SQL-GFG

SQL | 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	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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	XXXXXXXXXX18
2	RAMESH	GURGAON	XXXXXXXXXX18
3	SUJIT	ROHTAK	XXXXXXXXXX20
4	SURESH	DELHI	XXXXXXXXXX18
3	SUJIT	ROHTAK	XXXXXXXXXX20
2	RAMESH	GURGAON	XXXXXXXXXX18

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	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	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.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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.

4. table_name: name of the table.
```

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

6. **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:

```
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	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	945512345	118
2	RAMESH	GURGAON	956243154	318
3	SUJIT	ROHTAK	915625313	120
4	SURESH	Delhi	915676897	118
3	SUJIT	ROHTAK	915625313	120
2	RAMESH	GURGAON	956243154	318
5	HARSH	WEST BENGAL	875977047	719

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	945512345	118
2	RAMESH	GURGAON	956243154	318
3	SUJIT	ROHTAK	915625313	120
4	SURESH	Delhi	915676897	118
3	SUJIT	ROHTAK	915625313	120
2	RAMESH	GURGAON	956243154	318
5	HARSH	WEST BENGAL	875977047	719

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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
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	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
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
2	RAMESH	GURGAON	XXXXXXXXXX	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	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
7	SOUVIK	DUMDUM	XXXXXXXXXX	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

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	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!

SQL Quiz

SQL | 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	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	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	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	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	XXXXXXXXXX	18
2	PRATIK	GURGAON	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
4	PRATIK	Delhi	XXXXXXXXXX	18
3	PRATIK	ROHTAK	XXXXXXXXXX	20
2	PRATIK	GURGAON	XXXXXXXXXX	18

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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	18
2	RAMESH	GURGAON	XXXXXXXXXX	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	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	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
```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL - 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
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
2	PRATIK	BIHAR	XXXXXXXXXX	19
1	HARSH	DELHI	XXXXXXXXXX	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	XXXXXXXXXX	18
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
1	HARSH	DELHI	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
2	PRATIK	BIHAR	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
3	RIYANKA	SILIGURI	XXXXXXXXXX	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	XXXXXXXXXX	18
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
1	HARSH	DELHI	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
2	PRATIK	BIHAR	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
3	RIYANKA	SILIGURI	XXXXXXXXXX	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))* | SQLQuery2.sql - DE

```sql
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:

Results | Messages

	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.

SQL | 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:

NAMEGrade
SUJIT O

SQL | 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	XXXXXXXXXX	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	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	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	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	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	XXXXXXXXX18	

- 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	XXXXXXXXX20	
3	SUJIT	ROHTAK	XXXXXXXXX20	

- 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	XXXXXXXXX20	

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | Join (Inner, Left, Right and Full Joins)

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

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

Consider the two tables below:

Student

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

StudentCourse

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

The simplest Join is INNER JOIN.

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

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

3. FROM table1

4. INNER JOIN table2

5. ON table1.matching_column = table2.matching_column;

6.

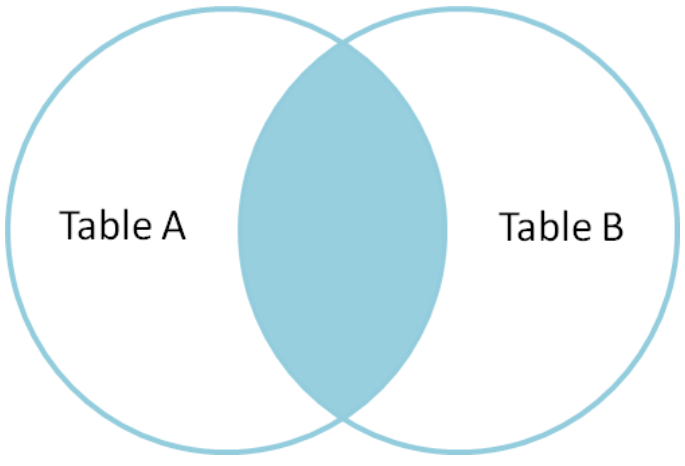
7.

8. table1: First table.

9. table2: Second table

10.   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:

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

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

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

13.   FROM table1

14.   LEFT JOIN table2

15.   ON table1.matching_column = table2.matching_column;

16.

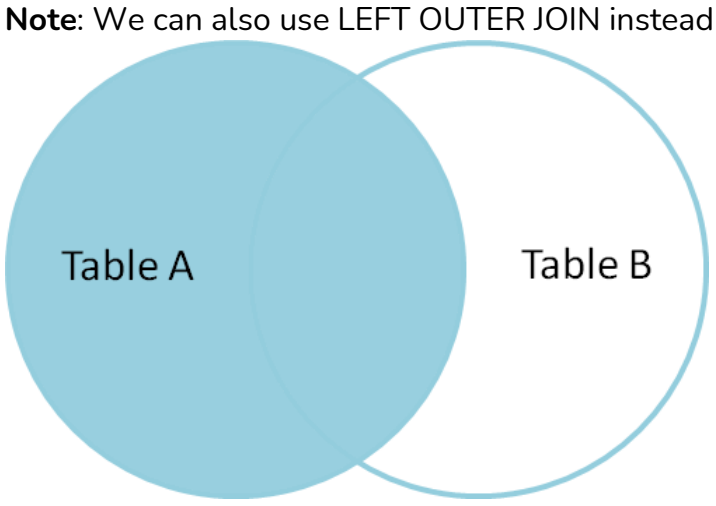
17.

18.   table1: First table.

19.   table2: Second table

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

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



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student
```

```
LEFT JOIN StudentCourse

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

Output:

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

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

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

23.   FROM table1

24.   RIGHT JOIN table2

25.   ON table1.matching_column = table2.matching_column;

26.

27.

28.   table1: First table.

29.   table2: Second table

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

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

Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student
```

```
RIGHT JOIN StudentCourse

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

Output:

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

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

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

33.   FROM table1

34.   FULL JOIN table2

35.   ON table1.matching_column = table2.matching_column;

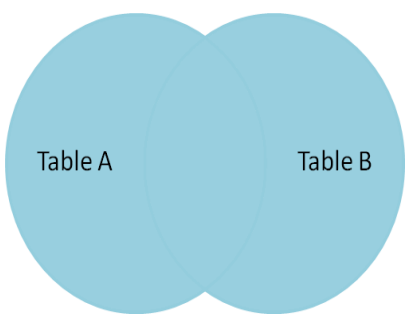
36.

37.

38.   table1: First table.

39.   table2: Second table

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



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

FULL JOIN StudentCourse
```

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

Output:

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

https://youtu.be/zGYLUogRe-M
Left JOIN (Video) Right JOIN (Video) Full JOIN (Video) SQL | JOIN (Cartesian Join, Self Join)
This article is contributed by **Harsh Agarwal** . If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | CREATE

There are two CREATE statements available in SQL:

1. CREATE DATABASE
2. 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.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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

(

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:

Orders

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

Customers

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

As we can see clearly that the field C_ID in 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)

)
```

(i) CHECK -
Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.

For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18. Check constraint in detail

```
CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int NOT NULL CHECK (AGE >= 18)

);
```

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

```
CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);
```

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | Comments
As is any programming languages comments matter a lot in SQL also. In this set we will learn about writing comments in any SQL snippet.

Comments can be written in the following three formats:

1. Single line comments.
2. Multi line comments
3. In line comments

- **Single line comments:** Comments starting and ending in a single line are considered as single line comments. Line starting with '--' is a comment and will not be executed.
  Syntax:

```
-- single line comment

-- another comment

SELECT * FROM Customers;
```

- **Multi line comments:** Comments starting in one line and ending in different line are considered as multi line comments. Line starting with '/*' is considered as starting point of comment and are terminated when '*/' is encountered.
  Syntax:

```
/* multi line comment

another comment */

SELECT * FROM Customers;
```

- **In line comments:** In line comments are an extension of multi line comments, comments can be stated in between the statements and are enclosed in between '/*' and '*/'.
  Syntax:

```
SELECT * FROM /* Customers; */
```

More examples:

```
Multi line comment ->

/* SELECT * FROM Students;

SELECT * FROM STUDENT_DETAILS;

SELECT * FROM Orders; */

SELECT * FROM Articles;



In line comment ->

SELECT * FROM Students;

SELECT * FROM /* STUDENT_DETAILS;

SELECT * FROM Orders;

SELECT * FROM */ Articles;
```

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

SQL | 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.

Syntax:

```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

ORDER BY column1, column2;



function_name: Name of the function used for example, SUM() , AVG().

table_name: Name of the table.

condition: Condition used.
```

Sample Table:

Employee

SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Student

SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

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:

```
SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME;
```

The above query will produce the below output:

NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

As you can see in the above output, 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. Consider the below query:

```
SELECT SUBJECT, YEAR, Count(*)

FROM Student

GROUP BY SUBJECT, YEAR;
```

Output:

SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

As you can see in the above output the students with both same SUBJECT and YEAR are placed in same group. And those whose only SUBJECT is same but not YEAR belongs to different groups. So here we have grouped the table according to two columns or more than one column.

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 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.

Example:

```
SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME

HAVING SUM(SALARY)>3000;
```

Output:

NAME	SUM(SALARY)
HARSH	5500

As you can see in the above output only one group out of the three groups appears in the result-set as it is the only group where sum of SALARY is greater than 3000. So we have used HAVING clause here to place this condition as the condition is required to be placed on groups not columns.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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.

In this article we will learn about creating , deleting and updating Views.

Sample Tables:

StudentDetails

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

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	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

Examples:

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

```
CREATE VIEW DetailsView AS

SELECT NAME, ADDRESS

FROM StudentDetails

WHERE S_ID < 5;
```

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

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

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

```
CREATE VIEW StudentNames AS

SELECT S_ID, NAME

FROM StudentDetails

ORDER BY NAME;
```

If we now query the view as,

```
SELECT * FROM StudentNames;
```

Output:

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. Query:

```
CREATE VIEW MarksView AS

SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS

FROM StudentDetails, StudentMarks
```

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

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Output:

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

DELETING VIEWS

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. 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.
```

For example, if we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

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.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. 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.

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
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
```

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

If we fetch all the data from MarksView now as:

```
SELECT * FROM MarksView;
```

Output:

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.**Syntax**:

```
• INSERT INTO view_name(column1, column2 , column3,..)
• VALUES(value1, value2, value3..);
•
• view_name: Name of the View
```

Example:
In the below example we will insert a new row in the View DetailsView which we have created above in the example of "creating views from a single table".

```
INSERT INTO DetailsView(NAME, ADDRESS)

VALUES("Suresh","Gurgaon");
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

- **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.**Syntax**:

```
• DELETE FROM view_name
• WHERE condition;
```

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

Example:
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

```
DELETE FROM DetailsView

WHERE NAME="Suresh";
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

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.
- 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;
```

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.
For example,though the View is updatable but then also the below query for this View is not valid:

```
INSERT INTO SampleView(S_ID)

VALUES(6);
```

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 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.

SQL | Functions (Aggregate and Scalar Functions)

For doing operations on data sql has many built-in functions, they are categorised in two categories and further sub-categorised in different seven functions under each category. The categories are:

1. **Aggregate functions:** These functions are used to do operations from the values of the column and a single value is returned.
    1. AVG()
    2. COUNT()
    3. FIRST()
    4. LAST()
    5. MAX()
    6. MIN()
    7. SUM()
2. **Scalar functions:** These functions are based on user input, these too returns single value.
    1. UCASE()
    2. LCASE()
    3. MID()
    4. LEN()
    5. ROUND()
    6. NOW()
    7. FORMAT()

Students-Table

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

Aggregate Functions

- **AVG()**: It returns average value after calculating from values in a numeric column.
  Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Queries:

1. Computing average marks of students.

```
2. SELECT AVG(MARKS) AS AvgMarks FROM Students;
```

Output:

AvgMarks
80

3. Computing average age of students.

```
4. SELECT AVG(AGE) AS AvgAge FROM Students;
```

Output:

AvgAge
19.4

- **COUNT():** It is used to count the number of rows returned in a SELECT statement. It can't be used in MS ACCESS.
  Syntax:

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

Queries:

1. Computing total number of students.

```
2. SELECT COUNT(*) AS NumStudents FROM Stuents;
```

Output:

NumStudents
5

3. Computing number of students with unique/distinct age.

```
4. SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;
```

Output:

NumStudents
4

- **FIRST():** The FIRST() function returns the first value of the selected column.
  Syntax:

```
SELECT FIRST(column_name) FROM table_name;
```

Queries:

1. Fetching marks of first student from the Students table.

```
2. SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

Output:

MarksFirst
90

3. Fetching age of first student from the Students table.

```
4. SELECT FIRST(AGE) AS AgeFirst FROM Students;
```

Output:

AgeFirst
19

- **LAST():** The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.
Syntax:

- `SELECT LAST(column_name) FROM table_name;`

Queries:

1. Fetching marks of last student from the Students table.

```
2. SELECT LAST(MARKS) AS MarksLast FROM Students;
```

Output:

MarksLast
82

3. Fetching age of last student from the Students table.

```
4. SELECT LAST(AGE) AS AgeLast FROM Students;
```

Output:

AgeLast
18

- **MAX():** The MAX() function returns the maximum value of the selected column.
Syntax:

- `SELECT MAX(column_name) FROM table_name;`

Queries:

1. Fetching maximum marks among students from the Students table.

```
2. SELECT MAX(MARKS) AS MaxMarks FROM Students;
```

Output:

MaxMarks
95

3. Fetching max age among students from the Students table.

```
4. SELECT MAX(AGE) AS MaxAge FROM Students;
```

Output:

MaxAge
21

- **MIN():** The MIN() function returns the minimum value of the selected column.
Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Queries:

1. Fetching minimum marks among students from the Students table.

```
2. SELECT MIN(MARKS) AS MinMarks FROM Students;
```

Output:

MinMarks
50

3. Fetching minimum age among students from the Students table.

```
4. SELECT MIN(AGE) AS MinAge FROM Students;
```

Output:

MinAge
18

- **SUM():** The SUM() function returns the sum of all the values of the selected column.
  Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Queries:

1. Fetching summation of total marks among students from the Students table.

```
2. SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

Output:

TotalMarks
400

3. Fetching summation of total age among students from the Students table.

```
4. SELECT SUM(AGE) AS TotalAge FROM Students;
```

Output:

TotalAge
97

Scalar Functions

- **UCASE()**: It converts the value of a field to uppercase.
  Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Queries:

1. Converting names of students from the table Students to uppercase.

```
2. SELECT UCASE(NAME) FROM Students;
```

Output:

NAME
HARSH

SURESH
PRATIK
DHANRAJ
RAM

- **LCASE()**: It converts the value of a field to lowercase.
  Syntax:

```
SELECT LCASE(column_name) FROM table_name;
```

Queries:

1. Converting names of students from the table Students to lowercase.

```
2. SELECT LCASE(NAME) FROM Students;
```

Output:

NAME
harsh
suresh
pratik
dhanraj
ram

- **MID():** The MID() function extracts texts from the text field.
  Syntax:

```
SELECT MID(column_name,start,length) AS some_name FROM table_name;

specifying length is optional here, and start signifies start position ( starting from 1 )
```

Queries:

1. Fetching first four characters of names of students from the Students table.

```
2. SELECT MID(NAME,1,4) FROM Students;
```

Output:

NAME
HARS
SURE
PRAT
DHAN
RAM

- **LEN():** The LEN() function returns the length of the value in a text field.
  Syntax:

```
SELECT LENGTH(column_name) FROM table_name;
```

Queries:

1. Fetching length of names of students from Students table.

```
2. SELECT LENGTH(NAME) FROM Students;
```

Output:

NAME
5
6
6
7
3

- **ROUND():** The ROUND() function is used to round a numeric field to the number of decimals specified.NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

  Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;

decimals- number of decimals to be fetched.
```

  Queries:

  1. Fetching maximum marks among students from the Students table.

```
2. SELECT ROUND(MARKS,0) FROM table_name;
```

  Output:

MARKS
90
50
80
95
85

- **NOW():** The NOW() function returns the current system date and time.
  Syntax:

```
SELECT NOW() FROM table_name;
```

  Queries:

  1. Fetching current system time.

```
2. SELECT NAME, NOW() AS DateTime FROM Students;
```

  Output:

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM
PRATIK	1/13/2017 1:30:11 PM
DHANRAJ	1/13/2017 1:30:11 PM
RAM	1/13/2017 1:30:11 PM

- **FORMAT():** The FORMAT() function is used to format how a field is to be displayed.
  Syntax:

```
SELECT FORMAT(column_name,format) FROM table_name;
```

  Queries:

  1. Formatting current date as 'YYYY-MM-DD'.

```
2. SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;
```

  Output:

NAME	Date
HARSH	2017-01-13
SURESH	2017-01-13
PRATIK	2017-01-13
DHANRAJ	2017-01-13
RAM	2017-01-13

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article

appearing on the GeeksforGeeks main page and help other Geeks.

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

1. What is SQL?
SQL stands for Structured Query Language. It is a language used to interact with the database, i.e to create a database, to create a table in the database, to retrieve data or update a table in the database, etc. SQL is an ANSI(American National Standards Institute) standard. Using SQL, we can do many things. For example - we can execute queries, we can insert records into a table, we can update records, we can create a database, we can create a table, we can delete a table, etc.

2. What is a database?
A Database is defined as a structured form of data that is stored database a computer or data in an organized manner and can be accessed in various ways. It is also the collection of schemas, tables, queries, views, etc. Databases help us with easily storing, accessing, and manipulating data held on a computer. The Database Management System allows a user to interact with the database.

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

4. What are the differences between SQL and PL/SQL?
Ans: Some common differences between SQL and PL/SQL are as shown below:

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

5. What is the difference between BETWEEN and IN operators in SQL?
BETWEEN
The **BETWEEN** operator is used to fetch rows based on a range of values.
For example,

```
SELECT * FROM Students

WHERE ROLL_NO BETWEEN 20 AND 30;
```

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

The **IN** operator is used to check for values contained in specific sets.
For example,

```
SELECT * FROM Students

WHERE ROLL_NO IN (20,21,23);
```

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

6. Write an SQL query to find names of employees start with 'A'?
The LIKE operator of SQL is used for this purpose. It is used to fetch filtered data by searching for a particular pattern in the where clause.
The Syntax for using LIKE is,

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



LIKE: operator name

pattern: exact value extracted from the pattern to get related data in

result set.
```

The required query is:

```
SELECT * FROM Employees WHERE EmpName like 'A%' ;
```

You may refer to this article WHERE clause for more details on the LIKE operator.

7. What is the difference between CHAR and VARCHAR2 datatype in SQL?
Both of these data types are used for characters, but varchar2 is used for character strings of variable length, whereas char is used for character strings of fixed length. For example, if we specify the type as char(5) then we will not be allowed to store a string of any other length in this variable, but if we specify the type of this variable as varchar2(5) then we will be allowed to store strings of variable length. We can store a string of length 3 or 4 or 2 in this variable.

8. Name different types of case manipulation functions available in SQL.
There are three types of case manipulation functions available in SQL. They are,

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

```
LOWER('string')
```

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

```
UPPER('string')
```

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

```
INITCAP('string')
```

9. What do you mean by data definition language?
Data definition language or DDL allows to execution of queries like CREATE, DROP and ALTER. That is those queries that define the data.

10. What do you mean by data manipulation language?
Data manipulation Language or DML is used to access or manipulate data in the database.
It allows us to perform the below-listed functions:

- Insert data or rows in a database
- Delete data from the database
- Retrieve or fetch data
- Update data in a database.

11. What is the difference between primary key and unique constraints?
The primary key cannot have NULL values, the unique constraints can have NULL values. There is only one primary key in a table, but there can be multiple unique constraints. The primary key creates the clustered index automatically but the unique key does not.

12. What is the view in SQL?

Views in SQL are a kind of virtual table. A view also has rows and columns as they are on 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 conditions.

The CREATE VIEW statement of SQL is used for creating views.

Basic 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

For more details on how to create and use view, please refer to this article.

13. What do you mean by foreign key?

A Foreign key is a field that can uniquely identify each row in another table. And this constraint is used to specify a field as a Foreign key. That is this field points to the primary key of another table. This usually creates a kind of link between the two tables.

Consider the two tables as shown below:

Orders

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

Customers

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

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

Syntax:

```
CREATE TABLE Orders

(

O_ID int NOT NULL,

ORDER_NO int NOT NULL,

C_ID int,
```

```
PRIMARY KEY (O_ID),

FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)

)
```

14. What is a join in SQL? What are the types of joins?
An SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- **INNER JOIN**: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e the value of the common field will be the same.
- **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will be null. LEFT JOIN is also known as LEFT OUTER JOIN
- **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.
- **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.

15. What is an index?
A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and the use of more storage space to maintain the extra copy of data. Data can be stored only in one order on a disk. To support faster access according to different values, a faster search like binary search for different values is desired. For this purpose, indexes are created on tables. These indexes need extra space on the disk, but they allow faster search according to different frequently searched values.

16. What is table and Field?

Table: A table has a combination of rows and columns. Rows are called records and columns are called fields. In MS SQL Server, the tables are being designated within the database and schema names.

Field: In DBMS, a database field can be defined as - a single piece of information from a record.

17. What is a primary key?

A Primary Key is one of the candidate keys. One of the candidate keys is selected as most important and becomes the primary key. There cannot be more than one primary key in a table.

18. What is a Default constraint?

The **DEFAULT** constraint is used to fill a column with default and fixed values. The value will be added to all new records when no other value is provided. For more details please refer to the SQL | Default Constraint article.

19. What is On Delete cascade constraint?

An 'ON DELETE CASCADE' constraint is used in MySQL to delete the rows from the child table automatically when the rows from the parent table are deleted. For more details, please read MySQL - On Delete Cascade constraint article.

20. What is normalization?

It is a process of analyzing the given relation schemas based on their functional dependencies and primary keys to achieve the following desirable properties:
1) Minimizing Redundancy
2) Minimizing the Insertion, Deletion, And Update Anomalies
Relation schemas that do not meet the properties are decomposed into smaller relation schemas that could meet desirable properties.

21. What is Denormalization?

Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization.

In a traditional normalized database, we store data in separate logical tables and attempt to minimize redundant data. We may strive to have only one copy of each piece of data in the database.

22. Explain WITH clause in SQL?

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

23. What are all the different attributes of indexes?

The indexing has various attributes:

- **Access Types**: This refers to the type of access such as value-based search, range access, etc.
- **Access Time**: It refers to the time needed to find a particular data element or set of elements.
- **Insertion Time**: It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time**: Time is taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

24. What is a Cursor?

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

25. Write down various types of relationships **in SQL?**

There are various relationships, namely:

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

26. What is a query?

An **SQL** query is used to retrieve the required data from the database. However, there may be multiple SQL queries that yield the same results but with different levels of efficiency. An inefficient query can drain the database resources, reduce the database speed or result in a loss of service for other users. So it is very important to optimize the query to obtain the best database performance.

27. What is subquery?

In SQL a Subquery can be simply defined as a query within another query. In other words, we can say that a Subquery is a query that is embedded in the WHERE clause of another SQL query.

28. What are the different operators available in SQL?

There are three operators available in SQL namely:

1. Arithmetic Operators
2. Logical Operators
3. Comparison Operators

29. What is a trigger?

Trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.

30. What is the difference between DELETE and TRUNCATE commands?

The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.	TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
DELETE command is slower than TRUNCATE command.	While the TRUNCATE command is faster than the DELETE command.
To use Delete you need DELETE permission on the table.	To use Truncate on a table we need at least ALTER permission on the table.
Identity of column retains the identity after using DELETE Statement on table.	Identity of the column is reset to its seed value if the table contains an identity column.
The delete can be used with indexed views.	Truncate cannot be used with indexed views.

31. What are local and global variables and their differences?

- **Global Variable:**

In contrast, global variables are variables that are defined outside of functions. These variables have global scope, so they can be used by any function without passing them to the function as parameters.

- **Local Variable:**

Local variables are variables that are defined within functions. They have local scope, which means that they can only be used within the functions that define them.

32. What is a constraint?

Constraints are the rules that we can apply to the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints. For more details please refer to SQL|Constraints article.

33. What is Data Integrity?

Data integrity is defined as the data contained in the database is both correct and consistent. For this purpose, the data stored in the database must satisfy certain types of procedures (rules). The data in a database must be correct and consistent. So, data stored in the database must satisfy certain types of procedures (rules). DBMS provides different ways to implement such types of constraints (rules). This improves data integrity in a database. For more details please refer difference between data security and data integrity article.

34. What is Auto Increment?

Sometimes, while creating a table, we do not have a unique identifier within the table, hence we face difficulty in choosing Primary Key. So as to resolve such an issue, we've to manually provide unique keys to every record, but this is often also a tedious task. So we can use the the the Auto-Increment feature that automatically generates a numerical Primary key value for every new record inserted. The Auto Increment feature is supported by all the Databases. For more details please refer SQL Auto Increment article.

35. What is the difference between Cluster and Non-Cluster Index?

CLUSTERED INDEX	NON-CLUSTERED INDEX
The clustered index is faster.	The non-clustered index is slower.
The clustered index requires less memory for operations.	The non-Clustered index requires more memory for operations.
In a clustered index, the index is the main data.	In the Non-Clustered index, the index is a copy of data.
A table can have only one clustered index.	A table can have multiple non-clustered index.
The clustered index has an inherent ability to store data on the disk.	The non-Clustered index does not have the inherent ability to store data on the disk.
Clustered indexes store pointers to block not data.	The non-Clustered index store both value and a pointer to actual row that holds data.

CLUSTERED INDEX	NON-CLUSTERED INDEX
In Clustered index leaf nodes are actual data itself.	In Non-Clustered index, leaf nodes are not the actual data itself rather they only contain included columns.
In the Clustered index, the Clustered key defines the order of data within the table.	In the Non-Clustered index, the index key defines the order of data within the index.
A Clustered index is a type of index in which table records are physically reordered to match the index.	A Non-Clustered index is a special type of index in which the logical order of index does not match the physical stored order of the rows on the disk.

For more details please refer Difference between Clustered index and No-Clustered index article.

36. What is MySQL collation?

A MySQL collation is a well-defined set of rules which are used to compare characters of a particular character set by using their corresponding encoding. Each character set in MySQL might have more than one collation, and has, at least, one default collation. Two character sets cannot have the same collation. For more details please refer What is collation and character set in MySQL? article.

37. What are user-defined functions?

We can use User-defined functions in PL/SQL or Java to provide functionality that is not available in SQL or SQL built-in functions. SQL functions and User-defined functions can appear anywhere, that is, wherever an expression occurs.

For example, it can be used in:

- Select a list of SELECT statements.
- Condition of WHERE clause.
- CONNECT BY, ORDER BY, START WITH, and GROUP BY
- The VALUES clause of the INSERT statement.
- The SET clause of the UPDATE statement.

38. What are all types of user-defined functions?

User-Defined Functions allow people to define their own T-SQL functions that can accept 0 or more parameters and return a single scalar data value or a table data type.
Different Kinds of User-Defined Functions created are:

1. Scalar User-Defined Function A Scalar user-defined function returns one of the scalar data types. Text, ntext, image and timestamp data types are not supported. These are the type of user-defined functions that most developers are used to in other programming languages. You pass in 0 to many parameters and you get a return value.

2. Inline Table-Value User-Defined Function An Inline Table-Value user-defined function returns a table data type and is an exceptional alternative to a view as the user-defined function can pass parameters into a T-SQL select command and, in essence, provide us with a parameterized, non-updateable view of the underlying tables.

3. Multi-statement Table-Value User-Defined Function A Multi-Statement Table-Value user-defined function returns a table and is also an exceptional alternative to a view, as the function can support multiple T-SQL statements to build the final result where the view is limited to a single SELECT statement. Also, the ability to pass parameters into a TSQL select command or a group of them gives us the capability to, in essence, create a parameterized, non-updateable view of the data in the underlying tables. Within the create function command you must define the table structure that is being returned. After creating this type of user-defined function, it can be used in the FROM clause of a T-SQL command, unlike the behavior found when using a stored procedure which can also return record sets.

39. Name the function which is used to remove spaces at the end of a string?

In SQL the spaces at the end of the string are removed by a trim function.

```
Syntax:
       Trim(s)

       Where s is a any string.
```

40. What is a stored procedure?

Stored Procedures are created to perform one or more DML operations on databases. It is nothing but a group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not return a value. For more details please refer what is Stored procedures in SQL article.

41. What are Union, minus and Interact commands?

Set Operations in SQL eliminate duplicate tuples and can be applied only to the relations which are union compatible. Set Operations available in SQL are :

- Set Union
- Set Intersection
- Set Difference

UNION Operation: This operation includes all the tuples which are present in either of the relations. For example: To find all the customers who have a loan or an account or both in a bank.

```
SELECT CustomerName FROM Depositor

UNION

SELECT CustomerName FROM Borrower ;
```

The union operation automatically eliminates duplicates. If all the duplicates are supposed to be retained, UNION ALL is used in the place of UNION.

INTERSECT Operation: This operation includes the tuples which are present in both of the relations. For example: To find the customers who have a loan as well as an account in the bank:

```
SELECT CustomerName FROM Depositor

INTERSECT

SELECT CustomerName FROM Borrower ;
```

The Intersect operation automatically eliminates duplicates. If all the duplicates are supposed to be retained, INTERSECT ALL is used in the place of INTERSECT.

EXCEPT Operation: This operation includes tuples that are present in one relation but should not be present in another relationship. For example: To find the customers who have an account but no loan at the bank:

```
SELECT CustomerName FROM Depositor

EXCEPT

SELECT CustomerName FROM Borrower ;
```

The Except operation automatically eliminates the duplicates. If all the duplicates are supposed to be retained, EXCEPT ALL is used at the place of EXCEPT.

42. What is an ALIAS command?

Aliases are the temporary names given to a table or column for the purpose of a particular SQL query. It is used when the name of a column or table is used other than their original name, 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 the 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 isthe more than one table involved in a query.

For more details, please read SQL| Aliases article.

43. What is the difference between TRUNCATE and DROP statements?

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

For more details, please read the Difference between DROP and TRUNCATE in the SQL article.

44. What are aggregate and scalar functions?

For doing operations on data SQL has many built-in functions, they are categorized into two categories and further sub-categorized into seven different functions under each category. The categories are:

Aggregate functions:
These functions are used to do operations from the values of the column and a single value is returned.

Scalar functions:
These functions are based on user input, these too return a single value.

For more details, please read the SQL | Functions (Aggregate and Scalar Functions) article.

45. Which operator is used in queries for pattern matching?

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

```
Syntax:


SELECT column1,column2 FROM table_name WHERE column_name LIKE pattern;


LIKE: operator name
```

46. Define SQL Order by the statement?

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.

For more details please read SQL | ORDER BY article.

47. Explain SQL Having statement?

HAVING is used to specify a condition for a group or an aggregate function used in the select statement. The WHERE clause selects before grouping. The HAVING clause selects rows after grouping. Unlike the HAVING clause, the WHERE clause cannot contain aggregate functions. (See this for examples). See Having vs Where Clause?

48. Explain SQL AND OR statement with example?

In SQL, the AND & OR operators are used for filtering the data and getting precise results based on conditions.

The AND and OR operators are used with the WHERE clause.

These two operators are called conjunctive operators.

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

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.**

For more details please read the SQL | AND and OR operators article.

49. Define BETWEEN statements in SQL?

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.

For more details please read SQL | Between & I operator article.

50. Why do we use Commit and Rollback command?

COMMIT	ROLLBACK
COMMIT permanently saves the changes made by the current transaction.	ROLLBACK undo the changes made by the current transaction.
The transaction can not undo changes after COMMIT execution.	Transaction reaches its previous state after ROLLBACK.
When the transaction is successful, COMMIT is applied.	When the transaction is aborted, ROLLBACK occurs.

For more details please read the Difference between Commit and Rollback in SQL article.

51. What are ACID properties?

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

52. What is a T-SQL?

T-SQL is an abbreviation for Transact Structure Query Language. It is a product by Microsoft and is an extension of SQL Language which is used to interact with relational databases. It is considered to perform best with Microsoft SQL servers. T-SQL statements are used to perform the transactions to the databases. T-SQL has huge importance since all the communications with an instance of an SQL server are done by sending Transact-SQL statements to the server. Users can also define functions using T-SQL.

Types of T-SQL functions are :

- **Aggregate** functions.
- **Ranking** functions. There are different types of ranking functions.
- **Rowset** function.
- **Scalar** functions.

53. Are NULL values the same as zero or a blank space?

In SQL, zero or blank space can be compared with another zero or blank space. whereas one null may not be equal to another null. null means data might not be provided or there is no data.

54. What is the need for group functions in SQL?

In database management, group functions, also known as 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 Group Functions

```
1) Count()

2) Sum()

3) Avg()

4) Min()

5) Max()
```

For more details please read Aggregate functions in SQL article.

55. What is the need for a **MERGE statement?**

The **MERGE** command in SQL is actually a combination of three SQL statements: **INSERT, UPDATE and DELETE**. In simple words, the MERGE statement in SQL provides a convenient way to perform all these three operations together which can be very helpful when it comes to handling large running databases. But unlike INSERT, UPDATE and DELETE statements MERGE statement requires a source table to perform these operations on the required table which is called a target table. For more details please read the SQL | MERGE Statement article.

56. How can you fetch common records from two tables?

The below statement could be used to get data from multiple tables, so, we need to use join to get data from multiple tables.

Syntax :

```
SELECT tablenmae1.columnname, tablename2.columnnmae

FROM tablenmae1

JOIN tablename2

ON tablenmae1.columnmnam = tablename2.columnnmae

ORDER BY columnname;
```

For more details and examples, please read SQL | SELECT data from Multiple tables article.

57. What are the advantages of PL/SQL functions?

Advantages of PL / SQL functions as follows:

- We can make a single call to the database to run a block of statements. Thus, it improves the performance against running SQL multiple times. This will reduce the number of calls between the database and the application.
- We can divide the overall work into small modules which becomes quite manageable, also enhancing the readability of the code.
- It promotes reusability.
- It is secure since the code stays inside the database, thus hiding internal database details from the application(user). The user only makes a call to the PL/SQL functions. Hence, security and data hiding is ensured.

58. What is the SQL query to display the current date?

CURRENT_DATE returns to the current date. This function returns the same value if it is executed more than once in a single statement, which means that the value is fixed, even if there is a long delay between fetching rows in a cursor.

Syntax:

```
CURRENT_DATE

     or

CURRENT DATE
```

59. What is ETL in SQL?

ETL is a process in Data Warehousing and it stands for **Extract**, **Transform** and **Load**. It is a process in which an ETL tool extracts the data from various data source systems, transforms it in the staging area, and then finally, loads it into the Data Warehouse system. These are three database functions that are incorporated into one tool to pull data out from one database and to put data into another database.

60. What are Nested Triggers?

A trigger can also contain INSERT, UPDATE, and DELETE logic within itself, so when the trigger is fired because of data modification it can also cause another data modification, thereby firing another trigger. A trigger that contains data modification logic within itself is called a nested trigger.

61. How to find the available constraint information in the table?

In SQL Server the **data dictionary** is a set of database tables used to store information about a database's definition.
One can use these data dictionaries to check the constraints on an already existing table and to change them(if possible).
For more details please read SQL | Checking Existing Constraint on a table article.

62. What is SQL injection?

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

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

For more details, please read the SQL | Injection article.

63. How to copy tables in SQL?

Sometimes, in SQL, we need to create an exact copy of an already defined (or created) table. MySQL enables you to perform this operation. Because we may need such duplicate tables for testing the data without having any impact on the original table and the data stored in it.

```
CREATE TABLE Contact List(Clone_1) LIKE Original_table;
```

For more details, Please read Cloning Table in the MySQL article.

64. Can we disable a trigger? If yes, how?

Yes, we can disable a trigger in PL/SQL. If consider temporarily disabling a trigger and one of the following conditions is true:

- An object that the trigger references is not available.
- We must perform a large data load and want it to proceed quickly without firing triggers.
- We are loading data into the table to which the trigger applies.
- We disable a trigger using the ALTER TRIGGER statement with the DISABLE option.
- We can disable all triggers associated with a table at the same time using the ALTER TABLE statement with the DISABLE ALL TRIGGERS option.

65. What is a Live Lock?

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.

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

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

- Remove duplicates using row numbers.
- Remove duplicates using self-Join.
- Remove duplicates using group by. For more details, please read SQL | Remove duplicates without distinct articles.

67. The difference between NVL and NVL2 functions?

These functions work with any data type and pertain to the use of null values in the expression list. These are all single-row functions i.e. provide one result per row.

NVL(expr1, expr2): In SQL, NVL() converts a null value to an actual value. Data types that can be used are date, character, and number. Data types must match with each other. i.e. expr1 and expr2 must be of the same data type.

```
Syntax:



NVL (expr1, expr2)
```

NVL2(expr1, expr2, expr3): The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned i.e. If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.

```
Syntax:



NVL2 (expr1, expr2, expr3)
```

For more details please read SQL general functions | NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL, and NANVL article.

68. What is Case WHEN in SQL?

Control statements form an important part of most languages since they control the execution of other sets of statements. These are found in SQL too and should be exploited for uses such as query filtering and query optimization through careful selection of tuples that match our requirements. In this post, we explore the Case-Switch statement in SQL. The CASE statement is SQL's way of handling if/then logic.

```
syntax: 1

CASE case_value

    WHEN when_value THEN statement_list

    [WHEN when_value THEN statement_list] ...

    [ELSE statement_list]

END CASE
syntax: 2

CASE

    WHEN search_condition THEN statement_list



    [WHEN search_condition THEN statement_list] ...

    [ELSE statement_list]

END CASE
```

For more details, please read the SQL | Case Statement article.

69. What is the difference between COALESCE() & ISNULL()?

COALESCE(): COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.
Syntax:

```
SELECT column(s), CAOLESCE(expression_1,....,expression_n)

FROM table_name;
```

ISNULL(): The ISNULL function has different uses in SQL Server and MySQL. In SQL Server, ISNULL() function is used to replace NULL values.
Syntax:

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

FROM table_name;
```

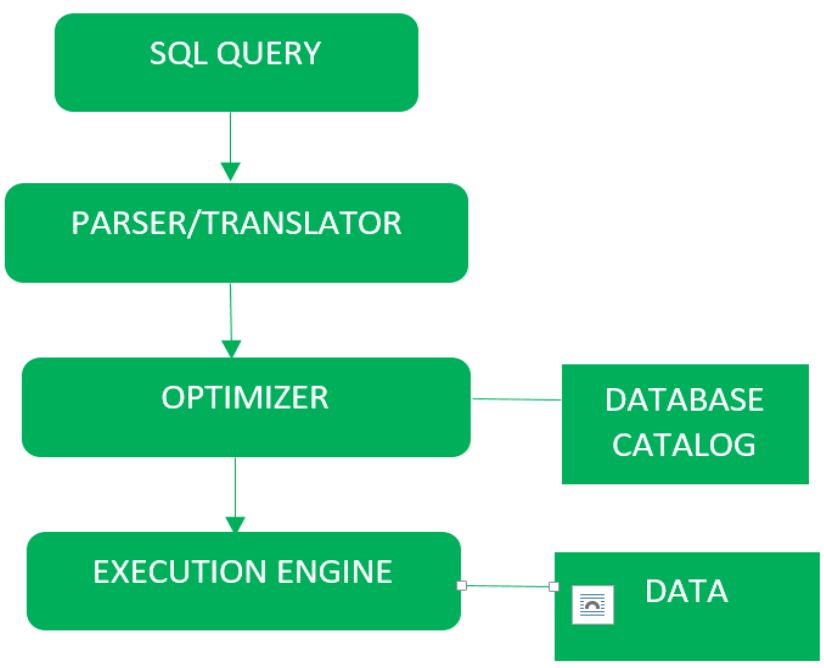
For more details, please read the SQL | Null functions article.

70. Name the operator which is used in the query for appending two strings?

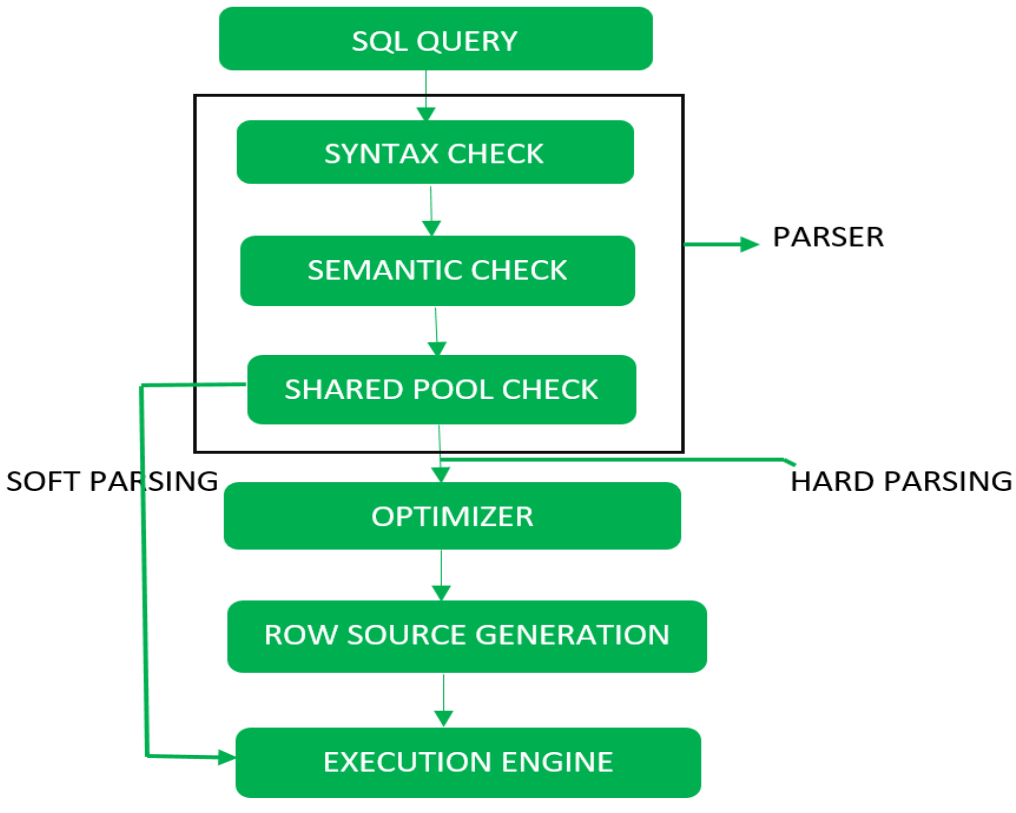
In SQL for appending two strings, " Concentration operator" is used and its symbol is " || ".

SQL | Query Processing
Query Processing includes translations on high level Queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result.

Block Diagram of Query Processing is as:



Detailed Diagram is drawn as:



It is done in the following steps:

- **Step-1:**
  Parser: During parse call, the database performs the following checks- Syntax check, Semantic check and Shared pool check, after converting the query into relational algebra.

  Parser performs the following checks as (refer detailed diagram):

  1. **Syntax check -** concludes SQL syntactic validity. Example:

     ```
     SELECT * FORM employee
     ```

     Here error of wrong spelling of FROM is given by this check.

  2. **Semantic check -** determines whether the statement is meaningful or not. Example: query contains a tablename which does not exist is checked by this check.
  3. **Shared Pool check -** Every query possess a hash code during its execution. So, this check determines existence of written hash code in shared pool if code exists in shared pool then database will not take additional steps for optimization and execution.

  Hard Parse and Soft Parse -
  If there is a fresh query and its hash code does not exist in shared pool then that query has to pass through from the additional steps known as hard parsing otherwise if hash code exists then query does not passes through additional steps. It just passes directly to execution engine (refer detailed diagram). This is known as soft parsing.
  Hard Parse includes following steps - Optimizer and Row source generation.

- **Step-2:**
  Optimizer: During optimization stage, database must perform a hard parse atleast for one unique DML statement and perform optimization during this parse. This database never optimizes DDL unless it includes a DML component such as subquery that require optimization.

  It is a process in which multiple query execution plan for satisfying a query are examined and most efficient query plan is satisfied for execution.
  Database catalog stores the execution plans and then optimizer passes the lowest cost plan for execution.

  Row Source Generation -
  The Row Source Generation is a software that receives a optimal execution plan from the optimizer and produces an iterative execution plan that is usable by the rest of the database. the iterative plan is the binary program that when executes by the sql engine produces the result set.

- **Step-3:**
  Execution Engine: Finally runs the query and display the required result.

SQL | WHERE Clause

WHERE keyword is used for fetching **filtered data** in a result set.

- It is used to fetch data according to a particular criteria.
- WHERE keyword can also be used to filter data by matching patterns.

Basic Syntax: SELECT column1,column2 FROM table_name WHERE column_name operator value;

```
column1 , column2: fields int the table

table_name: name of table

column_name: name of field used for filtering the data

operator: operation to be considered for filtering

value: exact value or pattern to get related data in result
```

List of operators that can be used with where clause:

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

Student

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

Queries

- To fetch record of students with age equal to 20

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

Output:

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

- To fetch Name and Address of students with ROLL_NO greater than 3

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

Output:

ROLL_NO	NAME	ADDRESS
4	SURESH	Delhi

BETWEEN operator

It is used to fetch filtered data in a given range inclusive of two values.
Basic Syntax: SELECT column1,column2 FROM table_name WHERE column_name BETWEEN value1 AND value2;

```
BETWEEN: operator name
```

value1 AND value2: exact value from value1 to value2 to get related data in result set.

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;
```

Output:

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

- 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;
```

Output:

NAME	ADDRESS
SUJIT	Rohtak
SUJIT	Rohtak

LIKE operator

It is used to fetch filtered data by searching for a particular pattern in where clause.
Basic Syntax: SELECT column1,column2 FROM table_name WHERE column_name LIKE pattern;

```
LIKE: operator name
```

pattern: exact value extracted from the pattern to get related data in result set.

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.

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20

- To fetch records of students where NAME contains the patter 'AM'.

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

Output:

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

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

```
IN: operator name
```

value1,value2,..: exact value matching the values given and get related data in result set.

Queries

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

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

Output:

NAME	ADDRESS
Ram	Delhi
RAMESH	GURGAON
SUJIT	ROHTAK
SURESH	Delhi
SUJIT	ROHTAK
RAMESH	GURGAON

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

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

Output:

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

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL AND and OR operators

In SQL, the 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.

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.

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
```

Now, we consider a table database to demonstrate AND & OR operators with multiple cases:

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

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:

Query:

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

Output:

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

Take another example, to fetch all the records from the Student table where NAME is Ram and Age is 18.

Query:

```
SELECT * FROM Student WHERE Age = 18 AND NAME = 'Ram';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

To fetch all the records from the Student table where NAME is Ram or NAME is SUJIT.

Query:

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

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

To fetch all the records from the Student table where NAME is Ram or Age is 20.

Query:

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

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

Combining AND and OR:

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);
```

Take an example to fetch all the records from the Student table where Age is 18 NAME is Ram or RAMESH.

Query:

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

Output:

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

SQL | 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:

1. The fields to be used in both the select statements must be in same order, same number and same data type.
2. 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;
```

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

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

Queries

- To fetch distinct ROLL_NO from Student and Student_Details table.

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

Output:

ROLL_NO
1
2
3
4

- 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;
```

Output:

ROLL_NO
1
2
3
4
3
2

- 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.
```

Output:

ROLL_NONAME
1 Information Technology
2 Computer Science
4 SURESH

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | Join (Cartesian Join & Self Join)
SQL| JOIN(Inner, Left, Right and Full Joins) In this article, we will discuss about the remaining two JOINS:

- **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

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

1. **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

Example Queries(CARTESIAN JOIN):

- 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;
```

Output:

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

2. **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:**

```
3. SELECT a.coulmn1 , b.column2

4. FROM table_name a, table_name b

5. WHERE some_condition;

6.

7. table_name: Name of the table.

8. some_condition: Condition for selecting the rows.
```

Example Queries(SELF JOIN):

```
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

https://youtu.be/-t6tjGtQ7-8

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | DROP, DELETE, TRUNCATE

DROP: DROP is used to delete a whole database or just a table.
TRUNCATE: Truncate statement is also used for same purpose.

DROP vs TRUNCATE

- Truncate is normally ultra-fast and its ideal for cleaning out data from a temporary table.
- It also preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.

Basic Syntax for deleting a table:

```
DROP TABLE table_name;



table_name: Name of the table to be deleted.
```

Basic Syntax for deleting a whole data base:

```
DROP DATABASE database_name;



database_name: Name of the database to be deleted.
```

Basic Syntax for truncating a table:

```
TRUNCATE TABLE  table_name;



table_name: Name of the table to be truncated.
```

NOTE: Table or Database deletion using DROP statement cannot be rolled back, so it must be used wisely. Suppose we have following tables in a database.
DATABASE name - student_data

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

Student_Details

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

Queries

- To delete Student table from student_data database.

```
DELETE TABLE Student;
```

After running the above query we will be left with only Student_details table.

- To delete the whole database

```
DROP DATABASE student_data;
```

After running the above query whole database will be deleted.

- To truncate Student_details table from student_data database.

```
TRUNCATE TABLE Student_details;
```

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

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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. It was officially introduced in the SQL:2008 standard.
The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).
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.

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

Queries

- To delete the whole database

```
DROP DATABASE student_data;
```

After running the above query whole database will be deleted.

- To truncate Student_details table from student_data database.

```
TRUNCATE TABLE Student_details;
```

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

References:

- https://en.wikipedia.org/wiki/Truncate_(SQL)
- https://en.wikipedia.org/wiki/Data_definition_language#DROP_statement

SQL | Date functions

In SQL, dates are complicated for newbies, since 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();
```

Output:

```
2017-01-13 08:03:52
```

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

```
SELECT CURDATE();
```

Output:

```
2017-01-13
```

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

```
SELECT CURTIME();
```

Output:

```
08:05:15
```

- **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

```
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

Queries

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

Output:

NameBirthDay	
Pratik	26

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

Output:

NameBirthYear	
Pratik	1996

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

Output:

NameBirthSecond	
Pratik	581

- **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.

Example:
For the below table named 'Test'

Id	NameBirthTime	
4120	Pratik	1996-09-26 16:44:15.581

Queries

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

Output:

NameBirthTimeModified	
Pratik	1997-09-26 16:44:15.581

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

Output:

NameBirthDayModified	
Pratik	1996-10-26 16:44:15.581

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

Output:

NameBirthSecond	
Pratik	1996-10-26 20:44:15.581

- **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:

```
DATEDIFF(date1, date2);

date1 & date2- date/time expression
```

Example:

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

Output:

DateDiff
10

- **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)
- %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

Example:

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

Result:

```
13 Jan 17
```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | EXISTS

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no

tuples) or not. The result of EXISTS is a boolean value True or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Syntax:

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

Examples:
Consider the following two relation "Customers" and "Orders".

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Queries

1. **Using EXISTS condition with SELECT statement** To fetch the first and last name of the customers who placed atleast one order.

```
2. SELECT fname, lname

3. FROM Customers

4. WHERE EXISTS (SELECT *
5.               FROM Orders
                 WHERE Customers.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

6. **Using NOT with EXISTS** Fetch last and first name of the customers who has not placed any order.

```
7. SELECT lname, fname
```

```
8. FROM Customer

9. WHERE NOT EXISTS (SELECT *

10.                     FROM Orders
```

```
                    WHERE Customers.customer_id = Orders.c_id);
```

Output:

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

11. **Using EXISTS condition with DELETE statement** Delete the record of all the customer from Order Table whose last name is 'Mehra'.

```
12.   DELETE

13.   FROM Orders

14.   WHERE EXISTS (SELECT *

15.                 FROM customers

16.                 WHERE Customers.customer_id = Orders.cid
```

```
               AND Customers.lname = 'Mehra');
SELECT * FROM Orders;
```

Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

17. **Using EXISTS condition with UPDATE statement** Update the lname as 'Kumari' of customer in Customer Table whose customer_id is 401.

```
18.   UPDATE Customers

19.   SET lname = 'Kumari'

20.   WHERE EXISTS (SELECT *

21.                 FROM Customers
```

```
               WHERE customer_id = 401);
SELECT * FROM Customers;
```

Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | WITH clause

The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database. The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
- Queries that have an associated WITH clause can also be written using nested sub-queries but doing so add more complexity to read/debug the SQL query.
- WITH clause is not supported by all database system.
- The name assigned to the sub-query is treated as though it was an inline view or table
- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database.

Syntax:

```
WITH temporaryTable (averageValue) as

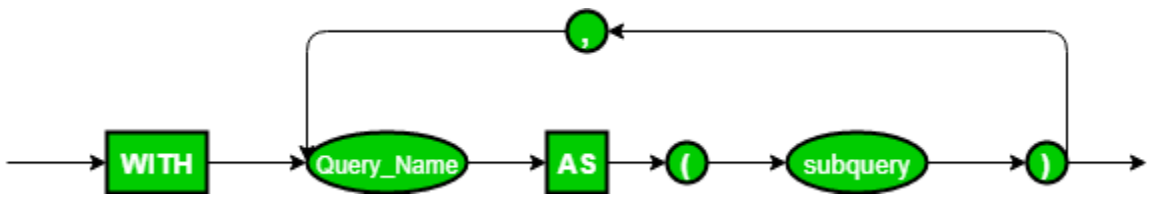
    (SELECT avg(Attr1)

    FROM Table)

    SELECT Attr1

    FROM Table, temporaryTable

    WHERE Table.Attr1 > temporaryTable.averageValue;
```



In this query, WITH clause is used to define a temporary relation temporaryTable that has only 1 attribute averageValue. averageValue holds the average value of column Attr1 described in relation Table. The SELECT statement that follows the WITH clause will produce only those tuples where the value of Attr1 in relation Table is greater than the average value obtained from the WITH clause statement.

Note: When a query with a WITH clause is executed, first the query mentioned within the clause is evaluated and the output of this evaluation is stored in a temporary relation. Following this, the main query associated with the WITH clause is finally executed that would use the temporary relation produced.

Queries

Example 1: Find all the employee whose salary is more than the average salary of all employees.
Name of the relation: **Employee**

EmployeeID	Name	Salary
100011	Smith	50000
100022	Bill	94000
100027	Sam	70550
100845	Walden	80000
115585	Erik	60000
1100070	Kate	69000

SQL Query:

```
WITH temporaryTable(averageValue) as

    (SELECT avg(Salary)

    from Employee)

        SELECT EmployeeID,Name, Salary

        FROM Employee, temporaryTable

        WHERE Employee.Salary > temporaryTable.averageValue;
```

Output:

EmployeeID	Name	Salary
100022	Bill	94000
100845	Walden	80000

Explanation: The average salary of all employees is 70591. Therefore, all employees whose salary is more than the obtained average lies in the output relation.

Example 2: Find all the airlines where the total salary of all pilots in that airline is more than the average of total salary of all pilots in the database.

Name of the relation: **Pilot**

EmployeeID	Airline	Name	Salary
70007	Airbus 380	Kim	60000
70002	Boeing	Laura	20000
10027	Airbus 380	Will	80050
10778	Airbus 380	Warren	80780
115585	Boeing	Smith	25000
114070	Airbus 380	Katy	78000

SQL Query:

```
WITH totalSalary(Airline, total) as

    (SELECT Airline, sum(Salary)

    FROM Pilot

    GROUP BY Airline),

    airlineAverage(avgSalary) as

    (SELECT avg(Salary)

    FROM Pilot )

    SELECT Airline

    FROM totalSalary, airlineAverage

    WHERE totalSalary.total > airlineAverage.avgSalary;
```

Output:

Airline
Airbus 380

Explanation: The total salary of all pilots of Airbus 380 = 298,830 and that of Boeing = 45000. Average salary of all pilots in the table Pilot = 57305. Since only the total salary of all pilots of Airbus 380 is greater than the average salary

obtained, so Airbus 380 lies in the output relation.

Important Points:

- The SQL WITH clause is good when used with complex SQL statements rather than simple ones
- It also allows you to break down complex SQL queries into smaller ones which make it easy for debugging and processing the complex queries.
- The SQL WITH clause is basically a drop-in replacement to the normal sub-query.

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL NULL Values

In SQL there may be some records in a table that do not have values or data for every field. This could be possible because at a time of data entry information is not available. So SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple. SQL places a NULL value in the field in the absence of a user-defined value. For example, the Apartment_number attribute of an address applies only to address that are in apartment buildings and not to other types of residences.

Importance of NULL value:

- It is important to understand that a NULL value is different from a zero value.
- A NULL value is used to represent a missing value, but that it usually has one of three different interpretations:
    - The value unknown (value exists but is not known)
    - Value not available (exists but is purposely withheld)
    - Attribute not applicable (undefined for this tuple)
- It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL.

Principles of NULL values:

- Setting a NULL value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is character.
- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, CHECK constraints will ignore by SQL.

In general, each NULL value is considered to be different from every other NULL in the database. When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN. Hence, SQL uses a three-valued logic with values **True, False**, and **Unknown.** It is, therefore, necessary to define the results of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

How to test for NULL Values?

SQL allows queries that check whether an attribute value is NULL. Rather than using = or to compare an attribute value to NULL, SQL uses **IS** and **IS NOT**. This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate.

Now, consider the following Employee Table,

Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

Suppose if we find the Fname, Lname of the Employee having no Super_ssn then the query will be:

Query

```
SELECT Fname, Lname FROM Employee WHERE Super_ssn IS NULL;
```

Output:

Fname	Lname
James	Borg

Now if we find the Count of the number of Employees having Super_ssn.

Query:

```
SELECT COUNT(*) AS Count FROM Employee WHERE Super_ssn IS NOT NULL;
```

Output:

Count
7

SQL | ALL and ANY

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

ALL

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 iff 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);
```

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,

Products Table

ProductID	ProductName	SupplierID	CotegoryID	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

OrderDetails Table

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

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

```
SELECT ALL ProductName

FROM Products

WHERE TRUE;
```

Output:

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

- **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);
```

Output:

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);
```

Output:

OrderID
10248
10250

ANY

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);
```

Output:

CategoryID
1
2
7
6

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

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                       FROM OrderDetails
                       WHERE Quantity = 9);
```

ProductName
Organic Dried Pears
Northwoods Cranberry Sauce

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

SQL | 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;
```

Examples:
Consider the following Employee Table,

Fname	Lname	SSN	Salary	DOB
John	Smith	123456789	30000	1988-05-02
Franklin	Wong	333445555	40000	1986-01-02
Joyce	English	453453453	80000	1977-12-08
Ramesh	Narayan	666884444	38000	1987-03-05
James	Borg	888665555	55000	1982-10-10
Jennifer	Wallace	987654321	43000	1985-08-07
Ahmad	Jabbar	987987987	25000	1990-06-28
Alicia	Zeala	999887777	25000	1980-09-14

Queries

- **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;
```

Output:

Fname	Lname
John	Smith
Franklin	Wong
Ramesh	Narayan
Jennifer	Wallace

- **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';
```

Output:

Fname	Lname
John	Smith
Franklin	Wong
Ramesh	Narayan
Jennifer	Wallace
Ahmad	Jabbar

- **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 Emplyoee

WHERE Salary

NOT BETWEEN 30000 AND 45000;
```

Output:

Fname	Lname
Joyce	English
James	Borg
Ahmad	Jabbar
Alicia	Zeala

IN

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

Queries

- 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);
```

Output:

Fname	Lname
John	Smith
Franklin	Wong
Ahmad	Jabbar
Alicia	Zeala

- 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);
```

Output:

Fname	Lname
Franklin	Wong
Joyce	English
Ramesh	Narayan
James	Borg
Jennifer	Wallace

SQL | Arithmetic Operators

Prerequisite: Basic Select statement, Insert into clause, Sql Create Clause, SQL Aliases
We can use various Arithmetic Operators on the data stored in the tables.

Arithmetic Operators are:

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

Addition (+) :

It is used to perform **addition 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 addition;
```

Output:

employee_id	employee_name	salary	salary+100
1	alex	25000	25100
2	rr	55000	55100
3	jpm	52000	52100
4	ggshmr	12312	12412

Here we have done addition of 100 to each Employee's salary i.e, addition operation on single column.

Let's perform **addition of 2 columns**:

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

Output:

employee_id	employee_name	salary	salary+employee_id
1	alex	25000	25001
2	rr	55000	55002
3	jpm	52000	52003
4	ggshmr	12312	12316

Here we have done addition of 2 columns with each other i.e, each employee's employee_id is added with its salary.

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;
```

Output:

employee_id	employee_name	salary	salary-100
12	Finch	15000	14900
22	Peter	25000	24900
32	Warner	5600	5500
42	Watson	90000	89900

Here we have done subtraction of 100 to each Employee's salary i.e, subtraction operation on single column.

Let's perform **subtraction of 2 columns**:

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

Output:

employee_id	employee_name	salary	salary - employee_id
12	Finch	15000	14988
22	Peter	25000	24978
32	Warner	5600	5568
42	Watson	90000	89958

Here we have done subtraction of 2 columns with each other i.e, each employee's employee_id is subtracted from its salary.

Division (/) : For **Division** refer this link- Division in SQL

Multiplication (*) :

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

Implementation:

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

Output:

employee_id	employee_name	salary	salary * 100
1	Finch	25000	2500000
2	Peter	55000	5500000
3	Warner	52000	5200000
4	Watson	1231	121200

Here we have done multiplication of 100 to each Employee's salary i.e, multiplication operation on single column.

Let's perform **multiplication of 2 columns**:

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

Output:

employee_id	employee_name	salary	salary * employee_id
1	Finch	25000	25000
2	Peter	55000	110000
3	Warner	52000	156000
4	Watson	1231	249248

Here we have done multiplication of 2 columns with each other i.e, each employee's employee_id is multiplied with its salary.

Modulus (%) :

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

Implementation:

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

Output:

employee_id	employee_name	salary	salary % 25000
1	Finch	25000	0
2	Peter	55000	5000
3	Warner	52000	2000
4	Watson	12312	12312

Here we have done modulus of 100 to each Employee's salary i.e, modulus operation on single column.

Let's perform **modulus operation between 2 columns**:

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

Output:

employee_id	employee_name	salary	salary % employee_id
1	Finch	25000	0
2	Peter	55000	0
3	Warner	52000	1
4	Watson	12312	0

Here we have done modulus of 2 columns with each other i.e, each employee's salary is divided with its id and corresponding remainder is shown.

Basically, **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;
```

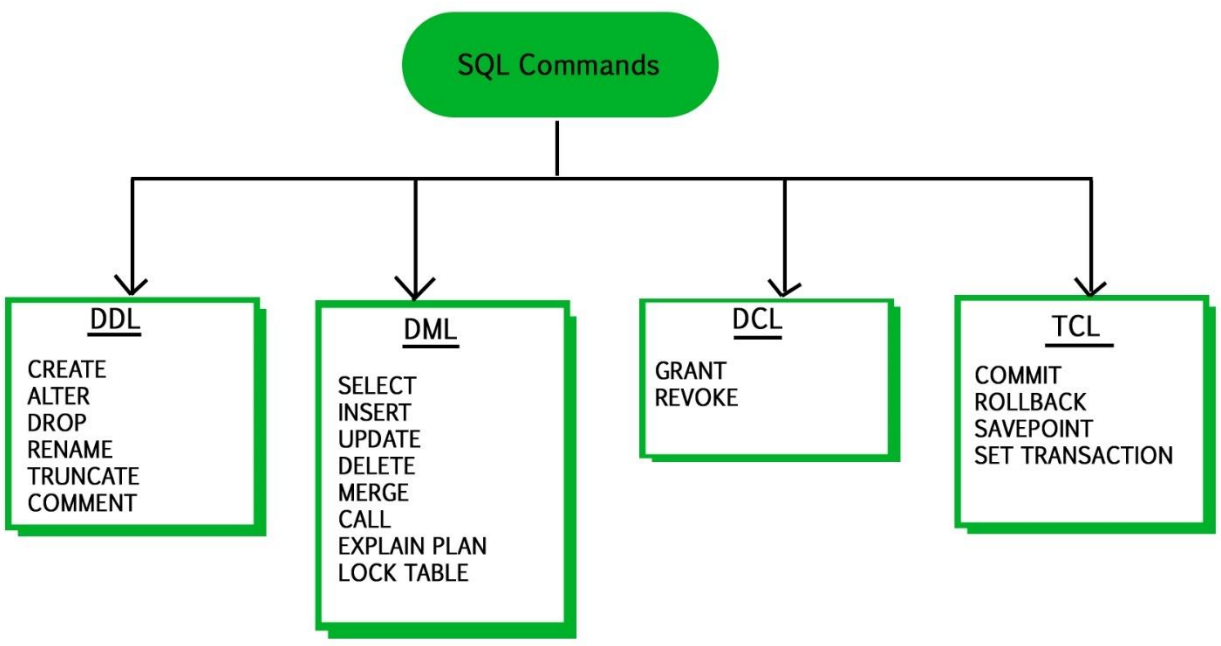
Output:

employee_id	employee_name	salary	type	type + 100
1	Finch	25000	NULL	NULL
2	Peter	55000	NULL	NULL
3	Warner	52000	NULL	NULL
4	Watson	12312	NULL	NULL

Here output always came null, since performing any operation on null will always result in a *null value*.

Note: Make sure that NULL is **unavailable**, **unassigned**, **unknown**. Null is **not** same as *blank space* or *zero*.
To get in depth understanding of NULL, refer THIS link.

References: Oracle Docs
SQL | DDL, DML, TCL and DCL
In this article, we'll be discussing Data Definition Language, Data Manipulation Language, Transaction Control Language, and Data Control Language.



DDL (Data Definition Language) :

Data Definition Language is used to define the database structure or schema. DDL is also used to specify additional properties of the data. The storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schema, which are usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints.
For example, suppose the university requires that the account balance of a department must never be negative. The DDL provides facilities to specify such constraints. The database system checks these constraints every time the database is updated. In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to the test. Thus, the database system implements integrity constraints that can be tested with minimal overhead.

1. **Domain Constraints :** A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as the constraints on the values that it can take.

2. **Referential Integrity :** There are cases where we wish to ensure that a value appears in one relation for a given set of attributes also appear in a certain set of attributes in another relation i.e. Referential Integrity. For example, the department listed for each course must be one that actually exists.

3. **Assertions :** An assertion is any condition that the database must always satisfy. Domain constraints and Integrity constraints are special form of assertions.

4. **Authorization :** We may want to differentiate among the users as far as the type of access they are permitted on various data values in database. These differentiation are expressed in terms of Authorization. The most common being :
   read authorization - which allows reading but not modification of data ;
   insert authorization - which allow insertion of new data but not modification of existing data
   update authorization - which allows modification, but not deletion.

Some Commands:

```
CREATE : to create objects in database
ALTER : alters the structure of database
DROP : delete objects from database
RENAME : rename an objects
```

Following SQL DDL-statement defines the department table :

```
create table department
(dept_name  char(20),
  building   char(15),
  budget     numeric(12,2));
```

Execution of the above DDL statement creates the department table with three columns - dept_name, building, and budget; each of which has a specific datatype associated with it.

DML (Data Manipulation Language) :

DML statements are used for managing data with in schema objects.
DML are of two types -

1. **Procedural DMLs** : require a user to specify what data are needed and how to get those data.
2. **Declerative DMLs** (also referred as **Non-procedural DMLs**) : require a user to specify what data are needed without specifying how to get those data.

   Declarative DMLs are usually easier to learn and use than procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data.

Some Commands :

```
SELECT: retrieve data from the database
INSERT: insert data into a table
UPDATE: update existing data within a table
DELETE: deletes all records from a table, space for the records remain
```

Example of SQL query that finds the names of all instructors in the History department :

```
select instructor.name
 from instructor
 where instructor.dept_name = 'History';
```

The query specifies that those rows from the table instructor where the dept_name is History must be retrieved and the name attributes of these rows must be displayed.

TCL (Transaction Control Language) :

Transaction Control Language commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

Examples of TCL commands -

```
COMMIT: Commit command is used to permanently save any transaction
            into the database.
ROLLBACK: This command restores the database to last committed state.
            It is also used with savepoint command to jump to a savepoint
            in a transaction.
SAVEPOINT: Savepoint command is used to temporarily save a transaction so
            that you can rollback to that point whenever necessary.
```

DCL (Data Control Language) :

A Data Control Language is a syntax similar to a computer programming language used to control access to data stored in a database (Authorization). In particular, it is a component of Structured Query Language (SQL).

Examples of DCL commands :

```
GRANT: allow specified users to perform specified tasks.
REVOKE: cancel previously granted or denied permissions.
```

The operations for which privileges may be granted to or revoked from a user or role apply to both the Data definition language (DDL) and the Data manipulation language (DML), and may include CONNECT, SELECT, INSERT, UPDATE, DELETE, EXECUTE and USAGE.

In the Oracle database, executing a DCL command issues an implicit commit. Hence, you cannot roll back the command.

References : kakeboksen.td.org.uit.no

SQL | 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.

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
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.

SQL | Difference between functions and stored procedures in PL/SQL

Prerequisite:

- Procedures in PL/SQL

- Functions in PL/SQL.

Differences between Stored procedures(SP) and Functions(User defined functions (UDF)):

1. **SP** *may or may not* return a value but **UDF must** return a value.
   Example:

```
SP ->
create or replace procedure GEEKS(x int)
is
y int;
begin
.....

UDF->
FUNCTION GEEKS(x int)
/*return statement so we must return value in function */
RETURN int
IS
  y int;
BEGIN
.....
```

2. **SP** can have *input/output* parameter but **UDF only** has input parameter.
   Example:

```
 SP ->
 CREATE OR REPLACE PROCEDURE Factorial(x IN NUMBER, result OUT NUMBER)
 is
 begin
 ....

UDF ->
FUNCTION Factorial(x IN NUMBER) /* only input parameter */
return  NUMBER
is
result NUMBER;
begin
.....
```

3. We can call **UDF** from **SP** but **cannot** call **SP** from a function.

   Example:

```
Calling UDF cal() inside SP square() but reverse is not possible.

set serveroutput on;
declare
a int;
c int;

function cal(temp  int)
return int
as
ans int;
begin
ans:=temp* temp;
return ans;
end;

procedure square(x in int, ans out int)
is
begin
dbms_output.put_line('calling function in procedure');
ans:= cal(x);
end;

begin
a:=6;
square(a, c);
dbms_output.put_line('the answer is '|| c);
end;
```

   OUTPUT:

```
calling function in procedure
the answer is 36
```

4. We **cannot** use **SP** in SQL statement like SELECT, INSERT, UPDATE, DELETE, MERGE etc. but we can use them with UDF.

5. We can use try-catch exception handling in SP but we **cannot** do that in **UDF**.

6. We can use transaction in **SP** but it is not possible in **UDF**.

SQL Trigger | Book Management Database

Prerequisite - SQL Trigger | Student Database For example, given Library Book Management database schema with Student database schema. In these databases, if any student borrows a book from library then the count of that specified book should be decremented. To do so,

Suppose the schema with some data,

```
mysql> select * from book_det;

+-----+-------------+--------+

| bid | btitle      | copies |

+-----+-------------+--------+

|   1 | Java        |     10 |

|   2 | C++         |      5 |

|   3 | MySql       |     10 |

|   4 | Oracle DBMS |      5 |

+-----+-------------+--------+

4 rows in set (0.00 sec)



mysql> select * from book_issue;

+------+------+--------+

| bid  | sid  | btitle |

+------+------+--------+

1 row in set (0.00 sec)
```

To implement such procedure, in which if the system inserts the data into the book_issue database a trigger should automatically invoke and decrements the copies attribute by 1 so that a proper track of book can be maintained.

Trigger for the system -

```
create trigger book_copies_deducts

after INSERT

on book_issue

for each row

update book_det set copies = copies - 1 where bid = new.bid;
```

Above trigger, will be activated whenever an insertion operation performed in a book_issue database, it will update the book_det schema setting copies decrements by 1 of current book id(bid).

Results -

```
mysql> insert into book_issue values(1, 100, "Java");

Query OK, 1 row affected (0.09 sec)



mysql> select * from book_det;

+-----+-------------+--------+

| bid | btitle      | copies |
```

```
+-----+-------------+--------+

|   1 | Java        |      9 |

|   2 | C++         |      5 |

|   3 | MySql       |     10 |

|   4 | Oracle DBMS |      5 |

+-----+-------------+--------+

4 rows in set (0.00 sec)



mysql> select * from book_issue;

+------+------+--------+

| bid  | sid  | btitle |

+------+------+--------+

|    1 |  100 | Java   |

+------+------+--------+

1 row in set (0.00 sec)
```

As above results show that as soon as data is inserted, copies of the book deducts from the book schema in the system.