SQL Interview Bank

EASY:

1. What is Database?

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

2. What is DBMS?

DBMS stands for Database Management System. DBMS is a system software responsible for the creation, retrieval, updation, and management of the database. It ensures that our data is consistent, organized, and is easily accessible by serving as an interface between the database and its end-users or application software.

3. What is RDBMS? How is it different from DBMS?

RDBMS stands for Relational Database Management System. The key difference here, compared to DBMS, is that RDBMS stores data in the form of a collection of tables, and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2, and Amazon Redshift are based on RDBMS.

4. What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

5. What is the difference between SQL and MySQL?

SQL vs MySQL

SQL	MySQL
SQL is a standard language which stands for Structured Query Language based on the English language	MySQL is a database management system.
SQL is the core of the relational database which is used for accessing and managing database	MySQL is an RDMS (Relational Database Management System) such as SQL Server, Informix etc.

6. What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

7. What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

NOT NULL - Restricts NULL value from being inserted into a column.

CHECK - Verifies that all values in a field satisfy a condition.

DEFAULT - Automatically assigns a default value if no value has been specified for the field.

UNIQUE - Ensures unique values to be inserted into the field.

INDEX - Indexes a field providing faster retrieval of records.

PRIMARY KEY - Uniquely identifies each record in a table.

FOREIGN KEY - Ensures referential integrity for a record in another table.

8. What is a Primary Key?

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.

A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

```
CREATE TABLE Students (   /* Create table with a single field as primary key */
  ID INT NOT NULL
  Name VARCHAR(255)
  PRIMARY KEY (ID)
);
```

```
CREATE TABLE Students (   /* Create table with multiple fields as primary key */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL,
  CONSTRAINT PK_Student
  PRIMARY KEY (ID, FirstName)
);
```

```
ALTER TABLE Students   /* Set a column as primary key */
ADD PRIMARY KEY (ID);
ALTER TABLE Students   /* Set multiple columns as primary key */
ADD CONSTRAINT PK_Student   /*Naming a Primary Key*/
PRIMARY KEY (ID, FirstName);
```

9. What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, there can be multiple unique constraints defined per table. The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

```
CREATE TABLE Students (   /* Create table with a single field as unique */
```

```
  ID INT NOT NULL UNIQUE
  Name VARCHAR(255)
);

CREATE TABLE Students (   /* Create table with multiple fields as unique */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL
  CONSTRAINT PK_Student
  UNIQUE (ID, FirstName)
);

ALTER TABLE Students   /* Set a column as unique */
ADD UNIQUE (ID);
ALTER TABLE Students   /* Set multiple columns as unique */
ADD CONSTRAINT PK_Student   /* Naming a unique constraint */
UNIQUE (ID, FirstName);
```

10. What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refers to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables.

The table with the foreign key constraint is labeled as the child table, and the table containing the candidate key is labeled as the referenced or parent table.

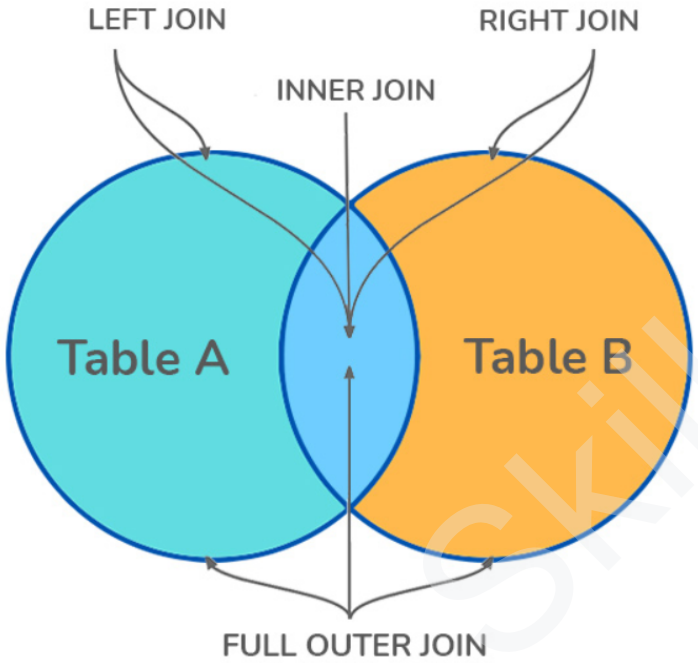
```
CREATE TABLE Students (   /* Create table with foreign key - Way 1 */
  ID INT NOT NULL
  Name VARCHAR(255)
  LibraryID INT
```

```
  PRIMARY KEY (ID)
  FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

CREATE TABLE Students (   /* Create table with foreign key - Way 2 */
  ID INT NOT NULL PRIMARY KEY
  Name VARCHAR(255)
  LibraryID INT FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

ALTER TABLE Students   /* Add a new foreign key */
ADD FOREIGN KEY (LibraryID)
REFERENCES Library (LibraryID);
```

11. What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.

There are mainly four different types of JOINs in SQL:



(INNER) JOIN: Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.

```
SELECT *
FROM Table_A
JOIN Table_B;
```

```
SELECT *
FROM Table_A
INNER JOIN Table_B;
```

LEFT (OUTER) JOIN: Retrieves all the records/rows from the left and the matched records/rows from the right table.

```
SELECT *
FROM Table_A A
LEFT JOIN Table_B B
ON A.col = B.col;
```

RIGHT (OUTER) JOIN: Retrieves all the records/rows from the right and the matched records/rows from the left table.

```
SELECT *
FROM Table_A A
RIGHT JOIN Table_B B
ON A.col = B.col;
```

FULL (OUTER) JOIN: Retrieves all the records where there is a match in either the left or right table.

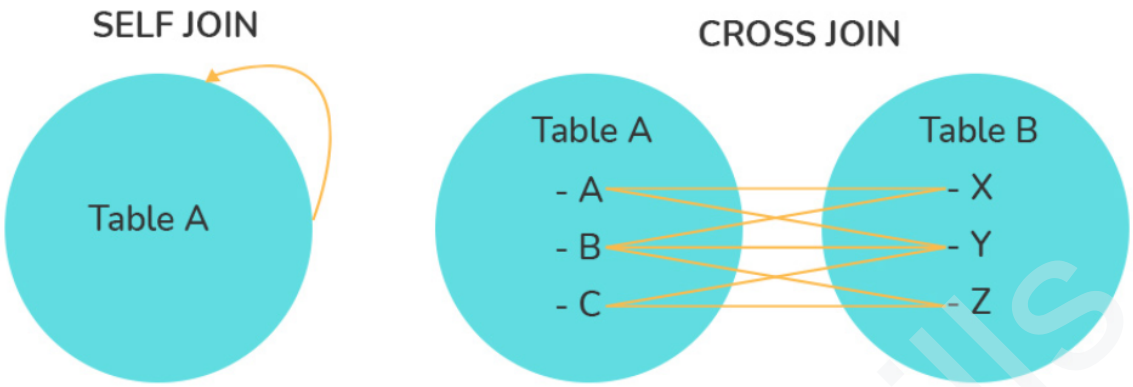
```
SELECT *
FROM Table_A A
FULL JOIN Table_B B
ON A.col = B.col;
```

12. What is a Self-Join?

A self JOIN is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

```
SELECT A.emp_id AS "Emp_ID",A.emp_name AS "Employee",
B.emp_id AS "Sup_ID",B.emp_name AS "Supervisor"
FROM employee A, employee B
WHERE A.emp_sup = B.emp_id;
```

13. What is a Cross-Join?



Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

```
SELECT stu.name, sub.subject
FROM students AS stu
CROSS JOIN subjects AS sub;
```

14: How do you create a new table in SQL?

A: You can create a new table in SQL using the CREATE TABLE statement. For example:

```
CREATE TABLE Employees (
    ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Salary DECIMAL(10, 2)
);
```

15: How would you insert a new record into the "Customers" table with values for Name, Email, and Phone?

A: You can use the INSERT INTO statement to insert a new record. For example:

INSERT INTO Customers (Name, Email, Phone)

VALUES ('John Smith', 'john@example.com', '123-456-7890');

16: Write a SQL query to retrieve all columns for customers whose last name is "Smith."

A: Here's a query to achieve this:

SELECT * FROM Customers WHERE LastName = 'Smith';

17: How do you filter records in a table to only include those with a salary greater than $50,000?

A: You can use the WHERE clause to filter records based on the salary. For example:

SELECT * FROM Employees WHERE Salary > 50000;

18: Write an SQL query to retrieve a list of products sorted by price in descending order.

A: SELECT *

FROM Products

ORDER BY Price DESC;

19: Explain the difference between GROUP BY and ORDER BY clauses.

A: GROUP BY is used to group rows that have the same values in specified columns, typically used with aggregate functions. ORDER BY is used to sort the result set based on specified columns.

20: How do you filter for NULL values in SQL?

A: You can filter for NULL values using the IS NULL or IS NOT NULL operators. For example:

SELECT * FROM Employees WHERE Department IS NULL;

21: What is the difference between INNER JOIN and LEFT JOIN?

A: An INNER JOIN retrieves records that have matching values in both tables, while a LEFT JOIN retrieves all records from the left table and the matched records from the right table, filling in with NULLs if there's no match.

22: How can you combine multiple conditions in a WHERE clause?

A: You can combine conditions in a WHERE clause using logical operators such as AND, OR, and parentheses to control the order of evaluation. For example:

```
SELECT * FROM Employees WHERE Salary > 50000 AND Department = 'Sales';
```

23: Write a query to calculate the average salary of all employees in the "Employees" table.

A: Here's a query to calculate the average salary:

```
SELECT AVG(Salary) FROM Employees;
```

24: How do you find the highest salary in the "Salaries" column of a table using SQL?

A: You can use the MAX function to find the highest salary. For example:

```
SELECT MAX(Salary) FROM Salaries;
```

25: Write an SQL query to calculate the total price of all products in a "Products" table.

A: To calculate the total price, you can use the SUM function:

```
SELECT SUM(Price) FROM Products;
```

26: Explain how to use the DATEADD function to add 7 days to a given date.

A: SELECT DATEADD(DAY, 7, SomeDate) FROM YourTable;

27: Write a query to find the absolute value of a number stored in a column.

A: You can use the ABS function to find the absolute value. For example:

SELECT ABS(Number) FROM YourTable;

28: How would you round a decimal number to the nearest whole number using SQL?

A: You can use the ROUND function to round a decimal number. For example:

SELECT ROUND(DecimalColumn) FROM YourTable;

29: Write a query to retrieve the current date and time in SQL.

A: You can use the GETDATE() function:

SELECT GETDATE();

30: How can you calculate the difference in days between two dates using DATEDIFF?

A: You can use DATEDIFF like this:

SELECT DATEDIFF(DAY, StartDate, EndDate) FROM YourTable;

32: Write a query to extract the month from a date column using DATEPART.

A: You can use DATEPART to extract the month:

SELECT DATEPART(MONTH, DateColumn) FROM YourTable;

33: Explain how the CONVERT function can be used to change a date's format in SQL.

A: You can use CONVERT to change the format of a date:

SELECT CONVERT(VARCHAR, DateColumn, 103) FROM YourTable;

MEDIUM:

1. What is Cursor? How to use a Cursor?

After any variable declaration, DECLARE a cursor. A SELECT Statement must always be coupled with the cursor definition.

To start the result set, move the cursor over it. Before obtaining rows from the result set, the OPEN statement must be executed.

To retrieve and go to the next row in the result set, use the FETCH command.

To disable the cursor, use the CLOSE command.

Finally, use the DEALLOCATE command to remove the cursor definition and free up the resources connected with it.

2. What is an Index? Explain its different types.

A database index is a data structure that provides a quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

```
CREATE INDEX index_name   /* Create Index */
ON table_name (column_1, column_2);
DROP INDEX index_name;   /* Drop Index */
```

There are different types of indexes that can be created for different purposes:

- Unique and Non-Unique Index:

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index.

```
CREATE UNIQUE INDEX myIndex
ON students (enroll_no);
```

Non-unique indexes, on the other hand, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

- Clustered and Non-Clustered Index:

Clustered indexes are indexes whose order of the rows in the database corresponds to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas, multiple non-clustered indexes can exist in the table.

The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the database in the same order as the corresponding keys appear in the clustered index.

Clustering indexes can improve the performance of most query operations because they provide a linear-access path to data stored in the database.

3. What is the difference between 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.
- Clustered index is used for easy and speedy retrieval of data from the database, whereas, fetching records from the non-clustered index is relatively slower.
- In SQL, a table can have a single clustered index whereas it can have multiple non-clustered indexes.

4. What is Data Integrity?

Data Integrity is the assurance of accuracy and consistency of data over its entire life-cycle and is a critical aspect of the design, implementation, and usage of any system which stores, processes, or retrieves data. It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database.

5. What is a Query?

A query is a request for data or information from a database table or combination of tables. A database query can be either a select query or an action query.

```
SELECT fname, lname    /* select query */
FROM myDb.students
```

```
WHERE student_id = 1;


UPDATE myDB.students    /* action query */
SET fname = 'Captain', lname = 'America'
WHERE student_id = 1;
```

6. What is a Subquery? What are its types?

A subquery is a query within another query, also known as a nested query or inner query. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. For example, here we fetch the contact information for students who have enrolled for the maths subject:

```
SELECT name, email, mob, address
FROM myDb.contacts
WHERE roll_no IN (
 SELECT roll_no
 FROM myDb.students
 WHERE subject = 'Maths');
```

There are two types of subquery - Correlated and Non-Correlated.

- A correlated subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM of the main query.
- A non-correlated subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

7. What is the SELECT statement?

SELECT operator in SQL is used to select data from a database. The data returned is stored in a result table, called the result-set.

```
SELECT * FROM myDB.students;
```

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

Some common SQL clauses used in conjuction with a SELECT query are as follows:

- WHERE clause in SQL is used to filter records that are necessary, based on specific conditions.
- ORDER BY clause in SQL is used to sort the records based on some field(s) in ascending (ASC) or descending order (DESC).

```
SELECT *

FROM myDB.students

WHERE graduation_year = 2019

ORDER BY studentID DESC;
```

- GROUP BY clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarized results from the database.
- HAVING clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since the WHERE clause cannot filter aggregated records.

```
SELECT COUNT(studentId), country

FROM myDB.students

WHERE country != "INDIA"

GROUP BY country

HAVING COUNT(studentID) > 5;
```

9. Differentiate between the WHERE clause and HAVING clause in SQL.

The WHERE clause and the HAVING clause are both used in SQL to filter rows in a query, but they are used at different stages of query processing and serve different purposes:

WHERE Clause:

- Usage: The WHERE clause is used to filter rows from the result set based on conditions applied to individual rows before the grouping or aggregation.

- Timing: The WHERE clause is evaluated before the GROUP BY clause (if present) and before any aggregation functions (e.g., SUM, AVG, COUNT) are applied.
- Filtering: It filters rows based on conditions applied to individual columns. For example, you can use it to filter rows where a specific column meets a certain condition.
- Aggregation: The WHERE clause cannot be used to filter based on aggregate functions like SUM, AVG, MAX, or MIN.

Example: Filtering employees with a salary greater than $50,000:

```
SELECT * FROM Employees WHERE Salary > 50000;
```

HAVING Clause:

- Usage: The HAVING clause is used to filter rows from the result set based on conditions applied to the result of aggregate functions after grouping.
- Timing: The HAVING clause is evaluated after the GROUP BY clause, and it is used to filter the result set after aggregation.
- Filtering: It filters groups of rows based on conditions applied to the result of aggregate functions. For example, you can use it to filter groups with a sum greater than a certain value.
- Aggregation: The HAVING clause is specifically designed for filtering based on aggregate functions. It allows you to filter groups based on their aggregate values.

Example: Filtering departments with a total salary expense greater than $1,000,000:

```
SELECT Department, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Department
HAVING SUM(Salary) > 1000000;
```

In summary, the WHERE clause is used for row-level filtering before grouping and aggregation, while the HAVING clause is used for filtering groups of rows after aggregation based on the result of aggregate functions.

10. 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.

Certain conditions need to be met before executing either of the above statements in SQL -

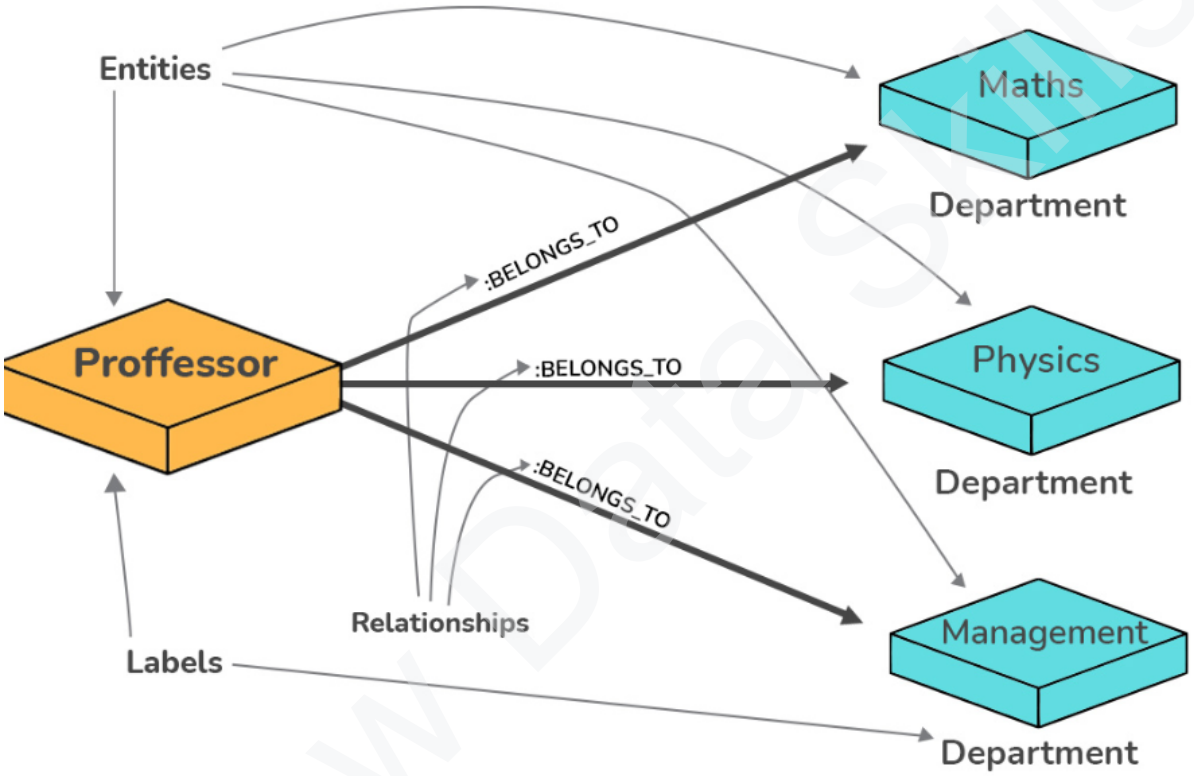
- Each SELECT statement within the clause must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement should necessarily have the same order

```
SELECT name FROM Students   /* Fetch the union of queries */
UNION
SELECT name FROM Contacts;
SELECT name FROM Students   /* Fetch the union of queries with duplicates*/
UNION ALL
SELECT name FROM Contacts;

SELECT name FROM Students   /* Fetch names from students */
MINUS     /* that aren't present in contacts */
SELECT name FROM Contacts;

SELECT name FROM Students   /* Fetch names from students */
INTERSECT    /* that are present in contacts as well */
SELECT name FROM Contacts;
```

11. What are Entities and Relationships?

Entity: An entity can be a real-world object, either tangible or intangible, that can be easily identifiable. For example, in a college database, students, professors, workers, departments, and projects can be referred to as entities. Each entity has some associated properties that provide it an identity.

Relationships: Relations or links between entities that have something to do with each other. For example - The employee's table in a company's database can be associated with the salary table in the same database.



12. List the different types of relationships in SQL.

- One-to-One - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.
- One-to-Many & Many-to-One - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.
- Many-to-Many - This is used in cases when multiple instances on both sides are needed for defining a relationship.
- Self-Referencing Relationships - This is used when a table needs to define a relationship with itself.

13. What is an Alias in SQL?

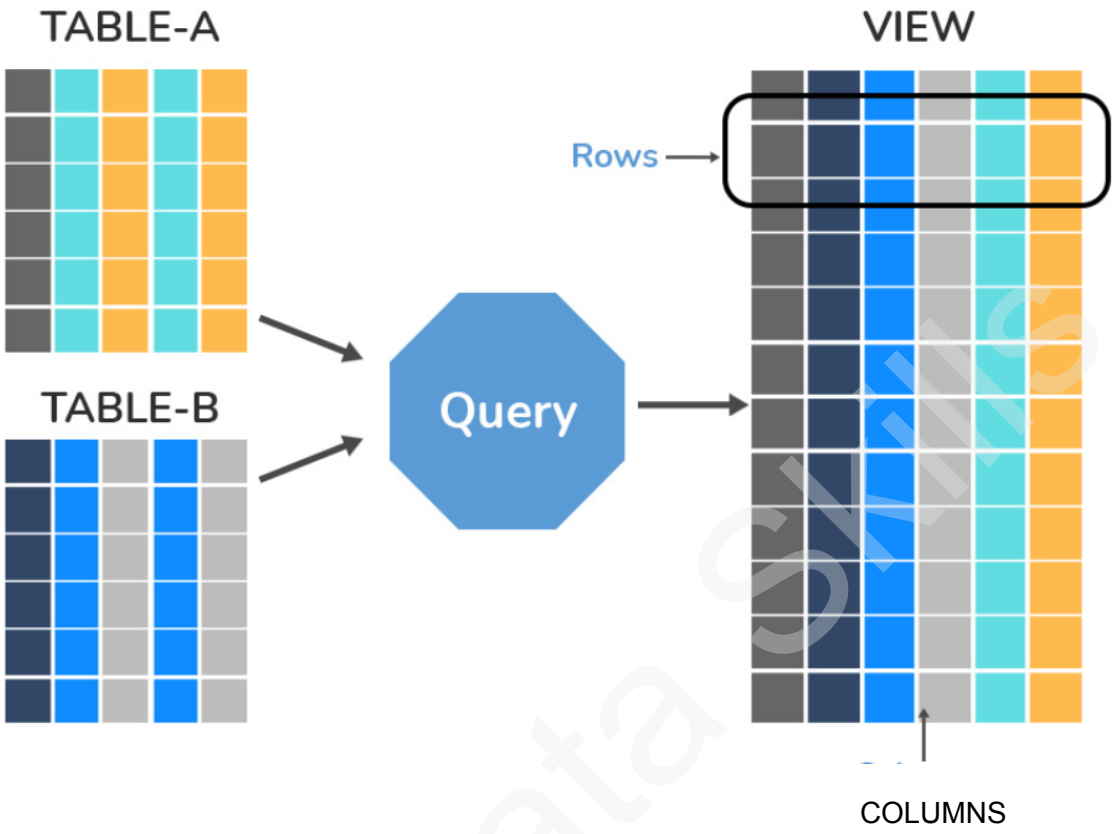
An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a correlation name.

An alias is represented explicitly by the AS keyword but in some cases, the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

```
SELECT A.emp_name AS "Employee"  /* Alias using AS keyword */
B.emp_name AS "Supervisor"
FROM employee A, employee B   /* Alias without AS keyword */
WHERE A.emp_sup = B.emp_id;
```

14. 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.



15. What is Normalization?

Normalization represents the way of organizing structured data in the database efficiently. It includes the creation of tables, establishing relationships between them, and defining rules for those relationships. Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

16. What is Denormalization?

Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema that has redundant information. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in the query processor by an over-normalized structure.

17. What are the various forms of Normalization?

Normal Forms are used to eliminate or reduce redundancy in database tables. The different forms are as follows:

- First Normal Form:

A relation is in first normal form if every attribute in that relation is a single-valued attribute. If a relation contains a composite or multi-valued attribute, it violates the first normal form. Let's consider the following students table. Each student in the table, has a name, his/her address, and the books they issued from the public library -

Students Table

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter), Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho), Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward), Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

As we can observe, the Books Issued field has more than one value per record, and to convert it into 1NF, this has to be resolved into separate individual records for each book issued. Check the following table in 1NF form -

Students Table (1st Normal Form)

Students Table (1st Normal Form)

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter)	Ms.
Sara	Amanora Park Town 94	Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho)	Mr.
Ansh	62nd Sector A-10	Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward)	Mrs.
Sara	24th Street Park Avenue	Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

- Second Normal Form:

A relation is in second normal form if it satisfies the conditions for the first normal form and does not contain any partial dependency. A relation in 2NF has no partial dependency, i.e., it has no non-prime attribute that depends on any proper subset of any candidate key of the table. Often, specifying a single column Primary Key is the solution to the problem. Examples -

Example 1 - Consider the above example. As we can observe, the Students Table in the 1NF form has a candidate key in the form of [Student, Address] that can uniquely identify all records in the table. The field Books Issued (non-prime attribute) depends partially on the Student field. Hence, the table is not in 2NF. To convert it into the 2nd Normal Form, we will partition the tables into two while specifying a new Primary Key attribute to identify the individual records in the Students table. The Foreign Key constraint will be set on the other table to ensure referential integrity.

Students Table (2nd Normal Form)

Student_ID	Student	Address	Salutation
1	Sara	Amanora Park Town 94	Ms.
2	Ansh	62nd Sector A-10	Mr.
3	Sara	24th Street Park Avenue	Mrs.
4	Ansh	Windsor Street 777	Mr.

Books Table (2nd Normal Form)

Student_ID	Book Issued
1	Until the Day I Die (Emily Carpenter)
1	Inception (Christopher Nolan)
2	The Alchemist (Paulo Coelho)
2	Inferno (Dan Brown)
3	Beautiful Bad (Annie Ward)
3	Woman 99 (Greer Macallister)
4	Dracula (Bram Stoker)

Example 2 - Consider the following dependencies in relation to R(W,X,Y,Z)

WX -> Y [W and X together determine Y]

XY -> Z [X and Y together determine Z]

Here, WX is the only candidate key and there is no partial dependency, i.e., any proper subset of WX doesn't determine any non-prime attribute in the relation.

- Third Normal Form

A relation is said to be in the third normal form, if it satisfies the conditions for the second normal form and there is no transitive dependency between the non-prime attributes, i.e., all non-prime attributes are determined only by the candidate keys of the relation and not by any other non-prime attribute.

Example 1 - Consider the Students Table in the above example. As we can observe, the Students Table in the 2NF form has a single candidate key Student_ID (primary key) that can uniquely identify all records in the table. The field Salutation (non-prime attribute), however, depends on the Student Field rather than the candidate key. Hence, the table is not in 3NF. To convert it into the 3rd Normal Form, we will once again partition the tables into two while specifying a new Foreign Key constraint to identify the salutations for individual records in the Students table. The Primary Key constraint for the same will be set on the Salutations table to identify each record uniquely.

Students Table (3rd Normal Form)

Student_ID	Student	Address	Salutation_ID
1	Sara	Amanora Park Town 94	1
2	Ansh	62nd Sector A-10	2
3	Sara	24th Street Park Avenue	3
4	Ansh	Windsor Street 777	1

Books Table (3rd Normal Form)

Student_ID	Book Issued
1	Until the Day I Die (Emily Carpenter)
1	Inception (Christopher Nolan)
2	The Alchemist (Paulo Coelho)
2	Inferno (Dan Brown)
3	Beautiful Bad (Annie Ward)
3	Woman 99 (Greer Macallister)
4	Dracula (Bram Stoker)

Salutations Table (3rd Normal Form)

Salutation_ID	Salutation
1	Ms.
2	Mr.
3	Mrs.

Example 2 - Consider the following dependencies in relation to R(P,Q,R,S,T)

P -> QR [P together determine C]

RS -> T [B and C together determine D]

Q -> S

T -> P

For the above relation to exist in 3NF, all possible candidate keys in the above relation should be {P, RS, QR, T}.

- Boyce-Codd Normal Form

A relation is in Boyce-Codd Normal Form if satisfies the conditions for third normal form and for every functional dependency, Left-Hand-Side is super key. In other words, a relation in BCNF has non-trivial functional dependencies in form X –> Y, such that X is always a super key. For

example - In the above example, Student_ID serves as the sole unique identifier for the Students Table and Salutation_ID for the Salutations Table, thus these tables exist in BCNF. The same cannot be said for the Books Table and there can be several books with common Book Names and the same Student_ID.

18. What are the TRUNCATE, DELETE and DROP statements?

DELETE statement is used to delete rows from a table.

```
DELETE FROM Candidates
WHERE CandidateId > 1000;
```

TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

```
TRUNCATE TABLE Candidates;
```

DROP command is used to remove an object from the database. If you drop a table, all the rows in the table are deleted and the table structure is removed from the database.

```
DROP TABLE Candidates;
```

19. 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.

20. 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.

21. What are Aggregate and Scalar functions?

An aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement. Following are the widely used SQL aggregate functions:

AVG() - Calculates the mean of a collection of values.

COUNT() - Counts the total number of records in a specific table or view.

MIN() - Calculates the minimum of a collection of values.

MAX() - Calculates the maximum of a collection of values.

SUM() - Calculates the sum of a collection of values.

FIRST() - Fetches the first element in a collection of values.

LAST() - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT 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.

22: Write a SQL query to retrieve all information about employees along with their department names using an INNER JOIN.

A:

```
SELECT Employees.*, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

23: How does a LEFT JOIN differ from an INNER JOIN, and when would you use each?

A: A LEFT JOIN retrieves all records from the left table and the matched records from the right table, filling in with NULLs if there's no match. An INNER JOIN retrieves only the records with matching values in both tables. Use LEFT JOIN when you want to retrieve all records from the left table, even if there are no matches in the right table.

24: Explain how to perform a self-join in SQL.

A: A self-join is performed when a table is joined with itself. You need to use table aliases to distinguish between the two instances of the table. For example:

```
SELECT e1.Name, e2.Name
FROM Employees e1
INNER JOIN Employees e2 ON e1.ManagerID = e2.ID;
```

25: Write a query to find all customers who have not placed an order using a RIGHT JOIN.

A: Here's an example query:

```
SELECT Customers.Name
FROM Customers
RIGHT JOIN Orders ON Customers.ID = Orders.CustomerID
WHERE Orders.CustomerID IS NULL;
```

26: What is the result of a FULL OUTER JOIN between two tables, and when would you use it?

A: A FULL OUTER JOIN returns all rows when there is a match in either the left or the right table. When there's no match, NULL values are included in the result. You would use it when you want to include all records from both tables, whether they match or not.

27: How do you perform a cross join between two tables?

A: A cross join, also known as a Cartesian join, combines all rows from the first table with all rows from the second table. You can do it like this:

```
SELECT * FROM Table1
CROSS JOIN Table2;
```

28: Write a SQL query to retrieve a list of products and their suppliers using an INNER JOIN.

A: Here's an example query:

```
SELECT Products.ProductName, Suppliers.SupplierName
FROM Products
INNER JOIN Suppliers ON Products.SupplierID = Suppliers.ID;
```

29: How can you join three or more tables together in a single query?

A: You can join three or more tables by extending the JOIN clauses. For example:

```
SELECT *
FROM Table1
INNER JOIN Table2 ON Table1.ID = Table2.Table1ID
INNER JOIN Table3 ON Table2.ID = Table3.Table2ID;
```

30: Write a query to find the common records between two tables using an INNER JOIN.

A: Here's an example query:

```
SELECT *
FROM Table1
```

INNER JOIN Table2 ON Table1.CommonColumn = Table2.CommonColumn;

31: Write a SQL query that uses a subquery to find the details of employees with salary higher than average salary among employees.

A: You can use a subquery like this:

Select *

From Employees where Salary > (SELECT AVG(Salary) FROM Employees);

32: Explain the difference between a single-row subquery and a multi-row subquery.

A: A single-row subquery returns only one value or row, typically used with single-value comparisons. A multi-row subquery returns multiple rows, often used with IN or EXISTS operators.

33: Write a subquery to find all orders placed by a customer with a specific ID.

A: SELECT *

FROM Orders

WHERE CustomerID = (SELECT ID FROM Customers WHERE Name = 'John Smith');

34: What is a correlated subquery, and when would you use it?

A: A correlated subquery refers to the outer query's columns within the subquery. It's used when you need to compare values from the outer query with values in the subquery. For example, finding employees whose salary is greater than the department's average salary.

35: How can you use a subquery to find records that are not present in another table?

A: You can use a subquery with the NOT IN or NOT EXISTS operators to find records that don't match a condition in another table. For example:

SELECT *

FROM Customers

WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);

36: Write a SQL query with a subquery to calculate the average order total for each customer.

A:

```
SELECT Customers.Name, (SELECT AVG(Total)
FROM Orders
WHERE CustomerID = Customers.ID) AS AverageOrderTotal
FROM Customers;
```

37: Explain the concept of a scalar subquery.

A: A scalar subquery returns a single value and can be used in expressions, assignments, or comparisons.

38: How would you rewrite a subquery as a JOIN statement for performance optimization?

A: Subqueries can often be rewritten as JOIN statements, which can improve performance. For example, a subquery like this:

```
SELECT *
FROM Orders
WHERE CustomerID = (SELECT ID FROM Customers WHERE Name = 'John Smith');
```

can be rewritten as:

```
SELECT Orders.*
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.ID
WHERE Customers.Name = 'John Smith';
```

39: Write a subquery to find the product with the highest price.

A: Here's a subquery to achieve that:

```
SELECT ProductName
FROM Products
WHERE Price = (SELECT MAX(Price) FROM Products);
```

40: How do you use subqueries with the IN and EXISTS operators?

A: You can use the IN operator to check if a value exists in the result set of a subquery. The EXISTS operator checks if a subquery returns any rows. For example:

```
SELECT * FROM Customers WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

41: Write a query that uses ROW_NUMBER to assign a unique number to each row in a result set.

A: Here's an example query:

```
SELECT Name, ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNum FROM Employees;
```

42: Explain the purpose of the PARTITION BY clause in window functions.

A: The PARTITION BY clause divides the result set into partitions to which the window function is applied separately. It's used for performing calculations within specific groups.

43: Write a query that calculates the average salary for each department using the AVG() window function.

A: You can use the AVG() window function with PARTITION BY like this:

```
SELECT Department, AVG(Salary) OVER (PARTITION BY Department) AS AvgSalary
FROM Employees;
```

44: How do you use the LAG() function to access the value of a previous row in a result set?

A:

```
SELECT Name, Salary, LAG(Salary) OVER (ORDER BY Salary) AS PrevSalary
FROM Employees;
```

45: Write a query to find the top 3 products with the highest sales using window functions.

A:

```
SELECT ProductName, TotalSales FROM (
    SELECT ProductName, SUM(Quantity * Price) AS TotalSales,
        ROW_NUMBER() OVER (ORDER BY SUM(Quantity * Price) DESC) AS RowNum
    FROM OrderDetails
    GROUP BY ProductName
) AS RankedProducts
WHERE RowNum <= 3;
```

46: Explain the difference between ROW_NUMBER(), RANK(), and DENSE_RANK() functions.

A: ROW_NUMBER() assigns a unique number to each row, RANK() assigns a unique rank to rows with the same values, and DENSE_RANK() assigns a unique rank but doesn't leave gaps for duplicate values.

47: How can you calculate a running total using window functions?

A: You can use the SUM() window function with an ORDER BY clause to calculate a running total. For example:

```
SELECT Date, Revenue, SUM(Revenue) OVER (ORDER BY Date) AS RunningTotal
 FROM Sales;
```

48: Write a query that uses the FIRST_VALUE() function to retrieve the earliest order date for each customer.

A: You can use FIRST_VALUE() like this:

```
SELECT CustomerID, OrderDate, FIRST_VALUE(OrderDate) OVER (PARTITION BY
CustomerID ORDER BY OrderDate) AS EarliestOrderDate
FROM Orders;
```

49: Explain the concept of a window frame in window functions.

A: A window frame defines the set of rows used for each calculation in a window function. It can be defined using ROWS BETWEEN or RANGE BETWEEN to specify a range of rows relative to the current row.

50: How do you use the LEAD() function to access the value of a subsequent row in a result set?

A: You can use LEAD() like this:

SELECT Name, Salary, LEAD(Salary) OVER (ORDER BY Salary) AS NextSalary

FROM Employees;

ADVANCED:

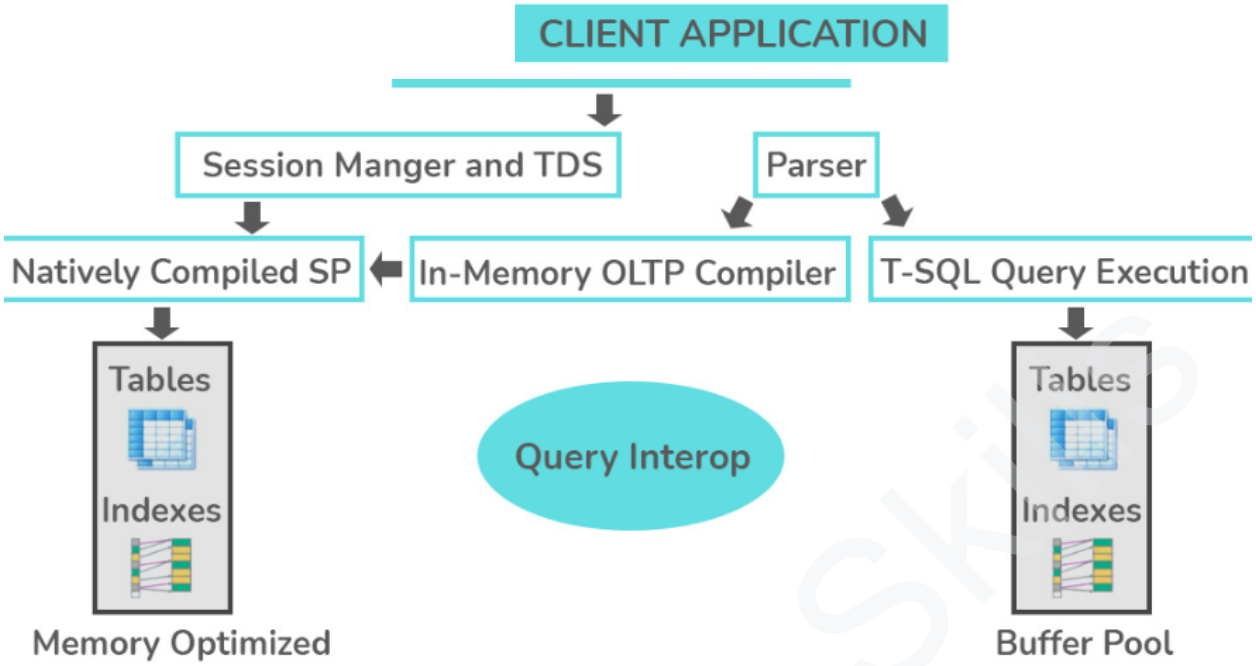
1. What is User-defined function? What are its various types?

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.

2. What is OLTP?

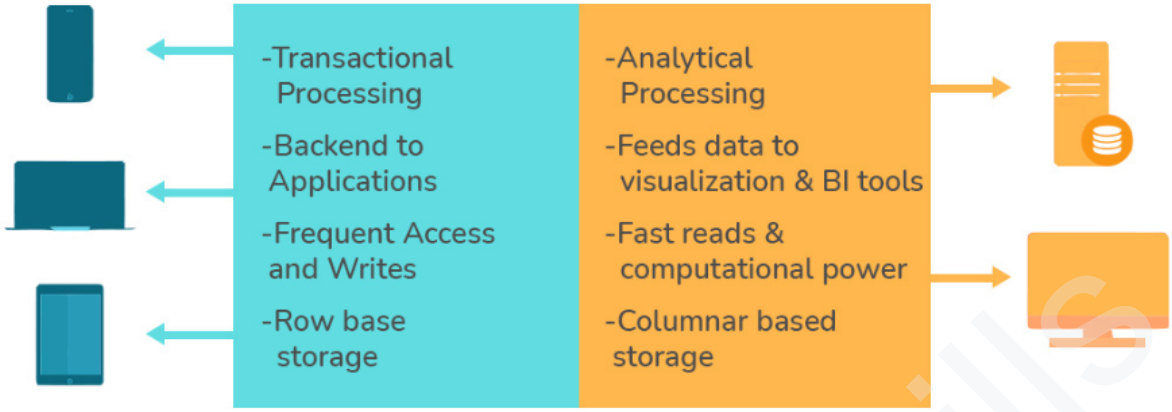
OLTP stands for Online Transaction Processing, is a class of software applications capable of supporting transaction-oriented programs. An essential attribute of an OLTP system is its ability to maintain concurrency. To avoid single points of failure, OLTP systems are often decentralized. These systems are usually designed for a large number of users who conduct short transactions. Database queries are usually simple, require sub-second response times, and return relatively few records. Here is an insight into the working of an OLTP system



3. What are the differences between OLTP and OLAP?

OLTP stands for Online Transaction Processing, is a class of software applications capable of supporting transaction-oriented programs. An important attribute of an OLTP system is its ability to maintain concurrency. OLTP systems often follow a decentralized architecture to avoid single points of failure. These systems are generally designed for a large audience of end-users who conduct short transactions. Queries involved in such databases are generally simple, need fast response times, and return relatively few records. A number of transactions per second acts as an effective measure for such systems.

OLAP stands for Online Analytical Processing, a class of software programs that are characterized by the relatively low frequency of online transactions. Queries are often too complex and involve a bunch of aggregations. For OLAP systems, the effectiveness measure relies highly on response time. Such systems are widely used for data mining or maintaining aggregated, historical data, usually in multi-dimensional schemas.



4. What is Collation? What are the different types of Collation Sensitivity?

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. It incorporates options for specifying case sensitivity, accent marks, kana character types, and character width. Below are the different types of collation sensitivity:

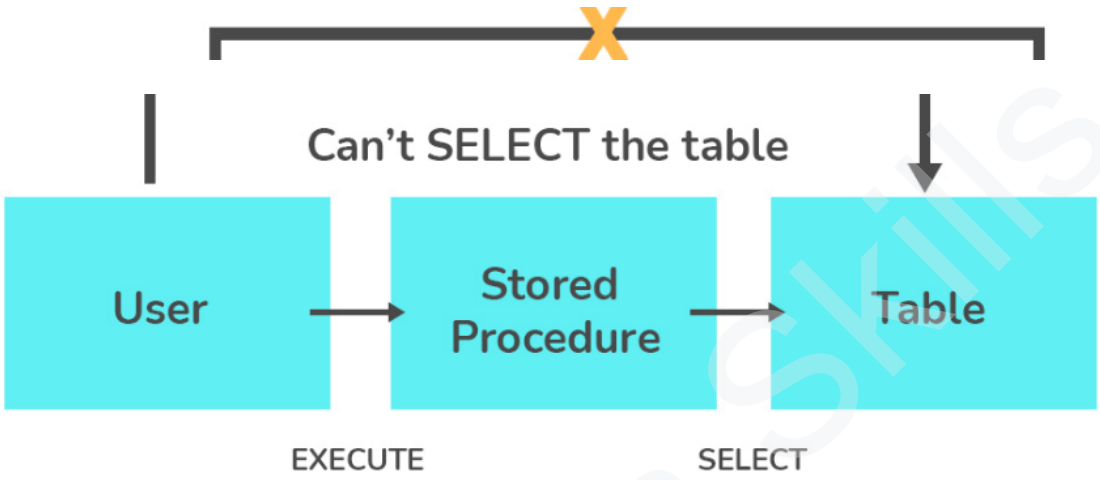
- Case sensitivity: A and a are treated differently.
- Accent sensitivity: a and á are treated differently.
- Kana sensitivity: Japanese kana characters Hiragana and Katakana are treated differently.
- Width sensitivity: Same character represented in single-byte (half-width) and double-byte (full-width) are treated differently.

5. What is a Stored Procedure?

A stored procedure is a subroutine available to applications that access a relational database management system (RDBMS). Such procedures are stored in the database data dictionary. The sole disadvantage of stored procedure is that it can be executed nowhere except in the database and occupies more memory in the database server. It also provides a sense of security and functionality as users who can't access the data directly can be granted access via stored procedures.

```
DELIMITER $$

CREATE PROCEDURE FetchAllStudents()
```

```
BEGIN
SELECT *  FROM myDB.students;
END $$
DELIMITER ;
```



6. What is a Recursive Stored Procedure?

A stored procedure that calls itself until a boundary condition is reached, is called a recursive stored procedure. This recursive function helps the programmers to deploy the same set of code several times as and when required. Some SQL programming languages limit the recursion depth to prevent an infinite loop of procedure calls from causing a stack overflow, which slows down the system and may lead to system crashes.

```
DELIMITER $$     /* Set a new delimiter => $$ */
CREATE PROCEDURE calctotal( /* Create the procedure */
  IN number INT,   /* Set Input and Ouput variables */
  OUT total INT
) BEGIN
DECLARE score INT DEFAULT NULL;   /* Set the default value => "score" */
SELECT awards FROM achievements   /* Update "score" via SELECT query */
WHERE id = number INTO score;
IF score IS NULL THEN SET total = 0;   /* Termination condition */
```

```
ELSE
CALL calctotal(number+1);   /* Recursive call */
SET total = total + score;   /* Action after recursion */
END IF;
END $$     /* End of procedure */
DELIMITER ;     /* Reset the delimiter */
```

7. How to create empty tables with the same structure as another table?

Creating empty tables with the same structure can be done smartly by fetching the records of one table into a new table using the INTO operator while fixing a WHERE clause to be false for all records. Hence, SQL prepares the new table with a duplicate structure to accept the fetched records but since no records get fetched due to the WHERE clause in action, nothing is inserted into the new table.

```
SELECT * INTO Students_copy
FROM Students WHERE 1 = 2;
```

8. What is Pattern Matching in SQL?

SQL pattern matching provides for pattern search in data if you have no clue as to what that word should be. This kind of SQL query uses wildcards to match a string pattern, rather than writing the exact word. The LIKE operator is used in conjunction with SQL Wildcards to fetch the required information.

- Using the % wildcard to perform a simple search

The % wildcard matches zero or more characters of any type and can be used to define wildcards both before and after the pattern. Search a student in your database with first name beginning with the letter K:

```
SELECT *
FROM students
WHERE first_name LIKE 'K%'
```

- Omitting the patterns using the NOT keyword

Use the NOT keyword to select records that don't match the pattern. This query returns all students whose first name does not begin with K.

```
SELECT *
FROM students
WHERE first_name NOT LIKE 'K%'
```

- Matching a pattern anywhere using the % wildcard twice

Search for a student in the database where he/she has a K in his/her first name.

```
SELECT *
FROM students
WHERE first_name LIKE '%Q%'
```

- Using the _ wildcard to match pattern at a specific position

The _ wildcard matches exactly one character of any type. It can be used in conjunction with % wildcard. This query fetches all students with letter K at the third position in their first name.

```
SELECT *
FROM students
WHERE first_name LIKE '__K%'
```

- Matching patterns for a specific length

The _ wildcard plays an important role as a limitation when it matches exactly one character. It limits the length and position of the matched results. For example -

```
SELECT *   /* Matches first names with three or more letters */
FROM students
```

WHERE first_name LIKE '___%'

SELECT * /* Matches first names with exactly four characters */

FROM students

WHERE first_name LIKE '____'

Scaler Courses

9. Does SQL support programming language features?

It is true that SQL is a language, but it does not support programming as it is not a programming language, it is a command language. We do not have conditional statements in SQL like for loops or if..else, we only have commands which we can use to query, update, delete, etc. data in the database. SQL allows us to manipulate data in a database.

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

- BETWEEN: The BETWEEN operator is used to fetch rows based on a range of values.

For example:

SELECT * FROM Students

WHERE ROLL_NO BETWEEN 20 AND 30;

This query will select all those rows from the table. Students where the value of the field ROLL_NO lies between 20 and 30.

- IN :The IN operator is used to check for values contained in specific sets.

For example,:

SELECT * FROM Students

WHERE ROLL_NO IN (20,21,23);

This query will select all those rows from the table Students where the value of the field ROLL_NO is either 20 or 21 or 23.

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

Both of these data types are used for characters, but varchar2 is used for character strings of variable length, whereas char is used for character strings of fixed length. For example, if we specify the type as char(5) then we will not be allowed to store a string of any other length in this variable, but if we specify the type of this variable as varchar2(5) then we will be allowed to store strings of variable length. We can store a string of length 3 or 4 or 2 in this variable.

12. Name different types of case manipulation functions available in SQL.

There are three types of case manipulation functions available in SQL. They are:

LOWER: The purpose of this function is to return the string in lowercase. It takes a string as an argument and returns the string by converting it into lower case.

Syntax:

LOWER('string')

UPPER: The purpose of this function is to return the string in uppercase. It takes a string as an argument and returns the string by converting it into uppercase.

Syntax:

UPPER('string')

INITCAP: The purpose of this function is to return the string with the first letter in uppercase and the rest of the letters in lowercase.

Syntax:

INITCAP('string')

13. 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.

14. What is Auto Increment?

Sometimes, while creating a table, we do not have a unique identifier within the table, hence we face difficulty in choosing Primary Key. So as to resolve such an issue, we’ve to manually provide unique keys to every record, but this is often also a tedious task. So we can use the

Auto-Increment feature that automatically generates a numerical Primary key value for every new record inserted. The Auto Increment feature is supported by all the Databases.

15. **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.

16. What are ACID properties?

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read-and-write operations. In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties. ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably.

17. How do we avoid getting duplicate entries in a query without using the distinct keyword?

DISTINCT is useful in certain circumstances, but it has drawbacks that it can increase the load on the query engine to perform the sort (since it needs to compare the result set to itself to remove duplicates). We can remove duplicate entries using the following options:

- Remove duplicates using row numbers.
- Remove duplicates using self-Join.
- Remove duplicates using group by.

18. 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.

Syntax:

SELECT column(s), ISNULL(column_name, value_to_replace)

FROM table_name;

19: Write an SQL statement to create a clustered index on a table.

A: CREATE CLUSTERED INDEX IX_EmployeeID ON Employees (EmployeeID);

20: How can you check if an index exists on a specific table and column?

A: You can use the sys.indexes catalog view to check for the existence of an index. For example:

SELECT * FROM sys.indexes WHERE object_id = OBJECT_ID('YourTable') AND name = 'YourIndex';

21: Write a query that uses an index to improve the performance of a SELECT statement.

A: An index can improve the performance of a query by allowing faster data retrieval. Ensure that the columns in your WHERE clause or JOIN conditions are indexed. For example, if you have an index on the "LastName" column:

SELECT * FROM Customers WHERE LastName = 'Smith';

22: What are the advantages and disadvantages of using indexes?

A:

Advantages: Faster data retrieval, improved query performance, efficient sorting, and reduced disk I/O.

Disadvantages: Increased storage requirements, slower data modification (INSERT, UPDATE, DELETE) operations, and maintenance overhead.

23: How do you rebuild or reorganize an index in SQL?

A: You can use the ALTER INDEX statement to rebuild or reorganize an index. For example:

```
ALTER INDEX YourIndex ON YourTable REBUILD;
```

24: Explain the concept of covering indexes and their benefits.

A: A covering index includes all the columns needed to satisfy a query, allowing the database engine to retrieve data directly from the index without accessing the table itself. This can significantly improve query performance by reducing I/O.

25. How can you disable or drop an existing index?

A: To disable an index temporarily:

```
ALTER INDEX YourIndex ON YourTable DISABLE;
```

To drop an index:

```
DROP INDEX YourTable.YourIndex;
```

26: What is index cardinality, and why is it important for query optimization?

A: Index cardinality represents the uniqueness of values in a column. High cardinality means that most values are unique, while low cardinality means many values are the same. High cardinality indexes are more selective and generally improve query optimization.

27: Write a SQL statement to create a stored procedure that retrieves employee information.

A:

```
CREATE PROCEDURE GetEmployeeInfo
AS
BEGIN
```

```
    SELECT * FROM Employees;
END;
```

28: How do you pass parameters to a stored procedure in SQL?

A: You can pass parameters to a stored procedure using the @parameter_name syntax. For example:

```
CREATE PROCEDURE GetEmployeeByID
    @EmployeeID INT
AS
BEGIN
    SELECT * FROM Employees WHERE EmployeeID = @EmployeeID;
END;
```

29: Write a stored procedure that updates the salary of an employee based on their ID.

A:

```
CREATE PROCEDURE UpdateEmployeeSalary
    @EmployeeID INT,
    @NewSalary DECIMAL(10, 2)
AS
BEGIN
    UPDATE Employees SET Salary = @NewSalary WHERE EmployeeID = @EmployeeID;
END;
```

30: Explain the difference between input and output parameters in stored procedures.

A: Input parameters are used to pass values into the stored procedure, while output parameters are used to return values from the stored procedure back to the caller.

31: How can you return a result set from a stored procedure?

A: You can use a SELECT statement within a stored procedure to return a result set. The result set can be consumed by the calling program or query.

32: Write a stored procedure that deletes all orders placed by a specific customer.

A: Here's an example of a stored procedure to delete orders by customer ID:

```
CREATE PROCEDURE DeleteOrdersByCustomer
    @CustomerID INT
AS
BEGIN
    DELETE FROM Orders WHERE CustomerID = @CustomerID;
END;
```

33: How do you modify an existing stored procedure in SQL?

A: To modify an existing stored procedure, you can use the ALTER PROCEDURE statement. For example:

```
ALTER PROCEDURE YourProcedureName
AS
BEGIN
    -- Updated procedure code here
END;
```

34: How can you handle exceptions and errors within a stored procedure?

A: You can use TRY...CATCH blocks to handle exceptions and errors within a stored procedure. In the CATCH block, you can log the error or perform other error-handling actions.

35: Write a SQL query to undo all changes made within a transaction and roll it back.

A: To roll back a transaction, you can use the ROLLBACK statement. For example:

```
BEGIN TRANSACTION;
```

```
-- Perform some operations
IF SomeCondition
BEGIN
    ROLLBACK;
END;
ELSE
BEGIN
    COMMIT;
END;
```

Transactions:

36: Write a SQL query to start a new transaction.

A: You can start a new transaction using the BEGIN TRANSACTION statement:

```
BEGIN TRANSACTION;
```

37: How do you commit a transaction, and what is its purpose?

A: You can commit a transaction using the COMMIT statement. Committing a transaction saves all changes made within the transaction to the database.

38: Explain the concept of a rollback in SQL transactions.

A: A rollback is used to undo all changes made within a transaction and return the database to its previous state before the transaction started.

39: How can you set a savepoint within a transaction?

A: You can set a savepoint within a transaction using the SAVE TRANSACTION statement. For example:

```
SAVE TRANSACTION SavepointName;
```

40: Write a query to set the isolation level of a transaction to "READ COMMITTED."

A: You can set the isolation level using the SET TRANSACTION ISOLATION LEVEL statement:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

41: How do you handle concurrent transactions and prevent conflicts?

A: Concurrent transactions can be handled by setting appropriate isolation levels, using locks to control access, and handling conflicts through proper error handling and retry mechanisms.

42: Explain the difference between implicit and explicit transactions.

A: Implicit transactions are automatically started by the database system for each SQL statement. Explicit transactions are explicitly started and managed using BEGIN TRANSACTION, COMMIT, and ROLLBACK statements.

43: How can you use the @@TRANCOUNT variable to manage nested transactions?

A: The @@TRANCOUNT variable is used to keep track of the number of nested transactions. You can use it to determine whether to commit or roll back a transaction based on the nesting level.

44: Write a SQL query to undo all changes made within a transaction and roll it back.

A: As mentioned earlier, you can use the ROLLBACK statement to undo changes made within a transaction and roll it back to the previous state:

ROLLBACK;

45: Write an SQL statement to create a view that combines data from two tables.

A:

CREATE VIEW EmployeeDetails AS

SELECT Employees.Name, Departments.DepartmentName

FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.ID;

46: How do you modify the structure of an existing view?

A: To modify an existing view, you can use the ALTER VIEW statement. For example:

```
ALTER VIEW YourViewName AS
SELECT NewColumns
FROM NewTables
WHERE NewConditions;
```

47: Write a query to list all views in a database.

A: You can retrieve information about views from the sys.views catalog view:

```
SELECT * FROM sys.views;
```

48: Explain the advantages of using views in SQL.

A: Views simplify complex queries, provide an additional layer of security by restricting direct table access, and can enhance performance by precomputing and caching results.

49: How can you drop a view from a database?

A: To drop a view, you can use the DROP VIEW statement. For example:

```
DROP VIEW YourViewName;
```

50: Write a SQL query to update data in a table through a view.

A: You can update data through a view as long as it meets certain criteria, such as having an updatable underlying query. For example:

```
UPDATE YourViewName
SET ColumnName = NewValue
WHERE Condition;
```

51: Explain the concept of indexed views (materialized views) and their benefits.

A: Indexed views are views that are physically stored as tables and can have indexes. They are used to improve query performance by precomputing and storing the results of complex queries.

52: How do you create an indexed view in SQL?

A: To create an indexed view, you can use the CREATE VIEW statement with the WITH SCHEMABINDING option, and then create a unique clustered index on the view. For example:

```
CREATE VIEW YourIndexedView
WITH SCHEMABINDING
AS
SELECT ...
GO
CREATE UNIQUE CLUSTERED INDEX IX_YourIndexedView ON YourIndexedView (Column1, Column2);
```

53: What is the difference between a view and a table in SQL?

A: A table stores data physically, while a view is a virtual table that is the result of a query. Views do not store data themselves but provide a convenient way to access and manipulate data from one or more tables.

54: How can you use views to restrict access to specific columns of a table?

A: You can create a view that includes only the columns you want to expose to users and grant them access to the view instead of the underlying table. This restricts their access to specific columns.

FAQ (Frequently Asked Questions):

1. Question: Write an SQL query to find the second highest salary from an "Employee" table.

Answer:

```
SELECT DISTINCT Salary
FROM Employee
ORDER BY Salary DESC
```

```
LIMIT 1 OFFSET 1;
```

2. Question: Write an SQL query to find the names of employees who earn more than their manager.

Answer:

```
SELECT e.Name
FROM Employee e, Employee m
WHERE e.ManagerId = m.Id AND e.Salary > m.Salary;
```

3. Question: Write an SQL query to find the third maximum salary from a "Salary" table.

Answer:

```
SELECT DISTINCT Salary
FROM Salary
ORDER BY Salary DESC
LIMIT 1 OFFSET 2;
```

4. Question: Write an SQL query to find the average salary of employees in each department.

Answer:

```
SELECT DepartmentId, AVG(Salary) AS AvgSalary
FROM Employee
GROUP BY DepartmentId;
```

5. Question: Write an SQL query to find the names of employees who have the same salary as the employee named "Joe".

Answer:

```
SELECT Name
FROM Employee
WHERE Salary = (SELECT Salary FROM Employee WHERE Name = 'Joe');
```

6. Question: Write an SQL query to find the employee with the highest salary in each department.

Answer:

```
SELECT DepartmentId, MAX(Salary) AS MaxSalary
FROM Employee
GROUP BY DepartmentId;
```

7. Question: Write an SQL query to find all duplicate emails in a "Person" table.

Answer:

```
SELECT Email
FROM Person
GROUP BY Email
HAVING COUNT(Email) > 1;
```

8. Question: Write an SQL query to find the nth highest salary from a "Salary" table.

Answer:

```
SELECT DISTINCT Salary
FROM Salary
ORDER BY Salary DESC
LIMIT 1 OFFSET n-1;
```

9. Question: Write an SQL query to find the second highest order number from an "Orders" table.

Answer:

```
SELECT DISTINCT OrderNumber
FROM Orders
ORDER BY OrderNumber DESC
LIMIT 1 OFFSET 1;
```

10. Question: Write an SQL query to find the department with the highest average salary.

Answer:

SELECT DepartmentId, AVG(Salary) AS AvgSalary

FROM Employee

GROUP BY DepartmentId

ORDER BY AvgSalary DESC LIMIT 1;

11. Question: Write an SQL query to find all customers who have made at least two orders in a "Customers" table.

Answer:

SELECT CustomerName F

ROM Customers

WHERE CustomerId IN (SELECT CustomerId FROM Orders GROUP BY CustomerId HAVING COUNT(OrderId) >= 2);

12. Question: Write an SQL query to find the difference between the number of customers and the number of orders in a "Customers" and "Orders" table.

Answer:

SELECT COUNT(DISTINCT CustomerId) - COUNT(DISTINCT OrderId) AS Difference

FROM Customers, Orders;

13. Question: Write an SQL query to find the average order value for each customer in an "Orders" table.

Answer:

SELECT CustomerId, AVG(OrderValue) AS AvgOrderValue

FROM (SELECT o.CustomerId, o.OrderId, SUM(od.Quantity * od.Price) AS OrderValue FROM Orders o JOIN OrderDetails od ON o.OrderId = od.OrderId GROUP BY o.CustomerId, o.OrderId) AS Subquery

GROUP BY CustomerId;

14. Question: Write an SQL query to find the products that have never been ordered.

Answer:

SELECT ProductName FROM Products

WHERE ProductId NOT IN (SELECT DISTINCT ProductId FROM OrderDetails);

15. Question: Write an SQL query to find the total number of employees in each department, including departments with zero employees.

Answer:

SELECT DepartmentId, COUNT(EmployeeId) AS TotalEmployees

FROM (SELECT DISTINCT DepartmentId, EmployeeId FROM Employee) AS Subquery

GROUP BY DepartmentId;

16. Question: Write an SQL query to find the number of students who scored higher than the average score in an "Exam" table.

Answer:

SELECT COUNT(StudentId) AS AboveAverage

FROM (SELECT StudentId, Score, AVG(Score) OVER () AS AvgScore FROM Exam) AS Subquery WHERE Score > AvgScore;

17. Question: Write an SQL query to find the total sales for each month in a "Sales" table.

Answer:

SELECT DATE_FORMAT(SaleDate, '%Y-%m') AS Month, SUM(SaleAmount) AS TotalSales

FROM Sales

GROUP BY Month;

18. Question: Write an SQL query to find the top N products by sales in an "OrderDetails" table.

Answer:

SELECT ProductId, SUM(Quantity * Price) AS TotalSales

FROM OrderDetails

GROUP BY ProductId

ORDER BY TotalSales DESC LIMIT N;

19. Question: Write an SQL query to find the first N customers who placed an order in a "Orders" table.

Answer:

SELECT DISTINCT CustomerId FROM Orders LIMIT N;

20. Question: Write an SQL query to find the average waiting time for a customer to place an order after their first visit in a "Customers" and "Orders" table.

Answer:

SELECT AVG(DATEDIFF(MIN(OrderDate), VisitDate)) AS AvgWaitingTime

FROM (SELECT C.CustomerId, C.VisitDate, O.OrderDate FROM Customers C LEFT JOIN (SELECT CustomerId, MIN(OrderDate) AS OrderDate FROM Orders GROUP BY CustomerId) O ON C.CustomerId = O.CustomerId) Subquery;

SCENARIO BASED QUESTIONS (INTERVIEWS):

1. Question:

You are given a "Customers" table with columns Id, Name, and Country. Write an SQL query to find the customers from the United States (Country = 'USA') who have made at least three orders. Display their names and the total number of orders they have made.

Answer:

SELECT c.Name, COUNT(o.Id) AS TotalOrders

FROM Customers c

JOIN Orders o ON c.Id = o.CustomerId

WHERE c.Country = 'USA'

GROUP BY c.Id, c.Name

HAVING COUNT(o.Id) >= 3;

2. Question:

You are given an "Employee" table with columns Id, Name, Salary, and DepartmentId. Write an SQL query to find the employees who have the highest salary in each department. Display their names, salaries, and department names.

Answer:

```
SELECT e.Name, e.Salary, d.Name AS DepartmentName
FROM Employee e
JOIN Department d ON e.DepartmentId = d.Id
WHERE (e.DepartmentId, e.Salary) IN (
    SELECT DepartmentId, MAX(Salary)
    FROM Employee
    GROUP BY DepartmentId
);
```

3. Question:

You are given a "Logs" table with columns Id, Status, and LogDate. Write an SQL query to find the consecutive login days for each user. A consecutive login day is defined as a day where the user logged in on the day immediately following their previous login day. Display the user's Id and the start date of consecutive login days.

Answer:

```
SELECT UserId, MIN(LogDate) AS StartDate
FROM (
    SELECT Id, UserId, LogDate,
        DATE_SUB(LogDate, INTERVAL ROW_NUMBER() OVER (PARTITION BY UserId
ORDER BY LogDate) DAY) AS ConsecutiveGroup
    FROM Logs
) AS ConsecutiveLogins
GROUP BY UserId, ConsecutiveGroup
HAVING COUNT(Id) >= 2;
```

4. Question:

You are given a "Products" table with columns ProductId, ProductName, and Price. Write an SQL query to find the top N most expensive products. Display their names and prices in descending order of price.

Answer:

```
SELECT ProductName, Price
FROM Products
ORDER BY Price DESC
LIMIT N;
```

5. Question:

You are given a "Trips" table with columns Id, Client_Id, Driver_Id, City_Id, Status, and Request_at. The "Trips" table holds all taxi trips, and each trip has a unique Id. Write an SQL query to find the cancellation rate of taxi requests made on the Request_at date in "2013-10-01," to two decimal places. The cancellation rate is calculated as canceled requests divided by total requests.

Answer:

```
SELECT
    ROUND(SUM(CASE WHEN Status = 'completed' THEN 0 ELSE 1 END) / COUNT(Id), 2) AS 'Cancellation Rate'
FROM Trips
WHERE DATE(Request_at) = '2013-10-01';
```