

SQL

SQL is a standard language for storing, manipulating, and retrieving data in databases.

1). What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

2). What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

3). RDBMS

RDBMS stands for Relational Database Management System.

A relational database management system (RDBMS) is a program used to create, update, and manage relational databases. Some of the most well-known RDBMSs include MySQL, PostgreSQL, MariaDB, Microsoft SQL Server, and Oracle Database.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries, and it consists of columns and rows.

Example: - **Select * from Customer**

Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, Customer Name, Contact Name, Address, City, Postal Code and Country. A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

4). what is the difference between SQL and MySQL?

MySQL is the underlying technology that stores the data, while SQL is the language you use to read, modify, and delete that data.

Several relational database management systems use SQL, including Microsoft SQL Server, which is also very popular.

5). what is the difference between rdbms and dbms?

- RDBMS stores data in the form of tables.
- DBMS stores data in the form of files .
- Single users are supported by DBMS.
- Multiple users are supported by RDBMS.
- Client-server architecture is not supported by DBMS.
- Although it is supported by RDBMS.

6). Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database.
- **UPDATE** - updates data in a database.
- **DELETE** - deletes data from a database.
- **INSERT INTO** - inserts new data into a database.
- **CREATE DATABASE** - creates a new database.
- **ALTER DATABASE** - modifies a database.
- **CREATE TABLE** - creates a new table.
- **ALTER TABLE** - modifies a table.
- **DROP TABLE** - deletes a table.
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index.

7). SQL SELECT Statement

The **SELECT** statement is used to select data from a database.

Example: - **Select * from Customers**

Select Name, Address from Customers

8). SELECT DISTINCT Statement

The **SELECT DISTINCT** statement is used to return only distinct (different) values.

Example: - **SELECT DISTINCT column1, column2, ...FROM table_name;**

9).SQL WHERE Clause

The **WHERE** clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Example: - **SELECT * FROM Customers WHERE Country='Mexico';**

SELECT * FROM Customers WHERE Country='Mexico' and Name='Narasimha'

SELECT * FROM Customers WHERE Country='Mexico' or Name='Narasimha'

SELECT * FROM Customers WHERE Country not in ('Mexico') and Name='Narasimha'

The following operators can be used in the **WHERE** clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

10). SQL ORDER BY

The **ORDER BY** keyword is used to sort the result-set in ascending or descending order.

Example: - **Select * from Customer order by Name Asc/Desc.**

11). SQL INSERT INTO Statement

The **INSERT INTO** statement is used to insert new records in a table.

INSERT INTO Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

12). What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Example: - **ISNULL & IS NOT NULL**

13). SQL UPDATE Statement

The **UPDATE** statement is used to modify the existing records in a table.

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

14). SQL DELETE Statement

The **DELETE** statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

15). SQL SELECT TOP Clause

The **SELECT TOP** clause is used to specify the number of records to return.

The **SELECT TOP** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

```
SELECT TOP 3 * FROM Customers;
```

16). Aggregate Functions

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the **GROUP BY** clause of the **SELECT** statement. The **GROUP BY** clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

- **MIN()** - returns the smallest value within the selected column
- **MAX()** - returns the largest value within the selected column
- **COUNT()** - returns the number of rows in a set
- **SUM()** - returns the total sum of a numerical column
- **AVG()** - returns the average value of a numerical column

Aggregate functions ignore null values (except for **COUNT()**).

17). SQL MIN() and MAX() Functions

The **MIN()** function returns the smallest value of the selected column.

The **MAX()** function returns the largest value of the selected column.

```
SELECT MIN(Price) FROM Products
```

```
SELECT MAX(Price) FROM Products
```

18). SQL COUNT () Function

The **COUNT ()** function returns the number of rows that matches a specified criterion.

```
SELECT COUNT (*) FROM Products;
```

```
SELECT COUNT (column_name) FROM table_name WHERE condition;
```

19).SQL SUM () Function

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

```
SELECT SUM(Quantity) FROM OrderDetails;
```

20).SQL AVG () Function

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

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

21).SQL LIKE Operator

The **LIKE** operator is used in a **WHERE** clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the **LIKE** operator:

- The percent sign **%** represents zero, one, or multiple characters
- The underscore sign **_** represents one, single character

```
SELECT * FROM Customers WHERE CustomerName LIKE 'a%';
```

22). SQL IN Operator

The **IN** operator allows you to specify multiple values in a **WHERE** clause.

The **IN** operator is a shorthand for multiple **OR** conditions.

```
SELECT * FROM Customers  
WHERE Country IN ('Germany', 'France', 'UK')
```

23). SQL BETWEEN Operator

The **BETWEEN** operator selects values within a given range. The values can be numbers, text, or dates.

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
```

24). SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the **AS** keyword.

```
SELECT CustomerID AS ID FROM Customers;
```

25). SQL JOIN

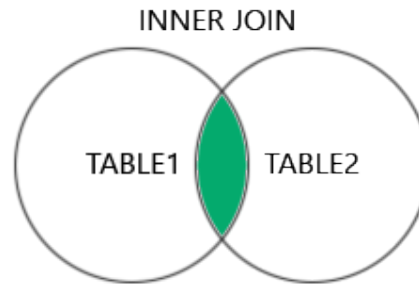
A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

26). INNER JOIN

The **INNER JOIN** keyword selects records that have matching values in both tables.

```
SELECT ProductID, ProductName, CategoryName FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```



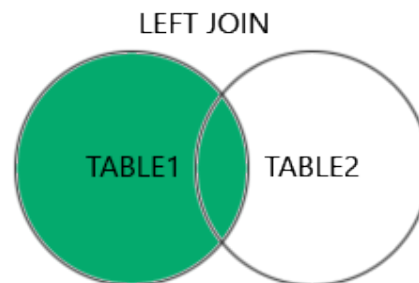
27). LEFT JOIN Keyword

The **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



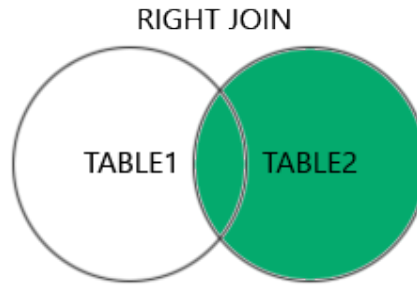
28). RIGHT JOIN Keyword

The **RIGHT JOIN** keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases **RIGHT JOIN** is called **RIGHT OUTER JOIN**.



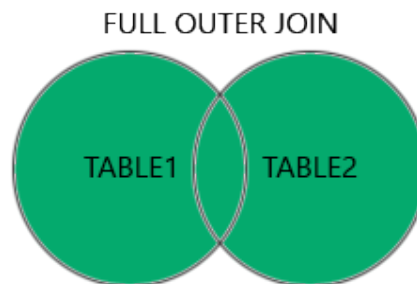
29). FULL OUTER JOIN Keyword

The **FULL OUTER JOIN** keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: **FULL OUTER JOIN** and **FULL JOIN** are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



30). Self-Join

A self join is a regular join, but the table is joined with itself.

Self Join Syntax

```
SELECT column_name(s) FROM table1 T1, table1 T2 WHERE condition;
```

31). SQL UNION Operator

The **UNION** operator is used to combine the result-set of two or more **SELECT** statements.

- Every **SELECT** statement within **UNION** must have the same number of columns
- The columns must also have similar data types
- The columns in every **SELECT** statement must also be in the same order

UNION Syntax


```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

The **UNION** operator selects only distinct values by default. To allow duplicate values, use **UNION ALL** :

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

32). SQL GROUP BY Statement

The **GROUP BY** statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The **GROUP BY** statement is often used with aggregate functions (**COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

33). SQL HAVING Clause

The **HAVING** clause was added to SQL because the **WHERE** keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

34). SQL EXISTS Operator

The **EXISTS** operator is used to test for the existence of any record in a subquery.

The **EXISTS** operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

35). SQL SELECT INTO Statement

The **SELECT INTO** statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

36). SQL INSERT INTO SELECT Statement

The **INSERT INTO SELECT** statement copies data from one table and inserts it into another table.

The **INSERT INTO SELECT** statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

37). SQL CASE Expression

The **CASE** expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the **ELSE** clause.

If there is no **ELSE** part and no conditions are true, it returns NULL.

CASE Syntax

CASE

```
WHEN condition1 THEN result1
WHEN condition2 THEN result2
WHEN conditionN THEN resultN
ELSE result
END;
```

38). 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.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'London';
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
```

39). SQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators -

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

40). SQL CREATE DATABASE Statement

The **CREATE DATABASE** statement is used to create a new SQL database.

Syntax

CREATE DATABASE *dbname*;

41). SQL DROP DATABASE Statement

The **DROP DATABASE** statement is used to drop an existing SQL database.

Syntax

DROP DATABASE *dbname*;

42). SQL CREATE TABLE Statement

The **CREATE TABLE** statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

43). **SQL DROP TABLE Statement**

The **DROP TABLE** statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

44). **ALTER TABLE Statement**

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

The **ALTER TABLE** statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

```
ALTER TABLE table_name  
RENAME COLUMN old_name to new_name;
```

45). **SQL Constraints**

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- [NOT NULL](#) - Ensures that a column cannot have a NULL value
- [UNIQUE](#) - Ensures that all values in a column are different
- [PRIMARY KEY](#) - A combination of a **NOT NULL** and **UNIQUE**. Uniquely identifies each row in a table
- [FOREIGN KEY](#) - Prevents actions that would destroy links between tables
- [CHECK](#) - Ensures that the values in a column satisfies a specific condition
- [DEFAULT](#) - Sets a default value for a column if no value is specified
- [CREATE INDEX](#) - Used to create and retrieve data from the database very quickly

46). SQL NOT NULL Constraint

By default, a column can hold NULL values.

The **NOT NULL** constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);
```

47). SQL UNIQUE Constraint

The **UNIQUE** constraint ensures that all values in a column are different.

Both the **UNIQUE** and **PRIMARY KEY** constraints provide a guarantee for uniqueness for a column or set of columns.

A **PRIMARY KEY** constraint automatically has a **UNIQUE** constraint.

However, you can have many **UNIQUE** constraints per table, but only one **PRIMARY KEY** constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a **UNIQUE** constraint on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);
```

48). SQL PRIMARY KEY Constraint

The **PRIMARY KEY** constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a **PRIMARY KEY** on the "ID" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (ID)  
);
```

49). SQL FOREIGN KEY Constraint

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

A **FOREIGN KEY** is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](#) in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

50). SQL CHECK Constraint

The **CHECK** constraint is used to limit the value range that can be placed in a column.

If you define a **CHECK** constraint on a column it will allow only certain values for this column.

If you define a **CHECK** constraint on a table it can limit the values in certain columns based on values in other columns in the row.

51). SQL DEFAULT Constraint

The **DEFAULT** constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

52). SQL CREATE INDEX Statement

The **CREATE INDEX** statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name  
ON table_name (column1, column2, ...);
```

53). Difference between clustered and non clustered index?

A clustered index is used to define the order or to sort the table or arrange the data by alphabetical order just like a dictionary. A non-clustered index collects the data at one place and records at another place. It is faster than a non-clustered index.

54). What is Cursor? How to use a 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.

Working with SQL Cursor:

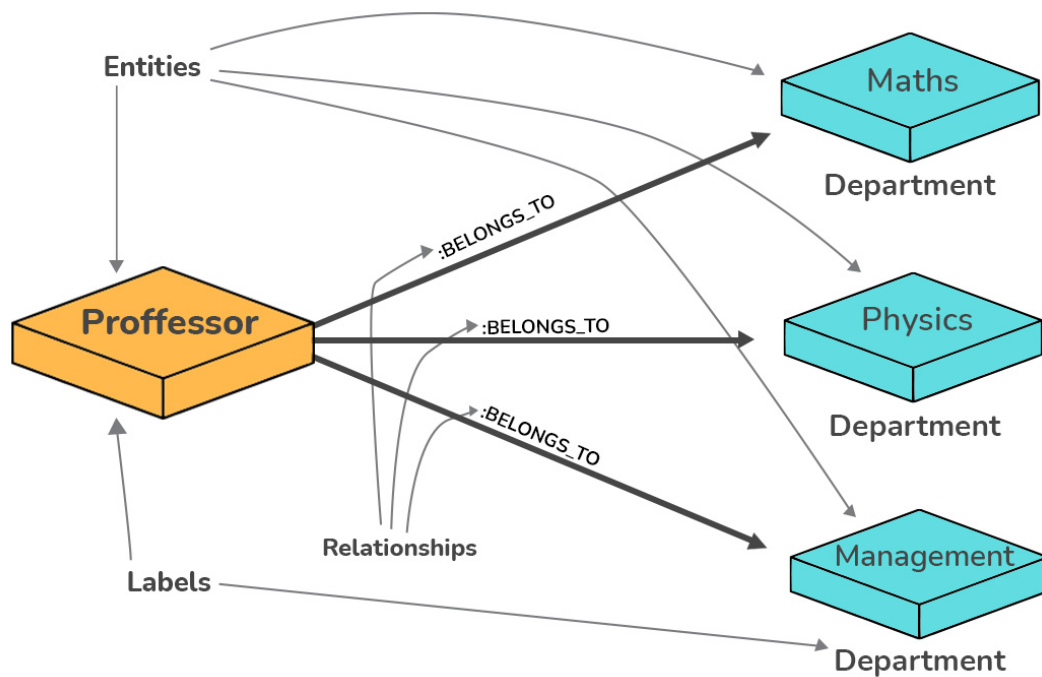
1. **DECLARE** a cursor after any variable declaration. The cursor declaration must always be associated with a SELECT Statement.
2. Open cursor to initialize the result set. The **OPEN** statement must be called before fetching rows from the result set.
3. **FETCH** statement to retrieve and move to the next row in the result set.
4. Call the **CLOSE** statement to deactivate the cursor.
5. Finally use the **DEALLOCATE** statement to delete the cursor definition and release the associated resources.

```
DECLARE @name VARCHAR(50) /* Declare All Required Variables */
DECLARE db_cursor CURSOR FOR /* Declare Cursor Name*/
SELECT name
FROM myDB.students
WHERE parent_name IN ('Sara', 'Ansh')
OPEN db_cursor /* Open cursor and Fetch data into @name */
FETCH next
FROM db_cursor
INTO @name
CLOSE db_cursor /* Close the cursor and deallocate the resources */
DEALLOCATE db_cursor
```

55. 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.



56. 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.

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

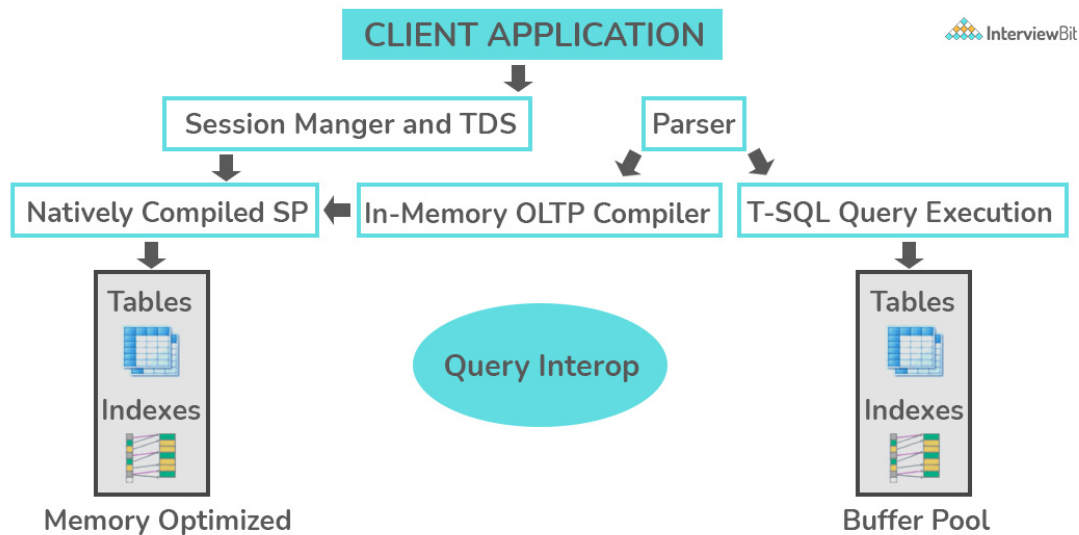
The user-defined functions in SQL are like functions in any other programming language that accept parameters, perform complex calculations, and return a value. They are written to use the logic repetitively whenever required. There are two types of SQL user-defined functions:

- **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.

58. 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.

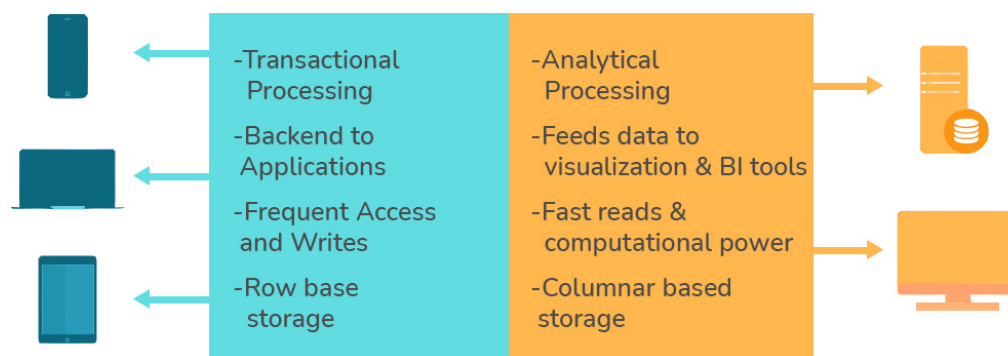
Database queries are usually simple, require sub-second response times, and return relatively few records. Here is an insight into the working of an OLTP system [Note - The figure is not important for interviews] -



59. 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.



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

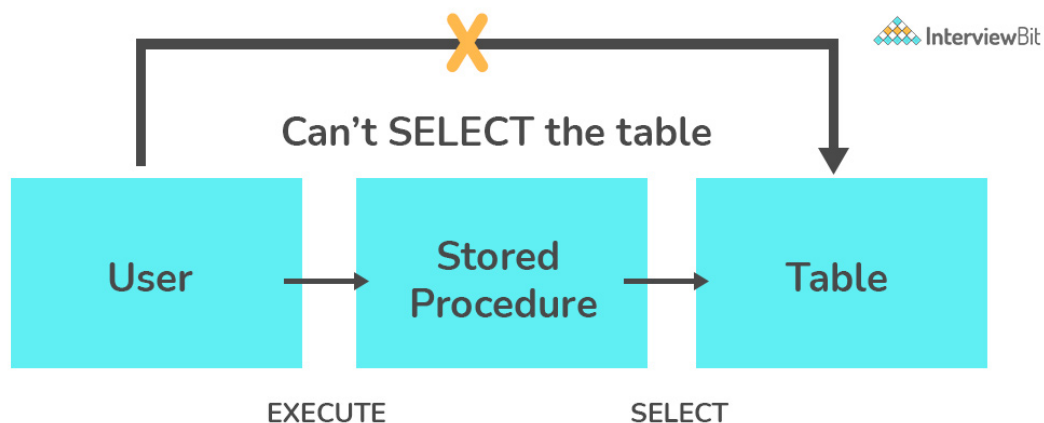
Collation refers to a set of rules that determine how data is sorted and compared. Rules defining the correct character sequence are used to sort the character data. It incorporates options for specifying case sensitivity, accent marks, kana character types, and character width. Below are the different 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.

61. 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 ;
```



62. 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 Output variables */
  OUT total INT
) BEG
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 */

```

40. 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;

```

41. 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 '%K%'
```

- 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

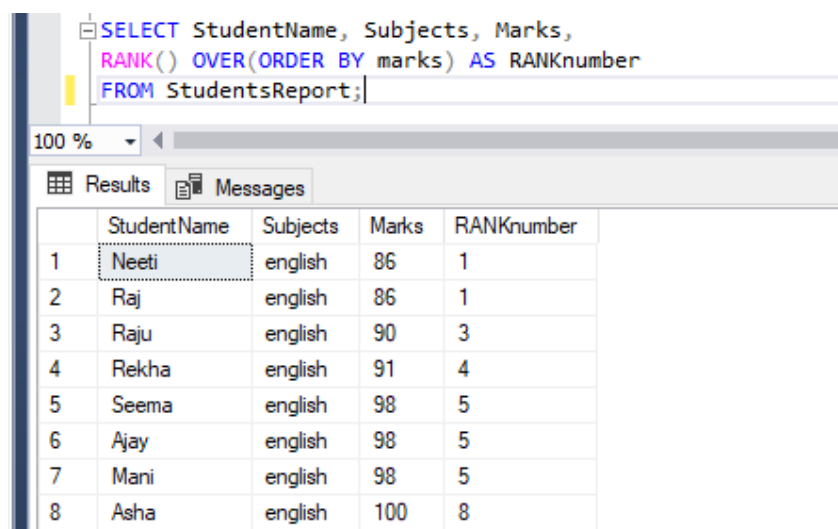
The _ wildcard plays an important role as a limitation when it matches exactly one character. It limits the length and position of the matched results. For example -

```
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 '____'
```

65). RANK Function

```
SELECT StudentName, Subjects, Marks,  
RANK() OVER(ORDER BY marks) AS RANKnumber  
FROM StudentsReport;  
SQL
```



The screenshot shows a SQL query editor with the following query:

```
SELECT StudentName, Subjects, Marks,  
RANK() OVER(ORDER BY marks) AS RANKnumber  
FROM StudentsReport;
```

Below the query, there is a 'Results' tab showing a table with 5 columns: StudentName, Subjects, Marks, and RANKnumber. The table contains 8 rows of data, with the first row highlighted.

	StudentName	Subjects	Marks	RANKnumber
1	Neeti	english	86	1
2	Raj	english	86	1
3	Raju	english	90	3
4	Rekha	english	91	4
5	Seema	english	98	5
6	Ajay	english	98	5
7	Mani	english	98	5
8	Asha	english	100	8

```
66). SELECT StudentName, Subjects, Marks,  
ROW_NUMBER() OVER(ORDER BY Marks) Rownumber
```

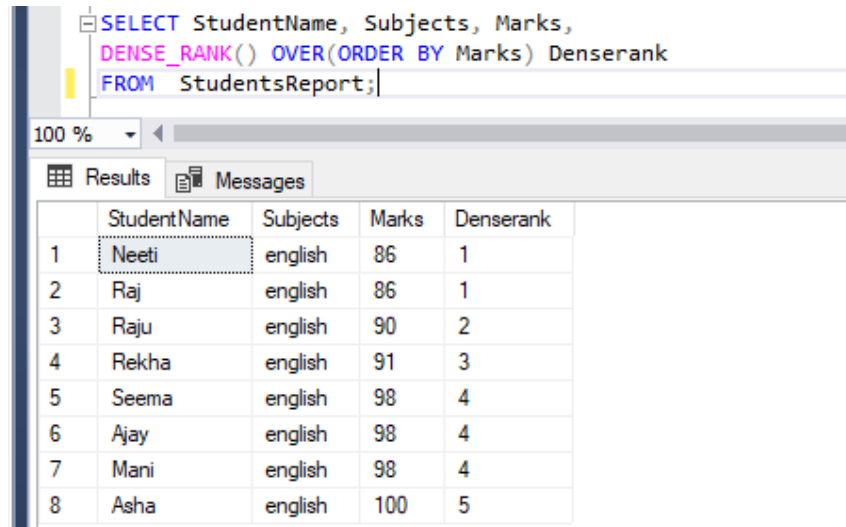
```
FROM StudentsReport;
```

SQL

Copy

```
67). SELECT StudentName, Subjects, Marks,  
DENSE_RANK() OVER(ORDER BY Marks) Denserank  
FROM StudentsReport;
```

SQL



The screenshot shows a SQL query editor with a query window and a results window. The query window contains the following SQL code:

```
SELECT StudentName, Subjects, Marks,  
DENSE_RANK() OVER(ORDER BY Marks) Denserank  
FROM StudentsReport;
```

The results window displays the output of the query as a table with 5 columns: StudentName, Subjects, Marks, and Denserank. The table contains 8 rows of data, sorted by Marks in descending order. The first two rows (Neeti and Raj) have the same Marks (86) and are both ranked 1. The subsequent rows have unique Marks and are ranked accordingly.

	StudentName	Subjects	Marks	Denserank
1	Neeti	english	86	1
2	Raj	english	86	1
3	Raju	english	90	2
4	Rekha	english	91	3
5	Seema	english	98	4
6	Ajay	english	98	4
7	Mani	english	98	4
8	Asha	english	100	5

68). What is SQL injection?

SQL injection is a technique used to exploit user data through web page inputs by injecting SQL commands as statements. Basically, these statements can be used to manipulate the application's web server by malicious users.

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

69). Explain WITH clause in SQL?

The WITH clause provides a way relationship of defining a temporary relationship whose definition is available only to the query in which the with clause occurs. SQL applies predicates in the WITH clause after groups have been formed, so aggregate functions may be used.

70). What is a Cursor?

The cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML operations on the Table by the User. Cursors are used to store Database Tables

71). Write down various types of relationships in SQL?

There are various relationships, namely:

- One-to-One Relationship.
- One to Many Relationships.
- Many to One Relationship.
- Self-Referencing Relationship.

72. What is a trigger?

The trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.(before/after)-insert,update,delete.

What types of SQL commands (or SQL subsets) do you know?

- **Data Definition Language (DDL)** – to define and modify the structure of a database.
- **Data Manipulation Language (DML)** – to access, manipulate, and modify data in a database.
- **Data Control Language (DCL)** – to control user access to the data in the database and give or revoke privileges to a specific user or a group of users.
- **Transaction Control Language (TCL)** – to control transactions in a database.
- **Data Query Language (DQL)** – to perform queries on the data in a database to retrieve the necessary information from it.
- **DDL:** CREATE, ALTER TABLE, DROP, TRUNCATE, and ADD COLUMN
- **DML:** UPDATE, DELETE, and INSERT
- **DCL:** GRANT and REVOKE
- **TCL:** COMMIT, SET TRANSACTION, ROLLBACK, and SAVEPOINT
- **DQL:** – SELECT