As Per CBSE Syllabus 2022-23

Computer Science

Class XII

Chapter-7

Strucrured Query Language (SQL)

What is a Structured Query Language (SQL)?

SQL is a standard language for storing, retrieving and manipulating data on a relational database. All the relational database like MySql, Oracle, MS Access, SQL server uses Structured query language(SQL) for accessing and manipulating data.

SQL provides wide range of effective command to perform all sort of required operations on data such as create tables, insert record, view recodes, update, alter, delete, drop, etc.

What is DDL and DML?

All the SQL commands are categorized into five categories: DDL,DML,DCL,DQL,TCL. In this course we are going to cover only DDL and DML commands in detail.

Data definition Language(DDL): Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. Example: Create, Drop, Alter, Truncate.

Data Manipulation Language(DML): The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. Example: Insert, Delete, Update.

Data Types in MySQL

Data stored in a database table are of different types, As SQL developers we have chose the suitable data types for each field while defining a table. SQL offers supports a wide range of data types from which the developer can choose the most appropriate data types for each column.

char(size): used for fixed length string data.
Example: A column defined as char(10) , it can contain string values of maximum 10 length. SQL allocates 10 bytes of memory irrespective of legth of data to be stored.

varchar(size): used for variable length string data.
Example: If a column defined as varchar(10) , SQL allocates maximum 10 bytes to each value, but bytes allocated may vary depending on the length of data.

int( ): Used for integer/digits data without decimal. Can accommodate maximum 11 digit numbers.

float(M,D): Permits real numbers upto M digits, out of which may be D digits after decimal .

Example: a column data type defined as float(6,3) may have 234.684

Date: used to store date in YYYY-MM-DD format.

Constraints:

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

Not Null- Ensures that a column cannot have a NULL value

Unique- Ensures that all values in a column are different

Primary Key- A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

Foreign Key- Prevents actions that would destroy links between tables

Check - Ensures that the values in a column satisfies a specific condition

Default- Sets a default value for a column if no value is specified

MySql Commands:

CREATE Database: Used to create a new database.
Syntax: CREATE DATABASE <database name>
e.g. CREATE Database MySchool;

```
mysql> create database MySchool;
Query OK, 1 row affected (0.01 sec)
```

SHOW Databases: Used to list all existing databases.
Syntax: SHOW Databases;

```
mysql> use MySchool;
Database changed
```

DROP Database: Used to delete an existing database.
Syntax: DROP Database <databasename>
e.g. DROP Database MyStore;

USE Database: Used to select/enter a existing database.
e.g. USE MySchool;

MySql Commands:

Show Tables: After a database has been selected this command can be Used to list all the tables in the database. e.g. SHOW TABLES;

CREATE Table: Syntax: CREATE TABLE <table name>(column1 datatype, column2 datatype, column3 datatype, ... columnN datatype, PRIMARY KEY(one or more columns));

E.g. CREATE TABLE cs_students(sid int(3), sname varchar(30), sclass int(2), smark int(3), skill varchar(30), primary key(sid));

```
mysql> CREATE TABLE cs_students(sid int(3), sname varchar(30), sclass int(2),
smark int(3), skill varchar(30), primary key(sid));
Query OK, 0 rows affected, 3 warnings (0.03 sec)
```

Note: Constraints other then Primary Key can also be specified when required

MySql Commands:

Creating a table with multiple constraints:

```
CREATE TABLE Employee (Eid int(5) Primary Key,
                       Ename varchar(30) Not Null,
                       age int(2),
                       Dept varchar(20) Default “Manufacturing”,
                       contactno int(10) unique,
                       Constraint ChkAge Check(Age>18));
```

```
mysql> CREATE TABLE Employee (Eid int(5) Primary Key, Ename varchar(30) Not Null,
age int(2), Dept varchar(20) Default "Manufacturing",contactno int(10) unique,Con
straint ChkAge Check(Age>18));
Query OK, 0 rows affected, 3 warnings (0.02 sec)
```

MySql Commands:

DESCRIBE Tables: A DDL command to display the structure of the table.

Syntax: DESCRIBE <table name>;

```
mysql> desc employee;
+----------+------------+------+-----+---------------+-------+
| Field    | Type       | Null | Key | Default       | Extra |
+----------+------------+------+-----+---------------+-------+
| Eid      | int        | NO   | PRI | NULL          |       |
| Ename    | varchar(30)| NO   |     | NULL          |       |
| age      | int        | YES  |     | NULL          |       |
| Dept     | varchar(20)| YES  |     | Manufacturing |       |
| contactno| int        | YES  | UNI | NULL          |       |
+----------+------------+------+-----+---------------+-------+
5 rows in set (0.01 sec)
```

MySql Commands:

ALTER Tables: ALTER TABLE is a DDL command that can change the structure of the table. Using ALTER TABLE command we can add, delete or modify the attributes/constraints of a table.

Adding a column using Alter table:

Syntax: ALTER TABLE <table name> ADD column <Column Name Data type>;

```
mysql> ALTER TABLE cs_students ADD Column Address varchar(40);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Deleting a column using Alter table:

Syntax: ALTER TABLE <table name> DROP column <Column Name >;

```
mysql> ALTER TABLE cs_students DROP column Address;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

MySql Commands:

Modify a column using Alter table:

Syntax: ALTER TABLE <table name> MODIFY column <Column Name Data type>;

E.g. MODIFY the data type of an existing column

```
mysql> ALTER TABLE cs_students MODIFY column sid int(4);
Query OK, 0 rows affected, 1 warning (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Adding a Primary Key Constraint using Alter table:

Syntax: ALTER TABLE <table name> ADD Primary Key (<column names>);

```
mysql> ALTER TABLE cs_students ADD PRIMARY KEY(sid);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Drop Primary Key Constraint using Alter table:

Syntax: ALTER TABLE <table name> DROP Primary Key;

```
mysql> ALTER TABLE cs_students DROP PRIMARY KEY;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

MySql Commands:

DROP Tables: DROP TABLE is a DDL command used to delete a table from the database.

Syntax: DROP TABLE <table name>;

E.g. DROP Table Employee;

INSERT INTO: INSERT is a DML command used to insert a new record/row in an existing table.

Syntax: INSERT INTO <Table Name> values (val1,val2,val3..);

```
mysql> INSERT INTO cs_students values(101,"jatin",12,98,"Python");
Query OK, 1 row affected (0.01 sec)
```

Insert a new record in the table with specific field value.

Syntax: INSERT INTO <Table Name> (Column1,Column2,..ColumnN) values (val1,val2,..valN);

```
mysql> INSERT INTO cs_students (sid,sname,skill)values(102,"Amit","Networking");
Query OK, 1 row affected (0.01 sec)
```

MySql Commands:

SELECT Command: Used to retrieve and display data from the tables.

Syntax: SELECT column1, column2,..
FROM <Table Name>;

E.g. SELECT sid, sname FROM cs_students;

[Here only sid and sname column has been selected]

To Select all the columns from the table:

SELECT * FROM <Table Name>;

Table: cs_students

```
mysql> select * from cs_students;
+-----+----------+--------+-------+------------+
| sid | sname    | sclass | smark | skill      |
+-----+----------+--------+-------+------------+
| 101 | jatin    |     12 |    98 | Python     |
| 102 | Aman     |     12 |  NULL | Networking |
| 103 | Akash    |     12 |    72 | Database   |
| 104 | Igon     |     12 |    85 | Python     |
| 105 | Arnab    |     12 |    91 | Database   |
| 106 | Lungsome |     12 |    87 | Networking |
| 107 | Aditya   |     12 |    86 | Python     |
| 108 | Suprit   |     12 |    92 | Python     |
| 109 | Apsana   |     12 |    79 | Networking |
| 110 | Kaming   |     12 |    88 | Database   |
| 111 | Ankit    |     12 |    75 | Networking |
| 112 | Abu      |     12 |    96 | Python     |
| 113 | Sanket   |     12 |    95 | Python     |
| 114 | Lakhi    |     12 |    83 | Database   |
+-----+----------+--------+-------+------------+
14 rows in set (0.00 sec)
```

MySql Commands:

WHERE Clause: The WHERE Clause can be used with SELECT command to select the data from the table based on some condition.

Syntax: SELECT column1, column2,..
FROM <Table Name>
WHERE <condition>;

Operators for Where Clause :
Mathematical: +, -, *, /
Relational: >, >=, <, <=, =, <>
Logical: AND, OR, NOT

E.g. Select * From cs_students WHERE smark>90;

```
mysql> Select * From cs_students WHERE smark>90;
+-----+--------+--------+-------+----------+
| sid | sname  | sclass | smark | skill    |
+-----+--------+--------+-------+----------+
| 101 | jatin  |     12 |    98 | Python   |
| 105 | Arnab  |     12 |    91 | Database |
| 108 | Suprit |     12 |    92 | Python   |
| 112 | Abu    |     12 |    96 | Python   |
| 113 | Sanket |     12 |    95 | Python   |
+-----+--------+--------+-------+----------+
5 rows in set (0.00 sec)
```

Where Clause Examples:

To select ID and Name of the students whose skill is Database:

```
mysql> Select * from cs_students WHERE skill="Database";
+-----+--------+--------+-------+----------+
| sid | sname  | sclass | smark | skill    |
+-----+--------+--------+-------+----------+
| 103 | Akash  |     12 |    72 | Database |
| 105 | Arnab  |     12 |    91 | Database |
| 110 | Kaming |     12 |    88 | Database |
| 114 | Lakhi  |     12 |    83 | Database |
+-----+--------+--------+-------+----------+
4 rows in set (0.01 sec)
```

Using Logical Operators in Where clause:

```
mysql> Select * from cs_students WHERE skill="Python" and smark>95;
+-----+-------+--------+-------+--------+
| sid | sname | sclass | smark | skill  |
+-----+-------+--------+-------+--------+
| 101 | jatin |     12 |    98 | Python |
| 112 | Abu   |     12 |    96 | Python |
+-----+-------+--------+-------+--------+
2 rows in set (0.00 sec)
```

IN Operator: Used To specify multiple possible values for a column

E.g. Select * from cs_student where skill in(“Networking”, ”Database”);

```
mysql> Select * from cs_students where skill in("Networking", "Database");
+-----+---------+--------+-------+------------+
| sid | sname   | sclass | smark | skill      |
+-----+---------+--------+-------+------------+
| 102 | Aman    |     12 |  NULL | Networking |
| 103 | Akash   |     12 |    72 | Database   |
| 105 | Arnab   |     12 |    91 | Database   |
| 106 | Lungsome|     12 |    87 | Networking |
| 109 | Apsana  |     12 |    79 | Networking |
| 110 | Kaming  |     12 |    88 | Database   |
| 111 | Ankit   |     12 |    75 | Networking |
| 114 | Lakhi   |     12 |    83 | Database   |
+-----+---------+--------+-------+------------+
```

BETWEEN Operator: Used To specify values in a certain range.

E.g. Select * from cs_student where smark BETWEEN 95 AND 100;

```
mysql> Select * from cs_students where smark BETWEEN 95 AND 100;
+-----+--------+--------+-------+--------+
| sid | sname  | sclass | smark | skill  |
+-----+--------+--------+-------+--------+
| 101 | jatin  |     12 |    98 | Python |
| 112 | Abu    |     12 |    96 | Python |
| 113 | Sanket |     12 |    95 | Python |
+-----+--------+--------+-------+--------+
```

DISTINCT Clause: Used to retrieve the distinct values in a field.
Syntax: Select * from student where mark is null;

```
mysql> SELECT DISTINCT skill FROM cs_students;
+------------+
| skill      |
+------------+
| Python     |
| Networking |
| Database   |
+------------+
```

ORDER BY: It is used to sort the data in ascending or descending order. By default ORDER BY sort the data in ascending order, for descending order we need to use ”DESC”.

```
mysql> SELECT *  FROM cs_students ORDER BY smark;
+-----+----------+--------+-------+------------+
| sid | sname    | sclass | smark | skill      |
+-----+----------+--------+-------+------------+
| 103 | Akash    |     12 |    72 | Database   |
| 111 | Ankit    |     12 |    75 | Networking |
| 109 | Apsana   |     12 |    79 | Networking |
| 114 | Lakhi    |     12 |    83 | Database   |
| 102 | Aman     |     12 |    84 | Networking |
| 104 | Igon     |     12 |    85 | Python     |
| 107 | Aditya   |     12 |    86 | Python     |
| 106 | Lungsome |     12 |    87 | Networking |
| 110 | Kaming   |     12 |    88 | Database   |
| 105 | Arnab    |     12 |    91 | Database   |
| 108 | Suprit   |     12 |    92 | Python     |
| 113 | Sanket   |     12 |    95 | Python     |
| 112 | Abu      |     12 |    96 | Python     |
| 101 | jatin    |     12 |    98 | Python     |
+-----+----------+--------+-------+------------+
```

```
mysql> SELECT *  FROM cs_students ORDER BY smark DESC;
+-----+----------+--------+-------+------------+
| sid | sname    | sclass | smark | skill      |
+-----+----------+--------+-------+------------+
| 101 | jatin    |     12 |    98 | Python     |
| 112 | Abu      |     12 |    96 | Python     |
| 113 | Sanket   |     12 |    95 | Python     |
| 108 | Suprit   |     12 |    92 | Python     |
| 105 | Arnab    |     12 |    91 | Database   |
| 110 | Kaming   |     12 |    88 | Database   |
| 106 | Lungsome |     12 |    87 | Networking |
| 107 | Aditya   |     12 |    86 | Python     |
| 104 | Igon     |     12 |    85 | Python     |
| 102 | Aman     |     12 |    84 | Networking |
| 114 | Lakhi    |     12 |    83 | Database   |
| 109 | Apsana   |     12 |    79 | Networking |
| 111 | Ankit    |     12 |    75 | Networking |
| 103 | Akash    |     12 |    72 | Database   |
+-----+----------+--------+-------+------------+
```

Handling NULL Values: To handle NULL entries in a field we can use “IS” and “IS NOT”, as NULL value is a Value which is Unknown so we can use =, <> operators to select NULL values.

```
mysql> select * from employee;
+-----+--------+------+---------------+--------+
| Eid | Ename  | age  | Dept          | salary |
+-----+--------+------+---------------+--------+
| 201 | Sunil  |   34 | Accoutant     |  25000 |
| 202 | Kunal  |   32 | Sales         |   NULL |
| 203 | Sambit |   32 | Testing       |  23000 |
| 204 | Monaj  |   32 | Merchandise   |  32000 |
| 205 | Pratap |   33 | Manufacturing |   NULL |
+-----+--------+------+---------------+--------+
```

Lets Consider the Employee table above, to select all the employees whose salary is specified as NULL in the salary field we must use IS NULL operator.

```
mysql> SELECT * FROM employee WHERE salary IS NULL;
+-----+--------+------+---------------+--------+
| Eid | Ename  | age  | Dept          | salary |
+-----+--------+------+---------------+--------+
| 202 | Kunal  |   32 | Sales         |   NULL |
| 205 | Pratap |   33 | Manufacturing |   NULL |
+-----+--------+------+---------------+--------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM employee WHERE salary IS NOT NULL;
+-----+--------+------+-------------+--------+
| Eid | Ename  | age  | Dept        | salary |
+-----+--------+------+-------------+--------+
| 201 | Sunil  |   34 | Accoutant   |  25000 |
| 203 | Sambit |   32 | Testing     |  23000 |
| 204 | Monaj  |   32 | Merchandise |  32000 |
+-----+--------+------+-------------+--------+
```

LIKE OPERATOR: LIKE is used for string matching in MySql, it can be used for comparison of character strings using pattern. LIKE uses the following two wildcard characters to create string patterns.

- **Percent(%):** used to match a substring of any length.
- **Underscore(_):** Used to match any single character.

The LIKE keyword selects the rows having column values that matches with the wildcard pattern.

e.g. To select details of employees whose name start with the letter “S”.

```
mysql> SELECT * FROM employee WHERE ename Like "S%";
+-----+--------+------+-----------+--------+
| Eid | Ename  | age  | Dept      | salary |
+-----+--------+------+-----------+--------+
| 201 | Sunil  |   34 | Accoutant |  25000 |
| 203 | Sambit |   32 | Testing   |  23000 |
+-----+--------+------+-----------+--------+
```

To select details of employees whose name ends with the letter “l” and has exactly a 5 characters name.

```
mysql> SELECT * FROM employee WHERE ename Like "____l";
+-----+-------+------+-----------+--------+
| Eid | Ename | age  | Dept      | salary |
+-----+-------+------+-----------+--------+
| 201 | Sunil |   34 | Accoutant |  25000 |
| 202 | Kunal |   32 | Sales     |   NULL |
+-----+-------+------+-----------+--------+
```

Note: Patterns are case sensitive, upper case characters do not match with lower case characters or vice-versa.

Update Command : UPDATE is a DML command used to change values in the rows of a existing table. It specifies the rows to be changed using WHERE clause and the new values to be updated using SET keyword.

Syntax: UPDATE <Table Name> SET column=<new value> WHERE <condition>

E.g. To change the salary to 70000 of the employee having Eid 204.

UPDATE employee SET salary=70000 WHERE Eid=204.

To change the Department of an employee

UPDATE employee SET Dept="Marketing" where Ename="Kunal";

```
mysql> select * from employee;
+-----+--------+------+--------------+--------+
| Eid | Ename  | age  | Dept         | salary |
+-----+--------+------+--------------+--------+
| 201 | Sunil  |   34 | Accounts     |  25000 |
| 202 | Kunal  |   32 | Sales        |   NULL |
| 203 | Sambit |   32 | Testing      |  23000 |
| 204 | Monaj  |   32 | Merchandise  |  32000 |
| 205 | Pratap |   33 | Manufacturing|   NULL |
+-----+--------+------+--------------+--------+
```

Employee Table before Update.

```
mysql> select * from employee;
+-----+--------+------+--------------+--------+
| Eid | Ename  | age  | Dept         | salary |
+-----+--------+------+--------------+--------+
| 201 | Sunil  |   34 | Accounts     |  25000 |
| 202 | Kunal  |   32 | Marketing    |   NULL |
| 203 | Sambit |   32 | Testing      |  23000 |
| 204 | Monaj  |   32 | Merchandise  |  70000 |
| 205 | Pratap |   33 | Manufacturing|   NULL |
+-----+--------+------+--------------+--------+
```

Employee Table after Update

Delete Command :

Delete is a DML command used to delete rows of an existing table. It specifies the rows to be deleted using WHERE clause.

Syntax: DELETE FROM <Table Name> WHERE <condition;

To delete the record/row of the employee having Eid 204.
DELETE FROM employee WHERE Eid=204;

To delete the records of all the employee working in Sales Department.
DELETE FROM employee WHERE Dept="salary";

To delete all rows of employee table
DELETE FROM employee;

Aggregate Functions :

MySql supports the following aggregate/multiple row functions:

- **count( ):** returns the number of rows in the given column or expression.
- **min( ):** returns the minimum value in the given column or expression.
- **max( ):** returns the maximum value in the given column or expression.
- **sum( ):** returns the sum of values in the given column or expression.
- **avg( ):** returns the average of values in the given column or expression.

Aggregate Functions Example:

Let us Consider the employee table:

```
mysql> select * from employee;
+-----+--------+------+---------------+--------+
| Eid | Ename  | age  | Dept          | salary |
+-----+--------+------+---------------+--------+
| 201 | Sunil  |   34 | Accoutant     |  25000 |
| 202 | Kunal  |   32 | Sales         |   NULL |
| 203 | Sambit |   32 | Testing       |  23000 |
| 204 | Monaj  |   32 | Merchandise   |  32000 |
| 205 | Pratap |   33 | Manufacturing |   NULL |
+-----+--------+------+---------------+--------+
5 rows in set (0.00 sec)
```

Queries	Output
SELECT sum(salary) FROM employee;	80000
SELECT avg(salary) FROM employee;	26666.6666
SELECT max(salary) FROM employee;	32000
SELECT min(salary) FROM employee;	23000
SELECT count(salary) FROM employee;	3
SELECT count(*) FROM employee;	5

GROUP BY:

- GROUP BY clause combines all those records that have identical values in a particular field or a group of fields.
- It is used in SELECT statement to divide the table into groups. Grouping can be done by a column name or with aggregate functions.

For Example let up consider the cs_students table, To find the number of students in each skill, we can use the command.

SELECT skill, count(*) from cs_students GROUP BY skill;

```
mysql> SELECT skill, count(*) FROM cs_students GROUP BY skill;
+------------+----------+
| skill      | count(*) |
+------------+----------+
| Python     |        6 |
| Networking |        4 |
| Database   |        4 |
+------------+----------+
```

Table: cs_students

```
mysql> SELECT * FROM cs_students;
+-----+----------+--------+-------+------------+
| sid | sname    | sclass | smark | skill      |
+-----+----------+--------+-------+------------+
| 101 | jatin    |     12 |    98 | Python     |
| 102 | Aman     |     12 |    84 | Networking |
| 103 | Akash    |     12 |    72 | Database   |
| 104 | Igon     |     12 |    85 | Python     |
| 105 | Arnab    |     12 |    91 | Database   |
| 106 | Lungsome |     12 |    87 | Networking |
| 107 | Aditya   |     12 |    86 | Python     |
| 108 | Suprit   |     12 |    92 | Python     |
| 109 | Apsana   |     12 |    79 | Networking |
| 110 | Kaming   |     12 |    88 | Database   |
| 111 | Ankit    |     12 |    75 | Networking |
| 112 | Abu      |     12 |    96 | Python     |
| 113 | Sanket   |     12 |    95 | Python     |
| 114 | Lakhi    |     12 |    83 | Database   |
+-----+----------+--------+-------+------------+
```

Here the GROUP BY clause creates 3 groups based on skill values and then the aggregate function count(*) is independently applied to each group.

GROUP BY Examples:

When used with GROUP BY clause, the aggregate functions perform calculations on groups of rows (formed by GROUP BY), and hence returns a single value for each group.

Let us consider the employee table, To find the average salary of employees of each department, we can use the command.

SELECT dept, avg(salary) from employee GROUP BY dept;

TABLE: Employee

```
mysql> SELECT * FROM employee;
+-----+--------+------+-----------+--------+
| Eid | Ename  | age  | Dept      | salary |
+-----+--------+------+-----------+--------+
| 201 | Sunil  |   34 | Accounts  |  25000 |
| 202 | Kunal  |   32 | Marketing |   NULL |
| 203 | Sambit |   32 | Testing   |  23000 |
| 204 | Monaj  |   32 | Accounts  |  70000 |
| 205 | Pratap |   33 | Testing   |  31000 |
+-----+--------+------+-----------+--------+
```

```
mysql> SELECT dept,avg(salary) FROM employee GROUP BY dept;
+-----------+-------------+
| dept      | avg(salary) |
+-----------+-------------+
| Accounts  |  47500.0000 |
| Marketing |        NULL |
| Testing   |  27000.0000 |
+-----------+-------------+
```

Here the GROUP BY clause creates 3 groups based on department values and then the aggregate function avg(salary) is independently applied to the salary field of each group to calculate average salary of each group of rows.

HAVING Clause:

HAVING clause is used to apply conditions on groups in contrast to WHERE clause which is used to apply conditions on individual rows.

Let us consider the cs_students table, To find the average marks of the group of students having a particular skill , where the skill group must have at least 5 students.

SELECT skill, avg(smark) FROM cs_students GROUP BY skill HAVING count(*)>=5;

Table: cs_students

```
mysql> SELECT * FROM cs_students;
+-----+----------+--------+-------+------------+
| sid | sname    | sclass | smark | skill      |
+-----+----------+--------+-------+------------+
| 101 | jatin    |     12 |    98 | Python     |
| 102 | Aman     |     12 |    84 | Networking |
| 103 | Akash    |     12 |    72 | Database   |
| 104 | Igon     |     12 |    85 | Python     |
| 105 | Arnab    |     12 |    91 | Database   |
| 106 | Lungsome |     12 |    87 | Networking |
| 107 | Aditya   |     12 |    86 | Python     |
| 108 | Suprit   |     12 |    92 | Python     |
| 109 | Apsana   |     12 |    79 | Networking |
| 110 | Kaming   |     12 |    88 | Database   |
| 111 | Ankit    |     12 |    75 | Networking |
| 112 | Abu      |     12 |    96 | Python     |
| 113 | Sanket   |     12 |    95 | Python     |
| 114 | Lakhi    |     12 |    83 | Database   |
+-----+----------+--------+-------+------------+
```

```
mysql> SELECT skill,avg(smark) FROM cs_students GROUP BY skill HAVING count(*)>=5;
+--------+------------+
| skill  | avg(smark) |
+--------+------------+
| Python |    92.0000 |
+--------+------------+
1 row in set (0.00 sec)
```

Out of 3 groups created by GROUP BY clause, two groups have been removed from result set as they have not satisfied the condition mentioned in HAVING clause.

HAVING Examples:

Conditions in having clause may contain a Boolean expression or aggregate functions can be used to specify the condition.

We can also use more then one condition in HAVING clause by using logical operators.

Let us consider the employee table, To find the average salary of employees of each department, where the average age of all the employees working in the department is less then 32.

SELECT dept,avg(salary) FROM employee GROUP BY dept HAVING avg(age)>32;

Table: Employee

```
mysql> SELECT * FROM employee;
+-----+--------+------+-----------+--------+
| Eid | Ename  | age  | Dept      | salary |
+-----+--------+------+-----------+--------+
| 201 | Sunil  |   34 | Accounts  |  25000 |
| 202 | Kunal  |   32 | Marketing |   NULL |
| 203 | Sambit |   32 | Testing   |  23000 |
| 204 | Monaj  |   32 | Accounts  |  70000 |
| 205 | Pratap |   33 | Testing   |  31000 |
+-----+--------+------+-----------+--------+
```

```
mysql> SELECT dept,avg(salary) FROM employee GROUP BY dept HAVING avg(age)>32;
+----------+-------------+
| dept     | avg(salary) |
+----------+-------------+
| Accounts |  47500.0000 |
| Testing  |  27000.0000 |
+----------+-------------+
```

Out of 3 groups created by GROUP BY clause, one group is filtered out of the result because it doesn't satisfy the condition mentioned in HAVING clause.

JOIN:

A JOIN clause combines rows from two or more tables. In a join query, more then one table are listed in FORM clause.

Types of Join Operation:

- Cartesian product on two tables,
- Equi-join
- Natural join

Cartesian Product (X):

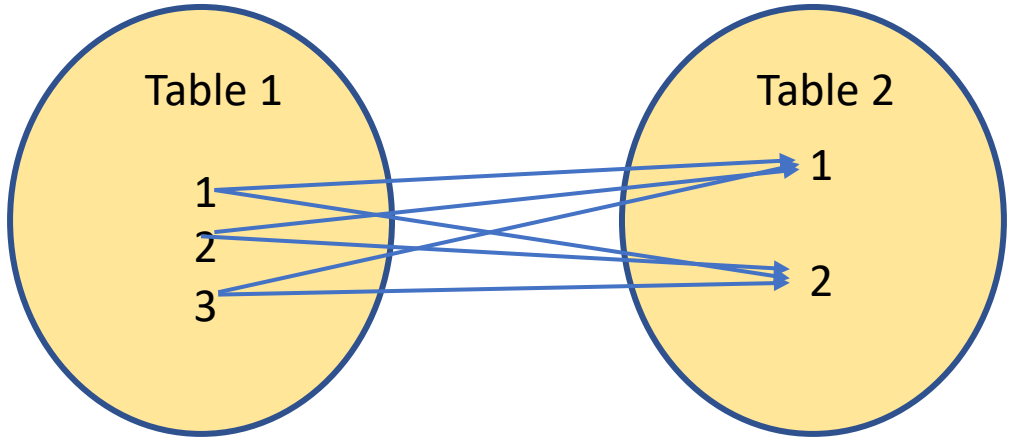
The Cartesian Product operation of two tables produces all possible concatenations of all the rows of both tables.

The Cartesian product(also known as Cross Join) multiplies all rows present in the first table with all the rows present in the second table

Syntax: SELECT * FROM Table1,Table2;

Or

SELECT * FROM Table1 CROSS JOIN Table2;



The Cardinality of cartesian product of two relations R1 and R2 is equal to the multiplication of cardinalities of R1 and R2. Whereas The Degree of cartesian Product is equal to addition of degrees of R1 and R2.

Cartesian Product (X) Example:

Let us Consider the Cartesian Product/Cross Join the of following Customer and Order Tables

Table: Customer

```
mysql> SELECT * FROM Customer;
+----------------+---------+--------+
| cname          | city    | custid |
+----------------+---------+--------+
| Vipul Raj      | Lucknow |      1 |
| Prakash Chandra| Siwan   |      2 |
| Dinesh Kumar   | Dehradun|      3 |
| Pralay Ghosh   | Kolkata |      4 |
+----------------+---------+--------+
```

Table: Orders

```
mysql> SELECT * FROM Orders;
+---------+----------+--------+
| orderid | quantity | custid |
+---------+----------+--------+
|    1001 |       21 |      2 |
|    1002 |       26 |      1 |
|    1003 |       12 |      3 |
+---------+----------+--------+
```

Result of Cartesian Product

```
mysql> SELECT * FROM Customer,Orders;
+----------------+---------+--------+---------+----------+--------+
| cname          | city    | custid | orderid | quantity | custid |
+----------------+---------+--------+---------+----------+--------+
| Vipul Raj      | Lucknow |      1 |    1003 |       12 |      3 |
| Vipul Raj      | Lucknow |      1 |    1002 |       26 |      1 |
| Vipul Raj      | Lucknow |      1 |    1001 |       21 |      2 |
| Prakash Chandra| Siwan   |      2 |    1003 |       12 |      3 |
| Prakash Chandra| Siwan   |      2 |    1002 |       26 |      1 |
| Prakash Chandra| Siwan   |      2 |    1001 |       21 |      2 |
| Dinesh Kumar   | Dehradun|      3 |    1003 |       12 |      3 |
| Dinesh Kumar   | Dehradun|      3 |    1002 |       26 |      1 |
| Dinesh Kumar   | Dehradun|      3 |    1001 |       21 |      2 |
| Pralay Ghosh   | Kolkata |      4 |    1003 |       12 |      3 |
| Pralay Ghosh   | Kolkata |      4 |    1002 |       26 |      1 |
| Pralay Ghosh   | Kolkata |      4 |    1001 |       21 |      2 |
+----------------+---------+--------+---------+----------+--------+
12 rows in set (0.01 sec)
```

The Cardinality of Customer Table is 4 and Degree is 3.
The Cardinality of Orders Table is 3 and Degree is 3.
Hence Cardinality of Cartesian Product Result set is 12 (4*3) and Degree is 6 (3+3)

Equi Join :

To perform Equi/Inner Join on two relations R1 and R2, we have to specify a equality condition using the common attribute in both the relations R1 and R2. Syntax: SELECT * FROM R1 Inner Join R2.

Table: Customer

```
mysql> SELECT * FROM Customer;
+----------------+---------+--------+
| cname          | city    | custid |
+----------------+---------+--------+
| Vipul Raj      | Lucknow |      1 |
| Prakash Chandra| Siwan   |      2 |
| Dinesh Kumar   | Dehradun|      3 |
| Pralay Ghosh   | Kolkata |      4 |
+----------------+---------+--------+
```

Table: Orders

```
mysql> SELECT * FROM Orders;
+---------+----------+--------+
| orderid | quantity | custid |
+---------+----------+--------+
|    1001 |       21 |      2 |
|    1002 |       26 |      1 |
|    1003 |       12 |      3 |
+---------+----------+--------+
```

Result of Equi/ Inner Join

```
mysql> SELECT * FROM Customer Inner Join Orders WHERE Customer.custid=Orders.Custid;
+----------------+---------+--------+---------+----------+--------+
| cname          | city    | custid | orderid | quantity | custid |
+----------------+---------+--------+---------+----------+--------+
| Prakash Chandra| Siwan   |      2 |    1001 |       21 |      2 |
| Vipul Raj      | Lucknow |      1 |    1002 |       26 |      1 |
| Dinesh Kumar   | Dehradun|      3 |    1003 |       12 |      3 |
+----------------+---------+--------+---------+----------+--------+
3 rows in set (0.00 sec)
```

Equi join resultset contains only those rows from both the tables which have matching value in the common attribute (Custid) mentioned in the equality condition.

Natural Join :

The Join in which only one of the identical columns(coming from joined tables) exists, is called as Natural Join.

The Equi Join and Natural join are equivalent except that duplicate columns are eliminated in the Natural Join that would otherwise appear in Equi Join.

Syntax: SELECT * FROM Table1 Natural Join Table2

Performing natural Join on Customer and Orders Table, results in selecting only those rows from both the tables which have matching values in common attribute CustId.

Result of Natural Join

```
mysql> SELECT * FROM Customer Natural Join Orders;
+--------+----------------+---------+---------+----------+
| custid | cname          | city    | orderid | quantity |
+--------+----------------+---------+---------+----------+
|      2 | Prakash Chandra | Siwan   |    1001 |       21 |
|      1 | Vipul Raj      | Lucknow |    1002 |       26 |
|      3 | Dinesh Kumar   | Dehradun |    1003 |       12 |
+--------+----------------+---------+---------+----------+
3 rows in set (0.00 sec)
```

Table: Orders

```
mysql> SELECT * FROM Orders;
+---------+----------+--------+
| orderid | quantity | custid |
+---------+----------+--------+
|    1001 |       21 |      2 |
|    1002 |       26 |      1 |
|    1003 |       12 |      3 |
+---------+----------+--------+
```

Table: Customer

```
mysql> SELECT * FROM Customer;
+----------------+----------+--------+
| cname          | city     | custid |
+----------------+----------+--------+
| Vipul Raj      | Lucknow  |      1 |
| Prakash Chandra | Siwan    |      2 |
| Dinesh Kumar   | Dehradun |      3 |
| Pralay Ghosh   | Kolkata  |      4 |
+----------------+----------+--------+
```

Examples on Join :

Table: Customer

```
mysql> SELECT * FROM Customer;
+----------------+---------+--------+
| cname          | city    | custid |
+----------------+---------+--------+
| Vipul Raj      | Lucknow |      1 |
| Prakash Chandra| Siwan   |      2 |
| Dinesh Kumar   | Dehradun|      3 |
| Pralay Ghosh   | Kolkata |      4 |
+----------------+---------+--------+
```

Table: Orders

```
mysql> SELECT * FROM Orders;
+---------+----------+--------+
| orderid | quantity | custid |
+---------+----------+--------+
|    1001 |       21 |      2 |
|    1002 |       26 |      1 |
|    1003 |       12 |      3 |
+---------+----------+--------+
```

To display the Customer Id, Customer Name and order Quantity of those Customers having minimum order quantity of 20.

Solution Using Cartesian Product:

```
mysql> SELECT A.Custid,Cname,Quantity FROM Customer as A ,Orders as B WHERE A.Custid=B.Custid and Quantity>=20;
+--------+----------------+----------+
| Custid | Cname          | Quantity |
+--------+----------------+----------+
|      2 | Prakash Chandra |       21 |
|      1 | Vipul Raj      |       26 |
+--------+----------------+----------+
```

Solution using Natural Join:

```
mysql> SELECT Custid,Cname,Quantity FROM Customer Natural Join Orders WHERE Quantity>=20;
+--------+----------------+----------+
| Custid | Cname          | Quantity |
+--------+----------------+----------+
|      2 | Prakash Chandra |       21 |
|      1 | Vipul Raj      |       26 |
+--------+----------------+----------+
```

Examples on Join :

Table: Customer

```
mysql> SELECT * FROM Customer;
+----------------+---------+--------+
| cname          | city    | custid |
+----------------+---------+--------+
| Vipul Raj      | Lucknow |      1 |
| Prakash Chandra| Siwan   |      2 |
| Dinesh Kumar   | Dehradun|      3 |
| Pralay Ghosh   | Kolkata |      4 |
+----------------+---------+--------+
```

Table: Orders

```
mysql> SELECT * FROM Orders;
+---------+----------+--------+
| orderid | quantity | custid |
+---------+----------+--------+
|    1001 |       21 |      2 |
|    1002 |       26 |      1 |
|    1003 |       12 |      3 |
+---------+----------+--------+
```

To display the OrderID along with the customer name and city of the customers having order quantity less then 20.

Solution Using Cartesian Product:

```
mysql> SELECT OrderId,Cname,City FROM Customer as A ,Orders as B WHERE A.Custid=B.Custid and Quantity<20;
+---------+-------------+----------+
| OrderId | Cname       | City     |
+---------+-------------+----------+
|    1003 | Dinesh Kumar | Dehradun |
+---------+-------------+----------+
```

Solution using Natural Join:

```
mysql> SELECT OrderID,Cname,City FROM Customer Natural Join Orders WHERE Quantity<20;
+---------+-------------+----------+
| OrderID | Cname       | City     |
+---------+-------------+----------+
|    1003 | Dinesh Kumar | Dehradun |
+---------+-------------+----------+
```