**Lab 01**

**OBJECTIVE:** Illustration of SELECT, FROM, WHERE CLAUSE

**SYNTAX:**

SELECT column1, column2, ……………….columnn

FROM <table\_name>

WHERE <condition>

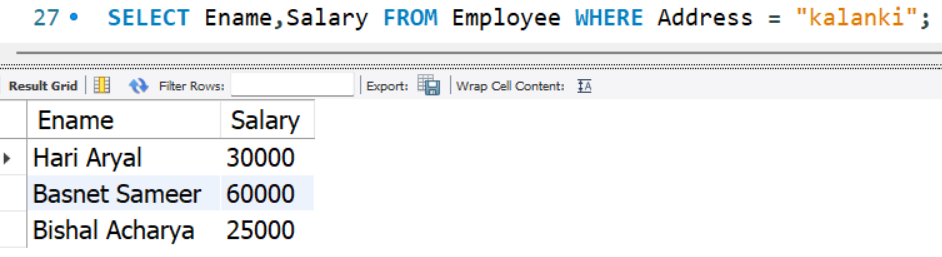
**QUERY**

SELECT Ename, Salary

FROM employee

WHERE Address = ‘Banasthali’;

**OUTPUT:**

****

**CONCLUSION:**

Hence, the SELECT, FROM, WHERE clause was executed in MYSQL.

**Lab 02**

**OBJECTIVE:** Illustration of SELECTING OF ALL COLUMNS

**SYNTAX:**

SELECT \*

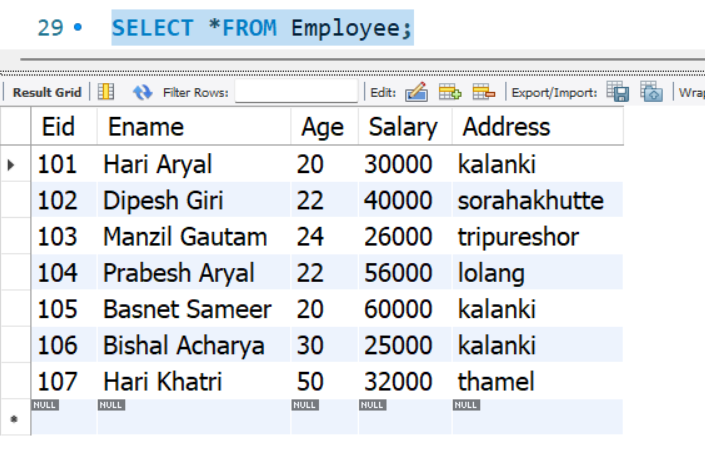
FROM <table\_name>

**QUERY**

SELECT \*

FROM employee;

**OUTPUT:**

****

**CONCLUSION:**

Hence, all the columns were selected using SELECT \* command in MYSQL.

**Lab 03**

**OBJECTIVE:** Illustration of SELECTING OF SPECIFIC COLUMNS.

**SYNTAX:**

SELECT column1, column2, …………, columnn

FROM <table\_name>

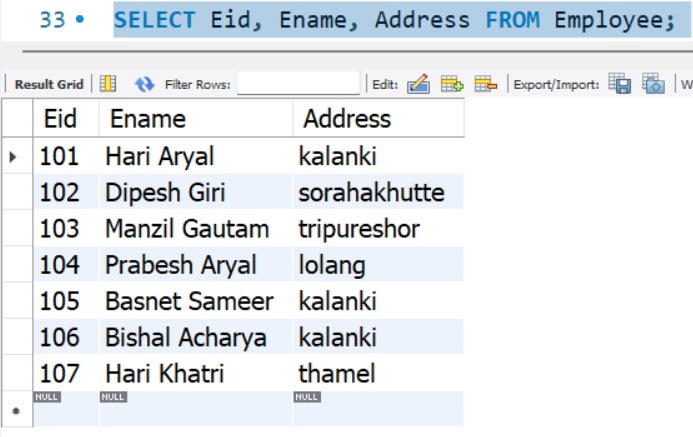
WHERE <condition>(optional)

**QUERY**

SELECT Eid, Ename, Address

FROM employee;

**OUTPUT:**

****

**CONCLUSION:**

Hence, specific columns were selected using MYSQL SELECT command.

**Lab 04**

**OBJECTIVE:** Illustration of ORDER BY clause

**SYNTAX:**

SELECT column1, column2, ……………….columnn

FROM <table\_name>

ORDERY BY column1[asc|desc] column2[asc|desc]…;

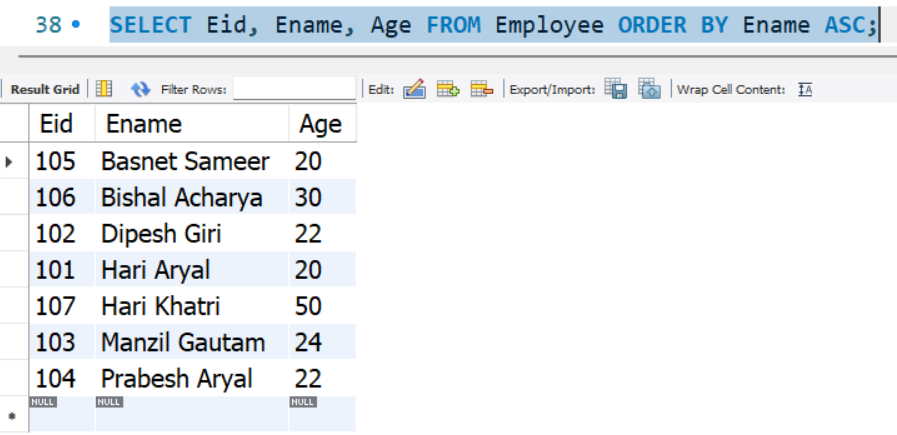
**QUERY1**

SELECT Eid, Ename, Age

FROM employee

ORDER BY Ename ASC;

**OUTPUT1:**

****

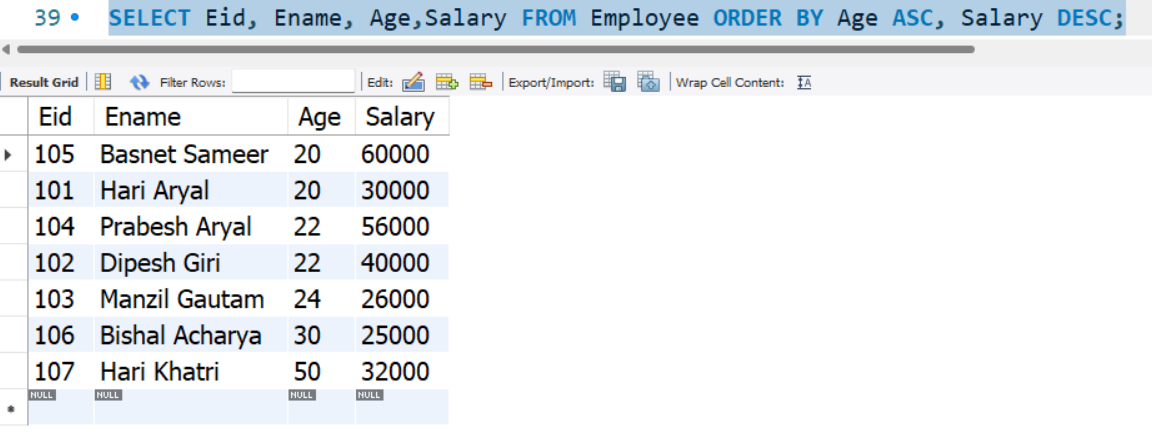
**QUERY2**

SELECT Eid, Ename, Age, Salary

FROM Employee

ORDER BY Age ASC, Salary DESC;

**OUTPUT2:**

****

**CONCLUSION:**

Hence, the working of ORDER BY clause was shown in MYSQL by sequencing the tuples in decreasing order of name.

**Lab 05**

**OBJECTIVE:** Illustration of Arithmetic Operators

**SYNTAX:**

SELECT <operand> OPERATOR (+, -, \*, /, %) <operand>

**QUERY:**

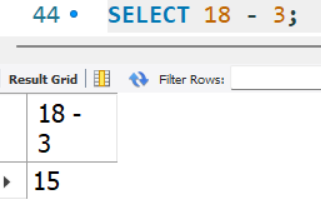
SELECT 17 + 5;

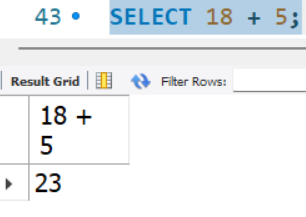
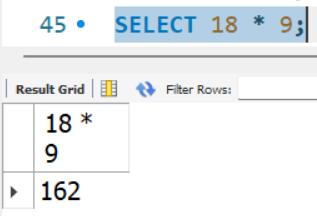
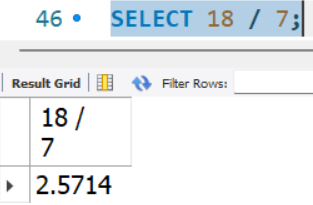
SELECT 17 - 5;

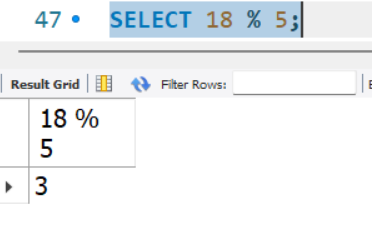
SELECT 17 \* 5;

SELECT 17 / 5;

SELECT 17 % 5;

**OUTPUT:**

****

****

**CONCLUSION:**

Hence, the arithmetic operators were shown to work in MYSQL command line terminal.

**Lab 06**

**OBJECTIVE:** Illustration of Operator Precedence in Arithmetic expression

**THEORY:**

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

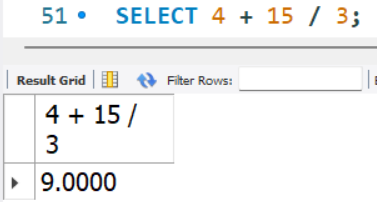
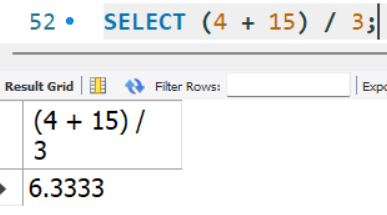
1. !
2. (unary minus), ~ (unary bit inversion)
3. ^
4. \*, /, DIV, %, MOD
5. -, +
6. <<, >>
7. &
8. |
9. = (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN, MEMBER OF
10. BETWEEN, CASE, WHEN, THEN, ELSE
11. NOT
12. AND, &&
13. XOR
14. OR, ||
15. = (assignment), :=

The precedence of = depends on whether it is used as a comparison operator ([=](https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html#operator_equal)) or as an assignment operator ([=](https://dev.mysql.com/doc/refman/8.0/en/assignment-operators.html#operator_assign-equal)). For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

**QUERY :**

SELECT 4 + 5 \* 4;

SELECT (4 + 5) \* 4;

**OUTPUT:**

**CONCLUSION:**

Hence, the operator precedence of arithmetic expression was shown in MYSQL.

**Lab 07**

**OBJECTIVE:** Illustration of aggregate functions

**SYNTAX:**

SELECT Aggregate\_function([DISTINCT|all]column)

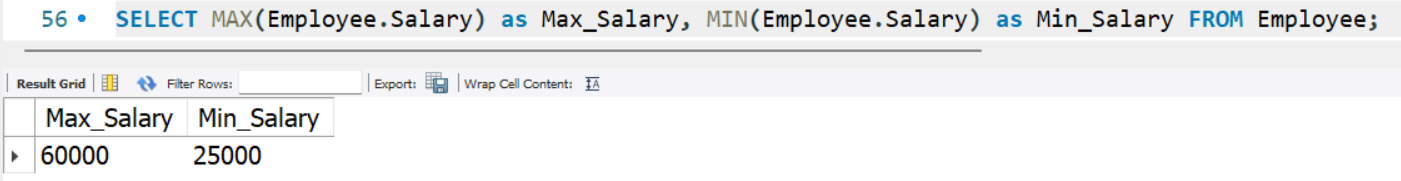
FROM <table name>

WHERE <condition>

**QUERY:**

SELECT MAX(Salary) as Max\_sal,MIN(Salary) as Min\_sal

FROM employee;

**OUTPUT:**

**CONCLUSION:**

Hence, an aggregate function used in MYSQL command.

**Lab 08**

**OBJECTIVE:** Illustration of GROUP BY clause

**SYNTAX:**

SELECT column1, column2,….., column

FROM <table name>

WHERE <condition>

GROUP BY expression1, expression2,…..;

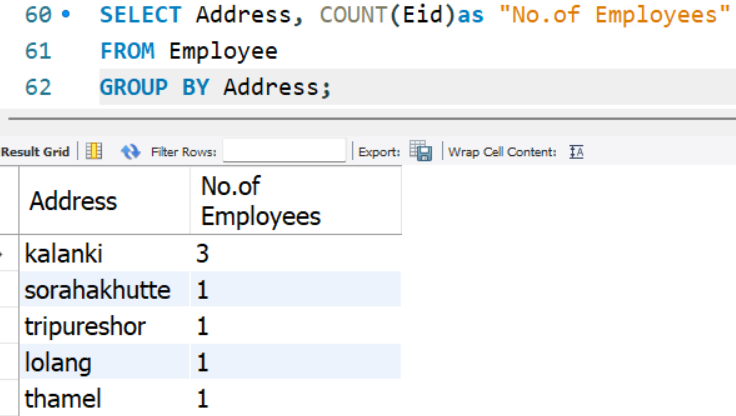
**QUERY**

SELECT Address, COUNT(Eid) as “No. of\_Employees”

FROM Employee

GROUP BY Address;

**OUTPUT:**

****

**CONCLUSION:**

Hence, the use of GROUP BY clause was illustrated in MYSQL.

**Lab 09**

**OBJECTIVE:** Illustration of Restricting Group Results with the HAVING Clause

**SYNTAX:**

SELECT column1, column2,…..,columnn

FROM <table name>

WHERE <condition>

GROUP BY expression1, expression2,…..;

HAVING having\_condition;

**QUERY:**

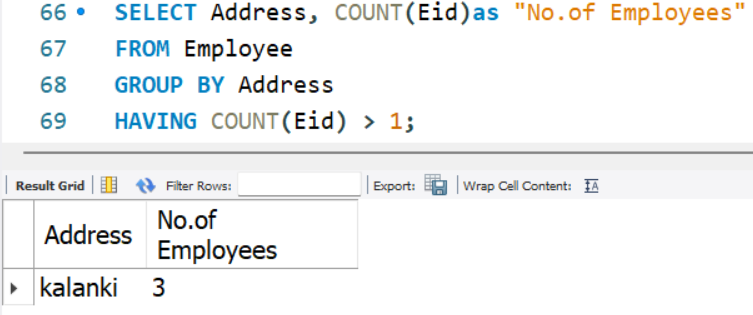
SELECT Address, COUNT(Eid)as "No. of Employees"

FROM Employee

GROUP BY Address

HAVING COUNT(Eid) > 1;

**OUTPUT:**

****

**CONCLUSION:**

Hence, the group result was restricted by introducing HAVING clause condition.

**Lab 10**

**OBJECTIVE:** Illustration of defining a NULL value

**SYNTAX:**

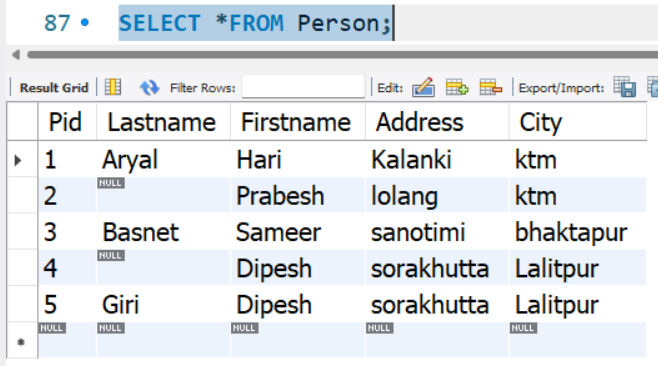
SELECT <column\_name>FROM <table\_name>   
WHERE <column \_name> IS NULL;

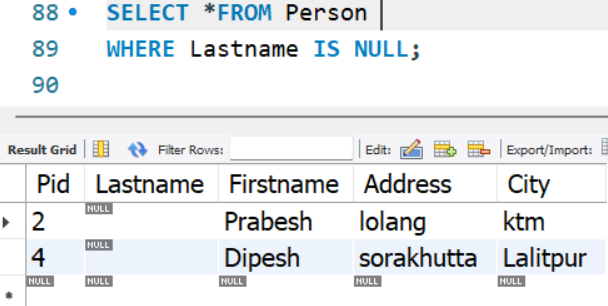
**QUERY:**

SELECT \*FROM Person

WHERE Lastname IS NULL;

**OUTPUT:**

****

****

**CONCLUSION:**

Hence, NULL values were defined and tuples having NULL values were accessed.

**Lab 11**

**OBJECTIVE:** Illustration of using Column Aliases

**SYNTAX:**

SELECT <column\_name> [AS] <column\_alias\_name>FROM <table\_name> [AS] <table\_alias\_name>

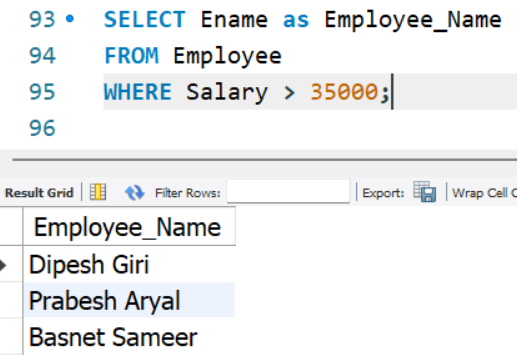
**QUERY**:

SELECT Ename as Employee\_Name

FROM Employee

WHERE Salary >35000;

**OUTPUT:**

****

**CONCLUSION:**

Hence, column can be renamed using aliases via AS keyword.

**Lab 12**

**OBJECTIVE:** Illustration of using Concatenation Operator

**SYNTAX:**

SELECT expression1 || expression2 || …

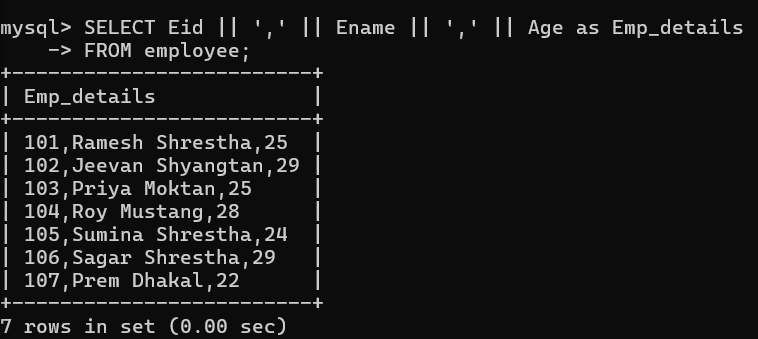
FROM <table\_name>

**QUERY**

SELECT Eid || ',' || Ename || ',' || Age as Emp\_details

FROM employee;

**OUTPUT:**

****

**CONCLUSION:**

Hence, concatenation operator “||” can be used to combine any two expressions.

**Lab 13**

**OBJECTIVE**: Illustration of using Literal Character Strings

**SYNTAX:**

Sequence of characters that are enclosed in single or double string:

<’CHARACTER\_STRING’>

<”CHARACTER\_STRING”>

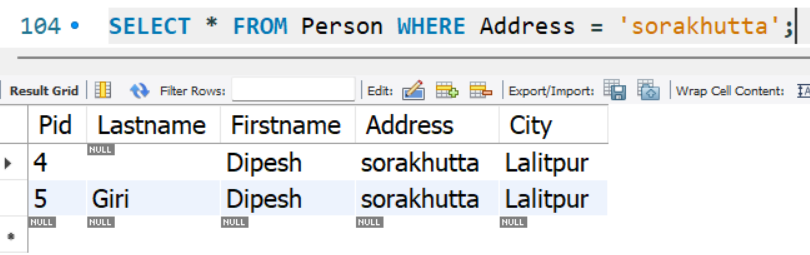
**QUERY**

SELECT \*

FROM Person

WHERE Address = ‘sorakhutta’;

**OUTPUT:**

****

**CONCLUSION:**

Hence, the literal character string was implemented.

**Lab 14**

**OBJECTIVE:** Illustration of Displaying Distinct Rows

**SYNTAX:**

SELECT DISTINCT column1, column2, ……………….columnn

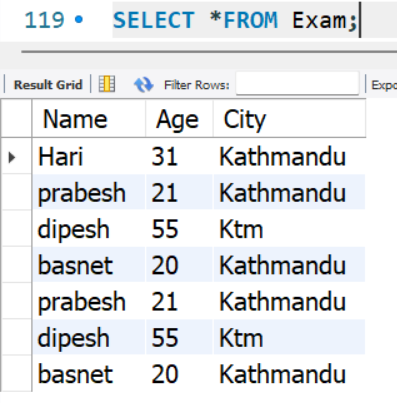
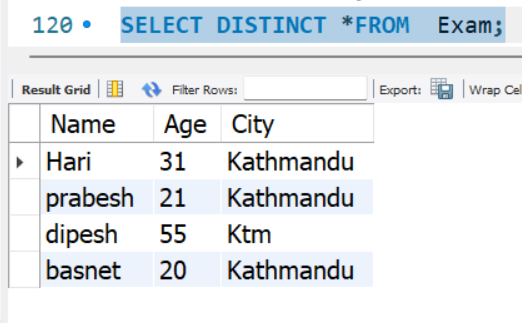
FROM <table name>

WHERE <condition>

**QUERY:**

SELECT DISTINCT \*

FROM Exam;

**OUTPUT:**

**CONCLUSION:**

Hence, only distinct rows were displayed by omitting the duplicate ones.

**Lab 15**

**OBJECTIVE:** Illustration of Displaying Table Structures

**SYNTAX:**

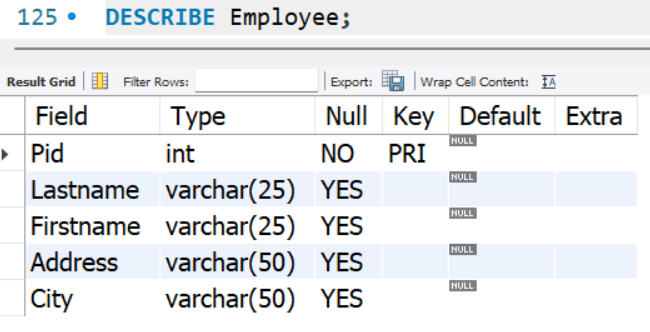
DESCRIBE <table\_name>;

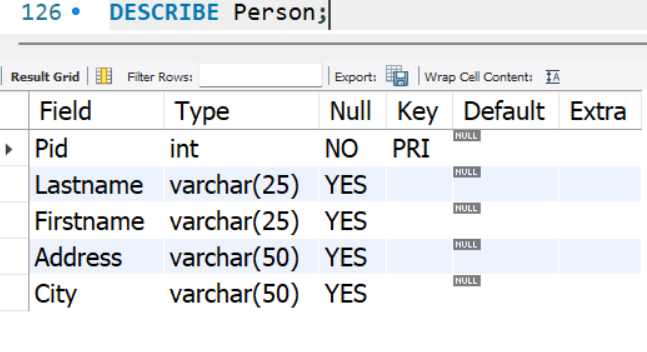
**QUERY**

DESCRIBE employee;

DESCRIBE PERSONS;

**OUTPUT:**

****

****

**CONCLUSION:**

Hence, table structure was displayed using keyword DESCRIBE.

**Lab 16**

**OBJECTIVE:** Illustration of Using BETWEEN operator

**SYNTAX:**

SELECT columns

FROM <table name>

WHERE <column\_name> BETWEEN value1 AND value2;

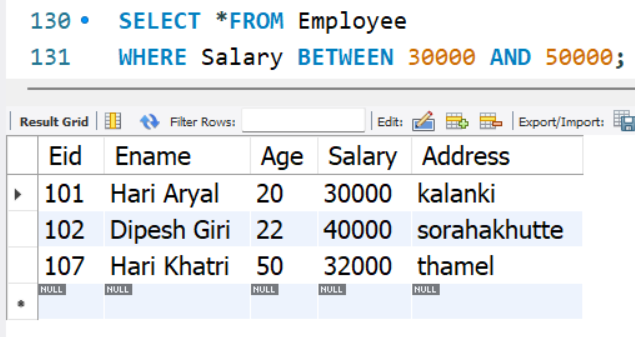
**QUERY**

SELECT \*

FROM employee

WHERE Salary BETWEEN 30000 AND 50000;

**OUTPUT:**

****

**CONCLUSION:**

Hence, the BETWEEN operator was implemented as WHERE condition to find the records between certain two values for an attribute.

**Lab 17**

**OBJECTIVE:** Illustration of Using IN operator

**SYNTAX:**

SELECT columns

FROM <table name>

WHERE <column\_name> IN(value1,value2,…..,valuen);

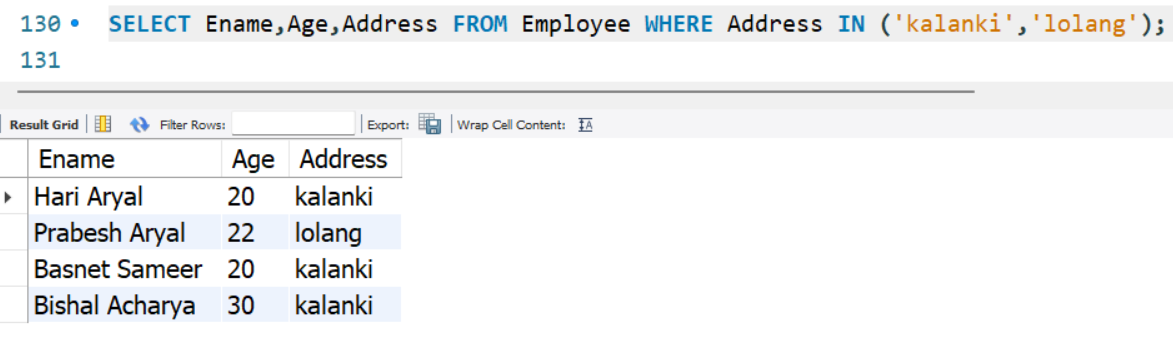
**QUERY**

SELECT Ename, Age, Address

FROM Employee

WHERE Address IN ('kalanki', 'lolang');

**OUTPUT:**

****

**CONCLUSION:**

Hence, IN operator was used as WHERE condition to retrieve only desired records.

**Lab 18**

**OBJECTIVE:** Illustration of Using LIKE operator

**SYNTAX:**

SELECT columns

FROM <table name>

WHERE <column\_name> LIKE pattern;

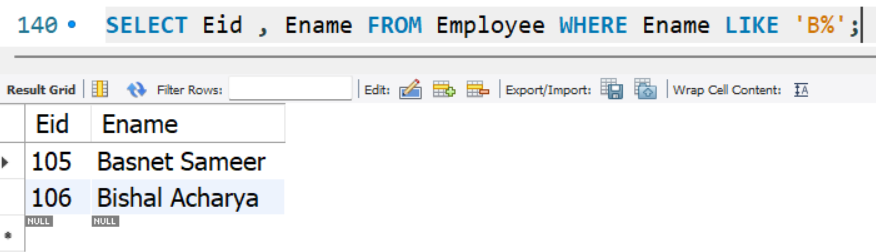
**QUERY**

SELECT Eid, Ename

FROM Employee

WHERE Ename LIKE 'B%';

**OUTPUT:**

****

**CONCLUSION:**

Hence, LIKE operator can be used with WHERE clause to retrieve certain records.

**Lab 19**

**OBJECTIVE:** Illustration of Using AND operator

**SYNTAX:**

SELECT column1,column2,…

FROM <table name>

WHERE condition1 AND condition2 AND …;

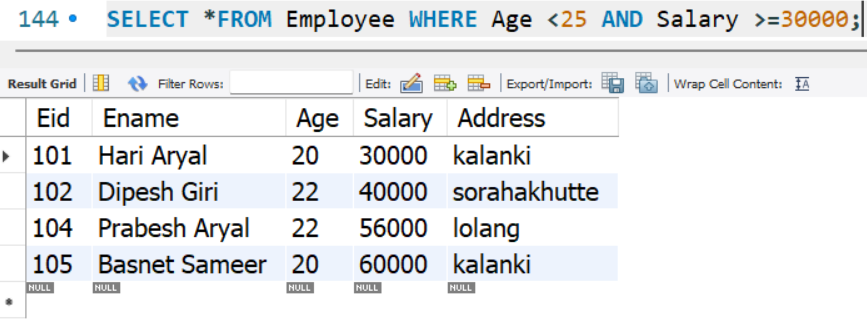
**QUERY**

SELECT \*

FROM Employee

WHERE Age < 25 AND Salary >=30000;

**OUTPUT:**

****

**CONCLUSION:**

Hence, tuples satisfying two or more conditions can be retrieved using AND operator.

**Lab 20**

**OBJECTIVE:** Illustration of Using OR operator

**SYNTAX:**

SELECT column1, column2,….

FROM <table name>

WHERE condition1 OR condition2 OR …;

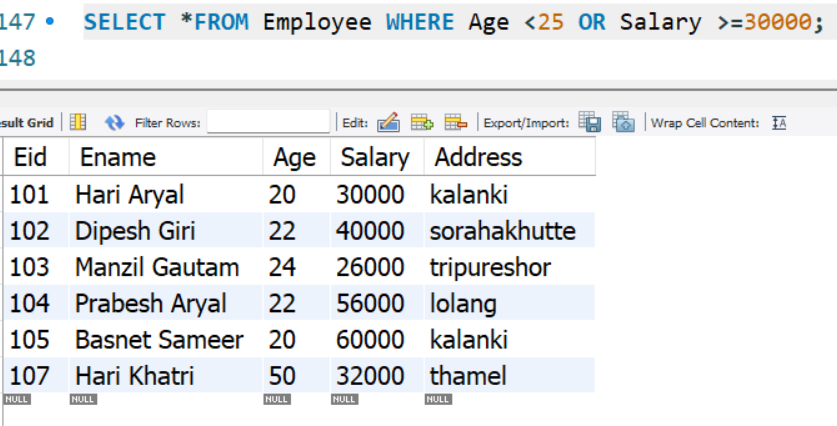
**QUERY**

SELECT \*

FROM employee

WHERE age < 25 OR Salary >=30000;

**OUTPUT:**

****

**CONCLUSION:**

Hence, OR operator was used to retrieve records that matched one of the following given conditions.

**Lab 21**

**OBJECTIVE:** Illustration of Using NOT operator

**SYNTAX:**

SELECT column1, column2,…

FROM <table name>

WHERE NOT condition;

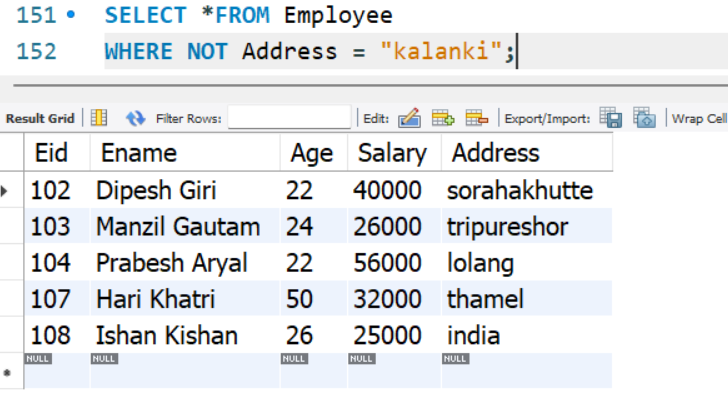
**QUERY**

SELECT \*

FROM employee

WHERE NOT Address = 'Banasthali';

**OUTPUT:**

****

**CONCLUSION:**

Hence, the records that is not required for a certain attribute value can be filtered using NOT operator.

**Lab 22**

**OBJECTIVE:** Illustration of Subquery

**SYNTAX:**

SELECT column1, column2,…

FROM <table name>

WHERE <column\_name> Comparison Operator, Relational Operator ALL| ANY| SOME| IN (SELECT column1, column2,… FROM <table name> WHERE inner\_condition);

**QUERY**

**Single Row Sub Query**

SELECT \*

FROM Employee

WHERE Salary = ( SELECT MAX(Salary)

FROM Employee);

**Multiple Rows Sub Query**

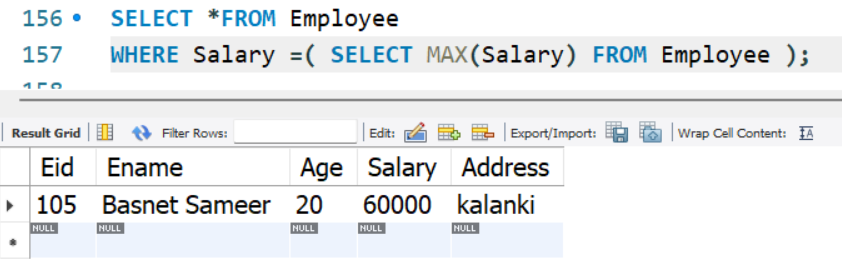
SELECT Ename

FROM employee

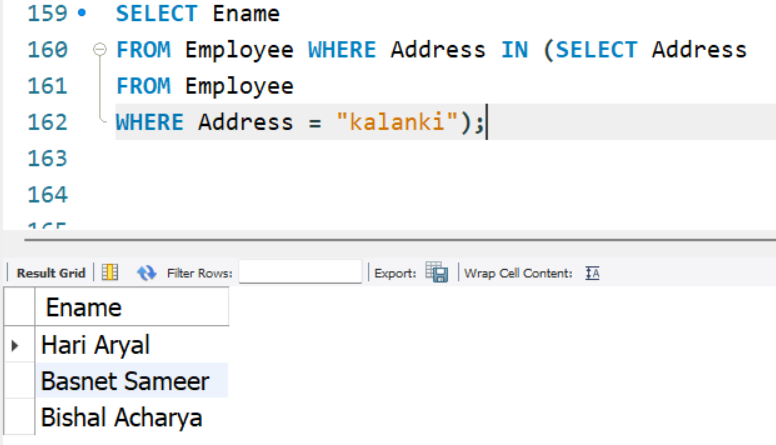
WHERE Address IN (SELECT Address

FROM employee

WHERE Address =”kalanki”);

****

**OUTPUT:**

****

**CONCLUSION:**

Hence, sub queries can be used to retrieve single or multiple records from the given table.

**Lab 23**

**OBJECTIVE:** Illustration of CROSS JOIN

**SYNTAX:**

SELECT column\_name\_list

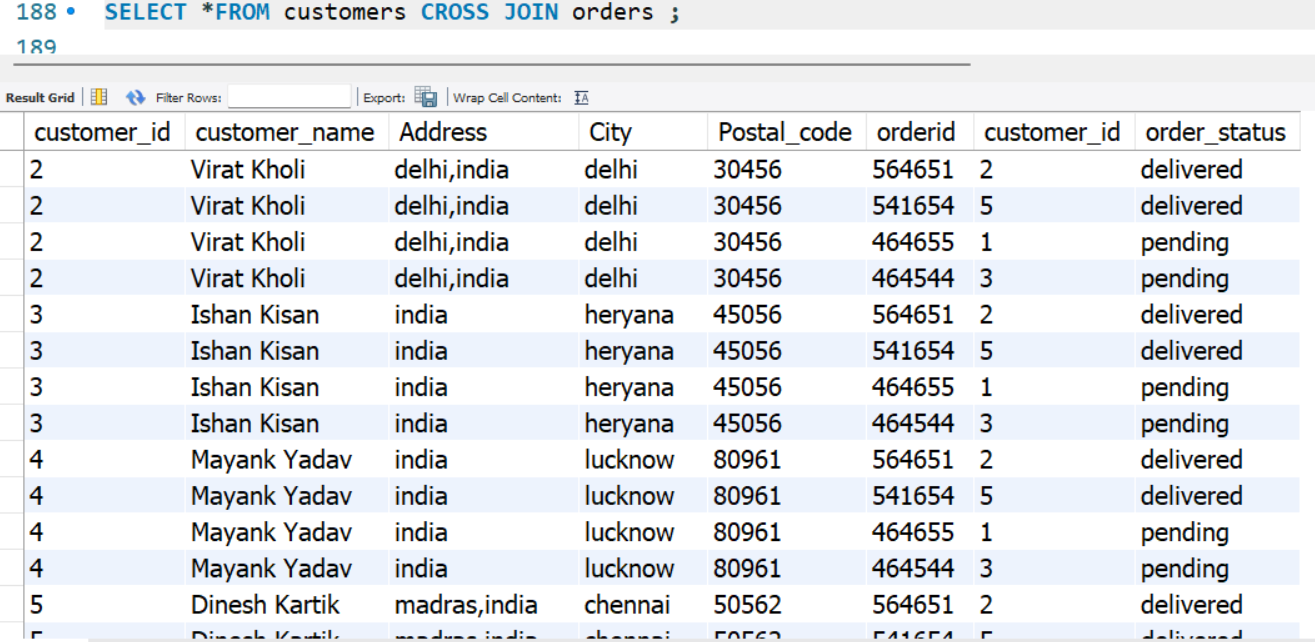
FROM <table\_name1> CROSS JOIN <table\_name2>

**QUERY**

SELECT \*

FROM customers CROSS JOIN orders ;

**OUTPUT:**

****

**CONCLUSION:**

Hence, cross join(cartesian product) was done combining all combinations of tuples.

**Lab 24**

**OBJECTIVE:** Illustration of NATURAL JOIN

**SYNTAX:**

SELECT column\_name\_list

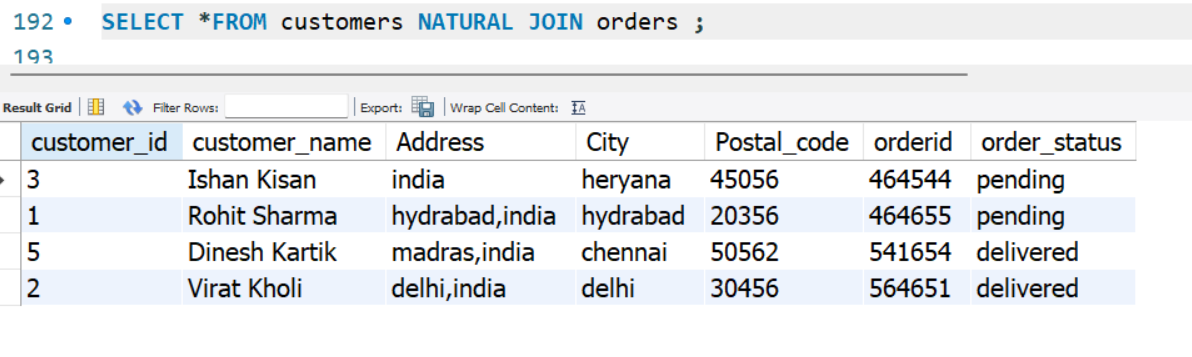
FROM <table\_name1> NATURAL JOIN <table\_name2>

**QUERY**

SELECT DISTINCT \*

FROM customers NATURAL JOIN orders ;

**OUTPUT:**

****

**CONCLUSION:**

Hence, natural join between two tables can be done for tables having a common attribute among the tables.

**Lab 25**

**OBJECTIVE:** Illustration of Creating JOINS with USING Clause

**SYNTAX:**

SELECT column\_name\_list

FROM <table\_name1> INNER JOIN <table\_name2>

USING (common\_column\_name);

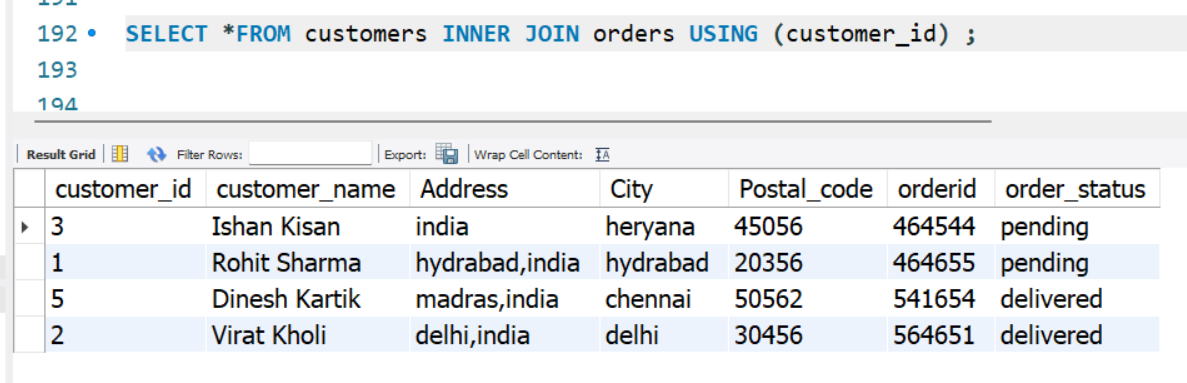
**QUERY**

SELECT Ename,Age,dept\_id,dept\_name

FROM employee INNER JOIN department

USING (Eid);

**OUTPUT:**

****

**CONCLUSION:**

Hence, two tables were joined via USING clause that accepts common attribute.

**Lab 26**

**OBJECTIVE:** Illustration of Creating JOINS with ON Clause

**SYNTAX:**

SELECT column\_name\_list

FROM <table\_name1> INNER JOIN <table\_name2>

ON table1.column = table2.column;

**QUERY**

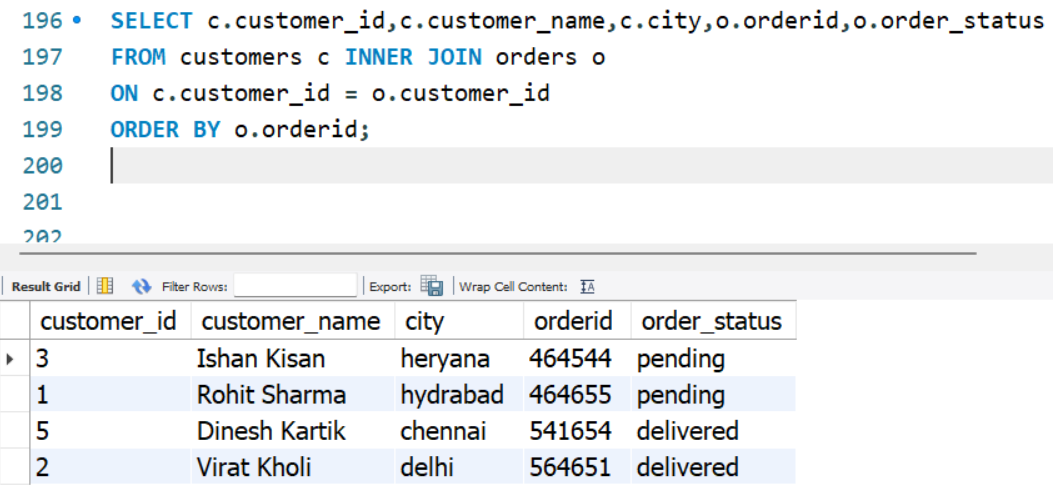
SELECT c.customer\_id, c.customer\_name, c.city, o.orderid, o.order\_status

FROM customers c INNER JOIN orders o

ON c.customer\_id = o.customer\_id

ORDER BY o.orderid;

**OUTPUT:**



**CONCLUSION:**

Hence, two tables can be combined using ON clause.

**Lab 27**

**OBJECTIVE:** Illustration of LEFT OUTER JOIN

**SYNTAX:**

SELECT column\_name\_list

FROM <table\_name1> LEFT OUTER JOIN <table\_name2>

ON table1.column = table2.column;

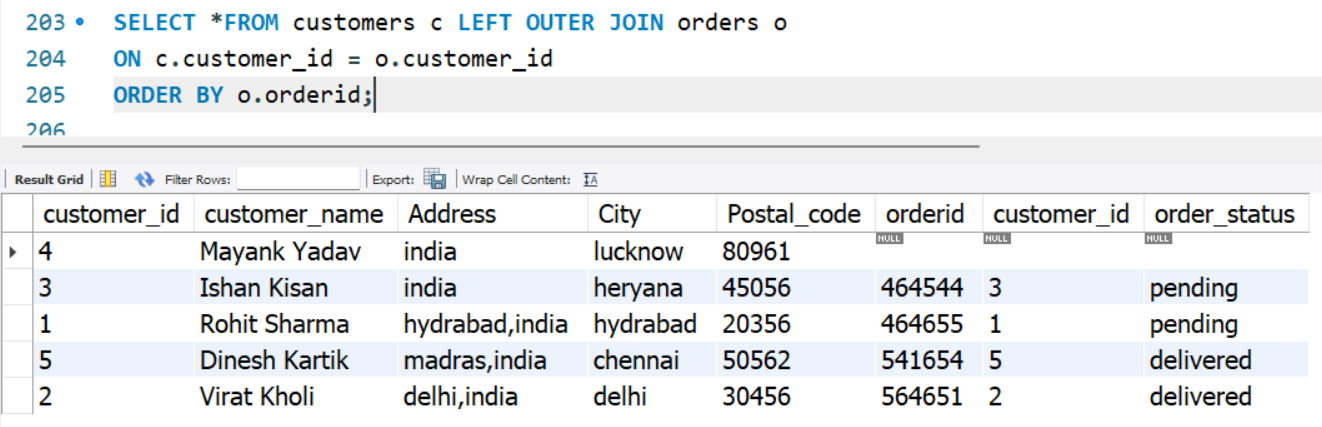
**QUERY**

SELECT \*FROM customers c LEFT OUTER JOIN orders o

ON c.customer\_id = o.customer\_id

ORDER BY o.orderid;

**OUTPUT:**

****

**CONCLUSION:**

Hence, left outer join was implemented in MYSQL that returns all tuples from first table and returns NULL for those values in second table that are not mapped with tuples from first table.

**Lab 28**

**OBJECTIVE:** Illustration of RIGHT OUTER JOIN

**SYNTAX:**

SELECT column\_name\_list

FROM <table\_name1>RIGHT OUTER JOIN <table\_name2>

ON table1.column = table2.column;

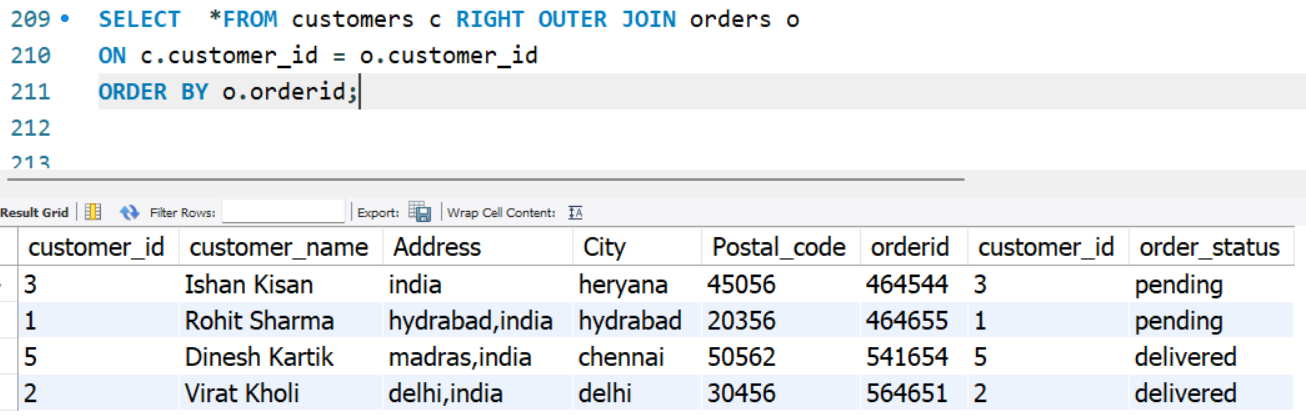
**QUERY**

SELECT \*FROM customers c RIGHT OUTER JOIN orders o

ON c.customer\_id = o.customer\_id

ORDER BY o.orderid;

**OUTPUT:**

****

**CONCLUSION:**

Hence, right outer join was implemented in MYSQL that returns all tuples from second table and returns NULL for those values in second table that are not mapped with tuples from second table.

**Lab 29**

**OBJECTIVE:** Illustration of FULL OUTER JOIN

**SYNTAX**

SELECT column\_name\_list

FROM <table\_name1> FULL OUTER JOIN <table\_name2>

ON table1.column = table2.column;

OR

SELECT column\_name\_list

FROM <table\_name1>LEFT OUTER JOIN <table\_name2>

ON table1.column = table2.column;

UNION

SELECT column\_name\_list

FROM <table\_name1>RIGHT OUTER JOIN <table\_name2>

ON table1.column = table2.column;

**QUERY**

SELECT

\*FROM customers c LEFT OUTER JOIN orders o

ON c.customer\_id = o.customer\_id

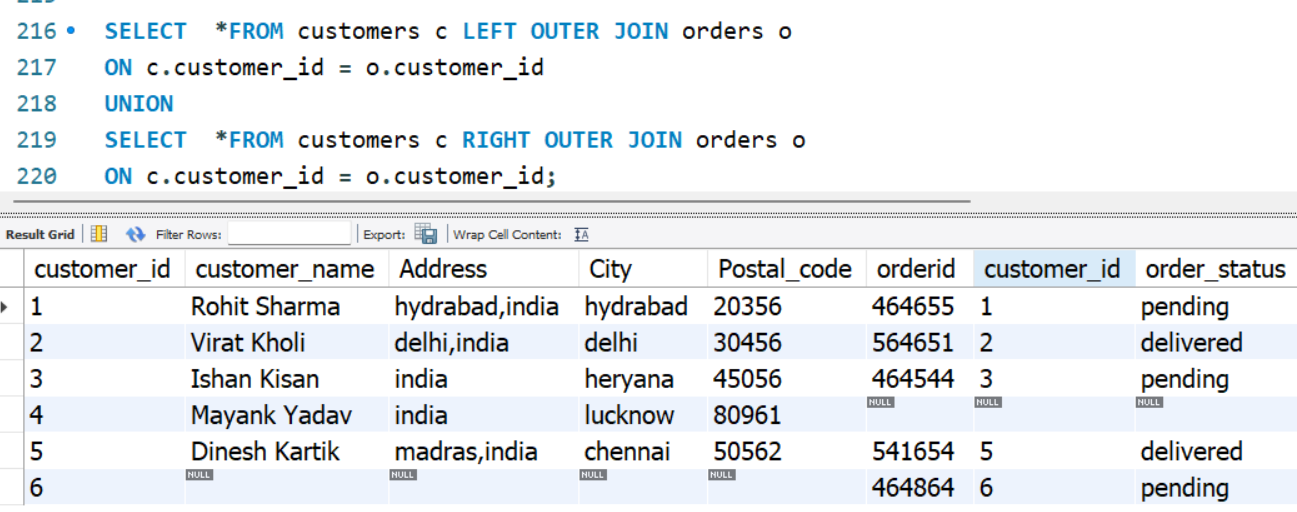
UNION

SELECT \*

FROM customers c RIGHT OUTER JOIN orders o

ON c.customer\_id = o.customer\_id

**OUTPUT:**

****

**CONCLUSION:**

Hence, full outer join was implemented by union of left outer and right outer join in MYSQL.

**Lab 30**

**OBJECTIVE:** Illustration of Creating Table with Enforcement of Integrity Constraints PRIMARY KEY, NOT NULL, UNIQUE, CHECK, REFERENTIAL INTEGRITY.

**SYNTAX:**

CREATE TABLE <table\_name>

(

column1 data\_type(size) CONSTRAINT,

column2 data\_type(size) CONSTRAINT,

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

columnn data\_type(size) CONSTRAINT

);

**QUERY**

CREATE TABLE POST (

Post\_code INT PRIMARY KEY,

Post VARCHAR(20) NOT NULL ,

Email VARCHAR(25) UNIQUE KEY,

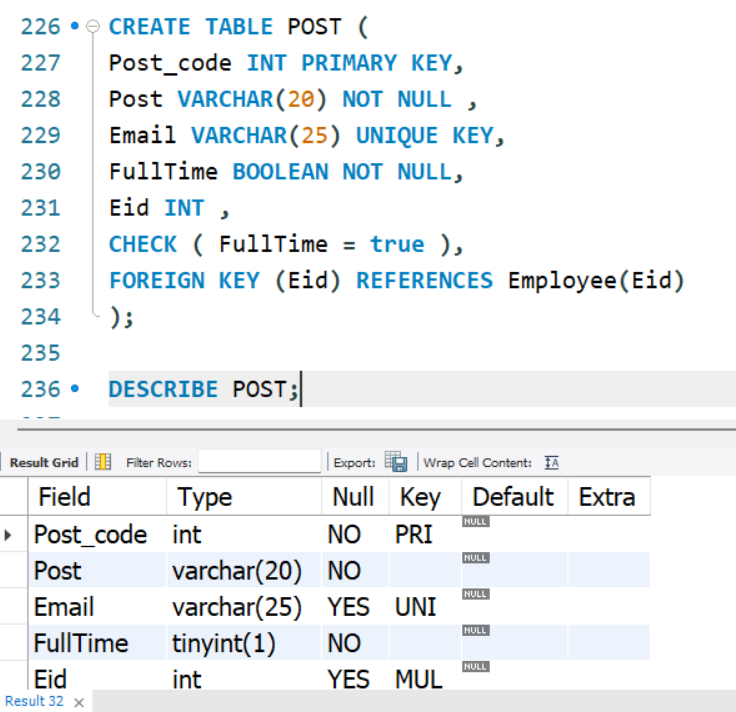
FullTime BOOLEAN NOT NULL,

Eid INT ,

CHECK ( FullTime = true ),

FOREIGN KEY (Eid) REFERENCES Employee(Eid)

);

****

**OUTPUT:**

**CONCLUSION:**

Hence, a table was created with enforcement of various integrity constraints in MYSQL.