

SQL- MOST IMPORTANT CONCEPTS

PLACEMENT PREPARATION

[EXCLUSIVE NOTES]

[SAVE AND SHARE]

Curated By- HIMANSHU KUMAR(LINKEDIN)

TOPICS COVERED-

PART-1 :-

- **SELECT Query In SQL**
- **Distinct Clause In SQL**
- **INSERT INTO Query In SQL**
- **INSERT INTO Statement In SQL**
- **DELETE Statement In SQL**
- **UPDATE Statement In SQL**
- **SELECT TOP Clause In SQL**
- **ORDER BY In SQL**
- **Aliases In SQL**
- **Wildcard operators In SQL**
- **Join (Inner, Left, Right and Full Joins)**
- **CREATE In SQL**



HIMANSHU KUMAR(LINKEDIN)

<https://www.linkedin.com/in/himanshukumarmahuri>

SELECT Query :-

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:

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

Basic 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;
```

Output:

ROLL_NO	NAME	Age
1	Ram	18
2	RAMESH	18
3	SUJIT	20
4	SURESH	18

- To fetch all the fields from the table Student:

```
SELECT * FROM Student;
```

Output:

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

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
```

column1, column2 : Names of the fields of the table.

table_name : Table from where we want to fetch the records.

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

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

Table - 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 unique names from the NAME field -

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

Output :

NAME
Ram
RAMESH
SUJIT
SURESH

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

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

Output :

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

Note : Without the keyword distinct in both the above examples 6 records would have been fetched instead of 4, since in the original table there are 6 records with the duplicate values.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

INSERT INTO Query :-

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:

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

Syntax:

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

3. **table_name:** name of the table.

4. **value1, value2,.. :** value of first column, second column,... for the new record

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

Syntax:

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

7. **table_name:** name of the table.

8. **column1:** name of first column, second column ...

value1, value2, value3 : value of first column, second column,... for the new record

Student				
ROLL_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

Queries:**Method 1 example:**

```
INSERT INTO Student VALUES ('5', 'HARSH', 'WEST BENGAL', '8759770477', '19');
```

Output: The table **Student** will now look like:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9562431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18
3	SUJIT	ROHTAK	9156253131	20
2	RAMESH	GURGAON	9562431543	18
5	HARSH	WEST BENGAL	8759770477	19

Method 2 (Inserting values in only specified columns):

```
INSERT INTO Student (ROLL_NO, NAME, Age) VALUES ('5', 'HARSH', '19');
```

Output: The table **Student** will now look like:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9562431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18
3	SUJIT	ROHTAK	9156253131	20
2	RAMESH	GURGAON	9562431543	18
5	HARSH	WEST BENGAL	8759770477	19

INSERT INTO Statement-

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:

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

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

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

**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

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:

Method 1 (Inserting only values) :

INSERT INTO Student VALUES ('5','HARSH','WEST BENGAL','XXXXXXXXXX','19');

Output: The table **Student** will now look like:

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

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
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');
```

Output: The table **Student** will now look like:

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
5	PRATIK	null	null	19

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 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.

INSERT INTO first_table SELECT * FROM second_table;

first_table: name of first table.

second_table: name of second table.

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.

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.

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.

Table2: LateralStudent

ROLL_NO	NAME	ADDRESS	PHONE	Age
7	SOUVIK	DUMDUM	XXXXXXXXXX	18
8	NIRAJ	NOIDA	XXXXXXXXXX	19
9	SOMESH	ROHTAK	XXXXXXXXXX	20

Queries:

Method 1(Inserting all rows and columns):

INSERT INTO Student SELECT * FROM LateralStudent;

Output: This query will insert all the data of the table LateralStudent in the table Student. The table Student will now look like,

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

ROLL_NO	NAME	ADDRESS	PHONE	Age
2	RAMESH	GURGAON	XXXXXXXXXX	18
7	SOUVIK	DUMDUM	XXXXXXXXXX	18
8	NIRAJ	NOIDA	XXXXXXXXXX	19
9	SOMESH	ROHTAK	XXXXXXXXXX	20

• Method 2(Inserting specific columns):

INSERT INTO Student(ROLL_NO,NAME,Age) SELECT ROLL_NO, NAME, Age FROM LateralStudent;

Output: This query will insert the data in the columns ROLL_NO, NAME and Age of the table LateralStudent in the table Student and the remaining columns in the Student table will be filled by *null* which is the default value of the remaining columns. The table Student will now look like,

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

ROLL_NO	NAME	ADDRESS	PHONE	Age
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
7	SOUVIK	null	null	18
8	NIRAJ	null	null	19
9	SOMESH	null	null	20

- Select specific rows to insert:**

INSERT INTO Student SELECT * FROM LateralStudent WHERE Age = 18;

Output: This query will select only the first row from table LateralStudent to insert into the table Student. The table Student will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
7	SOUVIK	DUMDUM	XXXXXXXXXXXX	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,.....),
      ..... ;
```

table_name: name of the table

Column1: name of first column, second column ...

Value1, Value2, Value3 : value of first column, second column,... for each new row inserted

You need To provide Multiple lists of values where each list is separated by ",". Every list of value corresponds to values to be inserted in each new row of the table.

Values in the next list tells values to be inserted in the next Row of the table.

Example:

The following SQL statement insert multiple rows in Student Table.

Input :

```
INSERT INTO STUDENT(ID, NAME,AGE,GRADE,CITY) VALUES(1,"AMIT KUMAR",15,10,"DELHI"),
                                                    (2,"GAURI RAO",18,12,"BANGALORE"),
                                                    (3,"MANAV BHATT",17,11,"NEW DELHI"),
                                                    (4,"RIYA KAPOOR",10,5,"UDAIPUR");
```

Output : STUDENT TABLE This query will insert all values in each successive row in the STUDENT TABLE . Thus STUDENT Table will look like this:

ID	NAME	AGE	GRADE	CITY
1	AMIT KUMAR	15	10	DELHI
2	GAURI RAO	16	12	BANGALORE
3	MANAV BHATT	17	11	NEW DELHI
4	RIYA KAPOOR	10	5	UDAIPUR

DELETE Statement –

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.

Sample Table:

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

Example Queries:

- **Deleting single record:** Delete the rows where NAME = 'Ram'. This will delete only the first row.
- `DELETE FROM Student WHERE NAME = 'Ram' ;`

Output: The above query will delete only the first row and the table **Student** will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
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

- **Deleting multiple records:** Delete the rows from the table Student where Age is 20. This will delete 2 rows(third row and

fifth row).

- `DELETE FROM Student WHERE Age = 20;`

Output: The above query will delete two rows(third row and fifth row) and the table **Student** will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXXXX	18

- **Delete all of the records:** There are two queries to do this as shown below,
 - query1: `"DELETE FROM Student";`
 - query2: `"DELETE * FROM Student";`

Output: All of the records in the table will be deleted, there are no records left to display. The table **Student** will become empty!

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.

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.

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

Example Queries

- **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 ;
```

Output: This query will update two rows(third row and fifth row) and the table **Student** will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX X	18
2	RAMESH	GURGAON	XXXXXXXXXX X	18
3	PRATIK	ROHTAK	XXXXXXXXXX X	20
4	SURESH	Delhi	XXXXXXXXXX X	18
3	PRATIK	ROHTAK	XXXXXXXXXX X	20
2	RAMESH	GURGAON	XXXXXXXXXX X	18

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;
```

Output

The above query will update two columns in the first row and the table **Student** will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	PRATIK	SIKKIM	XXXXXXXXXX X	18
2	RAMESH	GURGAON	XXXXXXXXXX X	18
3	PRATIK	ROHTAK	XXXXXXXXXX X	20
4	SURESH	Delhi	XXXXXXXXXX X	18
3	PRATIK	ROHTAK	XXXXXXXXXX X	20
2	RAMESH	GURGAON	XXXXXXXXXX X	18

Note: For updating multiple columns we have used comma(,) to separate the names and values of two columns.

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

- UPDATE Student SET NAME = 'PRATIK';

Output: The table **Student** will now look like,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	PRATIK	Delhi	XXXXXXXXXXXX	18
2	PRATIK	GURGAON	XXXXXXXXXXXX	18

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	PRATIK	ROHTAK	XXXXXXXXXXXX	20
4	PRATIK	Delhi	XXXXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXXXX	20
2	PRATIK	GURGAON	XXXXXXXXXXXX	18

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.

Basic Syntax:

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

Syntax using Percent

- `SELECT TOP value PERCENT column1,column2 FROM table_name;`
- `value`: percentage of number of rows to return from top
- `column1 , column2`: fields in the table
- `table_name`: name of table

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 first two data set from Student table.

```
SELECT TOP 2 * FROM Student;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX XXX	18
2	RAMESH	GURGAON	XXXXXXXXXX XXX	18

To fetch 50 percent of the total records from Student table.

```
SELECT TOP 50 PERCENT * FROM Student;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX XX	18
2	RAMESH	GURGAON	XXXXXXXXXX XX	18
3	SUJIT	ROHTAK	XXXXXXXXXX XX	20

NOTE: To get the same functionality on MySQL and Oracle databases there is a bit of difference in the basic syntax;

Equivalent Syntaxes are as follows:

For MySQL 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

For Oracle databases:

```
SELECT column1,column2 FROM table_name WHERE ROWNUM <= value;
```

column1 , column2: fields int the table

table_name: name of table

value: number of rows to return from top

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
```

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

Now consider the above database table and find the results of different queries.

Sort according to a single column:

In this example, we will fetch all data from the table Student and sort the result in descending order according to the column ROLL_NO.

Query:

```
SELECT * FROM Student ORDER BY ROLL_NO DESC;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
4	DEEP	RAMNAGAR	XXXXXXXXXX	18

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	RIYANKA	SILIGURI	XXXXXXXXXXXX	20
2	PRATIK	BIHAR	XXXXXXXXXXXX	19
1	HARSH	DELHI	XXXXXXXXXXXX	18

In the above example, if we want to sort in ascending order we have to use ASC in place of DESC.

Sort according to multiple columns:

In this example we will fetch all data from the table Student and then sort the result in ascending order first according to the column Age. and then in descending order according to the column ROLL_NO.

Query:

```
SELECT * FROM Student ORDER BY Age ASC , ROLL_NO DESC;
```

Output:

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

In the above output, we can see that first the result is sorted in ascending order according to Age. There are multiple rows of having the same Age.

Now, sorting further this result-set according to ROLL_NO will sort the rows with the same Age according to ROLL_NO in descending order.

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.

Take another example of the following query will give similar output as the above:

Query:

```
SELECT * FROM Student ORDER BY Age , ROLL_NO DESC;
```

Output:

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

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:

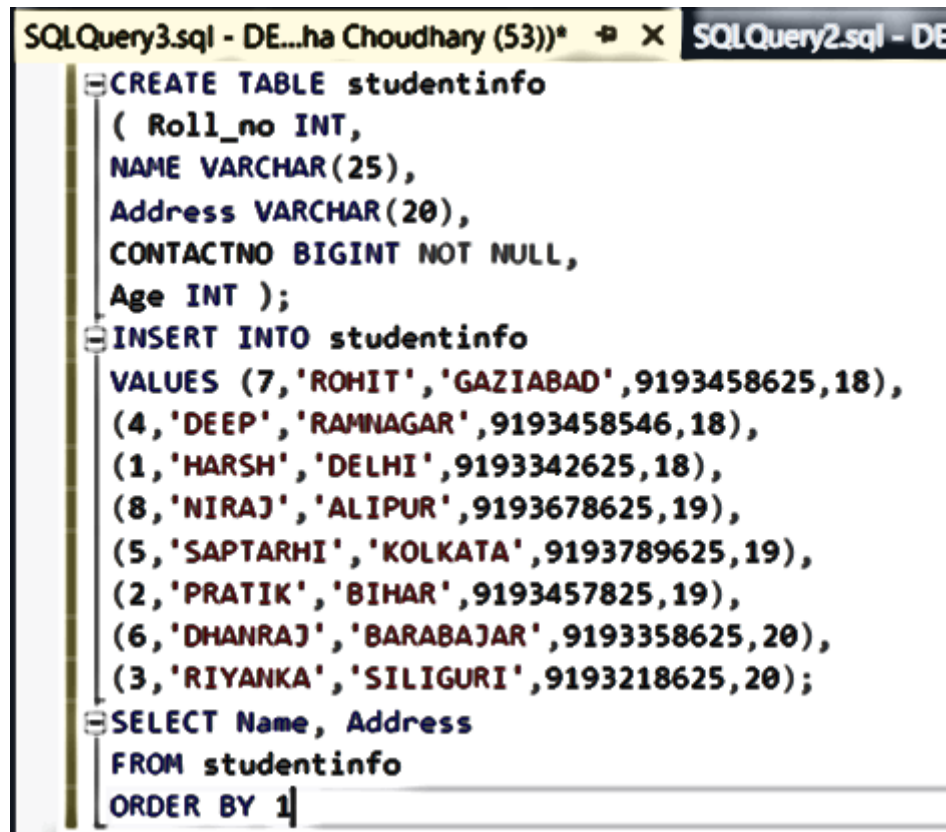
```
Order by Column_Number asc/desc
```

Here we take an example to sort a database table according to column 1 i.e Roll_Number. For this a query will be:

Query:

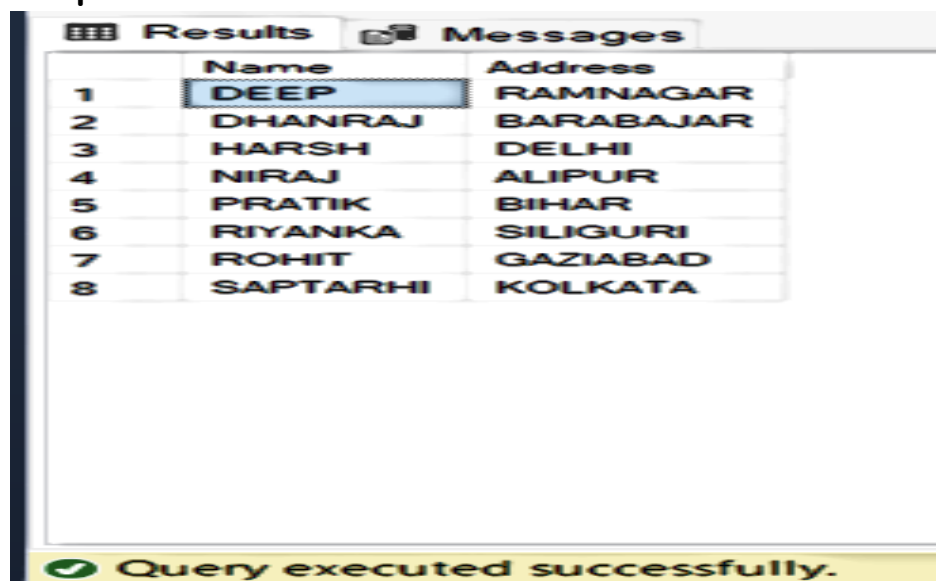
```
CREATE TABLE studentinfo
( Roll_no INT,
NAME VARCHAR(25),
Address VARCHAR(20),
CONTACTNO BIGINT NOT NULL,
Age INT );
INSERT INTO studentinfo
VALUES (7, 'ROHIT', 'GAZIABAD', 9193458625, 18),
(4, 'DEEP', 'RAMNAGAR', 9193458546, 18),
(1, 'HARSH', 'DELHI', 9193342625, 18),
(8, 'NIRAJ', 'ALIPUR', 9193678625, 19),
(5, 'SAPTARHI', 'KOLKATA', 9193789625, 19),
```

```
(2, 'PRATIK', 'BIHAR', 9193457825, 19),  
(6, 'DHANRAJ', 'BARABAJAR', 9193358625, 20),  
(3, 'RIYANKA', 'SILIGURI', 9193218625, 20);  
SELECT Name, Address  
FROM studentinfo  
ORDER BY 1
```



```
SQLQuery3.sql - DE...ha Choudhary (53)) * X SQLQuery2.sql - DE  
CREATE TABLE studentinfo  
( Roll_no INT,  
  NAME VARCHAR(25),  
  Address VARCHAR(20),  
  CONTACTNO BIGINT NOT NULL,  
  Age INT );  
INSERT INTO studentinfo  
VALUES (7, 'ROHIT', 'GAZIABAD', 9193458625, 18),  
(4, 'DEEP', 'RAMNAGAR', 9193458546, 18),  
(1, 'HARSH', 'DELHI', 9193342625, 18),  
(8, 'NIRAJ', 'ALIPUR', 9193678625, 19),  
(5, 'SAPTARHI', 'KOLKATA', 9193789625, 19),  
(2, 'PRATIK', 'BIHAR', 9193457825, 19),  
(6, 'DHANRAJ', 'BARABAJAR', 9193358625, 20),  
(3, 'RIYANKA', 'SILIGURI', 9193218625, 20);  
SELECT Name, Address  
FROM studentinfo  
ORDER BY 1
```

Output:



	Name	Address
1	DEEP	RAMNAGAR
2	DHANRAJ	BARABAJAR
3	HARSH	DELHI
4	NIRAJ	ALIPUR
5	PRATIK	BIHAR
6	RIYANKA	SILIGURI
7	ROHIT	GAZIABAD
8	SAPTARHI	KOLKATA

Query executed successfully.

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.

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

Student_Details

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

Queries for illustrating column alias

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

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

Output:

CODE
1
2
3
4

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 ;
```


Output:

Stream	CGPA
Information Technology	O
Computer Science	E
Computer Science	O
Mechanical Engineering	A

Queries for illustrating table alias

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

Generally table aliases are used to fetch the data from more than just single table and connect them through the field relations.

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;
```

Output:

NAME	Grade
SUJIT	O

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Wildcard operators –

Prerequisite: **SQL | WHERE Clause** In the above mentioned article WHERE Clause is discussed in which LIKE operator is also explained, where you must have encountered the word wildcards now lets get deeper into Wildcards.

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.

Basic 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

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 records from Student table with NAME ending with letter 'T'.

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

Output:

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

To fetch records from Student table with NAME ending any letter but starting from 'RAMES'.

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

Output:

2RAMESHGURGAONXXXXXXXXXX18

ROLL_NO	NAME	ADDRESS	PHONE	Age
2	RAMESH	GURGAON	XXXXXXXXXXXX	18

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

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

Output:

2RAMESHGURGAONXXXXXXXXXX18

ROLL_NO	NAME	ADDRESS	PHONE	Age
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXXXX	20

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

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

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXXXX	18

- 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%';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

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

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

Output:

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

- 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%';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXXXX	20

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

StudentCourse

The simplest Join is INNER JOIN.

A. 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.

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:**B. LEFT JOIN**

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.
```

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

Example Queries(LEFT JOIN):

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

Output:**C. 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;
```

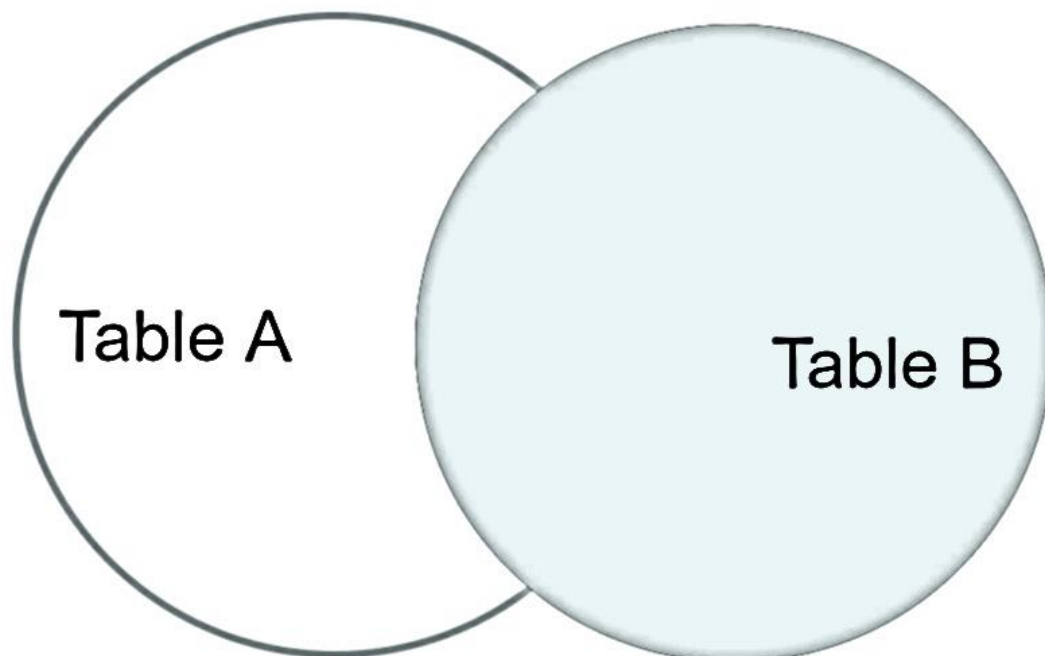


```
table1: First table.
```

```
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.



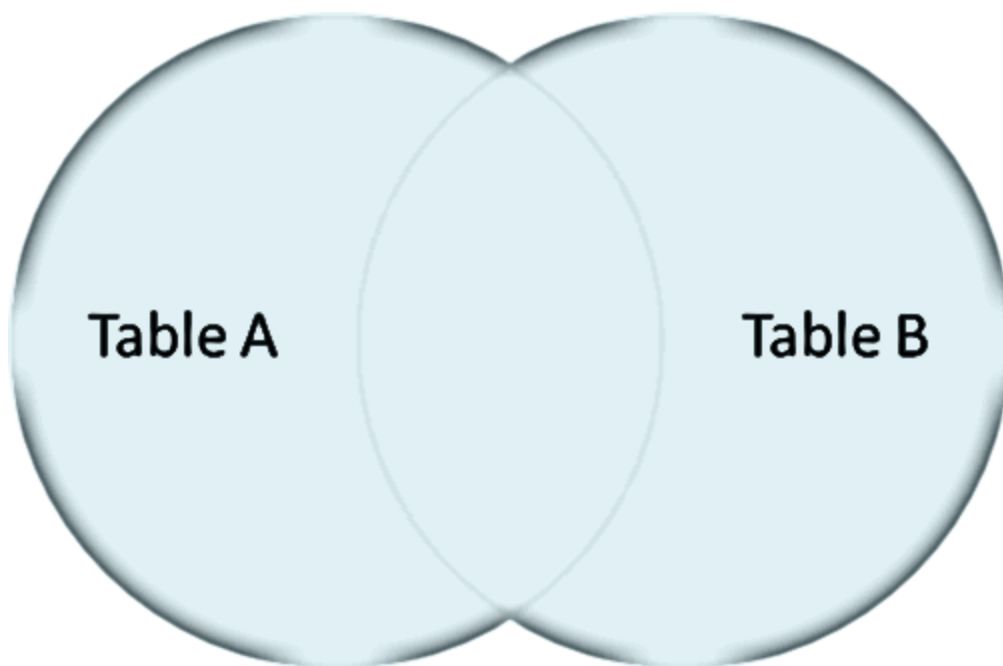
Example Queries **RIGHT JOIN**:

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

Output:

D. 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.



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	4
NULL	5
NULL	4

CREATE-

There are two CREATE statements available in SQL:

CREATE DATABASE

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 *my_database*.

```
CREATE DATABASE my_database;
```

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 of 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 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 number of maximum 10 digits.

Example Query: This query will create a table named Students with three columns, ROLL_NO, NAME and SUBJECT.

```
CREATE TABLE Students  
  
(  
ROLL_NO int(3),  
NAME varchar(20),  
SUBJECT varchar(20),);
```

This query will create a table named Students. The ROLL_NO field is of type int and can store an integer number of size 3. The next two columns NAME and SUBJECT are of type varchar and can store characters and the size 20 specifies that these two fields can hold maximum of 20 characters.

Part -2 topics (UPCOMING)

- Constraints
- Comments
- GROUP BY
- Views
- Functions (Aggregate and Scalar Functions)
- Query Processing
- WHERE Clause
- AND and OR operators
- Union Clause
- Join (Cartesian Join & Self Join)
- DROP, DELETE, TRUNCATE
- DROP, TRUNCATE
- Date functions
- EXISTS
- WITH clause
- NULL Values
- ALL and ANY
- BETWEEN & IN Operator
- Arithmetic Operators
- DDL, DML, TCL and DCL
- Creating Roles



HIMANSHU KUMAR(LINKEDIN)

<https://www.linkedin.com/in/himanshukumarmahuri>

CREDITS- INTERNET

DISCLOSURE- THE DATA AND IMAGES ARE TAKEN FROM GOOGLE AND INTERNET.

CHECKOUT AND DOWNLOAD MY ALL NOTES

LINK- https://linktr.ee/exclusive_notes