.
* All attributes in a relation are atomic, i.e., each cell of a relation contains exactly one value.
* Relation does not contain duplicate rows/tuples.

1. What is a schema?

* A database schema is the logical representation of a database, which shows how the data is stored logically in the entire database. It contains list of attributes and instruction that informs the database engine that how the data is organized and how the elements are related to each other.
* A database schema contains schema objects that may include **tables, fields, packages, views, relationships, primary key, foreign key,**
* It gives relationships between the tables and and show how they are related to to each other.

1. List the different types of relationships in SQL.

* One to One relationship: It is used to create a relationship between two tables in which a single row of the first table can only be related to one and only one records of a second table.(ex Customer table and order table )
* One to many or many to one relationship: Any single rows of the first table can be related to one or more rows of the second tables, but the rows of second tables can only relate to the only row in the first table. It is also known as a **many to one** relationship. Ex – person table and games table , a person can play many games
* Many to many relationships: Each record of the first table can relate to any records (or no records) in the second table. Similarly, each record of the second table can also relate to more than one record of the first table. It is also represented an **N:N** relationship. Ex - For example, there are many people involved in each project, and every person can involve more than one project.

1. What are tables and Fields?

* A table is a set of data that are organized in a model with Columns and Rows. A table is a set of data that are organized in a model with Columns and Rows.
* Columns are known as fields.
* Rows are called record.

1. What is a primary key?

* A primary key is a column or a set of columns that uniquely identifies each row in a table. It can not be null or cannot be duplicate .
* A foreign key is a column or a set of columns that references the primary key of another table. The purpose of a foreign key is to establish a relationship between two tables and ensure data integrity.

1. What is a unique key?

* A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.
* It is similar like primary key to identify a unique row, but it can contain only one null value.
* It cannot have duplicate values.
* A table can have more than one unique key.
* Unique keys can be referenced by foreign keys in other tables.

1. What is a foreign key?

* A foreign key is a column or a set of columns that references the primary key of another table. The purpose of a foreign key is to establish a relationship between two tables and ensure data integrity.
* A foreign key ensures **referential integrity**, meaning it maintains the consistency and validity of data across related tables.
* **Data Consistency**: It prevents actions that would break the connection between tables, such as deleting a record with associated child records.
* The table containing the foreign key is called the **child table**.
* FOREIGN KEY (PersonID) REFERENCES Persons (PersonID)
* it may have null values

Imagine an e-commerce platform where customers place orders. The database schema might include the following tables:

Orders: Contains order details (order ID, date, total amount, etc.).

Customers: Stores customer information (customer ID, name, address, etc.).

The foreign key here would be the customer ID in the Orders table, which references the customer ID in the Customers table. This ensures that each order is associated with a valid customer.

Employee Management System:

In an HR system, you have an Employees table and a Departments table.

Employees: Contains employee details (employee ID, name, department ID, etc.).

Departments: Stores department information (department ID, name, manager ID, etc.).

The foreign key would be the department ID in the Employees table, linking it to the corresponding department in the Departments table.

1. What is the difference between CHAR and VARCHAR2 datatype in SQL?

The CHAR datatype stores character strings of a fixed length specified during column creation.

The VARCHAR datatype stores character strings of variable length, up to a maximum specified length.

1. What is a constraint?

* Constraints in SQL means we are applying certain conditions or restrictions on the database. This further means that before inserting data into the database, we are checking for some conditions. If the condition we have applied to the database holds true for the data which is to be inserted, then only the data will be inserted into the database tables.
* **NOT NULL:** Ensures that a column cannot have a NULL value.
* **UNIQUE:** Ensures that all values in a column are different.
* **PRIMARY KEY:** Combines NOT NULL and UNIQUE, uniquely identifying each row in a table.
* **FOREIGN KEY:** Prevents actions that would break links between tables.
* **CHECK:** Ensures that column values satisfy specific conditions.
* **DEFAULT:** Sets a default value for a column if no value is specified.
* **CREATE INDEX:** Used to create and retrieve data from the database quickly.

1. Explain the constraints available in SQL?

There are total 7 constraints in SQL which are NOT NULL , UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK , DEFAULT, CREATE INDEX

1. Difference between normalization and de normalization

Normalization is a technique used to reduce data redundancy and improve data consistency within a database. **normalization focuses on maintaining data consistency and reducing redundancy.**

It involves dividing a database into multiple related tables (relations) to achieve these goals. Number of **tables increases.**

* Normalization ensures that data is stored in a set schema without unnecessary duplication.
* It prioritizes data integrity over query performance.

Denormalization aims to improve query performance by combining data from multiple tables into a single table**. denormalization prioritizes query speed by allowing some redundancy.**

* Method: It adds redundancy to the database schema.
* Denormalization allows for faster execution of queries because related data is already combined.

Reduced Tables: Unlike normalization, denormalization reduces the number of tables.

* It’s useful for scenarios where quick data retrieval is crucial.

1. Types of normalization ?

* 1NF- A relation is in 1NF if it contains atomic values (no repeating groups or arrays). It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* 2NF- 1 NF + In the second normal form, all non-key attributes are fully functional dependent on the primary key. Not dependent on any other key. There should not be any partial dependency .it means totally depends on primary key not the any part of primary key.
* 3NF- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency . dependency: **A → B and B → C** implies **A → C**.(this should not be in the data ). When non prime attribute is determined by non prime attribute then its called transitive dependency. It should not be there .
* BcNf- BCNF is the advance version of 3NF. LHS of functional dependency should be a candiadate key . In BCNF, every functional dependency must have a determinant that is a superkey.

The columns in a table are called fields while the rows can be referred to as records.

. **The columns in a table are called fields while the rows can be referred to as records**

1. What is SQL index ?

The Index in SQL is a special table used to speed up the searching of the data in the database tables. It also retrieves a vast amount of data from the tables frequently. It is the best SQL technique for improving the performance of queries. The drawback of using indexes is that they slow down the execution time of UPDATE and INSERT statements. But they have one advantage also as they speed up the execution time of SELECT and WHERE statements.

CREATE INDEX college\_index ON Colleges(college\_code);

* **Unique**  No two rows of data in a table have identical key values. **and Non-Unique Index:**
* **Clustered and Non-Clustered Index:** Clustered index modifies the way records are stored in a database based on the indexed column. A non-clustered index creates a separate entity within the table which references the original table.
* A table can have a single clustered index whereas it can have multiple non-clustered indexes.

1. What is data integrity?

Data Integrity is the assurance of accuracy and consistency of data over its entire life-cycle.

1. What are UNION, MINUS and INTERSECT commands?

The **UNION** operator combines and returns the result-set retrieved by two or more SELECT statements.  
The **MINUS** operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.  
The **INTERSECT** clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

1. What is cursor ?

A database cursor is a control structure that allows for the traversal of records in a database. Cursors, in addition, facilitates processing after traversal, such as retrieval, addition, and deletion of database records. They can be viewed as a pointer to one row in a set of rows.

1. What is a View?

A view in SQL is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

1. What is the difference between DELETE and TRUNCATE statements?

The **TRUNCATE** command is used to delete all the rows from the table and free the space containing the table.  
The **DELETE** command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

1. What is the difference between DROP and TRUNCATE statements?

If a table is dropped, all things associated with the tables are dropped as well. This includes - the relationships defined on the table with other tables, the integrity checks and constraints, access privileges and other grants that the table has. To create and use the table again in its original form, all these relations, checks, constraints, privileges and relationships need to be redefined. However, if a table is truncated, none of the above problems exist and the table retains its original structure

1. What is scaler function ?

A scalar function returns a single value based on the input value. Following are the widely used SQL scalar functions:

* **LEN()** - Calculates the total length of the given field (column).
* **UCASE()** - Converts a collection of string values to uppercase characters.
* **LCASE()** - Converts a collection of string values to lowercase characters.
* **MID()** - Extracts substrings from a collection of string values in a table.
* **CONCAT()** - Concatenates two or more strings.
* **RAND()** - Generates a random collection of numbers of a given length.
* **ROUND()** - Calculates the round-off integer value for a numeric field (or decimal point values).
* **NOW()** - Returns the current date & time.
* **FORMAT()** - Sets the format to display a collection of values

1. What is the main reason to add constraints to a table?

* The main reason behind adding constraints to a table is that before inserting any data we want to check some certain conditions to maintain data anomalies and referential integrity.
* By enforcing these rules, constraints prevent invalid or inconsistent data from being inserted, updated, or deleted.

1. What is the difference between primary key and unique constraints?

* The primary key will not accept NULL values whereas the Unique key can accept NULL values.
* A table can have only one primary key whereas there can be multiple unique keys on a table.
* A Clustered index is automatically created when a primary key is defined whereas a Unique key generates the [non-clustered index.](https://www.geeksforgeeks.org/sql-queries-on-clustered-and-non-clustered-indexes/)
* A Primary Key can be a Unique Key, but a Unique Key cannot be a primary key.

For a better understanding of the primary key, we take a table named Student table, having attributes such as Roll\_number, Name, Batch, Phone\_number, and Citizen\_ID.

**Roll no is primary key and citizen id is unique key .**

**Roll number attribute is already assigned with the primary key and Citizen\_ID can have unique constraints where each entry in a Citizen\_ID column should be unique because each citizen of a country must have his or her Unique identification number like an Aadhaar Number. But if the student is migrated to another country in that case, he or she would not have any Citizen\_ID and the entry could have a NULL value as only one NULL is allowed in the unique constraint.**

|  |  |
| --- | --- |
| **PRIMARY KEY** | **UNIQUE KEY** |
| Used to serve as a unique identifier for each row in a table. | Uniquely determines a row that isn’t the primary key. |
| **Cannot accept NULL values.** | **Can accept only one NULL value** |
| **Cannot accept duplicate value** | **Can not accept null value** |
| Only one primary key | **More than one unique key** |
| A Primary key supports auto-increment value. | **A unique key does not support auto-increment value.** |
| We cannot change or delete values stored in primary keys. | **We can change unique key values.** |
| The primary Key is used for indicating the rows uniquely. | The Unique Key is used for preventing duplicate entries. |
|  |  |

1. What is an index?

* In SQL, an index is a schema object that significantly enhances the efficiency of data retrieval from database tables.
* An index acts as a pointer used by the database server to swiftly locate rows.
* By providing a rapid path access method, it reduces disk I/O (input/output) during data retrieval.
* While indexes speed up SELECT queries and WHERE clauses, they can slightly slow down data insertion, updates, and deletions.
* Indexes allow databases to retrieve data very fast.
* Keep in mind that updating a table with indexes takes more time than updating a table without them.

1. -What is auto increment in sql?

* MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.
* Auto increment generate unique number automatically when a new record is inserted in the table
* It uses with primary key

1. What is the purpose of aggregate functions?

**An aggregate function performs operations on a collection of values to return a single scalar value.**

* An aggregate function in SQL performs a calculation on multiple values and returns a single value.
* SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
* It is also used to summarize the data.

1. **AVG()**
2. **COUNT()**
3. **FIRST()**
4. **LAST()**
5. **MAX()**
6. **MIN()**
7. **SUM()**
8. What is User-defined function? What are its various types?

The user-defined functions in SQL are like functions in any other programming language that accept parameters, perform complex calculations, and return a value. They are written to use the logic repetitively whenever required. There are two types of SQL user-defined functions:

* Scalar Function: As explained earlier, user-defined scalar functions return a single scalar value.
* Table-Valued Functions: User-defined table-valued functions return a table as output.
  + **Inline:** returns a table data type based on a single SELECT statement.
  + **Multi-statement:** returns a tabular result-set but, unlike inline, multiple SELECT statements can be used inside the function body.

1. Explain the aggregate functions available in SQL
2. What are the scalar functions in SQL? Give an example?

* A scalar function returns a single value based on the input value
* **These functions are based on user input, these too returns single value.**

1. **UCASE() It converts the value of a field to uppercase.**
2. **LCASE()It converts the value of a field to lowercase.**
3. **MID() The MID() function extracts texts from the text field.**
4. **LEN() The LEN() function returns the length of the value in a text field.**
5. **ROUND() The ROUND() function is used to round a numeric field to the number of decimals specified.** **NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.**
6. **NOW() The NOW() function returns the current system date and time.**
7. **FORMAT() The FORMAT() function is used to format how a field is to be displayed.**
8. What is the SELECT statement?

* it’s a Data Query Language (DQL) statement.
* it’s use to retrieve data from table .

1. What is CLAUSE?

* In SQL, a clause is a built-in function that helps retrieve specific records from a database table. Clauses allow you to filter and analyze data efficiently.
* The WHERE clause is used with the SELECT query to filter records based on specific conditions.
* It limits the number of rows displayed in the result set.
* The GROUP BY clause groups rows based on one or more columns.
* It’s often used with aggregate functions (e.g., SUM, COUNT, AVG) to summarize data. SELECT Department, AVG(Salary) FROM employees GROUP BY Department; This query calculates the average salary for each department.
* The HAVING clause filters grouped results.
* It’s similar to the WHERE clause but operates on aggregated data.
* SELECT Department, AVG(Salary) FROM employees GROUP BY Department HAVING AVG(Salary) > 60000;
* The ORDER BY clause sorts result rows based on specified columns.
* You can sort in ascending (default) or descending order.

1. What are some common clauses used with SELECT query in SQL?

The common clauses used with SELECT in SQL is WHERE , GROUP BY , ORDER BY , HAVING CALUSE .

1. OLAP v/s OLTP ?

* **OLAP** stands **for Online Analytical Processing**. OLAP systems have the capability to analyze database information of multiple systems at the current time. The primary goal of OLAP Service is data analysis and not data processing.  Spotify analyzed songs by users to come up with a personalized homepage of their songs and playlist.

Netflix movie recommendation system.

**OLTP** stands for **Online Transaction Processing**. OLTP has the work to administer day-to-day transactions in any organization. The main goal of OLTP is **data processing not data analysis.**

An example considered for OLTP System is ATM Center a person who authenticates first will receive the amount first and the condition is that the amount to be withdrawn must be present in the ATM.

1. What is Collation?

Collation refers to a set of rules that determine how data **is sorted and compared**. Rules defining the correct character sequence are used to sort the character data. A MySQL collation is a well-defined set of rules which are used to compare characters of a particular character set by using their corresponding encoding.

1. What is stored procedure ?

**Stored Procedures** are created to perform one or more DML operations on databases. It is nothing but a group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not return a value.

1. Which SQL clause is used to restrict the rows returned by a query?

The SQL clause used to restrict the rows returned by a query is the LIMIT clause. It allows you to specify the maximum number of rows to be included in the result set. The LIMIT clause is particularly useful when you want to retrieve only a subset of records from a table.

1. What are ACID properties?

* **Atomicity**: This property ensures that the transaction is completed in all-or-nothing way.
* **Consistency**: This ensures that updates made to the database is valid and follows rules and restrictions.
* **Isolation**: This property ensures integrity of transaction that are visible to all other transactions.
* **Durability**: This property ensures that the committed transactions are stored permanently in the database

1. What is the On Delete cascade constraint?

An ‘ON DELETE CASCADE’ constraint is used in MySQL to delete the rows from the child table automatically when the rows from the parent table are deleted.

1. Differentiate between commit and checkpoint.

The commit action ensures that the data consistency of the transaction is maintained and it ends the current transaction in the section. Commit adds a new record in the log that describes the COMMIT to the memory. Whereas, a checkpoint is used for writing all changes that were committed to disk up to SCN which would be kept in datafile headers and control files.

1. What is the difference between Cluster and Non-Cluster Index?

| **CLUSTERED INDEX** | **NON-CLUSTERED INDEX** |
| --- | --- |
| The clustered index is faster. | The non-clustered index is slower. |
| The clustered index requires less memory for operations. | The non-Clustered index requires more memory for operations. |
| In a clustered index, the index is the main data. | In the Non-Clustered index, the index is a copy of data. |
| A table can have only one clustered index. | A table can have multiple non-clustered indexes. |
| The clustered index has an inherent ability to store data on the disk. | The non-Clustered index does not have the inherent ability to store data on the disk. |

**Why do we use  Commit and Rollback commands?**

| **COMMIT** | **ROLLBACK** |
| --- | --- |
| COMMIT permanently saves the changes made by the current transaction. | ROLLBACK undo the changes made by the current transaction. |
| The transaction can not undo changes after COMMIT execution. | Transaction reaches its previous state after ROLLBACK. |
| When the transaction is successful, COMMIT is applied. | When the transaction is aborted, ROLLBACK occurs. |

1. What is the difference between COALESCE() & ISNULL()?

**COALESCE():**COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.  
**Syntax:**

*SELECT column(s), CAOLESCE(expression\_1,….,expression\_n)FROM table\_name;*

**ISNULL():**The ISNULL function has different uses in SQL Server and MySQL. In SQL Server, ISNULL() function is used to replace NULL values.

1. Which clause should you use to exclude group results?

* The HAVING clause filters grouped results after aggregation.
* It operates on aggregated data (e.g., using functions like SUM, COUNT, AVG).
* Unlike the WHERE clause, which filters individual rows before grouping, HAVING filters the grouped results.
* It’s commonly used with the GROUP BY clause.

1. Explain how GroupBy is used in query?

* All column names from the SELECT clause should either appear in the GROUP BY clause or be used in aggregate functions (e.g., COUNT(), SUM(), AVG()).
* Group by clause group the result based on the column
* In other words, every non-aggregated column in your SELECT statement must be present in the GROUP BY clause.

First, apply any filtering conditions using the WHERE clause.

Then, group the filtered data using the GROUP BY clause.

Finally, apply any additional filtering on the grouped data using the HAVING clause (if needed).

1. How “Having” clause will work?

Having clause filter the grouped result .What is the relationship between Having & GroupBy clause

1. Explain how OrderBy is used in query?

* The ORDER BY clause sorts result rows based on specified columns.
* You can sort in ascending (default) or descending order.
* By default, the ORDER BY clause sorts the data in ascending order.
* You can specify one or more columns by which to sort the result set.
* SELECT \* FROM Customers
* ORDER BY Country, CustomerName;

1. Which operator is used in query for pattern matching? Explaing with examples

LIKE operator is used in query for pattern matching.

SELECT \* FROM employee\_details WHERE Name LIKE 'Pr%';

1. What is the difference between BETWEEN and IN operators in SQL?

The BETWEEN operator is used to test if an expression falls within a specified range of values (inclusive).

It can be applied to text, date, or numeric data.

SELECT Name FROM Emp WHERE Age **BETWEEN** 22 AND 24;

**The IN** operator checks if an expression matches any value in a specified list.

It’s useful for simplifying queries with multiple OR conditions.

SELECT Name FROM Emp WHERE Salary **IN** (30000, 40000, 25000);

1. Are NULL values same as blank space?

In SQL, NULL, blank spaces, and zero have distinct meanings:

* NULL:
* Represents the absence of a value.
* It is not allocated any memory.
* You can’t perform arithmetic or comparisons directly with NULL.
* Example: If a field has not been set (e.g., no name given), it is NULL.
* Blank Spaces (Empty Strings):
* Blank spaces are strings with no characters (e.g., '').
* They occupy memory space.
* You can manipulate and compare them as strings.
* Example: An empty string represents a field with no visible characters.
* Zero:
* Zero is a numeric value (e.g., 0).
* It has precise mathematical properties.
* You can perform arithmetic operations on it.
* Example: A field with a value of zero (e.g., age = 0).

1. What is a join?

* In SQL, the JOIN operation allows you to combine data from two or more tables based on a related column.
* The INNER JOIN (or simply JOIN) retrieves records that have matching values in both tables.
* The LEFT JOIN returns all records from the left table and the matched records from the right table.
* The RIGHT JOIN is similar to the LEFT JOIN, but it returns all records from the right table and the matched records from the left table.
* The FULL JOIN returns all records when there is a match in either the left or right table.
* **SELF JOIN :**
* a self join operates on a single table. It’s particularly useful when you need to compare records within the same table based on specific conditions. Imagine we have a table called “GFGemployees” with columns like employee\_id, employee\_name, and manager\_id.
* Each employee in the company is assigned a manager, and the manager\_id column contains the manager ID for each employee.
* We want to extract a list of employees along with the names of their managers.
* **LETS SEE THE EXAMPLE**
* SELECT e.employee\_name AS employee, m.employee\_name AS manager
* FROM GFGemployees AS e
* JOIN GFGemployees AS m ON e.manager\_id = m.employee\_id;

SELECT Customers.CustomerName, Orders.OrderID, Orders.OrderDate

FROM Customers

FULL JOIN Orders

ON Customers.CustomerID = Orders.CustomerID;

1. What is Inner Join?

* The INNER JOIN (or simply JOIN) retrieves records that have matching values in both tables.
* It combines rows from two tables based on a common column (usually a foreign key).

SELECT Products.ProductID, Products.ProductName, Categories.CategoryName

FROM Products

JOIN Categories

ON Products.CategoryID = Categories.CategoryID;

1. How “Outer Join” works?

* **Left outer join:** Retrieve all the record from the left table and common/matched record from right table .it ensures all the rows from the left table are included.
* **Right outer join:** Retrieve all the record/rows from right table and matched record from
* **Full outer join :** In a full outer join, all rows from both tables are included.
* If there are any unmatched rows, they appear with NULL values.

1. What is Full Join?

* Full outer join : In a full outer join, all rows from both tables are included.
* If there are any unmatched rows, they appear with NULL values.
* Also called full outer join .
* Suppose we have two tables: Students and Comments. These tables contain information about students participating in an online forum and the comments they’ve posted. We want to retrieve all student records along with their associated comments, even if some students haven’t posted any comments or vice versa.
* **SELECT column\_list**
* **FROM table1**
* **FULL OUTER JOIN table2 ON table1.column = table2.column;-**

1. What is Self-Join? In what situation you will use Self Join?

* A self join operates on a single table. It’s particularly useful when you need to compare records within the same table based on specific conditions.
* Self joins are commonly used for hierarchical data structures.
* Examples include organizational charts, family trees, or product categories with parent-child relationships.
* By joining a table to itself, you can navigate through levels of hierarchy and retrieve related information.
* Imagine we have a table called “GFGemployees” with columns like employee\_id, employee\_name, and manager\_id.
* Each employee in the company is assigned a manager, and the manager\_id column contains the manager ID for each employee.
* We want to extract a list of employees along with the names of their managers. Then we can use self join .

1. What is a Natural Join?

* A **natural join** in SQL is an operation that combines data from two tables based on the common columns between them. Here are the key points about natural joins:
* Common Attribute: To perform a natural join, there must be at least one common attribute (column) between the two tables.
* SELECT \*
* FROM Employee
* **NATURAL JOIN** Department;

When you have two tables with related data, and they share a common attribute (column), a natural join simplifies combining them.

Example: Suppose you have a Customers table and an Orders table. Both tables have a common customer\_id column. A natural join would help merge customer details with their corresponding orders.

1. What is subquery?

* A subquery is a query nested inside another query.
* It can appear in various SQL clauses, such as the WHERE, HAVING, or FROM clause.
* Subqueries are used to retrieve data dynamically based on the results of another query.
* Subqueries provide a way to use dynamic values (results from other queries) in your main query.
* Example: **Retrieve all orders where the order amount exceeds the average order** amount.

SELECT \*

FROM Orders

WHERE amount > (SELECT AVG(amount) FROM Orders);

* **These subqueries reference columns from the outer query.**
* Example: Find employees whose salary is higher than the average salary in their department.
* SELECT emp\_name
* FROM Employees e
* WHERE salary > (SELECT AVG(salary) FROM Employees WHERE department\_id = e.department\_id);
* **Subqueries in EXISTS and NOT EXISTS Clauses**
* Example: Check if any orders exist for a specific customer.
* SELECT customer\_name
* FROM Customers c
* WHERE EXISTS (SELECT 1 FROM Orders WHERE customer\_id = c.customer\_id);

1. What are the types of subquery?

* **A scalar subquery returns a single value (one row and one column).**
* **Multiple row subqueries**
* **Multiple column subqueries**
* **Correlated subqueries**
* **Non correlated**
* **Nested subqueries**
* Subqueries can return either scalar (single) values or a row set, while joins always return rows.
* Commonly used in the WHERE clause to filter results based on dynamic conditions.

SELECT \*

FROM Sales

WHERE revenue > (SELECT AVG(revenue) FROM Sales);

**Here, the subquery calculates the average revenue, and the main query filters sales records with revenue greater than the average.**

* These subqueries return multiple rows but only one column.
* Used in scenarios where you need to compare a set of values.

SELECT product\_name

FROM Products

WHERE category\_id IN (SELECT category\_id FROM Categories WHERE category\_name = 'Electronics');

**This query retrieves product names from the “Electronics” category.**

These subqueries return multiple rows and columns.

Often used in the SELECT clause to provide additional data.

SELECT customer\_name,

(SELECT COUNT(\*) FROM Orders WHERE Customers.customer\_id = Orders.customer\_id) AS order\_count

FROM Customers;

**Here, the subquery calculates the order count for each customer.**

1. What are similarities between Subqueries & Joins?

similarities between subqueries and joins in SQL:

Combining Data:

Both subqueries and joins allow you to combine data from multiple tables into a single result set.

Subqueries achieve this by nesting one query inside another, whereas joins directly connect tables based on specified conditions.

Data Selection:

In both cases, you can select specific columns or all columns from the involved tables.

Subqueries can return either scalar (single) values or a row set, while joins always return rows.

Complex Queries:

Both subqueries and joins are used in complex queries to retrieve data from multiple tables.

However, they do so in different ways.

Efficiency and Readability:

Joins are often more efficient than subqueries, especially when dealing with large datasets.

The retrieval time of a query using joins is usually faster than that of a subquery.

Additionally, as queries become more complex, joins tend to be easier to read and understand.

Use Cases:

**Subqueries are useful when you need to filter data based on conditions from another table or aggregate results.**

**Joins are commonly used to connect tables using primary and foreign keys.**

1. What are differences between Subqueries & Joins?

|  |  |
| --- | --- |
| Subqueries | Joins |
| Used to retrieve data that will be used as a condition in the main query. | Combine data from multiple tables into a single result set. |
| Nested within the main query (e.g., in the WHERE, HAVING, or FROM clause). | Explicitly specified in the FROM clause of the query. |
| Can return either scalar values or a row set. | Always returns rows. |
| Useful for filtering data based on conditions from another table or aggregating results. | Commonly used to connect tables using primary and foreign keys. |
| Cannot avoid joins when retrieving data from a normalized database. | Essential for retrieving data from related tables. |

1. What is the difference between DELETE and TRUNCATE commands?

TRUNCATE TABLE table\_name; DELETE FROM table\_name WHERE condition;

|  |  |
| --- | --- |
| **DELETE** | **TRUNCATE** |
| **Definition** | **DELETE**  The delete statement is used to remove single or multiple records from an existing table depending on the specified condition. | **TRUNCATE**  The truncate command removes the complete data from an existing table but not the table itself. It preserves the table structure or schema. |
| **Language** | It is a DML (Data Manipulation Language) command. | It is a DDL (Data Definition Language) command. |
| **WHERE** | It can use the WHERE clause to filter any specific row or data from the table. | It does not use the WHERE clause to filter records from the table. |
| **Permission** | We need to have DELETE permission to use this command. | We need to have ALTER permission to use this command. |
| **Working** | This command eliminates records one by one. | This command deletes the entire data page containing the records. |
| **Lock** | It will lock the row before deletion. | It will lock the data page before deletion. |
| **Table Identity** | This command does not reset the table identity because it only deletes the data. | It always resets the table identity. |
| **Transaction** | It maintains transaction logs for each deleted record. | It does not maintain transaction logs for each deleted data page. |
| **Speed** | Its speed is slow because it maintained the log. | Its execution is fast because it deleted entire data at a time without maintaining transaction logs. |
| **Trigger** | This command can also activate the trigger applied on the table and causes them to fire. | This command does not activate the triggers applied on the table to fire. |
| **Restore** | It allows us to restore the deleted data by using the COMMIT or ROLLBACK statement. | We cannot restore the deleted data after using executing this command. |
| **Indexed view** | It can be used with indexed views. | It cannot be used with indexed views. |
| **Space** | The DELETE statement occupies more transaction space than truncate because it maintains a log for each deleted row. | The TRUNCATE statement occupies less transaction space because it maintains a transaction log for the entire data page instead of each row. |

1. What is the difference between TRUNCATE and DROP statements?

|  |  |
| --- | --- |
| **DROP** | **TRUNCATE** |
| The DROP command in SQL removes the table definition and its data. | The TRUNCATE command in SQL deletes all data from the table. |
| This query frees the tablespace from the memory. | The TRUNCATE query does not free the tablespace from the storage. |
| The view of the table does not exist in the DROP command. | View of the table exists in the Truncate command. |
| The integrity constraints will be automatically removed from the table in the DROP command | The integrity constraints in this command will not be removed. |
| In the DROP query, deleted space is not used. | The deleted space is used but less than the DELETE statement. |
| The DROP query deletes data quickly, but there are so many complications. | The TRUNCATE query in SQL is faster than the DROP query. |

1. What are Entities and Relationships?

* An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
* Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.
* A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.
* When only one instance of an entity is associated with the relationship, then it is known **as one to one** relationship.For example, A female can marry to one male, and a male can marry to one female.
* When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a **one-to-many** relationship.For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.
* When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a **many-to-one relationship.**
* **For example,** Student enrolls for only one course, but a course can have many students.

1. What is data Integrity?

Data integrity is defined as the data contained in the database is both correct and consistent. For this purpose, the data stored in the database must satisfy certain types of procedures (rules). The data in a database must be correct and consistent. DBMS provides different ways to implement such types of constraints (rules). It can be implemented by rules i.e., **Primary Key, Secondary Key, Foreign key. This improves data integrity in a database.**

Its work is to check the data is correct and not corrupt.

It can be implemented by following rule :

It avoids human error when data is entered.

Some of the means to preserve integrity are backing up, error detection, designing a suitable user interface and correcting data.

Primary Key, Foreign Key ,Relationship

**Entity Integrity:** There are columns, rows, and tables in a database. These elements should be as numerous as possible for the data to be correct in a primary key, but not more than that. A database of workers, for example, should have primary key data such as their name and a unique "employee number."

**Referential Integrity:** A foreign key table in a database is a second table that may refer to a database's primary key table. Foreign keys are used to connect data that is either shared or null. Employees can, for example, have the same job title or work in the same department.

**Domain Integrity:** In a database, all categories and values are set, including nulls (e.g., N/A). The common ways to input and read data in a database are referred to as domain integrity. Three decimal places would not be allowed in a database that uses monetary values such as dollars and cents.

1. What is de-normalization, and when do you go for it?

Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean ‘reversing normalization’ or ‘not to normalize’. It is an optimization technique that is applied after normalization.

Retrieving data is faster since we do fewer joins

Queries to retrieve can be simpler(and therefore less likely to have bugs),

since we need to look at fewer tables.

**Cons of Denormalization:**

Updates and inserts are more expensive.

Denormalization can make update and insert code harder to write.

Data may be inconsistent.

Data redundancy necessitates more storage.

1. What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

CREATE PROCEDURE procedure\_name  
AS  
sql\_statement  
GO;

EXEC procedure\_name;

CREATE TABLE Persons (

PersonID int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

INSERT INTO Persons (PersonID, LastName, FirstName, Address, City)

VALUES

(1, 'Smith', 'John', '123 Main St', 'New York'),

(2, 'Johnson', 'Emily', '456 Elm Ave', 'Los Angeles'),

(3, 'Lee', 'David', '789 Oak Rd', 'Chicago');

1. There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL. CREATE

commands that come under DDL:

CREATE

ALTER

DROP

TRUNCATE

some commands that come under DML:

INSERT

UPDATE

DELETE

some commands that come under DCL:

Grant

Revoke

some commands that come under TCL:

COMMIT

ROLLBACK

SAVEPOINT

DQL is used to fetch the data from the database.

It uses only one command: SELECT

 While it’s common to join tables based on a shared column, there are scenarios where you might need to combine tables **without a common column**. Let’s explore a couple of ways to achieve this:

1. **CROSS JOIN (Cartesian Product)**:
   * A **CROSS JOIN** produces the **Cartesian product** of two tables, meaning it combines every row from the first table with every row from the second table.
   * It doesn’t require any common column.
   * The resulting table will have **all possible combinations** of rows.
   * Example:

**SQL**

SELECT \*

FROM Table1

CROSS JOIN Table2;

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. **UNION** (Not a true join, but useful for merging):
   * **UNION** combines the result sets of two SELECT queries into a single result set.
   * It stacks rows on top of each other, even if the columns don’t match.
   * Example:

**SQL**

SELECT column1, column2 FROM Table1

UNION

SELECT column3, column4 FROM Table2;

* + In this example, we’re merging columns from Table1 and Table2 without a common column.

Scaler

* Scalar functions take one or more parameters and return a single value.
* They help simplify code by encapsulating complex calculations or formulas.
* Instead of repeating the same formula in multiple queries, you can create a scalar function and use it consistently.

CREATE FUNCTION sales.udfNetSale (

@quantity INT,

@list\_price DEC(10, 2),

@discount DEC(4, 2)

)

RETURNS DEC(10, 2)

AS

BEGIN

RETURN @quantity \* @list\_price \* (1 - @discount);

END;

1. **What Is an Index in SQL?**

* An **index** in SQL is like a **bookmark** for your database table.
* It helps speed up **searches** and **queries** by organizing data in a way that makes it easier to find specific rows.
* Imagine a library with thousands of books. Without an index, finding a specific book would be slow.
* Similarly, indexes help databases quickly locate relevant data.
* Suppose we have an Employee table with columns like Emp\_ID, Emp\_Name, and Emp\_State.
* To make searching faster, we can create an index on the Emp\_State column:
* CREATE INDEX index\_state ON Employee (Emp\_State);

1. **What is a window function?**

A **window function** performs calculations across a set of rows related to the current row.

Examples: RANK(), ROW\_NUMBER(), SUM().

1. **How to find duplicate records in a table?**

Using GROUP BY and HAVING:

SELECT Column1, COUNT(\*)

FROM MyTable

GROUP BY Column1

HAVING COUNT(\*) > 1;

1. **Operators in sql**

Arithmetic Operators: Addition substarction, division, multiplication and modulo.

Bitwise Operators: Bitwise AND, BitwiseOR,Bitwise exclusive OR .

Comparison Operators:

Equal to (=): Checks if two values are equal.

Greater than (>), Less than (<): Compare numeric or string values.

Greater than or equal to (>=), Less than or equal to (<=): Check for equality or inequality.

Not equal to (<> or !=): Tests if two values are not equal.

Compound Operators:

Add equals (+=): Adds a value to an existing one and assigns the result.

Subtract equals (-=): Subtracts a value and assigns the result.

Multiply equals (\*=): Multiplies and assigns.

Divide equals (/=): Divides and assigns.

Modulo equals (%=): Calculates the remainder and assigns.

Logical opearators :

AND: Returns true if all conditions are true.

OR: Returns true if any condition is true.

NOT: Negates a condition.

BETWEEN: Checks if a value falls within a specified range.

LIKE: Tests if a value matches a pattern (using wildcards).

IN: Checks if a value is in a list of expressions.

EXISTS: Determines if a subquery returns any records.

String operators :

Concatenation (||): Joins strings together.

LIKE: Used for pattern matching.

SUBSTRING(): Extracts a portion of a string.

LENGTH(): Calculates the length of a string.

1. What is pseudo class?

Definition: In CSS (Cascading Style Sheets), a pseudo-class is a selector that targets elements based on specific states or interactions.

Usage:

Hover State: A common use case is styling an element when a user hovers over it. For example, changing the color of a link when the mouse pointer hovers over it.

Visited Links: Pseudo-classes can also differentiate between visited and unvisited links.

Focus State: When an element receives focus (e.g., when a user clicks on an input field), you can apply specific styles.

selector:pseudo-class {

property: value;

}

1. What is DOM?
2. **What is the DOM?**:
   * The DOM represents the **structural pieces of a web document** as objects that can be accessed and manipulated.
   * In other words, the DOM allows you, as a software developer, to:
     + **Create and build web documents**.
     + **Navigate the structure of web documents**.
     + [**Add, modify, or delete elements and content within web documents**1](https://www.freecodecamp.org/news/what-is-the-dom-explained-in-plain-english/).
3. **Understanding the DOM**:
   * When an HTML file is loaded into a browser, the browser **automatically creates the DOM** by parsing the HTML document.
   * The process involves converting the HTML document into a **tree-like structure** composed of objects.
   * These objects represent various elements (such as headings, paragraphs, images, forms) and their relationships within the document.
   * JavaScript interacts with this tree-like structure, allowing you to manipulate the content, structure, and style of the webpage dynamically.
4. **Why is DOM Required?**:
   * HTML structures web pages, while JavaScript adds behavior to them.
   * JavaScript cannot directly understand the raw HTML document. Instead, it interacts with the DOM, which acts as a bridge.
   * The DOM provides a way for JavaScript to access and manipulate individual elements (like <h1>, <p>, etc.) within the HTML document.
   * It enables dynamic web experiences, such as updating content without reloading the entire page or responding instantly to user actions.
5. **Handling the DOM**:
   * To work with the DOM, follow these steps:
     + **Access Elements**:
       - Use methods like getElementById(), getElementsByClassName(), or querySelector() to select specific elements.
     + **Modify Content**:
       - Change text, attributes, or HTML structure using properties like textContent, innerHTML, or setAttribute().
     + **Create New Elements**:
       - Use methods like createElement() to create new elements (e.g., adding a new paragraph).
     + **Append and Remove Elements**:
       - Add elements to the DOM tree using appendChild() or remove them using removeChild().
     + **Respond to Events**:
       - Attach event listeners to elements (e.g., handling button clicks).
     + **Style Elements**:
       - Change CSS properties using style.
     + **Traverse the DOM Tree**:
       - Move up or down the tree to access parent, child, or sibling elements.
6. <head>
7. <link rel="stylesheet" href="style.css">
8. </head>
9. <head>
10. <title>My Document</title>
11. <script src="myscripts.js"></script>
12. </head>
13. **Choosing Between JSON and XML**:
    * **Use JSON When**:
      + Data interchange between web clients and servers.
      + Simplicity and ease of use are essential.
      + No need for extensive metadata or document markup.
    * **Use XML When**:
      + Document markup and metadata are crucial.
      + Compatibility with legacy systems or specific standards (e.g., SOAP).
      + Human readability is more important than compactness.

**JSON (JavaScript Object Notation)**:

* **Purpose**:
  + JSON is primarily used for **structured data interchange** between a client (usually a web browser) and a server (usually a web service).
  + It is widely used in web development due to its simplicity and compatibility with JavaScript.
* **Common Scenarios**:
  + **Web APIs (RESTful APIs)**: JSON is the dominant format for exchanging data between clients (web browsers, mobile apps) and servers.
  + **Front-End Development**: JavaScript applications often consume JSON data from APIs to display dynamic content.
  + **Configuration Files**: JSON is used for configuration settings in applications.

**SQL String functions** .

SQL provides a variety of **string functions** that allow you to manipulate and work with text data. Here are some commonly used string functions:

* ASCII: Returns the ASCII value for a specific character.
* CHAR: Returns the character based on the ASCII code.
* CHARINDEX: Finds the position of a substring within a string.
* CONCAT: Combines two or more strings together.
* CONCAT\_WS: Adds two or more strings together with a separator.
* DATALENGTH: Returns the number of bytes used to represent an expression.
* LEFT: Extracts a specified number of characters from the left side of a string.
* LEN: Returns the length of a string.
* LOWER: Converts a string to lowercase.
* LTRIM: Removes leading spaces from a string.
* REPLACE: Replaces occurrences of a substring within a string with a new substring.
* RIGHT: Extracts a specified number of characters from the right side of a string.
* RTRIM: Removes trailing spaces from a string.
* SUBSTRING: Extracts a portion of a string.
* UPPER: Converts a string to uppercase.

### ****Q1. Write a query to fetch the EmpFname from the EmployeeInfo table in upper case and use the ALIAS name as EmpName.****

|  |  |
| --- | --- |
| 1 | **SELECT** UPPER(EmpFname) **AS** EmpName **FROM** EmployeeInfo |

### ****Q2. Write a query to fetch the number of employees working in the department ‘HR’.****

|  |  |
| --- | --- |
| 1 | **SELECT** COUNT(\*) **FROM** EmployeeInfo **WHERE** Department = 'HR'; |

### ****Q3. Write a query to get the current date.****

You can write a query as follows in SQL Server:

|  |  |
| --- | --- |
| 1 | **SELECT** GETDATE(); |

You can write a query as follows in [MySQL](https://www.edureka.co/blog/mysql-tutorial/):

|  |  |
| --- | --- |
| 1 | **SELECT** SYSTDATE(); |

### ****Q4. Write a query to retrieve the first four characters of  EmpLname from the EmployeeInfo table.****

|  |  |
| --- | --- |
| 1 | **SELECT** SUBSTRING(EmpLname, 1, 4) **FROM** EmployeeInfo; |

### ****Q5. Write a query to fetch only the place name(string before brackets) from the Address column of EmployeeInfo table.****

Using the MID function in [MySQL](https://www.edureka.co/blog/what-is-mysql/)

|  |  |
| --- | --- |
| 1 | **SELECT** MID(Address, 0, LOCATE('(',Address)) **FROM** EmployeeInfo; |

Using SUBSTRING

|  |  |
| --- | --- |
| 1 | **SELECT** SUBSTRING(Address, 1, CHARINDEX('(',Address)) **FROM** EmployeeInfo; |

### ****Q6. Write a query to create a new table which consists of data and structure copied from the other table.****

Using the SELECT INTO command:

|  |  |
| --- | --- |
| 1 | **SELECT** \* **INTO** NewTable **FROM** EmployeeInfo **WHERE** 1 = 0; |

Using the [CREATE command](https://www.edureka.co/blog/create-table-in-sql/) in MySQL:

|  |  |
| --- | --- |
| 1 | **CREATE** **TABLE** NewTable **AS** **SELECT** \* **FROM** EmployeeInfo; |

### ****Q7. Write q query to find all the employees whose salary is between 50000 to 100000.****

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeePosition **WHERE** Salary BETWEEN '50000' AND '100000'; |

### ****Q8. Write a query to find the names of employees that begin with ‘S’****

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** EmpFname LIKE 'S%'; |

### ****Q9.**** Write a query to fetch top N records.

By using the TOP command in SQL Server:

|  |  |
| --- | --- |
| 1 | **SELECT** **TOP** N \* **FROM** EmployeePosition **ORDER** **BY** Salary **DESC**; |

By using the LIMIT command in MySQL:

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmpPosition **ORDER** **BY** Salary **DESC** LIMIT N; |

### ****Q10. Write a query to retrieve the EmpFname and EmpLname in a single column as “FullName”. The first name and the last name must be separated with space.****

|  |  |
| --- | --- |
| 1 | **SELECT** CONCAT(EmpFname, ' ', EmpLname) **AS** 'FullName' **FROM** EmployeeInfo; |

### ****Q11. Write a query find number of employees whose DOB is between 02/05/1970 to 31/12/1975 and are grouped according to gender****

|  |  |
| --- | --- |
| 1 | **SELECT** COUNT(\*), Gender **FROM** EmployeeInfo **WHERE** DOB BETWEEN '02/05/1970 ' AND '31/12/1975' **GROUP** **BY** Gender; |

### ****Q12. Write a query to fetch all the records from the EmployeeInfo table ordered by EmpLname in descending order and Department in the ascending order.****

To order the records in ascending and descnding order, you have to use the [ORDER BY statement in SQL](https://www.edureka.co/blog/order-by-in-sql).

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **ORDER** **BY** EmpFname **desc**, Department **asc**; |

### ****Q13. Write a query to fetch details of employees whose EmpLname ends with an alphabet ‘A’ and contains five alphabets.****

To fetch details mathcing a certain value, you have to use the [LIKE operator in SQL](https://www.edureka.co/blog/like-in-sql/).

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** EmpLname LIKE '\_\_\_\_a'; |

### ****Q14. Write a query to fetch details of all employees excluding the employees with first names, “Sanjay” and “Sonia” from the EmployeeInfo table.****

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** EmpFname NOT IN ('Sanjay','Sonia'); |

**Q15. Write a query to fetch details of employees with the address as “DELHI(DEL)”.**

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** Address LIKE 'DELHI(DEL)%'; |

**Q16. Write a query to fetch all employees who also hold the managerial position.**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** E.EmpFname, E.EmpLname, P.EmpPosition  **FROM** EmployeeInfo E **INNER** JOIN EmployeePosition P **ON**  E.EmpID = P.EmpID AND P.EmpPosition IN ('Manager'); |

**Q17.** **Write a query to fetch the department-wise count of employees sorted by department’s count in ascending order.**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** Department, count(EmpID) **AS** EmpDeptCount  **FROM** EmployeeInfo **GROUP** **BY** Department  **ORDER** **BY** EmpDeptCount **ASC**; |

**Q18. Write a query to calculate the even and odd records from a table.**

To retrieve the even records from a table, you have to use the MOD() function as follows:

|  |  |
| --- | --- |
| 1 | **SELECT** EmpID **FROM** (**SELECT** rowno, EmpID **from** EmployeeInfo) **WHERE** MOD(rowno,2)=0; |

Similarly, to retrieve the odd records from a table, you can write a query as follows:

|  |  |
| --- | --- |
| 1 | **SELECT** EmpID **FROM** (**SELECT** rowno, EmpID **from** EmployeeInfo) **WHERE** MOD(rowno,2)=1; |

**Q19.** **Write a SQL query to retrieve employee details from EmployeeInfo table who have a date of joining in the EmployeePosition table.**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** \* **FROM** EmployeeInfo E  **WHERE** EXISTS  (**SELECT** \* **FROM** EmployeePosition P **WHERE** E.EmpId = P.EmpId); |

**Q20. Write a query to retrieve two minimum and maximum salaries from the EmployeePosition table.**

To retrieve two minimum salaries, you can write a query as below:

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** **DISTINCT** Salary **FROM** EmployeePosition E1  **WHERE** 2 >= (SELECTCOUNT(**DISTINCT** Salary)**FROM** EmployeePosition E2  **WHERE** E1.Salary >= E2.Salary) **ORDER** **BY** E1.Salary **DESC**; |

To retrieve two maximum salaries, you can write a query as below:

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** **DISTINCT** Salary **FROM** EmployeePosition E1  **WHERE** 2 >= (SELECTCOUNT(**DISTINCT** Salary) **FROM** EmployeePosition E2  **WHERE** E1.Salary <= E2.Salary) **ORDER** **BY** E1.Salary **DESC**; |

**Q21.** **Write a query to find the Nth highest salary from the table without using TOP/limit keyword.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT** Salary  **FROM** EmployeePosition E1  **WHERE** N-1 = (  **SELECT** COUNT( **DISTINCT** ( E2.Salary ) )  **FROM** EmployeePosition E2  **WHERE** E2.Salary >  E1.Salary ); |

**Q22. Write a query to retrieve duplicate records from a table.**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** EmpID, EmpFname, Department COUNT(\*)  **FROM** EmployeeInfo **GROUP** **BY** EmpID, EmpFname, Department  **HAVING** COUNT(\*) > 1; |

**Q23. Write a query to retrieve the list of employees working in the same department.**

|  |  |
| --- | --- |
| 1  2  3 | **Select** **DISTINCT** E.EmpID, E.EmpFname, E.Department  **FROM** EmployeeInfo E, Employee E1  **WHERE** E.Department = E1.Department AND E.EmpID != E1.EmpID; |

**Q24. Write a query to retrieve the last 3 records from the EmployeeInfo table.**

|  |  |
| --- | --- |
| 1  2  3  4 | **SELECT** \* **FROM** EmployeeInfo **WHERE**  EmpID <=3 **UNION** **SELECT** \* **FROM**  (**SELECT** \* **FROM** EmployeeInfo E **ORDER** **BY** E.EmpID **DESC**)  **AS** E1 **WHERE** E1.EmpID <=3; |

**Q25. Write a query to find the third-highest salary from the EmpPosition table.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **SELECT** **TOP** 1 salary  **FROM**(  **SELECT** **TOP** 3 salary  **FROM** employee\_table  **ORDER** **BY** salary **DESC**) **AS** emp  **ORDER** **BY** salary **ASC**; |

**Q26. Write a query to display the first and the last record from the EmployeeInfo table.**

To display the first record from the EmployeeInfo table, you can write a query as follows:

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** EmpID = (**SELECT** **MIN**(EmpID) **FROM** EmployeeInfo); |

To display the last record from the EmployeeInfo table, you can write a query as follows:

|  |  |
| --- | --- |
| 1 | **SELECT** \* **FROM** EmployeeInfo **WHERE** EmpID = (**SELECT** **MAX**(EmpID) **FROM** EmployeeInfo); |

**Q27. Write a query to add email validation to your database**

|  |  |
| --- | --- |
| 1 | **SELECT** Email **FROM** EmployeeInfo **WHERE** NOT REGEXP\_LIKE(Email, ‘[A-Z0-9.\_%+-]+@[A-Z0-9.-]+.[A-Z]{2,4}’, ‘i’); |

**Q28. Write a query to retrieve Departments who have less than 2 employees working in it.**

|  |  |
| --- | --- |
| 1 | **SELECT** DEPARTMENT, COUNT(EmpID) **as** 'EmpNo' **FROM** EmployeeInfo **GROUP** **BY** DEPARTMENT **HAVING** COUNT(EmpD) < 2; |

**Q29. Write a query to retrieve EmpPostion along with total salaries paid for each of them.**

|  |  |
| --- | --- |
| 1 | **SELECT** EmpPosition, SUM(Salary) **from** EmployeePosition **GROUP** **BY** EmpPosition; |

**Q30. Write a query to fetch 50% records from the EmployeeInfo table.**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** \*  **FROM** EmployeeInfo **WHERE**  EmpID <= (**SELECT** COUNT(EmpID)/2 **from** EmployeeInfo); |