SQL Command: 1)

DDL(Data definition language)

Create,alter,rename,truncate

2) DML (data manipulation language) select , insert, update,delete

Constraints: primary key, Not Null, Foreign key,unique,default

SQL Datatype:

1) Numeric Datatype: int,integer, smallint, Decimal, float, double
2) String Datatype: char,varchar(size),blob,Text(size),longtext ,longblob
3) Date type: date,datetime,time,year,

Alter command: add,drop,modify(column)

//create table employee(id int primary key auto_increment,name varchar(20) not null,doj date);

// insert into employee values(5,'ajay','2020-09-24');

// insert into employee(name,doj) values('ajay','2020-09-24');

//show columns from employee; same as desc employee

// create table emp1(id int ,name varchar(20) not null default 'uname');

//Alter table employee add salary int;

//Alter table employee add bonus int,add company varchar(20) not null;

//**ALTER TABLE table_name DROP COLUMN column_name;**

//ALTER TABLE table_name MODIFY column_name column_type;

// alter table emp4 modify name char(30);

//alter table emp4 change column name firstname varchar(20);

// alter table emp4 rename to emp5;

Truncate:The truncate statement in mysql remvoes the complete data without

Removing its structure;It is a DDL;

truncate table employee;

Example #1

Let us create the developer's named table in our database saicoding that will hold all the above data.

```
mysql> desc devlopers;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| devloper_id | int(11)      | NO   | PRI | NULL    | auto_increment |
| team_id     | int(11)      | NO   |     | NULL    |                |
| name        | varchar(100) | YES  | UNI | NULL    |                |
| position    | varchar(100) | YES  |     | NULL    |                |
| technology  | varchar(100) | YES  |     | NULL    |                |
| salary      | int(11)      | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------
```

Example #2

Now, we will insert some records in it using the following query statements.

```
INSERT INTO `devlopers` VALUES
(1,1,'Payal','Developer','Angular',30000),
(2,1,'Heena','Developer','Angular',10000),
(3,3,'Vishnu','Manager','Maven',25000),
```

(4,3,'Rahul','Support','Digital Marketing',15000),

(5,3,'Siddhesh','Tester','Maven',20000),

(6,7,'Siddharth','Manager','Java',25000),

(7,4,'Brahma','Developer','Digital Marketing',30000),

(8,1,'Arjun','Tester','Angular',19000),

(9,2,'Nitin','Developer','MySQL',20000),

(10,2,'Ramesh','Administrator','MySQL',30000),

(11,2,'Rohan','Admin',NULL,20000),

(12,2,'Raj','Designer',NULL,30000);

Example #3

Let us first retrieve the records of the table using a simple select query statement.

```
SELECT * FROM developers;
```

Example #4

Now, suppose that we want to retrieve only those records from developers tables whose salary is greater than 10000 say. For this, we will have to mention a predicate/ condition in the WHERE clause of the SELECT query statement above. Our query statement will be as follows:

```
SELECT * FROM developers WHERE salary>10000;
```

Example #5

Now, consider a situation where you want to apply multiple conditions on more than one column in a query statement so that when all the specified conditions are fulfilled then only the row should be added into the final resultset of the query. In this case, we can use the AND operator in the WHERE clause. For example, suppose that we want to find out the names of all the developers whose technology is angular and salary is greater than 10000. Then our query statement will be as follows

Query:

```
SELECT * FROM developers WHERE salary>10000 AND
technology = "Angular";
```

Example #6

When you have to apply the conditions in such a way that if either of them gets fulfilled then you want that row to be retrieved in the final set then you can use OR operator to specify the conditions in the WHERE clause. Find the records of table developers having salary greater than 27000 or is of manager position then our query statement will be as follows –

Query:

```
SELECT * FROM developers WHERE salary>27000 OR position =
"Manager";
```

Example #7

When you want to specify the range of the values that will be allowed for a certain column then you can make the use of between keyword to specify so in WHERE clause. Consider that we have to find out the developers whose salary is between 15000 to 22000 then our query statement will be as follows

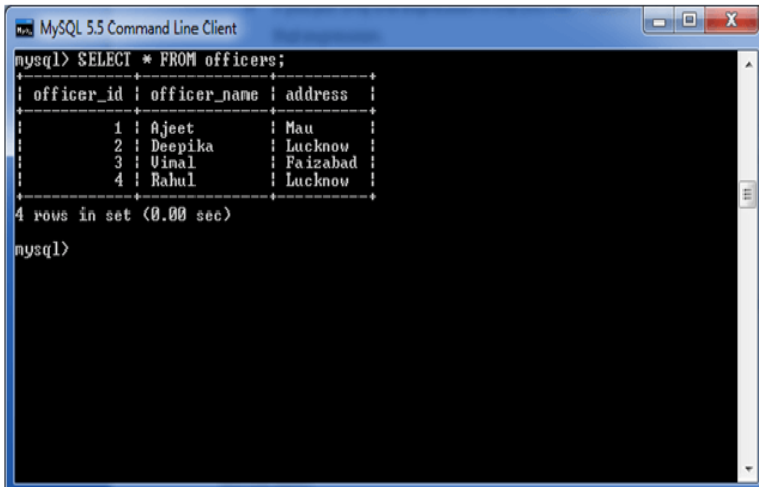
Query:

SELECT * FROM developers WHERE salary BETWEEN 15000 AND 22000;

Note: create a table officers with the following fields:

officer_id (primary key), officer_name , address

MySQL DISTINCT clause is used to remove duplicate records from the table and fetch only the unique records. The DISTINCT clause is only used with the SELECT statement.



Syntax:

SELECT DISTINCT expressions **FROM** tables [**WHERE** conditions];

1. **SELECT DISTINCT** address **FROM** officers;
2. **SELECT DISTINCT** officer_name, address **FROM** officers;

From clause:

The MySQL FROM Clause is used to select some records from a table. It can also be used to retrieve records from multiple tables using JOIN condition.

Syntax:

WHERE officer_id <= 3;

1. **FROM** table1

   [{ **INNER** JOIN | LEFT [OUTER] JOIN| RIGHT [OUTER] JOIN } table2

   ON table1.column1 = table2.column1]

1. **SELECT** * **FROM** officers

MySQL ORDER BY Clause:

The MYSQL ORDER BY Clause is used to sort the records in ascending or descending order.

Syntax:

1. **SELECT** expressions
2. **FROM** tables
3. [**WHERE** conditions]
4. **ORDER BY** expression [**ASC** | **DESC**];

1. **SELECT** * **FROM** officers **WHERE** address = 'Lucknow' **ORDER BY** officer_name;
2. **SELECT** * **FROM** officers **WHERE** address = 'Lucknow' **ORDER BY** officer_name asc;
3. **SELECT** * **FROM** officers **WHERE** address = 'Lucknow' **ORDER BY** officer_name desc;

MySQL Aggregate Functions

MySQL's aggregate function is used to **perform calculations on multiple values and return the result in a single value like the average of all values,** the sum of all values,

and maximum & minimum value among certain groups of values. We mostly use the aggregate functions with SELECT statements in the data query languages.

Aggregate Function	Descriptions
count()	It returns the number of rows, including rows with NULL values in a group.
sum()	It returns the total summed values (Non-NULL) in a set.
Avg()	It returns the average value of an expression.
min()	It returns the minimum (lowest) value in a set.
max()	It returns the maximum (highest) value in a set.

create table employee (id int primary key, name varchar (30) not null, occupation varchar (30), working_date date,working_hours time);

insert into employee1 values(1,'Sapna','Scientist','2020-05-12','08:30:30');

Query OK, 1 row affected (0.01 sec)

```
mysql> select * from employee1;
+----+---------+------------+--------------+---------------+
| id | name    | occupation | working_date | working_hours |
+----+---------+------------+--------------+---------------+
|  1 | Sapna   | Scientist  | 2020-05-12   | 08:30:30      |
|  2 | Ajay    | Teacher    | 2022-05-12   | 05:30:30      |
|  3 | Sanjay  | Devloper   | 2022-05-12   | 06:20:30      |
```

```
| 4 | Santosh | Doctor     | 2020-05-12   | 10:20:30      |
| 5 | Geeta   | Doctor     | 2018-05-12   | 10:20:30      |
| 6 | Geeta   | Teacher    | 2019-05-12   | 11:20:30      |
+----+---------+------------+--------------+---------------+1 row in set (0.00 sec)
select count(id) as 'Total no of employee' from employee1;
+----------------------+
| Total no of employee |
+----------------------+
|                    6 |
+----------------------+
alter table employee1 add salary int not null;
        update employee1 set salary=50000 where id=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update employee1 set salary=60000 where id=2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update employee1 set salary=40000 where id=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update employee1 set salary=50000 where id=4;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

select * from employee1;
+----+---------+------------+--------------+---------------+--------+
```

```
| id | name    | occupation | working_date | working_hours | salary |
+----+---------+------------+--------------+---------------+--------+
|  1 | Sapna   | Scientist  | 2020-05-12   | 08:30:30      |  50000 |
|  2 | Ajay    | Teacher    | 2022-05-12   | 05:30:30      |  60000 |
|  3 | Sanjay  | Devloper   | 2022-05-12   | 06:20:30      |  40000 |
|  4 | Santosh | Doctor     | 2020-05-12   | 10:20:30      |  50000 |
|  5 | Geeta   | Doctor     | 2018-05-12   | 10:20:30      |  30000 |
|  6 | Geeta   | Teacher    | 2019-05-12   | 11:20:30      |  70000 |
```


```
mysql> select sum(salary) from employee1 ;
+-------------+
| sum(salary) |
+-------------+
|      300000 |
+-------------+
1 row in set (0.00 sec)

mysql> select avg(salary) from employee1 ;
+-------------+
| avg(salary) |
+-------------+
|  50000.0000 |
+-------------+
1 row in set (0.00 sec)

mysql> select max(salary) from employee1 ;
```

```
+-------------+
| max(salary) |
+-------------+
|      70000 |
+-------------+
1 row in set (0.01 sec)

mysql> select min(salary) from employee1 ;
+-------------+
| min(salary) |
+-------------+
|      30000 |
+-------------+
1 row in set (0.00 sec)
```

1.SELECT COUNT(name) FROM employee;

2.SELECT SUM(working_hours) AS "Total working hours" FROM employee;

```
+---------------------+
| Total working hours |
+---------------------+
|                  33 |
+---------------------+
1 row in set (0.01 sec)
```

3.SELECT AVG(working_hours) AS "Average working hours" FROM employee;

```
+-----------------------+
| Average working hours |
+-----------------------+
|               8.2500 |
+-----------------------+
```

1 row in set (0.00 sec)

4.SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;

```
+-----------------------+
| Minimum_working_hours |
+-----------------------+
| 00:00:05              |
+-----------------------+
```

1 row in set (0.00 sec)

5. SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;

```
+-----------------------+
| Maximum_working_hours |
+-----------------------+
| 00:00:10              |
+-----------------------+
```

MySQL GROUP BY Clause

The MYSQL GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.

You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n;
```

```
desc officers;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| officer_id   | int(11)     | NO   | PRI | NULL    |       |
| officer_name | varchar(20) | YES  |     | NULL    |       |
| address      | varchar(40) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

```
select * from officers;
+------------+--------------+----------+
| officer_id | officer_name | address  |
+------------+--------------+----------+
|          1 | Ajeet        | Mau      |
|          2 | Deepika      | Lucknow  |
|          3 | Vimal        | Faizabad |
|          4 | Rahul        | Lucknow  |
```

1) count repetitive number of cities in the column address.

SELECT address, COUNT(*) as "no of emp" FROM officers

-> GROUP BY address;

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

```
| address  | COUNT(*) |
+----------+----------+
| Faizabad |        1 |
| Lucknow  |        2 |
| Mau      |        1 |
+----------+----------+
3 rows in set (0.01 sec)
```

2: use employee table and find the following query

the emp_name and total working hours of each employee.

```
SELECT name, SUM(working_hours) AS "Total working hours"
    -> FROM employee
    -> GROUP BY name;
+---------+--------------------+
| name    | Total working hours |
+---------+--------------------+
| Ajay    |                 10 |
| Anjana  |                  5 |
| Sanjana |                 10 |
| Sapna   |                  8 |
+---------+--------------------+
```

```
Insert into employee values(5,"Ajay","Doctor","2022-02-04",5);

select name,sum(working_hours) As "TW" from employee group by name;
+---------+------+
| name    | TW   |
+---------+------+
| Ajay    |   15 |
| Anjana  |    5 |
| Sanjana |   10 |
```

| Sapna | 8 |
+---------+------+
The following example specifies the minimum working hours of the employees form the table "employees".
Execute the following query:

SELECT name, MIN(working_hours) AS "Minimum working hour"
FROM employees
GROUP BY name;
----
create table Agents(agent_code varchar(10) primary key,name varchar(20),working_area varchar(20),commission float
);

```
mysql> select * from Agents;
+------------+-------+--------------+------------+
| agent_code | name  | working_area | commission |
+------------+-------+--------------+------------+
| A01        | Rahul | Pune         |       0.15 |
| A02        | Ravi  | Delhi        |        0.1 |
| A03        | Ajay  | Pune         |       0.15 |
| A04        | Sneha | Delhi        |        0.2 |
| A05        | Rakhi | Delhi        |        0.1 |
+------------+-------+--------------+------------+
```

To get data of 'working_area' and minimum value of 'commission' for the agents of each 'working_area' from the 'agents' table with the following condition -

1. the 'working_area' should come in a group.

mysql> select working_area,min(commission) from agents group by working_area;

```
+--------------+-----------------+
| working_area | min(commission) |
+--------------+-----------------+
| Pune         |            0.15 |
| Delhi        |             0.1 |
```

```
mysql> select * from officers;
+----+---------+----------+---------------+
| id | name    | address  | department    |
+----+---------+----------+---------------+
|  1 | Ajit    | Mau      | Account       |
|  2 | Deepika | Lucknow  | Computer      |
|  3 | Vimal   | Faizabad | Adminstrative |
|  4 | Rahul   | Lucknow  | Account       |
|  5 | Bharat  | Lucknow  | Computer      |
|  6 | NULL    | Delhi    | NULL          |
+----+---------+----------+---------------+
6 rows in set (0.00 sec)

mysql> select department,count(id) from officers group by department;
+---------------+-----------+
| department    | count(id) |
+---------------+-----------+
| Account       |         2 |
| Computer      |         2 |
| Adminstrative |         1 |
| NULL          |         1 |
+---------------+-----------+
4 rows in set (0.00 sec)
```

mysql> select department,count(id) from officers group by department having count(id)>1;

```
+------------+-----------+
| department | count(id) |
+------------+-----------+
| Account    |         2 |
| Computer   |         2 |
+------------+-----------+
2 rows in set (0.00 sec)
```

2 rows in set (0.00 sec)

Having clause: It is used to restrict the groups of returned rows. It shows only those groups in result set whose conditions are TRUE.

mysql> select working_area,min(commission) from agents group by working_area having min(commission)>.11;

```
+--------------+-----------------+
| working_area | min(commission) |
+--------------+-----------------+
| Pune         |            0.15 |
| Delhi        |             0.1 |
```

mysql> select count(id),name from employee1 group by name;

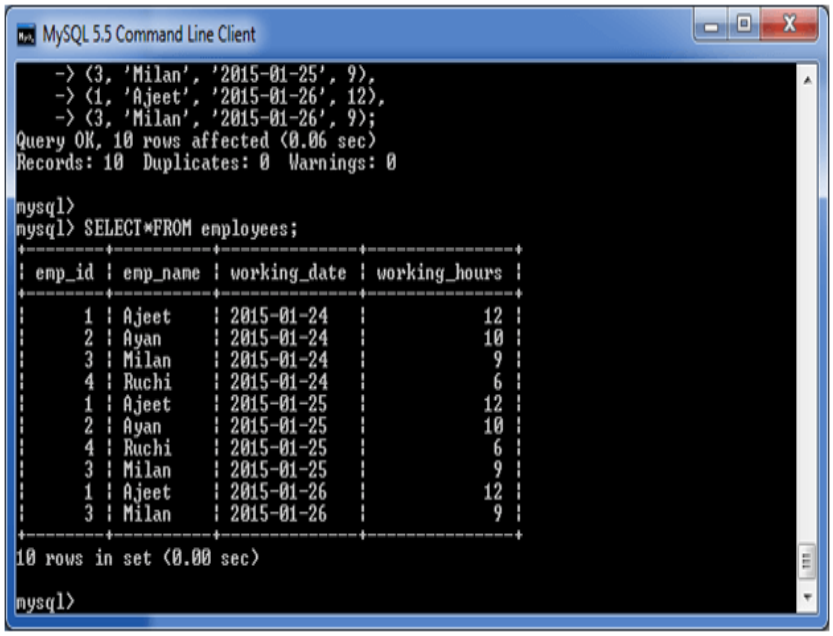
```
+-----------+---------+
| count(id) | name    |
+-----------+---------+
|         1 | Sapna   |
|         2 | Anjana  |
|         1 | Sanjana |
```

```
|        1 | Ajay    |
+-----------+---------+
4 rows in set (0.00 sec)
```

mysql> select count(id),name from employee1 group by name having count(id)>1;

```
+-----------+--------+
| count(id) | name   |
+-----------+--------+
|        2 | Anjana |
+-----------+--------+
1 row in set (0.00 sec)
```

2 rows in set (0.00 sec)



SELECT emp_name, SUM(working_hours) AS "Total working hours"
FROM employees GROUP BY emp_name HAVING SUM(working_hours) > 5;

Limit:

```
mysql> select * from officers limit 3;
+----+---------+----------+---------------+
| id | name    | address  | department    |
+----+---------+----------+---------------+
|  1 | Ajit    | Mau      | Account       |
|  2 | Deepika | Lucknow  | Computer      |
|  3 | Vimal   | Faizabad | Adminstrative |
+----+---------+----------+---------------+
3 rows in set (0.00 sec)
```

```
select officer_name from officers order by officer_name limit 2;
+--------------+
| officer_name |
+--------------+
| Ajeet        |
| Ajeet        |
+--------------+
```

Like : In MySQL, LIKE condition is used to perform pattern matching to find the correct result. It is used in SELECT, INSERT, UPDATE and DELETE statement with the combination of WHERE clause.

```
mysql> select name from officers where name like 'A%';
+------+
| name |
+------+
| Ajit |
+------+
1 row in set (0.00 sec)
```

1) Using % (percent) Wildcard:

```
mysql> select * from officers where address like '%un%';
+------------+--------------+---------+
| officer_id | officer_name | address |
+------------+--------------+---------+
|          5 | Ajay         | pune    |
|          6 | jay          | Dun     |
+------------+--------------+---------+
2 rows in set (0.00 sec)
```

```
SELECT officer_name  FROM officers WHERE address LIKE '%ck%';
select officer_name from officers where officer_name like '%l';
+--------------+
| officer_name |
+--------------+
| Vimal        |
| Rahul        |
+--------------+
2 rows in set (0.00 sec)
```

2) Using _ (Underscore) Wildcard:

```
SELECT officer_name  FROM officers  WHERE address LIKE 'Luc_now';
```

3) Using NOT Operator:

You can also use NOT operator with MySQL LIKE condition.

```
SELECT officer_name  FROM officers  WHERE address NOT LIKE 'Luck%';
```

MySQL IN Condition

The MySQL IN condition is used to reduce the use of multiple OR conditions in a SELECT, INSERT, UPDATE and DELETE statement.

1. expression IN (value1, value2, ... value_n);

```
SELECT *  FROM officers  WHERE officer_name IN ('Ajeet', 'Vimal', 'Deepika');
```

2. Is Null:

```
SELECT *  FROM officers  WHERE officer_name IS NULL;
```

```
insert into officers(id,address)values(6,"Delhi");
Query OK, 1 row affected (0.01 sec)

mysql> SELECT *  FROM officers;
+----+---------+----------+
| id | name    | address  |
+----+---------+----------+
|  1 | Ajit    | Mau      |
|  2 | Deepika | Lucknow  |
|  3 | Vimal   | Faizabad |
|  4 | Rahul   | Lucknow  |
|  5 | Bharat  | Lucknow  |
|  6 | NULL    | Delhi    |
+----+---------+----------+
6 rows in set (0.00 sec)

mysql> SELECT *  FROM officers where name is null;
+----+------+---------+
| id | name | address |
+----+------+---------+
|  6 | NULL | Delhi   |
+----+------+---------+
1 row in set (0.00 sec)
```

mysql>

3. **Is not null:**

SELECT * FROM officers WHERE officer_name IS NOT NULL;

Foreign key: A foreign is a field in one table,that refers to the primary key in another table.

Create two tables custormers and orders;

create table custormers(cid int auto_increment primary key,cname varchar(30),email varchar(50));

desc custormers;

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| cid   | int(11)     | NO   | PRI | NULL    | auto_increment |
| cname | varchar(30) | YES  |     | NULL    |                |
| email | varchar(50) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

create table orders(oid int auto_increment primary key, orderdate date, cid int, amount int, foreign key(cid) references custormers(cid));

desc orders;

```
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| oid       | int(11) | NO   | PRI | NULL    | auto_increment |
| orderdate | date    | YES  |     | NULL    |                |
| cid       | int(11) | YES  | MUL | NULL    |                |
```

```
| amount    | int(11) | YES  |     | NULL    |               |
+-----------+---------+------+-----+---------+---------------+
```

```
select * from custormers;
+-----+--------+-------------+
| cid | cname  | email       |
+-----+--------+-------------+
|   1 | Sanjay | s@gmail.com |
|   2 | Kajal  | k@gmail.com |
|   3 | Anjali | A@gmail.com |
|   4 | Sam    | s@gmail.com |
|   5 | Dev    | d@gmail.com |
+-----+--------+-------------+
5 rows in set (0.00 sec)+-----+--------+-------------+
```

```
mysql> select * from orders;
+-----+------------+------+--------+
| oid | orderdate  | cid  | amount |
+-----+------------+------+--------+
|   1 | 2019-04-21 |    1 |    700 |
|   2 | 2018-03-21 |    3 |   1000 |
|   3 | 2018-03-24 |    2 |   2000 |
|   4 | 2020-07-02 |    1 |   3000 |
+-----+------------+------+--------+
mysql>
```

using concept foreign key:

select * from custormers c , orders o where c.cid=o.cid;

```
+-----+--------+-------------+-----+------------+------+--------+
| cid | cname  | email       | oid | orderdate  | cid  | amount |
+-----+--------+-------------+-----+------------+------+--------+
|   1 | Sanjay | s@gmail.com |   1 | 2019-04-21 |    1 |    700 |
|   3 | Anjali | A@gmail.com |   2 | 2018-03-21 |    3 |   1000 |
|   2 | Kajal  | k@gmail.com |   3 | 2018-03-24 |    2 |   2000 |
|   1 | Sanjay | s@gmail.com |   4 | 2020-07-02 |    1 |   3000 |
+-----+--------+-------------+-----+------------+------+--------+
4 rows in set (0.00 sec)
```

Note: same thing we can find using inner join or join clause.

Join in mysql:

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

1) Inner join 2)outer join(left and right) and full outer join 3)cross join 4)self join

Inner join: The inner join keywords select records that have matching value in both tables.

select * from custormers c join orders o on c.cid=o.cid;

```
+-----+--------+-------------+-----+------------+------+--------+
| cid | cname  | email       | oid | orderdate  | cid  | amount |
+-----+--------+-------------+-----+------------+------+--------+
|   1 | Sanjay | s@gmail.com |   1 | 2019-04-21 |    1 |    700 |
|   3 | Anjali | A@gmail.com |   2 | 2018-03-21 |    3 |   1000 |
|   2 | Kajal  | k@gmail.com |   3 | 2018-03-24 |    2 |   2000 |
|   1 | Sanjay | s@gmail.com |   4 | 2020-07-02 |    1 |   3000 |
+-----+--------+-------------+-----+------------+------+--------+
4 rows in set (0.00 sec)
```

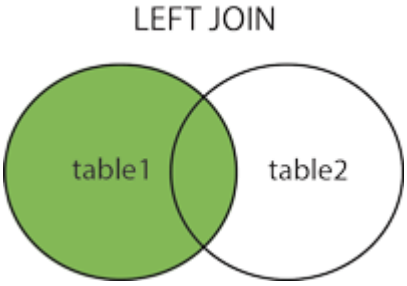
mysql> select cname,orderdate,amount from custormers inner join orders on custormers.cid=orders.cid;

```
+--------+------------+--------+
| cname  | orderdate  | amount |
+--------+------------+--------+
| Sanjay | 2019-04-21 |    700 |
| Anjali | 2018-03-21 |   1000 |
| Kajal  | 2018-03-24 |   2000 |
| Sanjay | 2020-07-02 |   3000 |
+--------+------------+--------+
4 rows in set (0.00 sec)
```

Outer join:

MySQL Outer JOINs return all records matching from both tables .

1) **Left outer join:** The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. Where no matches have been found in the table on the right, NULL is returned.



select all customers, and any orders they might have:

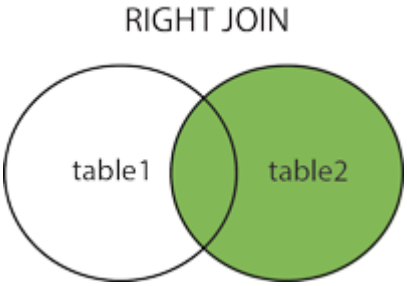
mysql> select * from custormers c left join orders o on c.cid=o.cid;

```
+-----+--------+-------------+------+------------+------+--------+
| cid | cname  | email       | oid  | orderdate  | cid  | amount |
+-----+--------+-------------+------+------------+------+--------+
```

```
|   1 | Sanjay | s@gmail.com |    1 | 2019-04-21 |    1 |    700 |
|   3 | Anjali | A@gmail.com |    2 | 2018-03-21 |    3 |   1000 |
|   2 | Kajal  | k@gmail.com |    3 | 2018-03-24 |    2 |   2000 |
|   1 | Sanjay | s@gmail.com |    4 | 2020-07-02 |    1 |   3000 |
|   4 | Sam    | s@gmail.com | NULL | NULL       | NULL |   NULL |
|   5 | Dev    | d@gmail.com | NULL | NULL       | NULL |   NULL |
+-----+--------+-------------+------+------------+------+--------+
6 rows in set (0.00 sec)
```

Right join:

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).




```
select * from custormers c right join orders o on c.cid=o.cid;
+------+--------+-------------+-----+------------+------+--------+
| cid  | cname  | email       | oid | orderdate  | cid  | amount |
+------+--------+-------------+-----+------------+------+--------+
|    1 | Sanjay | s@gmail.com |   1 | 2019-04-21 |    1 |    700 |
|    3 | Anjali | A@gmail.com |   2 | 2018-03-21 |    3 |   1000 |
2|     | Kajal  | k@gmail.com |   3 | 2018-03-24 |    2 |   2000 |
|    1 | Sanjay | s@gmail.com |   4 | 2020-07-02 |    1 |   3000 |
+------+--------+-------------+-----+------------+------+--------+

insert into orders(orderdate,amount)values("2022-04-02",60);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from custormers c right join orders o on c.cid=o.cid;
+------+--------+-------------+-----+------------+------+--------+
| cid  | cname  | email       | oid | orderdate  | cid  | amount |
+------+--------+-------------+-----+------------+------+--------+
|    1 | Sanjay | s@gmail.com |   1 | 2019-04-21 |    1 |    700 |
|    3 | Anjali | A@gmail.com |   2 | 2018-03-21 |    3 |   1000 |
|    2 | Kajal  | k@gmail.com |   3 | 2018-03-24 |    2 |   2000 |
|    1 | Sanjay | s@gmail.com |   4 | 2020-07-02 |    1 |   3000 |
| NULL | NULL   | NULL        |   5 | 2022-04-02 | NULL |     60 |
+------+--------+-------------+-----+------------+------+--------+
```

Full outer join:

The full outer join doesn't exist in MySQL, so you combine a `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` with the `UNION` operator.

```
mysql> (select * from custormers c left join orders o on c.cid=o.cid) union (select *
from custormers c right join orders o on c.cid=o.cid);
+------+--------+-------------+------+------------+------+--------+
| cid  | cname  | email       | oid  | orderdate  | cid  | amount |
+------+--------+-------------+------+------------+------+--------+
|    1 | Sanjay | s@gmail.com |    1 | 2019-04-21 |    1 |    700 |
|    3 | Anjali | A@gmail.com |    2 | 2018-03-21 |    3 |   1000 |
|    2 | Kajal  | k@gmail.com |    3 | 2018-03-24 |    2 |   2000 |
|    1 | Sanjay | s@gmail.com |    4 | 2020-07-02 |    1 |   3000 |
|    4 | Sam    | s@gmail.com | NULL | NULL       | NULL |   NULL |
|    5 | Dev    | d@gmail.com | NULL | NULL       | NULL |   NULL |
| NULL | NULL   | NULL        |    5 | 2022-04-02 | NULL |     60 |
+------+--------+-------------+------+------------+------+--------+
7 rows in set (0.04 sec)
```

Self join: A self join is a regular join, but the table is joined with itself.

Select column(names) from table t1,table t2 where condition;

Find customers that are from the same city:

```
alter table custormers add column city varchar(20);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> update custormers set city="pune" where cid=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update custormers set city="Delhi" where cid=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update custormers set city="Lucknow" where cid=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update custormers set city="pune" where cid=4;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update custormers set city="lucknow" where cid=5;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from custormers;
+-----+--------+-------------+---------+
| cid | cname  | email       | city    |
+-----+--------+-------------+---------+
|   1 | Sanjay | s@gmail.com | pune    |
|   2 | Kajal  | k@gmail.com | Delhi   |
|   3 | Anjali | A@gmail.com | Lucknow |
|   4 | Sam    | s@gmail.com | pune    |
|   5 | Dev    | d@gmail.com | lucknow |
```

```
+-----+--------+-------------+---------+
5 rows in set (0.00 sec)
```

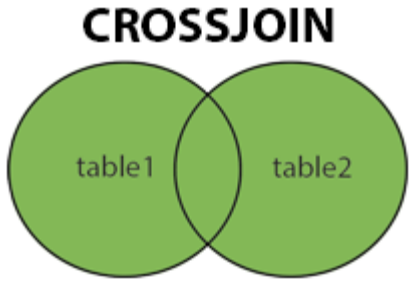
Find customers that are from the same city:

select t1.cname as cname1,t1.city from custormers t1,custormers t2 where t1.cid<>t2.cid and t1.city=t2.city order by city;

```
+--------+--------+---------+
| cname1 | cname2 | city    |
+--------+--------+---------+
| Anjali | Dev    | Lucknow |
| Dev    | Anjali | lucknow |
| Sanjay | Sam    | pune    |
| Sam    | Sanjay | pune    |
+--------+--------+---------+
4 rows in set (0.00 sec)
```

Cross join:

The cross-join keyword returns all records from both tables (table1 and table2).



Select custormers.cname ,orders.oid from custormers cross join orders;

```
+--------+-----+
| cname  | oid |
+--------+-----+
| Sanjay |   5 |
```

| Kajal | 5 |
| Anjali | 5 |
| Sam | 5 |
| Dev | 5 |
| Sanjay | 1 |
| Kajal | 1 |
| Anjali | 1 |
| Sam | 1 |
| Dev | 1 |
| Sanjay | 4 |
| Kajal | 4 |
| Anjali | 4 |
| Sam | 4 |
| Dev | 4 |
| Sanjay | 3 |
| Kajal | 3 |
| Anjali | 3 |
| Sam | 3 |
| Dev | 3 |
| Sanjay | 2 |
| Kajal | 2 |
| Anjali | 2 |
| Sam | 2 |
| Dev | 2 |

Note: The `CROSS JOIN` keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

If you add a `WHERE` clause (if table1 and table2 has a relationship), the `CROSS JOIN` will produce the same result as the `INNER JOIN` clause:

```
SELECT Custormers.cname,orders.oid
   -> FROM Custormers
   -> CROSS JOIN Orders
   -> WHERE Custormers.cid=Orders.cid;
+--------+-----+
| cname  | oid |
+--------+-----+
| Sanjay |   1 |
| Sanjay |   4 |
| Kajal  |   3 |
| Anjali |   2 |
+--------+-----+
4 rows in set (0.00 sec)
```

Extra practice

SQL Inner Join in action

Let's try to understand the concept of Inner Join through an interesting data sample that deals with a Pizza Company and its food distribution. I am going to create two tables first – table 'PizzaCompany' that manages different branches of Pizza outlets in a few cities and table 'Foods' that stores food distribution details across these companies. You can execute the code below to create and populate data into these two tables. All this data is hypothetical and you can create in any of your existing databases.

```
CREATE TABLE [dbo].[PizzaCompany]
(
 [CompanyId] [int] IDENTITY(1,1) PRIMARY KEY CLUSTERED,
 [CompanyName] [varchar](50) ,
 [CompanyCity] [varchar](30)
)
SET IDENTITY_INSERT [dbo].[PizzaCompany] ON;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(1,'Dominos','Los Angeles') ;
```

```
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(2,'Pizza Hut','San Francisco')
;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(3,'Papa johns','San Diego') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(4,'Ah Pizz','Fremont') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(5,'Nino Pizza','Las Vegas') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(6,'Pizzeria','Boston') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VALUES(7,'chuck e cheese','Chicago')
;
```

Data populated in PizzaCompany table.

Results | Messages

	CompanyId	CompanyName	CompanyCity
1	1	Dominos	Los Angeles
2	2	Pizza Hut	San Francisco
3	3	Papa johns	San Diego
4	4	Ah Pizz	Fremont
5	5	Nino Pizza	Las Vegas
6	6	Pizzeria	Boston
7	7	chuck e cheese	Chicago

```
CREATE TABLE [dbo].[Foods]
(
[ItemId] INT  PRIMARY KEY CLUSTERED ,
[ItemName]  Varchar(50),
[UnitsSold] int,
CompanyID int,
FOREIGN KEY(CompanyID) REFERENCES PizzaCompany(CompanyID)
)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(1,'Large Pizza',5,2)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(2,'Garlic Knots',6,3)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(3,'Large Pizza',3,3)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(4,'Medium Pizza',8,4)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(5,'Breadsticks',7,1)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(6,'Medium Pizza',11,1)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(7,'Small Pizza',9,6)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(8,'Small Pizza',6,7)

SELECT * FROM Foods
```

Results | Messages

	ItemId	ItemName	UnitsSold	CompanyID
1	1	Large Pizza	5	2
2	2	Garlic Knots	6	3
3	3	Large Pizza	3	3
4	4	Medium Pizza	8	4
5	5	Breadsticks	7	1
6	6	Medium Pizza	11	1
7	7	Small Pizza	9	6
8	8	Small Pizza	6	7

Inner join:

```
SELECT pz.CompanyCity, pz.CompanyName, pz.CompanyId AS PizzaCompanyId,
f.CompanyID AS FoodsCompanyId, f.ItemName, f.UnitsSold
FROM PizzaCompany pz
INNER JOIN Foods f
ON pz.CompanyId = f.CompanyId
```

	CompanyId	CompanyName	CompanyCity
1	1	Dominos	Los Angeles
2	2	Pizza Hut	San Francisco
3	3	Papa johns	San Diego
4	4	Ah Pizz	Fremont
5	5	Nino Pizza	Las Vegas
6	6	Pizzeria	Boston
7	7	chuck e cheese	Chicago

Table PizzaCompany

Results | Messages

	ItemId	ItemName	Units
1	1	Large Pizza	5
2	2	Garlic Knots	6
3	3	Large Pizza	3
4	4	Medium Pizza	8
5	5	Breadsticks	7
6	6	Medium Pizza	11
7	7	Small Pizza	9
8	8	Small Pizza	6

Table Foods

Inner join result set of the tables without CompanyId = 5
(Unmatched row)

Results | Messages

	CompanyCity	CompanyName	PizzaCompanyId	FoodsCompanyId	ItemName
1	San Francisco	Pizza Hut	2	2	Large Pizza
2	San Diego	Papa johns	3	3	Garlic Knots
3	San Diego	Papa johns	3	3	Large Pizza
4	Fremont	Ah Pizz	4	4	Medium Pizza
5	Los Angeles	Dominos	1	1	Breadsticks
6	Los Angeles	Dominos	1	1	Medium Pizza
7	Boston	Pizzeria	6	6	Small Pizza
8	Chicago	chuck e cheese	7	7	Small Pizza