MySQL Tutorial

PreparedBy:- Shubham Mishra

SQL:- Structured Query Language is a standard Database language which is used to create, maintain and retrieve the relational database.

- [ ] SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.
- [ ] We can write comments in SQL using "–-" (double hyphen) at the beginning of any line.
- [ ] SQL is the programming language for relational databases (explained below) like MySQL, Oracle, SQL Server, PostgreSQL, etc.

Relational Database:- Relational database means the data is stored as well as retrieved in the form of tables/Relations.

Table 1 shows the relational database with only one relation called **STUDENT** which stores **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

STUDENT Table

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham	DELHI	6954789654	26
2	Rohit	JAIPUR	8954678458	24
3	Chetan	NAGPUR	7845965478	31
4	Saket	PATNA	7800213640	45
5	Rakesh	MUMBAI	9461021032	52

Shubham Mishra

Important terminologies that are used in terms of relation.

- **Attribute:** Attributes are the properties that define a relation. e.g.; **ROLL_NO**, **NAME** etc.

- **Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	Shubham	Varanasi	7860481732	25

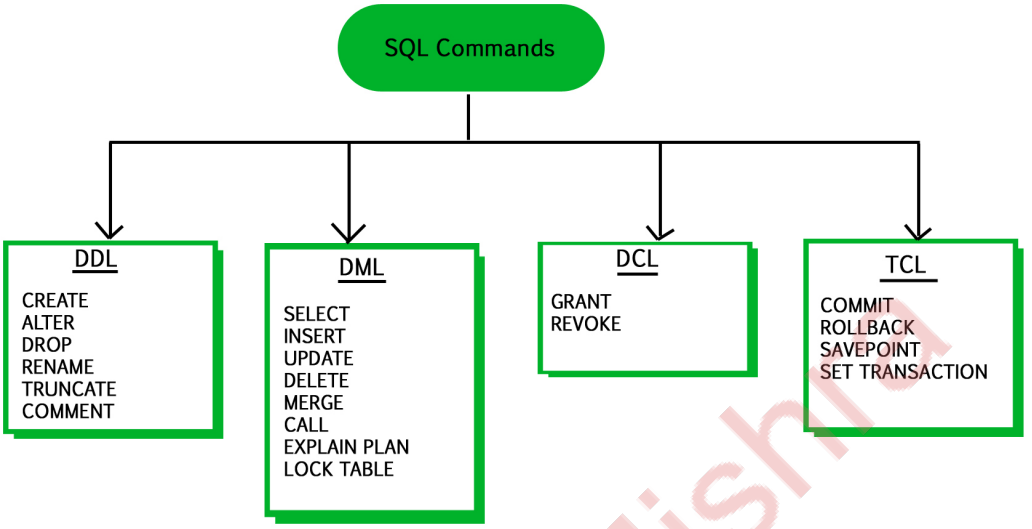
- **Degree:** The number of attributes in the relation is known as degree of the relation. The **STUDENT** relation defined above has degree 5.
- **Column:** Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT

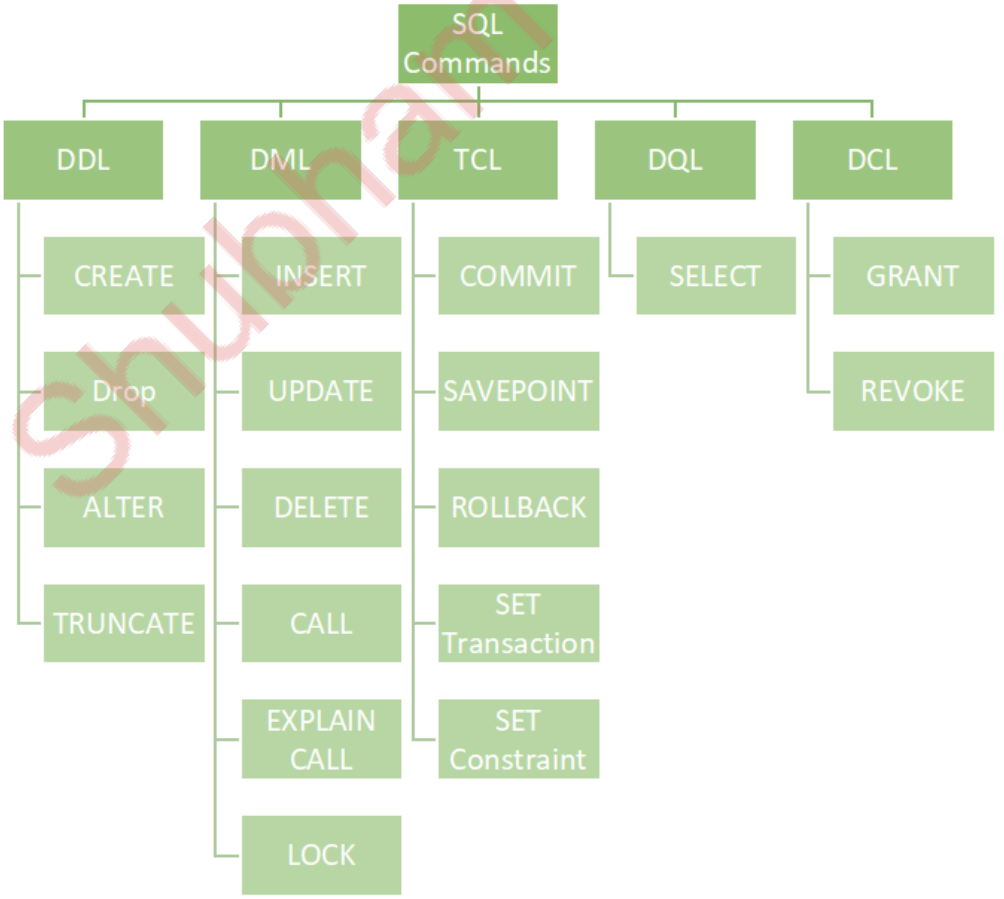
ROLL_NO
1
2
3
4
5

Shubham Mishra

The queries to deal with relational database can be categories as:

- **Data Definition Language (DDL) :-It is used to define the structure of the database. e.g; CREATE TABLE, ADD COLUMN, DROP COLUMN**
- **Data Manipulation Language (DML) :-It is used to manipulate data in the relations. e.g.; INSERT, DELETE, UPDATE**
- **Data Control Language (DCL) :-DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.**
- **Data Query Language (DQL) :- It is used to extract the data from the relations. e.g.; SELECT**
- **Transaction Control Language (TCL) :-TCL commands deal with the transaction within the database.**





Shubham Mishra

Data Definition Language (DDL)

- **CREATE:-** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP:-** This command is used to delete objects from the database.
- **ALTER**:- This is used to alter the structure of the database.
- **TRUNCATE**:- This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**:- This is used to add comments to the data dictionary.
- **RENAME**:- This is used to rename an object existing in the database.

Data Manipulation Language (DML)

- **INSERT:-** It is used to insert data into a table.
- **UPDATE:-** It is used to update existing data within a table.
- **DELETE:-** It is used to delete records from a database table.
- **LOCK:-** Table control concurrency.
- **CALL:-** Call a PL/SQL or JAVA subprogram.
- **EXPLAIN PLAN:-** It describes the access path to data.

Data Control Language (DCL)

- **GRANT:-** This command gives users access privileges to the database.
- **REVOKE:-** This command withdraws the user's access privileges given by using the GRANT command.

Data Query Language (DQL)

- **SELECT:-** It is used to retrieve data from the database.

Transaction Control Language (TCL)

- **COMMIT:-** Commits a Transaction.
- **ROLLBACK:-** Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**:- Sets a save-point within a transaction.
- **SET TRANSACTION:-** Specify characteristics for the transaction.

CREATE TABLE

A **Table** is combination of rows and columns. For creating table we have to define the structure of table by adding name to columns,

Shubham Mishra

providing data type and size of data to be stored in columns.

Syntax:-

```
CREATE table table_name
(
Column1 datatype (size),
column2 datatype (size),
.
.
columnN datatype(size)
);

Here table_name is name of the table, column is the name of column
```

Example Query:- Let us create a table to store data of Subjects, so the table name is **Subject**, Columns are Sub_ID, Sub_Name

```
CREATE TABLE Subject
(
Sub_ID INT,
Sub_Name varchar(20)
);
Here INT and varchar are datatype, Datatype means type of data we can store, like for
INT type,
we can store integer type data in the column
```

ADD DATA IN TABLE

To add data in table, we use **INSERT INTO**

Syntax:-

```
//Below query adds data in specific column, (like Column1=Value1)//
Insert into Table_name(Column1, Column2, Column3)
Values (Value1, value2, value3);

//Below query adds data in table in sequence of column name(Value1 will be added in
Column1 and so on)//
Insert into Table_name
Values (Value1, value2, value3);

//Adding multiple data in the table in one go//
Insert into Table_name
Values (Value01, value02, value03),
(Value11, value12, value13),
(Value21, value22, value23),
.
```

Shubham Mishra

```
.
(ValueN1, valueN2, valueN3)

Here table_name is name of the table, column is the name of columns
```

Example Query:- This query will add data in the table named Subject

```
//Adding data in first row//
Insert into Subject
Values (1,'English');

//Adding data in specific column/
Insert into Subject(Sub_Name)
Values ('Hindi');

//Adding multiple data in the table in one go//
Insert into Subject
Values (1,'English'),
(2,'French'),
(2,'Science'),
(2,'Maths');

Try creating table with five columns and add data using above queries
```

Shubham Mishra

SELECT

- **Select** is the most commonly used statement in SQL. The **SELECT** Statement in SQL is used to retrieve or fetch data from a database. We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called result-set.
- With the **SELECT** clause of a **SELECT** command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table
- The **select** clause is the first clause and is one of the last clauses of the **select** statement that the database server evaluates. The reason for this is that before we can determine what to include in the final result set, we need to know all of the possible columns that could be included in the final result set

Sample Table

Table_name:- Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham	DELHI	6954789654	26
2	Rohit	JAIPUR	8954678458	24
3	Chetan	NAGPUR	7845965478	31
4	Saket	PATNA	7800213640	45
5	Rakesh	MUMBAI	9461021032	52

Syntax:-

```
SELECT column1,column2 FROM table_name
column1 , column2: names of the fields of the table
table_name: from where we want to fetch
```

This query will return all the rows in the table with fields column1 , column2.

- To fetch the entire table or all the fields in the table:

```
 SELECT * FROM table_name;
```

- Query to fetch the fields ROLL_NO, NAME, AGE from the table Student:

```
SELECT ROLL_NO, NAME, AGE FROM Student;
```

- To fetch all the fields from the table Student

```
SELECT * FROM Student;
```

Shubham Mishra

WHERE

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.

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

Shubham Mishra

Queries:

- To fetch record of students with age equal to 20

```
SELECT * FROM Student WHERE Age=20;
```

```
SELECT ROLL_NO,NAME,ADDRESS FROM Student WHERE ROLL_NO > 3;
```

BETWEEN operator: It is used to fetch filtered data in a given range inclusive of two values.

Syntax:- SELECT column1,column2 FROM table_name WHERE column_name BETWEEN value1 AND value2;

Queries:

- To fetch records of students where ROLL_NO is between 1 and 3 (inclusive)

```
SELECT * FROM Student WHERE ROLL_NO BETWEEN 1 AND 3;
```

- To fetch NAME,ADDRESS of students where Age is between 20 and 30 (inclusive)

```
SELECT NAME,ADDRESS FROM Student WHERE Age BETWEEN 20 AND 30;
```

LIKE operator: It is used to fetch filtered data by searching for a particular pattern in where clause.

Syntax:- SELECT column1,column2 FROM table_name WHERE column_name LIKE pattern;

Pattern:- exact value extracted from the pattern to get related data in result set. **Note**: The character(s) in pattern are case sensitive.

Queries:

- To fetch records of students where NAME starts with letter S.

```
SELECT * FROM Student WHERE NAME LIKE 'S%';
```

- **The '%'(wildcard) signifies the later characters here which can be of any length and value.More about wildcards will be discussed in the later set**
- To fetch records of students where NAME contains the pattern 'AM'.

Shubham Mishra

```
SELECT * FROM Student WHERE NAME LIKE '%AM%';
```

IN operator: It is used to fetch filtered data same as fetched by '=' operator just the difference is that here we can specify multiple values for which we can get the result set.

Syntax:- SELECT column1,column2 FROM table_name WHERE column_name IN (value1,value2,..);

Queries:

- To fetch NAME and ADDRESS of students where Age is 18 or 20.

```
SELECT NAME,ADDRESS FROM Student WHERE Age IN (18,20);
```

- To fetch records of students where ROLL_NO is 1 or 4.

```
SELECT * FROM Student WHERE ROLL_NO IN (1,4);
```

DELETE

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.

Syntax:-

```
DELETE FROM table_name WHERE some_condition;
```

Queries:

- **Deleting single record**: Delete the rows where NAME = 'Rakesh'. This will delete only the last row.

```
DELETE FROM Student WHERE NAME = 'Rakesh';
```

- **Deleting multiple records**: Delete the rows from the table Student where Age is 20.

Shubham Mishra

```
DELETE FROM Student WHERE Age = 20;
```

- **Delete all of the records:** There are two queries to do this as shown below,

```
query1: "DELETE FROM Student";

query2: "DELETE * FROM Student";
```

INSERT INTO

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:

- **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,...);

- **Column names and values both:** In the second method we will specify both the columns which we want to fill and their corresponding values.

Syntax:- INSERT INTO table_name (column1, column2, column3,..) VALUES (value1, value2, value3,..);

Queries:

Method 1 (Inserting only values) : INSERT INTO Student VALUES ('5','HARSH','WEST BENGAL','XXXXXXXXXX','19');

Method 2 (Inserting values in only specified columns): INSERT INTO Student (ROLL_NO, NAME, Age) VALUES ('5','PRATIK','19');

Note:- Notice that the columns for which the values are not provided are filled by null. Which is the default values for those columns.

Using SELECT in INSERT INTO Statement

We can use the SELECT statement with INSERT INTO statement to copy rows from one table and insert them into another table.The use of this

Shubham Mishra

statement is similar to that of INSERT INTO statement. The difference is that the SELECT statement is used here to select data from a different table. The different ways of using INSERT INTO SELECT statement are shown below:

- **Inserting all columns of a table:** We can copy all the data of a table and insert into in a different table.

Syntax:- **INSERT INTO first_table SELECT * FROM second_table;**

first_table: name of first table.
second_table: name of second table
Note:- We have used the SELECT statement to copy the data from one table and INSERT INTO statement to insert in a different table.

- **Inserting specific columns of a table:** We can copy only those columns of a table which we want to insert into in a different table.

.

Syntax:- INSERT INTO first_table(names_of_columns1) SELECT names_of_columns2 FROM second_table;

first_table: name of first table.
second_table: name of second table.
names of columns1: name of columns separated by comma(,) for table 1.
names of columns2: name of columns separated by comma(,) for table 2.
Note:- We have used the SELECT statement to copy the data of the selected columns only from the second table and INSERT INTO statement to insert in first table.

- **Copying specific rows from a table**: We can copy specific rows from a table to insert into another table by using WHERE clause with the SELECT statement. We have to provide appropriate condition in the WHERE clause to select specific rows.

Shubham Mishra

Syntax:- INSERT INTO table1 SELECT * FROM table2 WHERE condition;

first_table: name of first table.
second_table: name of second table.
condition: condition to select specific rows

Queries:

Method 1(Inserting all rows and columns): INSERT INTO Student SELECT * FROM LateralStudent;
Method 2(Inserting specific columns): INSERT INTO Student(ROLL_NO,NAME,Age) SELECT ROLL_NO, NAME, Age FROM LateralStudent;

- **Select specific rows to insert**:- INSERT INTO Student SELECT * FROM LateralStudent WHERE Age = 18;
- **To insert multiple rows in a table using Single SQL Statement:-**

```
INSERT INTO table_name(Column1,Column2,Column3,.......)
VALUES (Value1, Value2,Value3,.....),
       (Value1, Value2,Value3,.....),
        (Value1, Value2,Value3,.....),
        ........................... ;
```

AND and OR operators:

- AND & OR operators are used for filtering the data and getting precise results based on conditions.
- The SQL **AND** & **OR** operators are also used to combine multiple conditions.
- These two operators can be combined to test for multiple conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

Note: When combining these conditions, it is important to use parentheses so that the database knows what order to evaluate each condition.

- The AND and OR operators are used with the WHERE clause.
- These two operators are called conjunctive operators.

Shubham Mishra

AND Operator:
This operator displays only those records where both the conditions condition1 and condition2 evaluates to True.

Syntax:-

```
SELECT * FROM table_name WHERE condition1 AND condition2 and ...conditionN;

table_name: name of the table
condition1,2,..N : first condition, second condition and so on
```

OR Operator:
This operator displays the records where either one of the conditions condition1 and condition2 evaluates to True. That is, either condition1 is True or condition2 is True.

Syntax:-

```
SELECT * FROM table_name WHERE condition1 OR condition2 OR... conditionN;

table_name: name of the table
condition1,2,..N : first condition, second condition and so on
```

- If suppose we want to fetch all the records from the Student table where Age is 18 and ADDRESS is Delhi. then the query will be:

Queries:

```
SELECT * FROM Student WHERE Age = 18 AND ADDRESS = 'Delhi';
```

- To fetch all the records from the Student table where NAME is Satvik or NAME is Niraj.

```
SELECT * FROM Student WHERE NAME = 'Satvik' OR NAME = 'Niraj';
```

- To fetch all the records from the Student table where NAME is Niraj or Age is 30.

```
SELECT * FROM Student WHERE NAME = 'Ram' OR Age = 20;
```

Combining AND and OR:

Shubham Mishra

We can combine AND and OR operators in the below manner to write complex queries.

Syntax:-

```
SELECT * FROM table_name WHERE condition1 AND (condition2 OR condition3);
```

Query:

```
SELECT * FROM Student WHERE Age = 18 AND (NAME = 'Niraj' OR NAME = 'satvik');
```

Shubham Mishra

DROP, TRUNCATE

DROP:-

- DROP is used to delete a whole database or just a table.The DROP statement destroys the objects like an existing database, table, index, or view.
- A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Syntax:-

```
DROP object object_name

Examples:
DROP TABLE table_name;
table_name: Name of the table to be deleted.

DROP DATABASE database_name;
database_name: Name of the database to be deleted.
```

TRUNCATE:-

- TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extents of a table for deallocation (empty for reuse).
- The result of this operation quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms.

Syntax:-

```
TRUNCATE TABLE  table_name;
table_name: Name of the table to be truncated.
DATABASE name - student_data
```

DROP vs TRUNCATE:

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
- Table or Database deletion using DROP statement **cannot** be rolled back, so it must be used wisely.

Queries:

To delete the whole database:-

```
DROP DATABASE student_data;
```

To truncate Student table from global database:-

```
TRUNCATE TABLE Student;
```

After running the above query Student table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

NOT Operator

Syntax:- SELECT column1, colomn2, ... FROM table_name WHERE NOT condition;

Arithmetic Operators

We can use various Arithmetic Operators on the data stored in the tables.

Arithmetic Operators are:

```
+           [Addition]
-           [Subtraction]
/           [Division]
*           [Multiplication]
%           [Modulus]
```

Addition (+) :

Shubham Mishra

It is used to perform **addition operation** on the data items, items include either single column or multiple columns.

Implementation:

addition of 1 columns

```
SELECT employee_id, employee_name, salary, salary + 100
   AS "salary + 100" FROM addition;
```

addition of 2 columns

```
SELECT employee_id, employee_name, salary, salary + employee_id
   AS "salary + employee_id" FROM addition;
```

Subtraction (-) :

It is use to perform **subtraction operation** on the data items, items include either single column or multiple columns.

Implementation:

```
SELECT employee_id, employee_name, salary, salary - 100
    AS "salary - 100" FROM subtraction;
```

subtraction of 2 columns:

```
SELECT employee_id, employee_name, salary, salary - employee_id
    AS "salary - employee_id" FROM subtraction;
```

Multiplication (*) :

It is use to perform **multiplication** of data items.

Implementation:

```
SELECT employee_id, employee_name, salary, salary * 100
    AS "salary * 100" FROM addition;
```

multiplication of 2 columns:

```
SELECT employee_id, employee_name, salary, salary * employee_id
     AS "salary * employee_id" FROM addition;
```

Modulus (%) :

It is use to get **remainder** when one data is divided by another.

Implementation:

Shubham Mishra

```
SELECT employee_id, employee_name, salary, salary % 25000
    AS "salary % 25000" FROM addition;
```

modulus operation between 2 columns:

```
SELECT employee_id, employee_name, salary, salary % employee_id
    AS "salary % employee_id" FROM addition;
```

Note:
***modulus** is use to check whether a number is **Even** or **Odd**. Suppose a given number if divided by 2 and gives 1 as remainder, then it is an odd number or if on dividing by 2 and gives 0 as remainder, then it is an even number.*

Concept of NULL :
If we perform any arithmetic operation on **NULL**, then answer is *always* null.
Implementation:

```
SELECT employee_id, employee_name, salary, type, type + 100
    AS "type+100" FROM addition;
```

Shubham Mishra

Wildcard operators

Wildcard operators are used with LIKE operator, there are four basic operators:

Operator	Description
%	It is used in substitute of zero or more characters.
_	It is used in substitute of one character.
_	It is used to substitute a range of characters.
[range_of_characters]	It is used to fetch matching set or range of characters specified inside the brackets.

[^range_of_characters] or [!range of characters]	It is used to fetch non-matching set or range of characters specified inside the brackets.

Syntax:

```
SELECT column1,column2 FROM table_name WHERE column LIKE wildcard_operator;
column1 , column2: fields in the table
table_name: name of table
column: name of field used for filtering data
```

Queries:-

- To fetch records from Student table with NAME ending with letter ‘T’.

```
SELECT * FROM Student WHERE NAME LIKE '%T';
```

- To fetch records from Student table with NAME ending any letter but starting from ‘RAMES’.

```
SELECT * FROM Student WHERE NAME LIKE 'RAMES_';
```

- To fetch records from Student table with address containing letters ‘a’, ‘b’, or ‘c’.

```
SELECT * FROM Student WHERE ADDRESS LIKE '%[A-C]%';
```

- To fetch records from Student table with ADDRESS not containing letters ‘a’, ‘b’, or ‘c’.

```
SELECT * FROM Student WHERE ADDRESS LIKE '%[^A-C]%';
```

- To fetch records from Student table with PHONE field having a ‘9’ in 1st position and a ‘5’ in 4th position.

```
SELECT * FROM Student WHERE PHONE LIKE '9__5%';
```

- To fetch records from Student table with ADDRESS containing total of 6 characters.

```
SELECT * FROM Student WHERE ADDRESS LIKE '______';
```

Shubham Mishra

```

```

- To fetch records from Student table with ADDRESS containing ‘OH’ at any position, and the result set should not contain duplicate data.

```
SELECT DISTINCT * FROM Student WHERE ADDRESS LIKE '%OH%';
```

UPDATE Statement

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.

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.

Updating single column: Update the column NAME and set the value to ‘PRATIK’ in all the rows where Age is 20.

```
UPDATE Student SET NAME = 'PRATIK' WHERE Age = 20;
```

Updating multiple columns: Update the columns NAME to ‘PRATIK’ and ADDRESS to ‘SIKKIM’ where ROLL_NO is 1

```
UPDATE Student SET NAME = 'PRATIK', ADDRESS = 'SIKKIM' WHERE ROLL_NO = 1;
```

Omitting WHERE clause: If we omit the WHERE clause from the update query then all of the rows will get updated.

Shubham Mishra

```
UPDATE Student SET NAME = 'PRATIK';
```

ALTER (ADD, DROP, MODIFY)

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);
```

Shubham Mishra

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:-

```
 ALTER TABLE table_name
MODIFY column_name column_type;
```

Query:-

```
ALTER TABLE Student ADD (COURSE varchar(40));
ALTER TABLE Student MODIFY COURSE varchar(20);
ALTER TABLE Student DROP COLUMN COURSE;
```

LIKE

Sometimes we may require tuples from the database which match certain patterns. For example, we may wish to retrieve all columns where the tuples start with the letter 'y', or start with 'b' and end with 'l', or even more complicated and restrictive string patterns. This is where the LIKE Clause comes to rescue, often coupled with the WHERE Clause in SQL.

There are two kinds of wildcards used to filter out the results:

- % : Used to match zero or more characters. (Variable Length)
- _ : Used to match exactly one character. (Fixed Length)

The following are the rules for pattern matching with the LIKE Clause:

Pattern Meaning	
'a%'	Match strings which start with 'a'
'%a'	Match strings with end with 'a'
'a%t'	Match strings which contain the start with 'a' and end with 't'.

Shubham Mishra

'_wow%'	Match strings which contain the substring 'wow' in them at the second position.
'_a%'	Match strings which contain 'a' at the second position.
'a_ _%'	Match strings which start with 'a' and contain at least 2 more characters.

```
SELECT roll_no, Name, Address
FROM student
WHERE Name LIKE 'Ro%';
```

```
SELECT *
FROM student
WHERE Address LIKE '%Patna%';
```

```
SELECT Name, Address, Phone
FROM student
WHERE Name LIKE '_usum%';
```

Shubham Mishra

Note: The LIKE operator is extremely resourceful in situations such as address filtering wherein we know only a segment or a portion of the entire address (such as locality or city) and would like to retrieve results based on that. The wildcards can be resourcefully exploited to yield even better and more filtered tuples based on the requirement.

BETWEEN & IN Operator

BETWEEN:-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;
```

Query:-

- **Using BETWEEN with Numeric Values:**

List all the Employee Fname, Lname who is having salary between 30000 and 45000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary
BETWEEN 30000 AND 45000;
```

- **Using BETWEEN with Date Values:**

Find all the Employee having Date of Birth Between 01-01-1985 and 12-12-1990.

```
SELECT Fname, Lname
FROM Employee
where DOB
BETWEEN '1985-01-01' AND '1990-12-30';
```

- **Using NOT operator with BETWEEN**

Find all the Employee name whose salary is not in the range of 30000 and 45000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary
NOT BETWEEN 30000 AND 45000;
```

IN Operator:

- 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.

Shubham Mishra

- You can also use NOT IN to exclude the rows in your list.
- We should note that any kind of duplicate entry will be retained.

Syntax:-

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

Query:-

- Find the Fname, Lname of the Employees who have Salary equal to 30000, 40000 or 25000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary IN (30000, 40000, 25000);
```

- Find the Fname, Lname of all the Employee who have Salary not equal to 25000 or 30000.

```
SELECT Fname, Lname
FROM Employee
WHERE Salary NOT IN (25000, 30000);
```

Distinct Clause

- The distinct keyword is used in conjunction with select keyword.
- It is helpful when there is a need of avoiding duplicate values present in any specific columns/table.
- When we use distinct keyword only the **unique values** are fetched.

Syntax:-

```
SELECT DISTINCT column1, column2
FROM table_name
```

NOTE: If distinct keyword is used with multiple columns, the distinct combination is displayed in the result set.

Shubham Mishra

Queries:-

- To fetch unique names from the NAME field –

```
SELECT DISTINCT NAME
FROM Student;
```

- To fetch a unique combination of rows from the whole table –

```
SELECT DISTINCT *
FROM Student;
```

Aggregate functions in SQL

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.

Various Aggregate Functions

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

Count():

Count(*): Returns total number of records .i.e 6.
Count(salary): Return number of Non Null values over the column salary. i.e 5.
Count(Distinct Salary): Return number of distinct Non Null values over the column salary .i.e 4

Sum():

sum(salary): Sum all Non Null values of Column salary i.e., 310
sum(Distinct salary): Sum of all distinct Non-Null values i.e., 250.

Avg():

Avg(salary) = Sum(salary) / count(salary) = 310/5***Avg(Distinct salary)*** = sum(Distinct salary) / Count(Distinct Salary) = 250/4

Shubham Mishra

Min():

Min(salary): Minimum value in the salary column except NULL i.e., 40.***Max(salary):*** Maximum value in the salary i.e., 80.

ORDER BY:

The ORDER BY statement in SQL is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Sort according to one column:

To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

Syntax:-

```
SELECT * FROM table_name ORDER BY column_name ASC|DESC
//Where
table_name: name of the table.
column_name: name of the column according to which the data is needed to be arranged.
ASC: to sort the data in ascending order.
DESC: to sort the data in descending order.
| : use either ASC or DESC to sort in ascending or descending order//
```

Sort according to multiple columns:

To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by the (,) operator.

Syntax:-

```
SELECT * FROM table_name ORDER BY column1 ASC|DESC , column2 ASC|DESC
```

Note:

ASC is the default value for the ORDER BY clause. So, if we don't specify anything after the column name in the ORDER BY clause, the output will be sorted in ascending order by default.

Shubham Mishra

Sorting by column number (instead of name):

An integer that identifies the number of the column in the SelectItems in the underlying query of the SELECT statement. Column number must be greater than 0 and not greater than the number of columns in the result table. In other words, if we want to order by a column, that column must be specified in the SELECT list.

The rule checks for ORDER BY clauses that reference select list columns using the column number instead of the column name. The column numbers in the ORDER BY clause impairs the readability of the SQL statement. Further, changing the order of columns in the SELECT list has no impact on the ORDER BY when the columns are referred by names instead of numbers.

Syntax:-

```
Order by Column_Number asc/desc
```

```
SELECT Name, Address
FROM studentinfo
ORDER BY 1
```

GROUP BY

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.
- In the query , Group BY clause is placed before Having clause .
- Place condition in the having clause

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
```

Shubham Mishra

```
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:

Group By single column: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:

Syntax:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME;
```

Note: the rows with duplicate NAMEs are grouped under same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

Group By multiple columns:

Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group.

Syntax:

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

HAVING Clause:

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups? This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So

Shubham Mishra

we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:

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

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

Query:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME
HAVING SUM(SALARY)>3000;
```

ALL and ANY

ALL & ANY are logical operators in SQL. They return boolean value as a result.

ALL operator:

ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery.

- The ALL operator returns TRUE if all of the subqueries values meet the condition. The ALL must be preceded by comparison operators and evaluates true if all of the subqueries values meet the condition
- ALL is used with SELECT, WHERE, HAVING statement.

ALL with SELECT Statement:

```
Syntax:
SELECT ALL field_name
FROM table_name
WHERE condition(s);
```

Shubham Mishra

ALL with WHERE or HAVING Statement:

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name comparison_operator ALL
(SELECT column_name
FROM table_name
WHERE condition(s));
```

Example:

Consider the following Products Table and OrderDetails Table,

ProductID	ProductName	SupplierID	CategoryID	Price
1	Chais	1	1	18
2	Chang	1	1	19
3	Aniseed Syrup	1	2	10
4	Chef Anton's Cajun Seasoning	2	2	22
5	Chef Anton's Gumbo Mix	2	2	21
6	Boysenberry Spread	3	2	25
7	Organic Dried Pears	3	7	30
8	Northwoods Cranberry Sauce	3	2	40
9	Mishi Kobe Niku	4	6	97

OrderDetailsID	OrderID	ProductID	Quantity
1	10248	1	12
2	10248	2	10
3	10248	3	15
4	10249	1	8
5	10249	4	4
6	10249	5	6
7	10250	3	5
8	10250	4	18
9	10251	5	2
10	10251	6	8
11	10252	7	9
12	10252	8	9
13	10250	9	20
14	10249	9	4

Queries:-

Shubham Mishra

- **Find the name of the all the product.**

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

ProductName
Chais
Chang
Aniseed Syrup
Chef Anton's Cajun Seasoning
Chef Anton's Gumbo Mix
Boysenberry Spread
Organic Dried Pears
Northwoods Cranberry Sauce
Mishi Kobe Niku

Shubham Mishra

- **Find the name of the product if all the records in the OrderDetails has Quantity either equal to 6 or 2.**

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductId
                       FROM OrderDetails
                       WHERE Quantity = 6 OR Quantity = 2);
```

ProductName
Chef Anton's Gumbo Mix

- **Find the OrderID whose maximum Quantity among all product of that OrderID is greater than average quantity of all OrderID.**

```
SELECT OrderID
FROM OrderDetails
GROUP BY OrderID
HAVING max(Quantity) > ALL (SELECT avg(Quantity)
```

```
                        FROM OrderDetails
                        GROUP BY OrderID);
```

OrderID
10248
10250

ANY Operator

ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row.

- ANY return true if any of the subqueries values meet the condition.
- ANY must be preceded by comparison operators.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name comparison_operator ANY
(SELECT column_name
FROM table_name
WHERE condition(s));
```

Queries:-

- Find the Distinct CategoryID of the products which have any record in OrderDetails Table.

```
SELECT DISTINCT CategoryID
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                       FROM OrderDetails);
```

CategoryID
1
2
7
6

- Finds any records in the OrderDetails table that Quantity = 9

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
```

Shubham Mishra

```
                        FROM OrderDetails
                        WHERE Quantity = 9);
```

ProductName
Organic Dried Pears
Northwoods Cranberry Sauce

SELECT TOP Clause:-

SELECT TOP clause is used to fetch limited number of rows from a database. This clause is very useful while dealing with large databases.

```
SELECT column1,column2 FROM table_name LIMIT value;
column1 , column2: fields int the table
table_name: name of table
value: number of rows to return from top
```

Union Clause

The Union Clause is used to combine two separate select statements and produce the result set as a union of both the select statements.

Note:

- The fields to be used in both the select statements must be in same order, same number and same data type.
- The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

Basic Syntax:

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

Resultant set consists of distinct values.
```

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

Shubham Mishra

```
Resultant set consists of duplicate values too.
```

Queries:

- To fetch distinct ROLL_NO from Student and Student_Details table.

```
SELECT ROLL_NO FROM Student UNION SELECT ROLL_NO FROM Student_Details;
```

- To fetch ROLL_NO from Student and Student_Details table including duplicate values.

```
SELECT ROLL_NO FROM Student UNION ALL SELECT ROLL_NO FROM Student_Details;
```

- To fetch ROLL_NO , NAME from Student table WHERE ROLL_NO is greater than 3 and ROLL_NO , Branch from Student_Details table WHERE ROLL_NO is less than 3 , including duplicate values and finally sorting the data by ROLL_NO.

```
SELECT ROLL_NO,NAME FROM Student WHERE ROLL_NO>3
UNION ALL
SELECT ROLL_NO,Branch FROM Student_Details WHERE ROLL_NO<3
ORDER BY 1;

Note:The column names in both the select statements can be different but the
 data type must be same.And in the result set the name of column used in the first
 select statement will appear.
```

Shubham Mishra

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

Student_Details

ROLL_NO	Branch	Grade
1	Information Technology	O
2	Computer Science	E
3	Computer Science	O
4	Mechanical Engineering	A

Shubham Mishra

Note: use above table to find insights.

Intersect & Except clause

INTERSECT clause : As the name suggests, the intersect clause is used to provide the result of the intersection of two select statements. This implies the result contains all the rows which are common to both the SELECT statements.

Syntex:

```
SELECT column-1, column-2 ……
FROM table 1
WHERE…..

INTERSECT

SELECT column-1, column-2 ……
FROM table 2
```

```
WHERE…..
```

Example : Table 1 containing Employee Details

ID	Name	Age	City
1	Suresh	24	Delhi
2	Ramesh	23	pune
3	Kashish	34	Agra

Table 2 containing details of employees who are provided bonus

Bonus_ID	Employee_ID	Bonus (in RS.)
43	1	20,000
45	3	30,000

Shubham Mishra

Query :

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID

INTERSECT

SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Result:

ID	Name	Bonus
1	Suresh	20,000
3	Kashish	30,000

EXCEPT clause : contains all the rows that are returned by the first SELECT operation, and not returned by the second SELECT operation.

Syntax:

```
SELECT column-1, column-2 ……
FROM table 1
WHERE…..

EXCEPT

SELECT column-1, column-2 ……
FROM table 2
WHERE…..
```

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID

EXCEPT

SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Shubham Mishra

Aliases

Aliases are the temporary names given to table or column for the purpose of a particular SQL query. It is used when name of column or table is used other than their original names, but the modified name is only temporary.

- Aliases are created to make table or column names more readable.
- The renaming is just a temporary change and table name does not change in the original database.
- Aliases are useful when table or column names are big or not very readable.
- These are preferred when there are more than one table involved in a query.
- Generally table aliases are used to fetch the data from more than just single table and connect them through the field relations.

Shubham Mishra

Basic Syntax:

- **For column alias:**

```
SELECT column as alias_name FROM table_name;
column: fields in the table
alias_name: temporary alias name to be used in replacement of original column name
table_name: name of table
```

- **For table alias:**

```
SELECT column FROM table_name as alias_name;
column: fields in the table
table_name: name of table
alias_name: temporary alias name to be used in replacement of original table name
```

Queries:

- To fetch ROLL_NO from Student table using CODE as alias name.

```
SELECT ROLL_NO AS CODE FROM Student;
```

- To fetch Branch using Stream as alias name and Grade as CGPA from table Student_Details.

```
SELECT Branch AS Stream,Grade as CGPA FROM Student_Details;
```

- To fetch Grade and NAME of Student with Age = 20.

```
SELECT s.NAME, d.Grade FROM Student AS s, Student_Details
AS d WHERE s.Age=20 AND s.ROLL_NO=d.ROLL_NO;
```

Join (Inner, Left, Right and Full Joins)

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 as follows:

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

Consider the two tables below:

Student Table & Student Course

Shubham Mishra

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

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

Shubham Mishra

Note:- The simplest Join is INNER JOIN.

INNER JOIN:

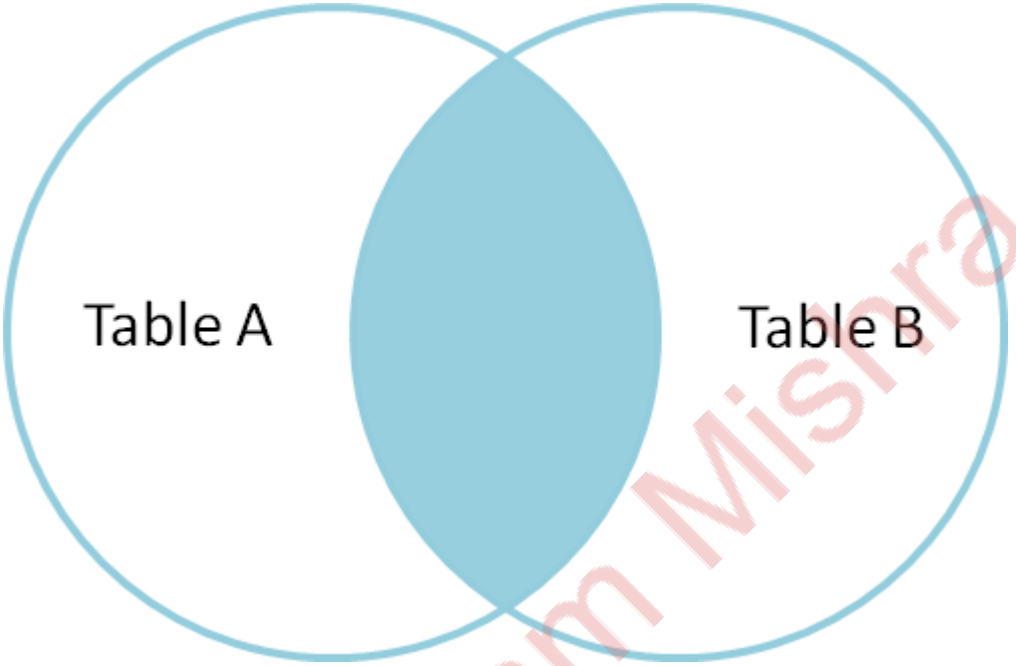
The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:-

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER 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 write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

INNER JOIN Query:-

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;
```

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

LEFT JOIN:

Shubham Mishra

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For 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.

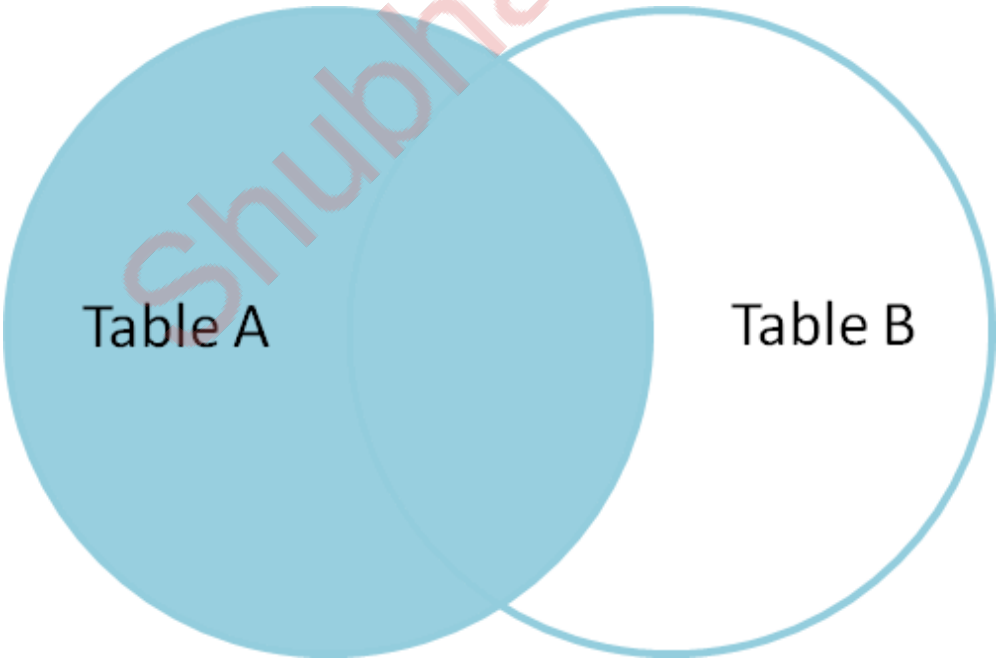
Syntax:

```
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.
```

Shubham Mishra

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



LEFT JOIN Query:-

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

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

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 the join. For 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.

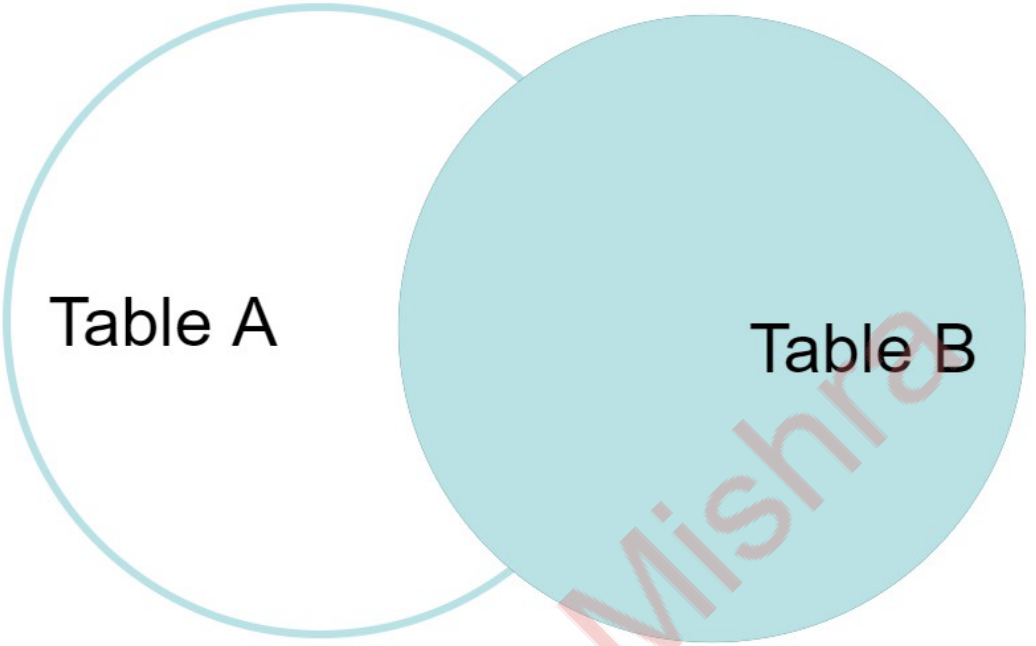
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT 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 RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.

Shubham Mishra



Shubham Mishra

RIGHT JOIN Query:

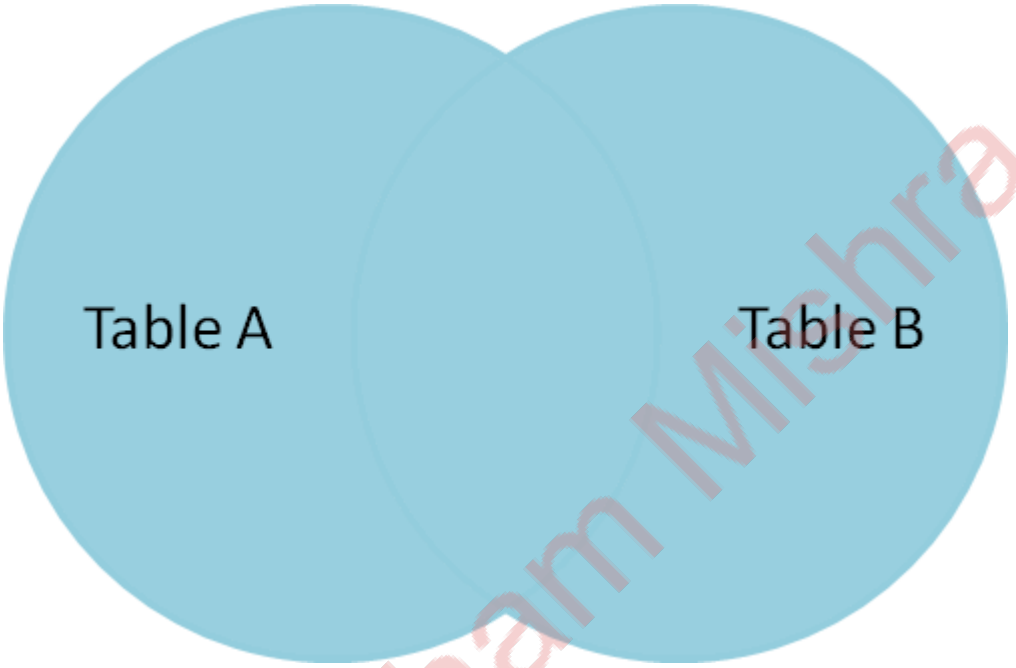
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

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

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 tables. For the rows for which there is no matching, the result-set will contain *NULL* values.



Shubham Mishra

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.
```

FULL JOIN Query:

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Join (Cartesian Join & Self Join):

Consider the two tables below:

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

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

Shubham Mishra

CARTESIAN JOIN:

The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

```
SELECT table1.column1 , table1.column2, table2.column1...
FROM table1
CROSS JOIN table2;


table1: First table.
table2: Second table
```

CARTESIAN JOIN Query:

In the below query we will select NAME and Age from Student table and COURSE_ID from StudentCourse table. In the output you can see that each row of the table Student is joined with every row of the table StudentCourse. The total rows in the result-set = 4 * 4 = 16.

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;
```

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

Shubham Mishra

SELF JOIN:

As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some

conditions. In other words we can say that it is a join between two copies of the same table.

Syntax:

```
SELECT a.coulmn1 , b.column2
FROM table_name a, table_name b
WHERE some_condition;

table_name: Name of the table.
some_condition: Condition for selecting the rows.
```

Self join Query:

```
SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;
```

Output:

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

Shubham Mishra

USING Clause:

If several columns have the same names but the datatypes do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an EQUIJOIN.

- USING Clause is used to match only one column when more than one column matches.
- NATURAL JOIN and USING Clause are mutually exclusive.
- It should not have a qualifier(table name or Alias) in the referenced columns.
- NATURAL JOIN uses all the columns with matching names and datatypes to join the tables.

The USING Clause can be used to specify only those columns that should be used for an EQUIJOIN.

Queries:-

- Write SQL query to find the working location of the employees. Also give their respective employee_id and last_name?

```
SELECT e.EMPLOYEE_ID, e.LAST_NAME, d.LOCATION_ID
FROM Employees e JOIN Departments d
USING(DEPARTMENT_ID);
Output :
```

Note:The example shown joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS
tables, and thus shows the location where an employee works.

- Write SQL query to find the location_id, street_address, postal_code and their respective country name?

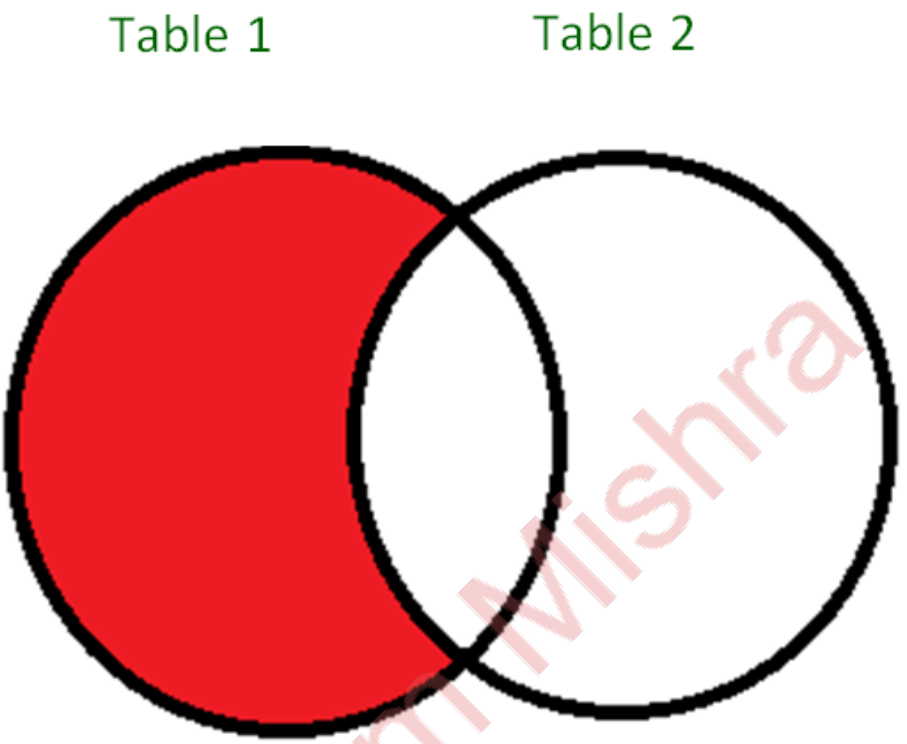
```
SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id);
```

NOTE: When we use the USING clause in a join statement, the join column is not qualified with table Alias. Do not Alias it even if the same column is used elsewhere in the SQL statement.

MINUS Operator

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries.

Shubham Mishra



Shubham Mishra

As you can see is in the above diagram, the MINUS operator will return only those rows which are present in the result set from Table1 and not present in the result set of Table2.

Syntax:

```
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition
MINUS
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition;

columnN: column1, column2.. are the name of columns of the table.
```

Note:

- The WHERE clause is optional in the above query.
- The number of columns in both SELECT statements must be same.
- The data type of corresponding columns of both SELECT statement must be same.

- **The MINUS operator is not supported with all databases. It is supported by Oracle database but not SQL server or PostgreSQL.**

Views

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 Table: StudentDetails & StudentMarks

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

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	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
```

Queries:-

- **Creating View from a single table:**
  In this example we will create a View named DetailsView from the table StudentDetails.

```
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.

Shubham Mishra

```
SELECT * FROM DetailsView;
```

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- In this example, we will create a view named StudentNames from the table StudentDetails.

```
CREATE VIEW StudentNames AS
SELECT S_ID, NAME
FROM StudentDetails
ORDER BY NAME;
```

- If we now query the view as,

```
SELECT * FROM StudentNames;
```

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

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.

```
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;
```

Shubham Mishra

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS:

SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax:

```
DROP VIEW view_name;

view_name: Name of the View which we want to delete.
```

UPDATING VIEWS:

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.

1. The SELECT statement should not have the DISTINCT keyword.
2. The View should have all NOT NULL values.
3. The view should not be created using nested queries or complex queries.
4. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

- We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,column2,..
FROM table_name
WHERE condition;
```

- For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

```
CREATE OR REPLACE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE
FROM StudentDetails, StudentMarks
```

Shubham Mishra

```
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

- If we fetch all the data from MarksView now as:

```
SELECT * FROM MarksView;
```

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

Inserting a row in a view: We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

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

view_name: Name of the View
```

Deleting a row from a View: Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

```
DELETE FROM view_name
WHERE condition;

view_name:Name of view from where we want to delete rows
condition: Condition to select rows
```

```
DELETE FROM DetailsView
WHERE NAME="Suresh";
```

WITH CHECK OPTION: The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

- The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.

Shubham Mishra

- If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example: In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause

```
CREATE VIEW SampleView AS
SELECT S_ID, NAME
FROM  StudentDetails
WHERE NAME IS NOT NULL
WITH CHECK OPTION;
```

NOTE: The default value of NAME column is *null*.
Uses of a View : A good database should contain views due to the given reasons:

1. **Restricting data access –** Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity –** A view can hide the complexity that exists in a multiple table join.
3. **Simplify commands for the user –** Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries –** Views can be used to store complex queries.
5. **Rename Columns –** Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to hide the names of the columns of the base tables.
6. **Multiple view facility –** Different views can be created on the same table for different users.

Creating Roles

A role is created to ease setup and maintenance of the security model. It is a named group of related privileges that can be granted to the user. When there are many users in a database it becomes difficult to grant or revoke privileges to users. Therefore, if you define roles:

- You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.
- You can either create Roles or use the system roles pre-defined.

Snubham Mishra

Some of the privileges granted to the system roles are as given below:

System Roles	
Privileges granted to the Role	
Connect	Create table, Create view, Create synonym, Create sequence, Create session etc.
Resource	Create Procedure, Create Sequence, Create Table, Create Trigger etc. The primary usage of the Resource role is to restrict access to database objects.
DBA	All system privileges

Creating and Assigning a Role –

First, the (Database Administrator)DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax –

```
CREATE ROLE manager;
Role created.
```

In the syntax:
'manager' is the name of the role to be created.

- Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.
- It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.
- If a role is identified by a password, then GRANT or REVOKE privileges have to be identified by the password.

Grant privileges to a role –

```
GRANT create table, create view
```

Shubham Mishra

```
TO manager;
Grant succeeded.
```

Grant a role to users

```
GRANT manager TO SAM, STARK;
Grant succeeded.
```

Revoke privilege from a Role :

```
REVOKE create table FROM manager;
```

Drop a Role :

```
DROP ROLE manager;
```

Explanation –
Firstly it creates a manager role and then allows managers to create tables and views. It then grants Sam and Stark the role of managers. Now Sam and Stark can create tables and views. If users have multiple roles granted to them, they receive all of the privileges associated with all of the roles. Then create table privilege is removed from role 'manager' using Revoke.The role is dropped from the database using drop.

Shubham Mishra

Constraints

Constraints are the rules that we can apply on 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 any more.
- **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 which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as 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 constraint in detail.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept 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
```

Shubham Mishra

```
(
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 columns 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 which uniquely identifies each row in the table. If a field in a table as 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 combination of NOT NULL and UNIQUE constraints.

A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as 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 which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Consider the two tables as shown below:

Shubham Mishra

OrdersTable

O_ID
ORDER_NO
C_ID
1
2253
3
2
3325
3
3
4521
2
4
8532
1

CustomersTable

C_ID
NAME
ADDRESS

Shubham Mishra

1
RAMESH
DELHI
2
SURESH
NOIDA
3
DHARMESH
GURGAON

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

- **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.

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

Shubham Mishra

```
AGE int NOT NULL CHECK (AGE >= 18)
);
```

- **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
);
```

TRANSACTIONS

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.
Example of a transaction to transfer $150 from account A to account B:
1. read(A)
2. A := A – 150
3. write(A)
4. read(B)
5. B := B + 150
6. write(B)

Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable.
These are popularly known as ACID properties. These properties can ensure the concurrent execution of multiple transactions without conflict.

How to implement Transactions using SQL?
Following commands are used to control transactions. It is important to note that these statements cannot be used while creating tables and are only used with the DML Commands such as – INSERT, UPDATE and DELETE.

Shubham Mishra

1. BEGIN TRANSACTION: It indicates the start point of an explicit or local transaction.

Syntax:

```
BEGIN TRANSACTION transaction_name ;
```

2. SET TRANSACTION: Places a name on a transaction.

Syntax:

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

3. COMMIT: If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

Syntax:

```
COMMIT;
```

Example: Sample table 1

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

Following is an example which would delete those records from the table which have age = 20 and then COMMIT the changes in the database. **Queries:**

```
DELETE FROM Student WHERE AGE = 20;
COMMIT;
```

Output: Thus, two rows from the table would be deleted and the SELECT statement would look like,

Shubham Mishra

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

4. ROLLBACK: If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Syntax:

```
ROLLBACK;
```

Example: From the above example **Sample table1**, Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database.

Queries:

```
DELETE FROM Student WHERE AGE = 20;
ROLLBACK;
```

Output:

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

5. SAVEPOINT: creates points within the groups of transactions in which to ROLLBACK. A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax for Savepoint command:

```
SAVEPOINT SAVEPOINT_NAME;
```

This command is used only in the creation of SAVEPOINT among all the transactions. In general ROLLBACK is used to undo a group of transactions. **Syntax for rolling back to Savepoint command:**

Shubham Mishra

```
ROLLBACK TO SAVEPOINT_NAME;
```

you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example: From the above example **Sample table1**, Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database by keeping Savepoints.

Queries:

```
SAVEPOINT SP1;
//Savepoint created.
DELETE FROM Student WHERE AGE = 20;
//deleted
SAVEPOINT SP2;
//Savepoint created.
```

Here SP1 is first SAVEPOINT created before deletion.In this example one deletion have taken place. After deletion again SAVEPOINT SP2 is created. Output:

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

Deletion have been taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP1 which is before deletion. deletion is undone by this statement ,

```
ROLLBACK TO SP1;
//Rollback completed.
```

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

6. RELEASE SAVEPOINT:- This command is used to remove a SAVEPOINT that you have created.

Shubham Mishra

Syntax:

```
RELEASE SAVEPOINT SAVEPOINT_NAME
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.
It is used to initiate a database transaction and used to specify characteristics of the transaction that follows.

SQL Server Mathematical functions (SQRT, PI, SQUARE, ROUND, CEILING & FLOOR)

Mathematical functions are present in SQL server which can be used to perform mathematical calculations. Some commonly used mathematical functions are given below:

1. SQRT(): SQRT() function is the most commonly used function. It takes any numeric values and returns the square root value of that number.

Syntax:

```
SELECT SQRT(..value..)
```

2. PI(): There are calculations which require use of pi. Using pi() function, value of PI can be used anywhere in the query.

Syntax:

```
SELECT PI()
```

3. POWER(): POWER() function is used to find the square of any number.

Syntax:

```
SELECT SQUARE(..value..)
```

4. ROUND(): ROUND() function is used to round a value to the nearest specified decimal place.

Syntax:

Shubham Mishra

```
SELECT ROUND(..value.., number_of_decimal_places)
```

5. CEILING(): CEILING() function is used to find the next highest value (integer).

Syntax:

```
SELECT CEILING(..value..)
```

6. FLOOR(): FLOOR() function returns the next lowest value (integer).

Syntax:

```
SELECT FLOOR(..value..)
```

Date functions

while working with database, the format of the date in table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

In MySql the default date functions are:

- **NOW():** Returns the current date and time. Example:

```
SELECT NOW();
```

- **CURDATE()**: Returns the current date. Example:

```
SELECT CURDATE();
```

- **CURTIME():** Returns the current time. Example:

```
SELECT CURTIME();
```

- **DATE()**: Extracts the date part of a date or date/time expression.
- Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Shubham Mishra

```
SELECT Name, DATE(BirthTime) AS BirthDate FROM Test;
```

Output:

Name	BirthDate
Pratik	1996-09-26

- **EXTRACT():** Returns a single part of a date/time.

  Syntax:

```
EXTRACT(unit FROM date);
```

There are several units that can be considered but only some are used such as:MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.And 'date' is a valid date expression.
Example:For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Shubham Mishra

```
SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;
```

```
SELECT Name, Extract(YEAR FROM BirthTime) AS BirthYear FROM Test;
```

```
SELECT Name, Extract(SECOND FROM BirthTime) AS BirthSecond FROM Test;
```

- **DATE_ADD() :** Adds a specified time interval to a date

Syntax:

```
DATE_ADD(date, INTERVAL expr type);
```

Where, date – valid date expression and expr is the number of interval we want to add.
and type can be one of the following:
MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.

Queries:

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL 1 YEAR) AS BirthTimeModified FROM Test;
```

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL 30 DAY) AS BirthDayModified FROM Test;
```

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL 4 HOUR) AS BirthHourModified FROM Test;
```

- **DATE_SUB():**Subtracts a specified time interval from a date. Syntax for DATE_SUB is same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.
- **DATEDIFF():** Returns the number of days between two dates.
- Syntax:
  Example:

```
SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;
```

- **DATE_FORMAT():** Displays date/time data in different formats.

Syntax:

```
DATE_FORMAT(date,format);
```

date is a valid date and format specifies the output format for the date/time. The formats that can be used are:

- %a-Abbreviated weekday name (Sun-Sat)
- %b-Abbreviated month name (Jan-Dec)
- %c-Month, numeric (0-12)
- %D-Day of month with English suffix (0th, 1st, 2nd, 3rd)
- %d-Day of month, numeric (00-31)

Shubham Mishra

- %e-Day of month, numeric (0-31)
- %f-Microseconds (000000-999999)
- %H-Hour (00-23)
- %h-Hour (01-12)
- %I-Hour (01-12)
- %i-Minutes, numeric (00-59)
- %j-Day of year (001-366)
- %k-Hour (0-23)
- %l-Hour (1-12)
- %M-Month name (January-December)
- %m-Month, numeric (00-12)
- %p-AM or PM
- %r-Time, 12-hour (hh:mm:ss followed by AM or PM)
- %S-Seconds (00-59)
- %s-Seconds (00-59)
- %T-Time, 24-hour (hh:mm:ss)
- %U-Week (00-53) where Sunday is the first day of week
- %u-Week (00-53) where Monday is the first day of week
- %V-Week (01-53) where Sunday is the first day of week, used with %X
- %v-Week (01-53) where Monday is the first day of week, used with %x
- %W-Weekday name (Sunday-Saturday)
- %w-Day of the week (0=Sunday, 6=Saturday)
- %X-Year for the week where Sunday is the first day of week, four digits, used with %V
- %x-Year for the week where Monday is the first day of week, four digits, used with %v
- %Y-Year, numeric, four digits
- %y-Year, numeric, two digits

```
DATE_FORMAT(NOW(),'%d %b %y')
```

Shubham Mishra