SQL

[Structured Query Language]

QUESTIONS COVERED:-

1. What is SQL?
2. What is Database ?
3. What are the differences between SQL and PL/SQL?
4. What is the difference between BETWEEN and IN operators in SQL?
5. What is the use of LIKE operator?
6. What is the use of 'WHERE' Clause?
7. What is the use of 'Having' Clause?
8. What are SQL Commands? Explain types of SQL Commands
9. What are normalization and denormalization and why do we need them?
10. What are Nested Queries in SQL?
11. What are different types of Normalization?
12. What is Stored Procedures in SQL ?
13. What are different types of case manipulation functions available in SQL.
14. What is the difference between CHAR and VARCHAR2 datatype in SQL?
15. What is the use of CREATE, INSERT INTO, UPDATE and DELETE Clauses?
16. What is the use of ADD, DROP and MODIFY Commands?
17. What are VIEWS in SQL?
18. What are JOINS in SQL?
19. What is the use of GROUP BY Clause?
20. What are Aggregate Functions?
21. What is Cursor in SQL ?
22. What is the difference between Implicit and Explicit Cursor?
23. What is the difference between VIEW and CURSOR in SQL?
24. What are the advantages of PL/SQL functions?
25. Explain BETWEEN and IN Clause.
26. What is the difference between DROP and TRUNCATE?
27. What are Constraints in SQL?
28. What is a TRIGGER?
29. What is the use of LIMIT and OFFSET in SQL?
30. What are different types of operators present in SQL?

What is SQL?

Structured Query Language is a computer language that we use to interact with a relational database.SQL is a tool for organizing, managing, and retrieving archived data from a computer database. The original name was given by IBM as Structured English Query Language, abbreviated by the acronym SEQUEL. When data needs to be retrieved from a database, SQL is used to make the request. The DBMS processes the SQL query retrieves the requested data and returns it to us. Rather, SQL statements describe how a collection of data should be organized or what data should be extracted or added to the database.

In common usage, SQL encompasses DDL and DML commands for create, updates, modified or other operations on database structure.

SQL uses:

- **Data definition:** It is used to define the structure and organization of the stored data and relationships among the stored data items.
- **Data retrieval:** SQL can also be used for data retrieval.
- **Data manipulation:** If the user wants to add new data, remove data, or modifying in existing data then SQL provides this facility also.
- **Access control:** SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- **Data sharing:** SQL is used to coordinate data sharing by concurrent users, ensuring that changes made by one user do not inadvertently wipe out changes made at nearly the same time by another user.

SQL also differs from other computer languages because it describes what the user wants the computer to do rather than how the computer should do it. (In more technical terms, SQL is a declarative or descriptive language rather than a procedural one.) SQL contains no IF statement for testing conditions, and no GOTO, DO, or FOR statements for program flow control. Rather, SQL statements describe how a collection of data is to be organized, or what data is to be retrieved or added to the database. The sequence of steps to do those tasks is left for the DBMS to determine.

Features of SQL:

- SQL may be utilized by quite a number of users, which include people with very little programming experience.
- SQL is a Non-procedural language.

- We can without difficulty create and replace databases in SQL. It isn't a time-consuming process.
- SQL is primarily based totally on ANSI standards.
- SQL does now no longer have a continuation individual.
- SQL is entered into SQL buffer on one or extra lines.
- SQL makes use of a termination individual to execute instructions immediately. It makes use of features to carry out a few formatting.
- It uses functions to perform some formatting.

Rules for SQL:

- A ';' is used to end SQL statements.
- Statements may be split across lines but keywords may not.
- Identifiers, operator names, literals are separated by one or more spaces or other delimiters.
- A comma(,) separates parameters without a clause.
- A space separates a clause.
- Reserved words can not be used as identifiers unless enclosed with double quotes.
- Identifiers can contain up to 30 characters.
- Identifiers must start with an alphabetic character.
- Characters and date literals must be enclosed within single quotes.
- Numeric literals can be represented by simple values.
- Comments may be enclosed between /* and */ symbols and maybe multi-line.

Role of SQL :

SQL plays many different roles:

- SQL is an interactive question language. Users type SQL instructions into an interactive SQL software to retrieve facts and show them on the screen, presenting a convenient, easy-to-use device for ad hoc database queries.
- SQL is a database programming language. Programmers embed SQL instructions into their utility packages to access the facts in a database. Both user-written packages and database software packages (consisting of document writers and facts access tools) use this approach for database access.
- SQL is a client/server language. Personal computer programs use SQL to communicate over a network with database servers that save shared facts. This client/server architecture is utilized by many famous enterprise-class applications.
- SQL is an Internet facts access language. Internet net servers that have interaction with company facts and Internet utility servers all use SQL as a widespread language for getting access to company databases, frequently through embedding SQL database get entry to inside famous scripting languages like PHP or Perl.

- SQL is a distributed database language. Distributed database control structures use SQL to assist distribute facts throughout many linked pc structures. The DBMS software program on every gadget makes use of SQL to speak with the opposite structures, sending requests for facts to get entry to.
- SQL is a database gateway language. In a pc community with a mixture of various DBMS products, SQL is frequently utilized in a gateway that lets in one logo of DBMS to speak with every other logo. SQL has for this reason emerged as a useful, effective device for linking people, pc packages, and pc structures to the facts saved in a relational database.

Finally, SQL is not a particularly structured language, especially when compared with highly structured languages such as C, Pascal, or Java. Instead, SQL statements resemble English sentences, complete with “noise words” that don’t add to the meaning of the statement but make it read more naturally. The SQL has quite a few inconsistencies and also some special rules to prevent you from constructing SQL statements that look perfectly legal but that don’t make sense.

Despite the inaccuracy of its name, SQL has emerged as the standard language for using relational databases. SQL is both a powerful language and one that is relatively easy to learn. So SQL is a database management language. The database administrator answerable for handling a minicomputer or mainframe database makes use of SQL to outline the database shape and manipulate get entry to to the saved data.

What is Database ?

The **Database** is an essential part of our life. As we encounter several activities that involve our interaction with databases, for example in the bank, in the railway station, in school, in a grocery store, etc. These are the instances where we need to store a large amount of data in one place and fetch these data easily.

A database is a collection of data that is organized, which is also called structured data. It can be accessed or stored in a computer system. It can be managed through a Database Management System (DBMS), a software used to manage data. Database refers to related data in a structured form.

In a database, data is organized into tables consisting of rows and columns and it is indexed so data can be updated, expanded, and deleted easily. Computer databases typically contain file records data like transactions money in one bank

account to another bank account, sales and customer details, fee details of students, and product details. There are different kinds of databases, ranging from the most prevalent approach, the relational database, to a distributed database, cloud database, and NoSQL databases.

- **Relational Database:**
  A relational database is made up of a set of tables with data that fits into a predefined category.

- **Distributed Database:**
  A distributed database is a database in which portions of the database are stored in multiple physical locations, and in which processing is dispersed or replicated among different points in a network.

- **Cloud Database:**
  A cloud database is a database that typically runs on a cloud computing platform. Database service provides access to the database. Database services make the underlying software-stack transparent to the user.

These interactions are the example of a traditional database where data is of one type-that is textual. In advancement of technology has led to new applications of database systems. New media technology has made it possible to store images, video clips. These essential features are making multimedia databases.

Nowadays, people are becoming smart - before taking any decisions they analyze facts and figures related to it, which come from these databases. As the databases have made it easier to manage information, we are able to catch criminals and do deep research.

What are the differences between SQL and PL/SQL?

SQL	PL/SQL
SQL is a query execution or commanding language	PL/SQL is a complete programming language
SQL is a data-oriented language.	PL/SQL is a procedural language
SQL is very declarative in nature.	PL/SQL has a procedural nature.
It is used for manipulating data.	It is used for creating applications.
We can execute one statement at a time in SQL	We can execute blocks of statements in PL/SQL
SQL tells databases, what to do?	PL/SQL tells databases how to do.
We can embed SQL in PL/SQL	We can not embed PL/SQL in SQL

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.

What is the use of LIKE operator?

The **LIKE** operator of SQL is used for this purpose. It is used to fetch filtered data by searching for a particular pattern in the where clause.
The Syntax for using LIKE is,

SELECT column1,column2 **FROM** table_name **WHERE** column_name **LIKE** pattern;

LIKE: operator name
pattern: exact value extracted from the pattern to get related data in
result set.
For Example: Find all employees from the table 'Employees' whose name start with an 'A'.
The required query is:
SELECT * FROM Employees WHERE EmpName like 'A%' ;

What is the use of 'WHERE' Clause?

WHERE keyword is used for fetching filtered data in a result set.
- It is used to fetch data according to a particular criteria.
- WHERE keyword can also be used to filter data by matching patterns.

Basic Syntax:

SELECT column1,column2 FROM table_name WHERE column_name operator value;
column1 , column2: fields int the table
table_name: name of table
column_name: name of field used for filtering the data
operator: operation to be considered for filtering
value: exact value or pattern to get related data in result

List of operators that can be used with where clause:

operator	description
>	Greater Than
>=	Greater than or Equal to
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

EXAMPLE -

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

Queries

To fetch record of students with age equal to 20

SELECT * FROM Student WHERE Age=20;

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

What is the use of 'Having' Clause?

Having clause extracts the rows based on the conditions given by the user in the query. Having clause has to be paired with the group by clause in order to extract the data. Otherwise, an error is produced.

Syntax:

select *select_list* from *table_name*
group by *group_list*
having *conditions*

Example:

select *roll_number*
from *student*
having name like 'R%'

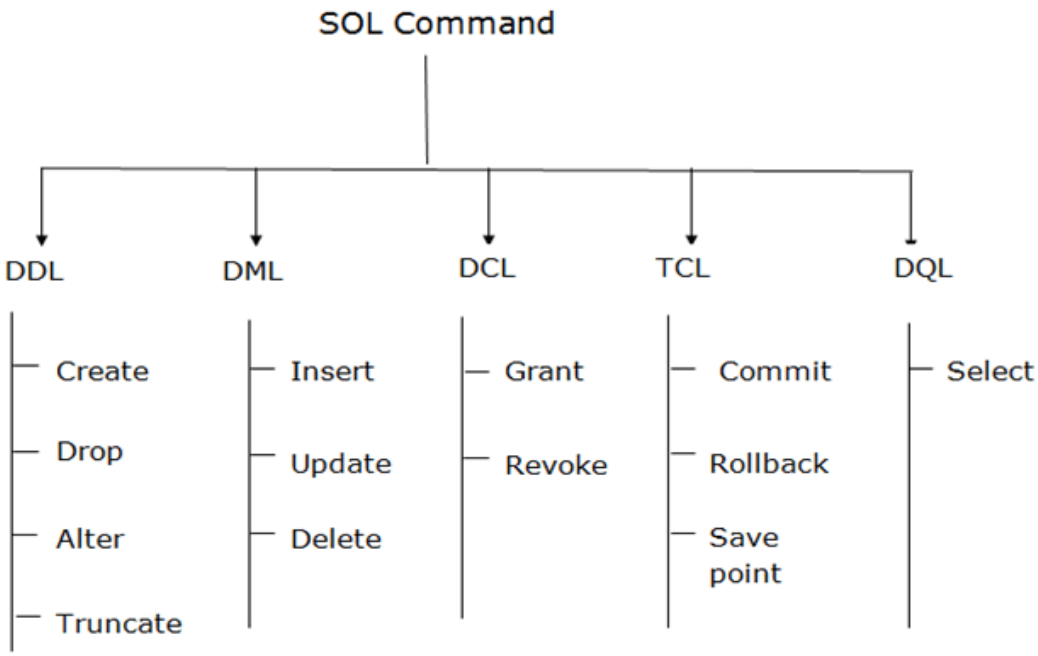
What are SQL Commands? Explain types of SQL Commands

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database, and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL - Data Definition Language
2. DQl - Data Query Language
3. DML - Data Manipulation Language
4. DCL - Data Control Language

Though many resources claim there to be another category of SQL clauses **TCL - Transaction Control Language**. So we will see in detail about TCL as well.



1. **DDL(Data Definition Language):** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

   Examples of DDL commands:
   - CREATE - is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
   - DROP - is used to delete objects from the database.
   - ALTER -is used to alter the structure of the database.
   - TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
   - COMMENT –is used to add comments to the data dictionary.

- RENAME –is used to rename an object existing in the database.

2. DQL (Data Query Language) :

DML statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- SELECT – is used to retrieve data from the database.

3. **DML(Data Manipulation Language):** The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

   Examples of DML:
   - INSERT – is used to insert data into a table.
   - UPDATE - is used to update existing data within a table.
   - DELETE – is used to delete records from a database table.

4. **DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

   Examples of DCL commands:
   - **GRANT** -gives user's access privileges to the database.
   - **REVOKE**-withdraw user's access privileges given by using the GRANT command.

5. **TCL(transaction Control Language):** TCL commands deal with the transaction within the database.

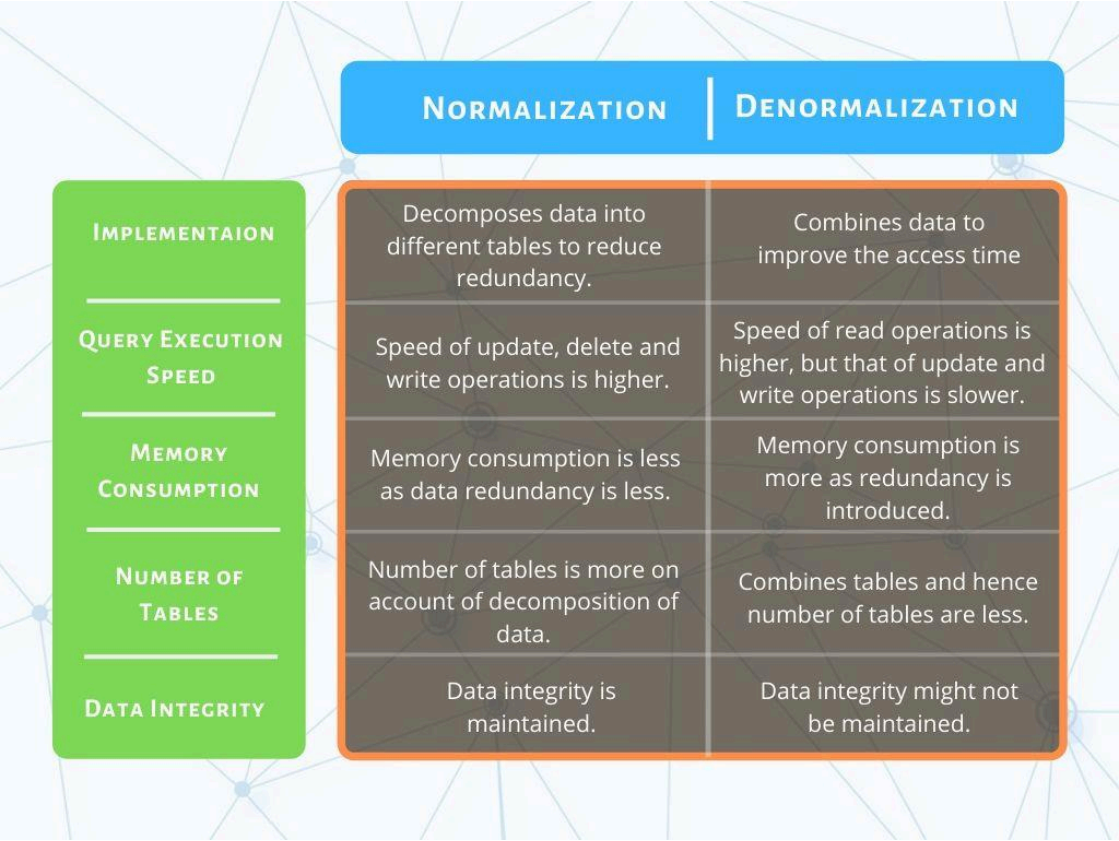
   Examples of TCL commands:
   - **COMMIT**– commits a Transaction.
   - **ROLLBACK**– rollbacks a transaction in case of any error occurs.
   - **SAVEPOINT** –sets a savepoint within a transaction.
   - **SET TRANSACTION** –specify characteristics for the transaction.

What are normalization and denormalization and why do we need them?

Normalization: Normalization is the method used in a database to reduce the data redundancy and data inconsistency from the table. It is the technique in which Non-redundancy and consistency data are stored in the set schema. By using normalization the number of tables is increased instead of decreased.

Denormalization: Denormalization is also the method that is used in a database. It is used to add redundancy to execute the query quickly. It is a technique in which data are combined to execute the query quickly. By using denormalization the number of tables is decreased which oppose to the normalization.

Difference between Normalization and Denormalization:



What are Nested Queries in SQL?

In nested queries, a query is written inside a query. The result of the inner query is used in the execution of the outer query. We will use **STUDENT, COURSE,**

STUDENT_COURSE tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of the inner query is independent of the outer query, but the result of the inner query is used in the execution of the outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

  IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_ID**s for finding **S_ID**s from **STUDENT_COURSE** TABLE.

  STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'

  Select **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME** = 'DBMS'

STEP 2: Using **C_ID** of step 1 for finding **S_ID**
Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME**='DBMS');

The inner query will return a set with members C1 and C3 and the outer query will return those **S_ID**s for which **C_ID** is equal to any member of the set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of **STUDENT**s who have either enrolled in 'DSA' or 'DBMS', it can be done as:

Select S_NAME from **STUDENT** where **S_ID** IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

NOT IN: If we want to find out **S_ID**s of **STUDENT**s who have neither enrolled in 'DSA' nor in 'DBMS', it can be done as:

Select **S_ID** from **STUDENT** where **S_ID** NOT IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S_ID**s where **S_ID** is not a member of set (S1, S2 and S4). So it will return S3.

- **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out **S_NAME** of **STUDENT**s who are enrolled in **C_ID** 'C1', it can be done with the help of co-related nested query as:

  Select S_NAME from **STUDENT** S where EXISTS
  (select * from **STUDENT_COURSE** SC where S.**S_ID**=SC.**S_ID** and SC.**C_ID**='C1');

For each row of **STUDENT** S, it will find the rows from **STUDENT_COURSE** where S.**S_ID** = SC.**S_ID** and SC.**C_ID**='C1'. If for a **S_ID** from **STUDENT** S, atleast a row exists in **STUDENT_COURSE** SC with **C_ID**='C1', then inner query will return true and corresponding **S_ID** will be returned as output.

What are different types of Normalization?

There are four types of normalization:
1. 1NF: It is known as the first normal form and is the simplest type of normalization that you can implement in a database. A table to be in its first normal form should satisfy the following conditions:
○ Every column must have a single value and should be atomic.
○ Duplicate columns from the same table should be removed.
○ Separate tables should be created for each group of related data and each row should be identified with a unique column.

2. 2NF : It is known as the second normal form. A table to be in its second normal form should satisfy the following conditions:
○ The table should be in its 1NF i.e. satisfy all the conditions of 1NF.
○ Every non-prime attribute of the table should be fully functionally dependent on the primary key i.e. every non-key attribute should be dependent on the primary key in such a way that if any key element is deleted then even the non_key element will be saved in the database.
3. 3NF : It is known as the third normal form. A table to be in its second normal form should satisfy the following conditions:
○ The table should be in its 2NF i.e. satisfy all the conditions of 2NF.
○ There is no transitive functional dependency of one attribute on any attribute in the same table.

4. BCNF: BCNF stands for Boyce-Codd Normal Form and is an advanced form of

3NF. It is also referred to as 3.5NF for the same reason. A table to be in its BCNF normal form should satisfy the following conditions:
○ The table should be in its 3NF i.e. satisfy all the conditions of 3NF.
○ For every functional dependency of any attribute A on B (A->B), A should be the super key of the table. It simply implies that A can't be a non-prime attribute if B is a prime attribute

What is Stored Procedures in SQL ?

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

Syntax : Creating a Procedure

```
CREATE or REPLACE PROCEDURE name(parameters)

IS

variables;

BEGIN

//statements;

END;
```

The most important part is parameters. Parameters are used to pass values to the Procedure. There are 3 different types of parameters, they are as follows:

1. **IN:**
   This is the Default Parameter for the procedure. It always receives the values from calling program.
2. **OUT:**
   This parameter always sends the values to the calling program.

3. **IN OUT:**
   This parameter performs both the operations. It Receives value from as well as sends the values to the calling program.

Example:
Imagine a table named with emp_table stored in Database. We are Writing a Procedure to update a Salary of Employee with 1000.

```
CREATE or REPLACE PROCEDURE INC_SAL(eno IN NUMBER, up_sal OUT NUMBER)

IS

BEGIN

UPDATE emp_table SET salary = salary+1000 WHERE emp_no = eno;

COMMIT;

SELECT sal INTO up_sal FROM emp_table WHERE emp_no = eno;

END;
```

- Declare a Variable to Store the value coming out from Procedure :

```
VARIABLE v NUMBER;
```

- Execution of the Procedure:

```
EXECUTE INC_SAL(1002, :v);
```

- To check the updated salary use SELECT statement:

```
 SELECT * FROM emp_table WHERE emp_no = 1002;
```

- or Use print statement :

```
PRINT :v
```

What are 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')*

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.

What is the use of CREATE, INSERT INTO, UPDATE and DELETE Clauses?

1. **CREATE Clause**

   There are two CREATE statements available in SQL:

   0. CREATE DATABASE
   1. CREATE TABLE

CREATE DATABASE

A **Database** is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database. The **CREATE DATABASE** statement is used to create a new database in SQL.
Syntax:

```
CREATE DATABASE database_name;

database_name: name of the database.
```

Example Query: This query will create a new database in SQL and name the database as *university*.

```
CREATE DATABASE university;
```

CREATE TABLE

We have learned above about creating databases. Now to store the data we need a table to do that. The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data, etc. Let us now dive into details on how to use the CREATE TABLE statement to create tables in SQL.

Syntax:

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

table_name:  name of the table.
column1 name of the first column.
data_type: Type of data we want to store in the particular
column.
            For example,int for integer data.
size: Size of the data we can store in a particular column. For
example, if for
a column we specify the data_type as int and size as 10 then this
column can store an integer
a number of maximum 10 digits.
```

Example Query: This query will create a table named Students with four columns, ROLL_NO, NAME, ADDRESS, and AGE.

```
CREATE TABLE Student
(
    ROLL_NO int,
    AGE int,
    NAME varchar(20),
    ADDRESS varchar(20)
);
```

This query will create a table named Student. The ROLL_NO and AGE field is of type int. The next two columns NAME and ADDRESS are of type varchar and can store characters and the size 20 specifies that these two fields can hold a maximum of 20 characters.

2. INSERT INTO Clause

The INSERT INTO statement of SQL is used to insert a new row in a table. There are two ways of using INSERT INTO statement for inserting rows:

0. **Only values:** First method is to specify only the value of data to be inserted without the column names.
   Syntax

   ```
   INSERT INTO table_name VALUES (value1, value2, value3,...);
   table_name: name of the table.
   value1, value2,.. : value of first column, second
   column,... for the new record
   ```

1. **Values with Column Name:** In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:
   Syntax:

   ```
   INSERT INTO table_name (column1, column2, column3,..)
   VALUES ( value1, value2, value3,..);
   table_name: name of the table.
   column1: name of first column, second column ...
   value1, value2, value3 : value of first column, second
   column,... for the new record
   ```

Empty Student table After Creation

```
 roll_no | name | address | age
---------+------+---------+-----
(0 rows)
```

Example - Method 1 (Inserting only values) :

```
INSERT INTO Students VALUES ('1','ALEX','NOIDA','19');
```

Student Table will look like this.

```
 roll_no | name | address | age
---------+------+---------+-----
       1 | ALEX | NOIDA   |  19
(1 row)
```

Example - Method 2 (Inserting Values with Column Name) :

```
INSERT INTO Students (ROLL_NO, NAME, Address,Age) VALUES
('2','ALLEN','DELHI','19');
```

Student Table will look like this.

```
 roll_no | name   | address | age
---------+--------+---------+-----
       1 | ALEX   | NOIDA   |  19
       2 | ALLEN  | DELHI   |  19
(2 rows)
```

3. UPDATE Clause

The UPDATE statement in SQL is used to update the data of an existing table in database. We can update single columns as well as multiple columns using UPDATE statement as per our requirement.

Basic Syntax

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

table_name: name of the table
column1: name of first , second, third column....
value1: new value for first, second, third column....
condition: condition to select the rows for which the
values of columns needs to be updated.
```

NOTE: In the above query the **SET** statement is used to set new values to the particular column and the **WHERE** clause is used to select the rows for which the columns are needed to be updated. If we have not used the

WHERE clause then the columns in **all** the rows will be updated. So the WHERE clause is used to choose the particular rows.

Example Query for Updating Multiple Columns

```
UPDATE Students SET NAME = 'BROOK', ADDRESS = 'GURUGRAM' WHERE
ROLL_NO = 1;
```

The Student Table will look like.

```
 roll_no | name  | address   | age
---------+-------+-----------+-----
       1 | BROOK | GURUGRAM  |  19
       2 | ALLEN | DELHI     |  19
(2 rows)
```

4. DELETE Clause

The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Basic Syntax

```
DELETE FROM table_name WHERE some_condition;
table_name: name of the table
some_condition: condition to choose particular record.
```

Note: We can delete single as well as multiple records depending on the condition we provide in WHERE clause. If we omit the WHERE clause then all of the records will be deleted and the table will be empty.

Example Query for Deleting Record

```
DELETE FROM Students WHERE NAME = 'ALLEN';
```

The Student table will look like this after deleting ALLEN's record.

```
 roll_no | name  | address   | age
---------+-------+-----------+-----
       1 | BROOK | GURUGRAM  |  19
(1 row)
```

What is the use of ADD, DROP and MODIFY Commands?

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

ALTER TABLE - ADD

ADD is used to add columns into the existing table. Sometimes we may require to add additional information, in that case we do not require to create the whole database again, **ADD** comes to our rescue.

Syntax:

```
ALTER TABLE table_name

            ADD (Columnname_1  datatype,

            Columnname_2  datatype,

            …

            Columnname_n  datatype);
```

ALTER TABLE - DROP

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

Syntax:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

ALTER TABLE-MODIFY

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

Syntax may vary slightly in different databases.* **Syntax(Oracle,MySQL,MariaDB):

```
ALTER TABLE table_name

MODIFY column_name column_type;
```

Syntax(SQL Server):

```
ALTER TABLE table_name

ALTER COLUMN column_name column_type;
```

Queries

Sample Table:

Student

ROLL_NO	NAME
1	Ram
2	Abhi
3	Rahul
4	Tanu

QUERY:

- To ADD 2 columns AGE and COURSE to table Student.

```
ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));
```

OUTPUT:

ROLL_NO	NAME	AGE	COURSE
1	Ram		
2	Abhi		
3	Rahul		
4	Tanu		

- MODIFY column COURSE in table Student

```
ALTER TABLE Student MODIFY COURSE varchar(20);
```

After running the above query maximum size of Course Column is reduced to 20 from 40.

- DROP column COURSE in table Student.

```
ALTER TABLE Student DROP COLUMN COURSE;
```

OUTPUT:

ROLL_NONAMEAGE

1 Ram

2 Abhi

3 Rahul

4 Tanu

What are VIEWS in SQL?

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

Sample Tables:

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Ashish	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Creating Views

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

SYNTAX:

```
CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;



view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows
```

Example

Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
Output:
```

OUTPUT

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

Creating View from multiple tables:

In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement.

Query:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Ashish	Durgapur	50	20
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

What are JOINS in SQL?

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

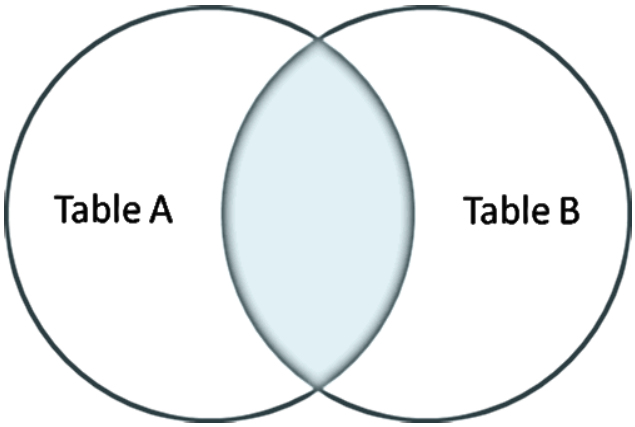
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e., value of the common field will be the same.
   Syntax:

```
2. SELECT table1.column1,table1.column2,table2.column1,....

3. FROM table1

4. INNER JOIN table2

5. ON table1.matching_column = table2.matching_column;
6. table1: First table.

7. table2: Second table

8. matching_column: Column common to both the tables.
```

Note: We can also write JOIN instead of INNER JOIN. JOIN is the same as INNER JOIN.



Example Queries(INNER JOIN)

- This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE
FROM Student

INNER JOIN StudentCourse

ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

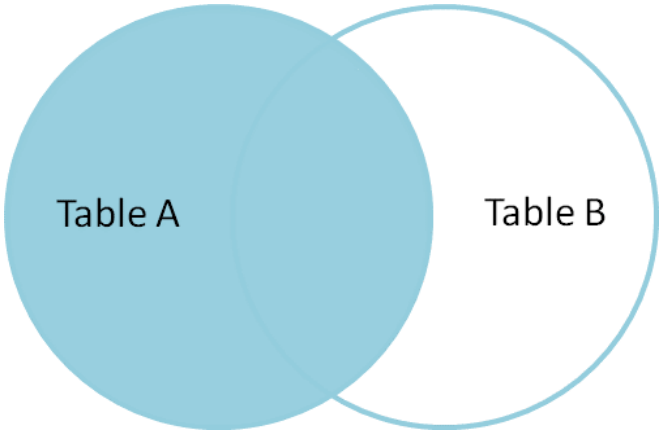
Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

9. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join. The rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN. **Syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;
table1: First table.

table2: Second table

matching_column: Column common to both the tables.
```

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

17. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

```
18. SELECT table1.column1,table1.column2,table2.column1,....

19. FROM table1

20. RIGHT JOIN table2

21. ON table1.matching_column = table2.matching_column;
22. table1: First table.

23. table2: Second table

24. matching_column: Column common to both the tables.
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.

Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

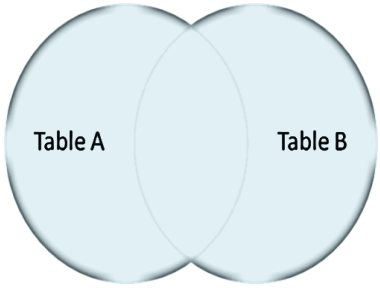
Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

25. **FULL JOIN:** FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;
table1: First table.

table2: Second table

matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

What is the use of GROUP BY Clause?

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
ORDER BY column1, column2;

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.
```

Example query to demonstrate GROUP BY Clause.

```
 SELECT address, AVG(age) FROM Students GROUP BY address;
```

Output of the Above query will look like this.

```
 address   |          avg
-----------+----------------------
 DELHI     | 19.0000000000000000
 NOIDA     | 20.0000000000000000
 GURUGRAM  | 19.0000000000000000
(3 rows)
```

This returns the Average Age for each address present in Students Table.

What are Aggregate Functions?

In database management an **aggregate function** is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

We will be using below Students table to explain the **Example Queries**.

Students Table

roll_no	name	address	age
1	BROOK	GURUGRAM	19
2	ALEX	NOIDA	19
3	ALLEN	NOIDA	21
4	ROBIN	DELHI	20
5	CALVIN	DELHI	18

Various Aggregate Functions

```
1) Count ()
2) Sum()
3) Avg()
4) Min()
5) Max()
```

1. **Count( ) Example** queries to demonstrate above function.

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

```
Output : 5
```

Returns total number of records .i.e 5.

```
SELECT COUNT(age) FROM Students;
Output : 5
```

Return number of Non Null values over the column age. i.e 5.

```
SELECT COUNT(DISTINCT age) FROM Students;
Output : 4
```

Return number of distinct Non Null values over the column age .i.e 4

2. **Sum( ) Example** queries to demonstrate above function.

```
 SELECT SUM(age) FROM Students;
Output : 97
```

Sum all Non Null values of Column salary i.e.,97

```
 SELECT SUM(DISTINCT age) FROM Students;
Output : 78
```

Sum of all distinct Non-Null values i.e., 78.

3. **Avg( ) Example** queries to demonstrate above function.

```
 SELECT AVG(age) FROM Students;
Output : 19.40
```

Avg(age) = sum(age) / Count(age) = 97/5 = 19.40

```
 SELECT AVG(DISTINCT age) FROM Students;
Output : 19.50
```

Avg(Distinct age) = sum(Distinct age) / Count(Distinct age) = 78/4 = 19.50

4. **Min( ) Example** query to demonstrate above function.

```
 SELECT MIN(age) FROM Students;
```

```
Output : 18
```

Return minimum non null value over the column age. i.e 18.

5. **Max( ) Example** query to demonstrate above function.

```
 SELECT MAX(age) FROM Students;
Output : 21
```

Return maximum non null value over the column age. i.e 21.

What is Cursor in SQL?

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:** Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :** Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

How to create Explicit Cursor:

1. **Declare Cursor Object. Syntax :** DECLARE cursor_name CURSOR FORSELECT * FROM table_name

```
DECLARE s1 CURSOR FOR SELECT * FROM studDetails
```

2. **Open Cursor Connection. Syntax :** OPEN cursor_connection

```
OPEN s1
```

3. **Fetch Data from cursor.** There are total 6 methods to access data from cursor. They are as follows :
FIRST is used to fetch only the first row from cursor table.

LAST is used to fetch only last row from cursor table.
NEXT is used to fetch data in forward direction from cursor table.
PRIOR is used to fetch data in backward direction from cursor table.
ABSOLUTE n is used to fetch the exact n^{th} row from cursor table.
RELATIVE n is used to fetch the data in incremental way as well as decremental way.
Syntax : FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor_name

```
FETCH FIRST FROM s1
FETCH LAST FROM s1
FETCH NEXT FROM s1

FETCH PRIOR FROM s1
FETCH ABSOLUTE 7 FROM s1

FETCH RELATIVE -2 FROM s1
```

4. **Close cursor connection. Syntax :** CLOSE cursor_name

```
CLOSE s1
```

5. **Deallocate cursor memory. Syntax :** DEALLOCATE cursor_name

```
DEALLOCATE s1
```

What is the difference between Implicit and Explicit Cursor?

Databases such as ORACLE have a memory area, where processing of instructions and fetched data takes place.A cursor is a pointer which is pointing to this area.The data contained in this memory area is also known as **Active Set**. Cursors can be broadly classified into **Implicit Cursors** and **Explicit Cursors**.

Difference between Implicit and Explicit Cursors :

Implicit cursor in Oracle	Explicit Cursor in Oracle
• Implicit Cursor in Oracle are declared by PL/SQL implicitly for all DML Statements and for single row queries • Example: Select Statement issued directly within the BEGIN.. END part of a block opens up an implicit cursor.	• Declared and named by the programmer • Explicit Cursors allow multiple rows to be processed from the query. • Use explicit cursor to individually process all the rows returned by multiple row select statement

Example
 	
DECLARE
v_salary number;
BEGIN
SELECT emp_salary INTO v_salary FROM emp where empno=1111;
DBMS_OUTPUT.PUT_LINE(v_salary);
END;
/	

Four attributes SQL%NOTFOUND SQL%FOUND SQL%ISOPEN SQL%ROWCOUNT	

What is the difference between VIEW and CURSOR in SQL?

1. View:

A view is a virtual table that not actually exist in the database but it can be produced upon request by a particular user. A view is an object that gives the user a logical view of data from a base table we can restrict to what user can view by allowing them to see an only necessary column from the table and hide the other database details. View also permits users to access data according to their requirements, so the same data can be access by a different user in a different way according to their needs.

2Cursor : A cursor is a temporary work area created in memory for processing and storing the information related to an SQL statement when it is executed. The temporary work area is used to store the data retrieved from the database and manipulate data according to need. It contains all the necessary information on data access by the select statement. It can hold a set of rows called active set but can access only a single row at a time. There are two different types of cursors -

1. Implicit Cursor
2. Explicit Cursor

Difference between View and Cursor in SQL :

S.No.	View	Cursor
1.	A view is a virtual table that gives logical view of data from base table.	A cursor is a temporary workstation create in database server when SQL statement is executed.
2.	Views are dynamic in nature which means any changes made in base table are immediately reflected in view.	Cursor can be static as well as dynamic in nature.
3.	We can perform CRUD operations on view like create, insert, delete and update.	There are some steps for creating a Explicit cursor - • Declare the cursor in declaration section. • Open the cursor in execution section. • Fetch the cursor to retrieve data into PL/SQL variable. • Close the cursor to release allocated memory.
4.	There are two types of view i.e. Simple View (created from single table) and Complex Cursor (created from multiple tables).	Cursor has two types i.e. Implicit Cursor (pre-defined) and Explicit Cursor(user defined).
5.	View is a database object similar to table so it can be used with both SQL and PL/SQL.	Cursor is defined and used within the block of stored procedure which means it can be only used with PL/SQL.
6.	General Syntax of Creating View : CREATE VIEW "VIEW_NAME" AS "SQL Statement";	General Syntax of Creating View : CURSOR cursor_name IS select_statement;

What are the advantages of PL/SQL functions?

Advantages of PL / SQL functions as follows:

- We can make a single call to the database to run a block of statements. Thus, it improves the performance against running SQL multiple times. This will reduce the number of calls between the database and the application.

- We can divide the overall work into small modules which becomes quite manageable, also enhancing the readability of the code.
- It promotes reusability.
- It is secure since the code stays inside the database, thus hiding internal database details from the application(user). The user only makes a call to the PL/SQL functions. Hence, security and data hiding is ensured.

Explain BETWEEN and IN Clause.

We will be using below Students table to explain the **Example Queries**.

Students Table

roll_no	name	address	age
1	BROOK	GURUGRAM	19
2	ALEX	NOIDA	19
3	ALLEN	NOIDA	21
4	ROBIN	DELHI	20
5	CALVIN	DELHI	18

- **BETWEEN Clause**

  The SQL BETWEEN condition allows you to easily test if an expression is within a range of values (inclusive). The values can be text, date, or numbers. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement. The SQL BETWEEN Condition will return the records where expression is within the range of value1 and value2.

  Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

  Example : Query to get Student Records, Where Student's age is between 19 and 21.

```
SELECT * FROM Students WHERE age BETWEEN 19 AND 21;
```

Output of the query will look like this

```
 roll_no | name   | address   | age
---------+--------+-----------+-----
       1 | BROOK  | GURUGRAM  |  19
       2 | ALEX   | NOIDA     |  19
       3 | ALLEN  | NOIDA     |  21
       4 | ROBIN  | DELHI     |  20
(4 rows)
```

Example : Query to get Student Records, Where Student's age is **not** between 19 and 21.

```
SELECT * FROM Students WHERE age NOT BETWEEN 19 AND 21;
```

Output of the query will look like this.

```
 roll_no |  name   | address | age
---------+---------+---------+-----
       5 | CALVIN  | DELHI   |  18
(1 row)
```

- **IN Clause**

IN operator allows you to easily test if the expression matches any value in the list of values. It is used to remove the need of multiple OR condition in SELECT, INSERT, UPDATE or DELETE. You can also use NOT IN to exclude the rows in your list.
Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (list_of_values);
```

Example : Query to get Student Records, Where Student's age is in the set {18,20,21}.

```
SELECT * FROM Students WHERE age IN (18,20,21);
```

Output of the query will look like this.

```
 roll_no |  name   | address | age
---------+---------+---------+-----
       3 | ALLEN   | NOIDA   |  21
       4 | ROBIN   | DELHI   |  20
       5 | CALVIN  | DELHI   |  18
(3 rows)
```

Example : Query to get Student Records, Where Student's age is **not** in the

set {18,20,21}.

```
SELECT * FROM Students WHERE age NOT IN (18,20,21);
```

Output of the query will look like this.

```
 roll_no | name   | address   | age
---------+--------+-----------+-----
       1 | BROOK  | GURUGRAM  |  19
       2 | ALEX   | NOIDA     |  19
(2 rows)
```

What is the difference between DROP and TRUNCATE?

S.NO.	DROP	TRUNCATE
1.	The DROP command is used to remove table definition and its contents.	Whereas the TRUNCATE command is used to delete all the rows from the table.
2.	In the DROP command, table space is freed from memory.	While the TRUNCATE command does not free the table space from memory.
3.	DROP is a DDL(Data Definition Language) command.	Whereas the TRUNCATE is also a DDL(Data Definition Language) command.
4.	In the DROP command, a view of the table does not exist.	While in this command, a view of the table exists.
5.	In the DROP command, integrity constraints will be removed.	While in this command, integrity constraints will not be removed.
6.	In the DROP command, undo space is not used.	While in this command, undo space is used but less than DELETE.
7.	The DROP command is quick to perform but gives rise to complications.	While this command is faster than DROP.

What are Constraints in SQL?

Constraints are the rules that we can apply to the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

NOT NULL: This constraint tells that we cannot store a null value in a column. That

is, if a column is specified as NOT NULL then we will not be able to store null in this particular column anymore.

UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.

PRIMARY KEY: A primary key is a field that can uniquely identify each row in a table. And this constraint is used to specify a field in a table as the primary key.

FOREIGN KEY: A Foreign key is a field that can uniquely identify each row in another table. And this constraint is used to specify a field as a Foreign key.

CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.

DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

How to specify constraints? We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax: Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);
sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL,
UNIQUE, PRIMARY KEY etc.
```

Let us see each of the constraints in detail.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept the null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

2. UNIQUE – This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE column in a table.
For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

3. PRIMARY KEY –
Primary Key is a field that uniquely identifies each row in the table. If a field in a table is the primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words, we can say that this is a combination of NOT NULL and UNIQUE constraints.
A table can have only one field as a primary key. The below query will create a table named Student and specifies the field ID as a primary key.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

4. FOREIGN KEY –

Foreign Key is a field in a table that uniquely identifies each row of another table. That is this field points to the primary key of another table. This usually creates a kind of link between the tables.
Consider the two tables as shown below:

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8523	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in the Orders table is the primary key in the Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in the Orders table.

Syntax:

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
```

```
)
```

(i) CHECK –

Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18. Check constraint in detail

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

(ii) DEFAULT –
This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

What is a TRIGGER?

A **trigger** is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Example:
Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.
Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.
Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SOLUTION

```
create trigger stud_marks
before INSERT
on
```

```
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. I.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

What is the use of LIMIT and OFFSET in SQL?

We will be using below Students table to explain the **Example Queries**.

Students Table

roll_no	name	address	age
1	BROOK	GURUGRAM	19
2	ALEX	NOIDA	19
3	ALLEN	NOIDA	21
4	ROBIN	DELHI	20
5	CALVIN	DELHI	18

If there are a large number of tuples satisfying the query conditions, it might be resourceful to view only a handful of them at a time.

- The **LIMIT** clause is used to set an upper limit on the number of tuples returned by SQL.
- It is important to note that this clause is not supported by all SQL versions.
- The LIMIT clause can also be specfied using the SQL 2008 OFFSET/FETCH FIRST clauses.
- The limit/offset expressions must be a non-negative integer.

Example Queries to demonstrate LIMIT Clause.

```
SELECT *
FROM Students
LIMIT 3;
```

```
 roll_no | name  | address   | age
---------+-------+-----------+-----
       1 | BROOK | GURUGRAM  |  19
       2 | ALEX  | NOIDA     |  19
       3 | ALLEN | NOIDA     |  21
(3 rows)
```

Output of the query gives First three Student Records.

```
SELECT *
FROM Students
ORDER BY age
LIMIT 3;
```

```
 roll_no |  name   | address   | age
---------+---------+-----------+-----
       5 | CALVIN  | DELHI     |  18
       1 | BROOK   | GURUGRAM  |  19
       2 | ALEX    | NOIDA     |  19
(3 rows)
```

Output of the query gives three Student Records in order of Ascending Ages.

The LIMIT operator can be used in situations such as the above, where we need to find the top **N** students in a class and based on any condition statements.

Using LIMIT along with OFFSET

LIMIT **x** OFFSET **y** simply means skip the first y entries and then return the next x entries.
OFFSET can only be used with the ORDER BY clause. It cannot be used on its own.
OFFSET value must be greater than or equal to zero. It cannot be negative, else returns error.
Example query to demonstrate LIMIT and OFFSET.

```
SELECT *
FROM Students
LIMIT 3 OFFSET 2
ORDER BY roll_no;
```

```
 roll_no |  name   | address | age
---------+---------+---------+-----
       3 | ALLEN   | NOIDA   |  21
       4 | ROBIN   | DELHI   |  20
       5 | CALVIN  | DELHI   |  18
(3 rows)
```

Returns 3 Student Records skipping first two records in Table.

What are different types of operators present in SQL?

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. SQL operators have three different categories.

-Arithmetic operator

-Comparison operator

-Logical operator

Arithmetic operators:

We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

S.No.	Operator	Description
1	+	The addition is used to perform an addition operation on the data values.
2	–	This operator is used for the subtraction of the data values.
3	/	This operator works with the 'ALL' keyword and it calculates division operations.
4	*	This operator is used for multiply data values.
5	%	Modulus is used to get the remainder when data is divided by another.

Comparison operators:

Another important operator in SQL is a comparison operator, which is used to compare one expression's value to other expressions. SQL supports different types of the comparison operator, which is described below:

S.No.	Operator	Description
1	=	Equal to.
2	>	Greater than.
3	<	Less than.
4	>=	Greater than equal to.
5	<=	Less than equal to.
6	<>	Not equal to.

Logical operators:

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

S.No	Operator	Description
1	AND	Logical AND compares between two Booleans as expressions and returns true when both expressions are true.
2	OR	Logical OR compares between two Booleans as expressions and returns true when one of the expressions is true.
3	NOT	Not takes a single Boolean as an argument and changes its value from false to true or from true to false.

Special operators:

S.No.	Operator	Description
1	ALL	ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list of results from a query. The ALL must be preceded by the comparison operators and evaluates to TRUE if the query returns no rows.
2	ANY	ANY compares a value to each value in a list of results from a query and evaluates to true if the result of an inner query contains at least one row.
3	BETWEEN	The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression.
4	IN	The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table which are matching.
5	EXISTS	The EXISTS checks the existence of a result of a subquery. The EXISTS subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns 'FALSE'.
6	SOME	SOME operator evaluates the condition between the outer and inner tables and evaluates to true if the final result returns any one row. If not, then it evaluates to false.