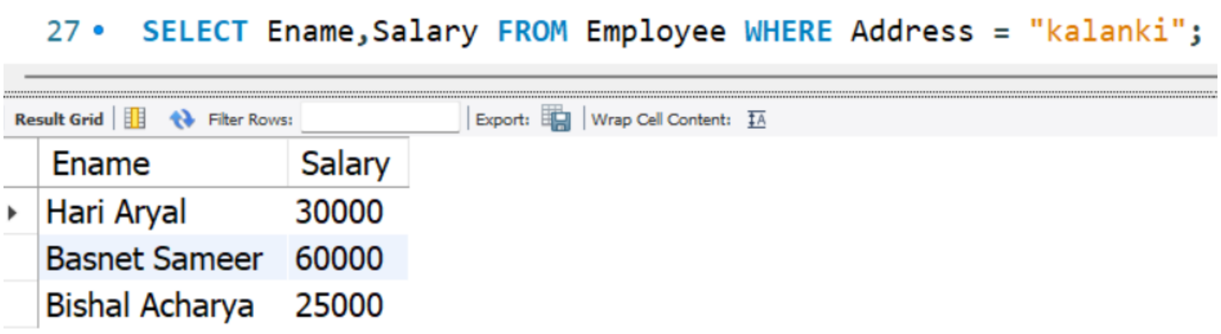
Lab 01

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:



```
SELECT Ename,Salary FROM Employee WHERE Address = "kalanki";
```

Ename	Salary
Hari Aryal	30000
Basnet Sameer	60000
Bishal Acharya	25000

CONCLUSION:

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

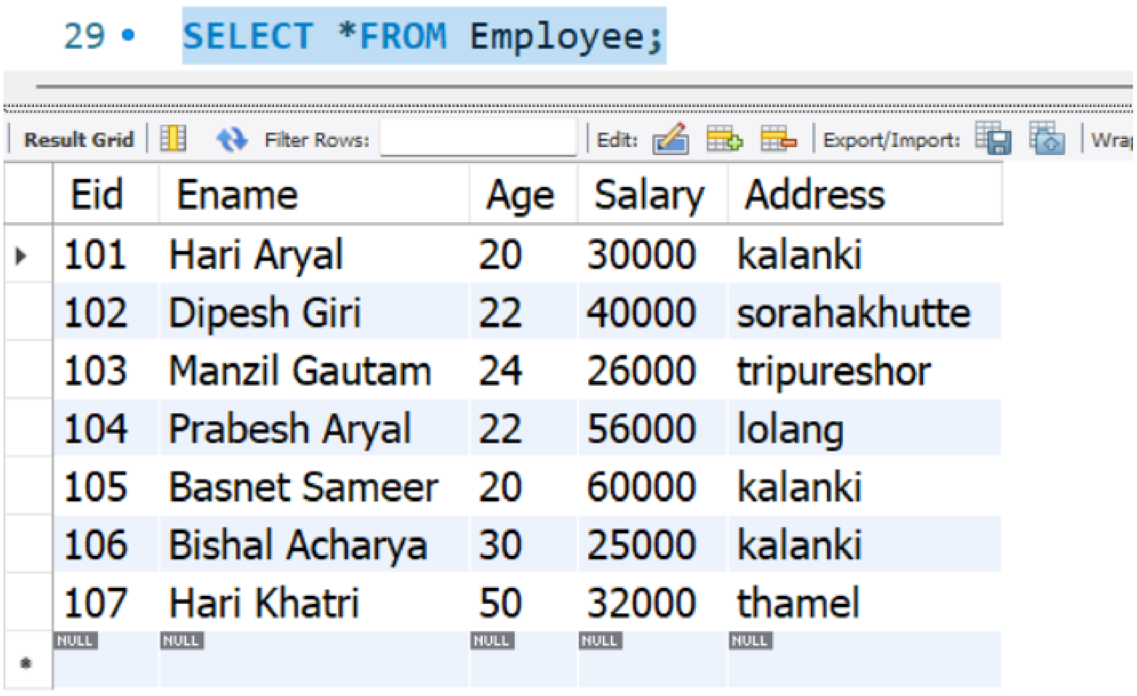
Lab 02

OBJECTIVE: Illustration of SELECTING OF ALL COLUMNS

SYNTAX:
SELECT *
FROM <table_name>

QUERY
SELECT *
FROM employee;

OUTPUT:



```
SELECT *FROM Employee;
```

Eid	Ename	Age	Salary	Address
101	Hari Aryal	20	30000	kalanki
102	Dipesh Giri	22	40000	sorahakhutte
103	Manzil Gautam	24	26000	tripureshor
104	Prabesh Aryal	22	56000	lolang
105	Basnet Sameer	20	60000	kalanki
106	Bishal Acharya	30	25000	kalanki
107	Hari Khatri	50	32000	thamel
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

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

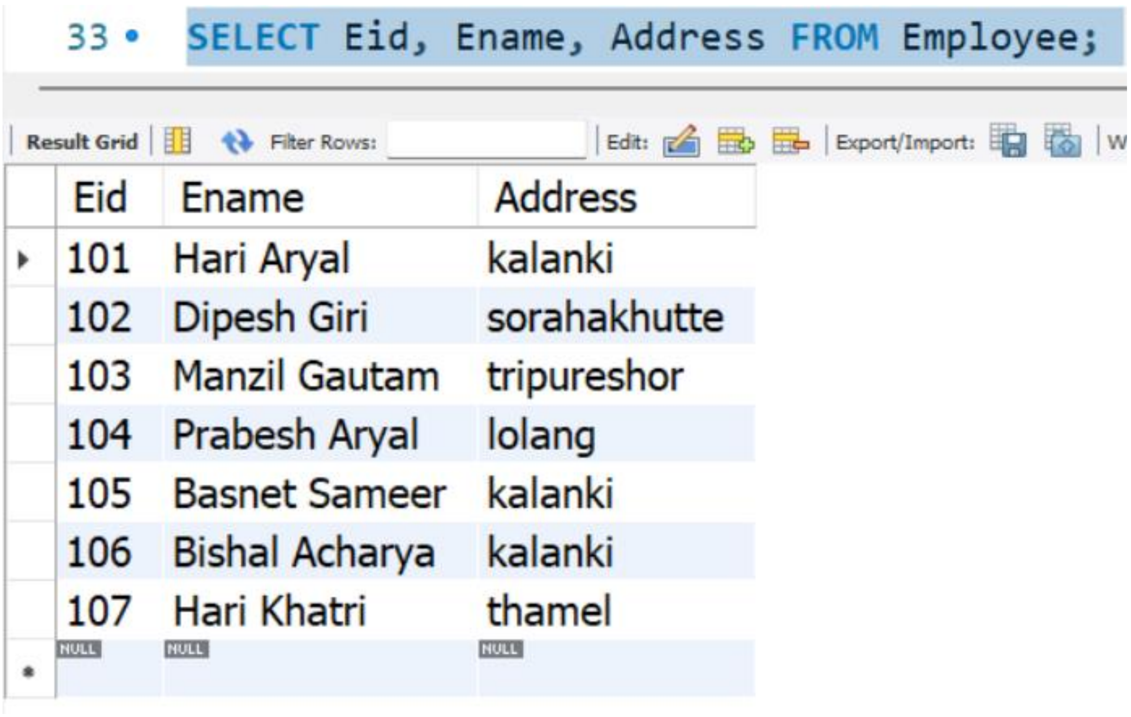
Lab 03

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:



```
SELECT Eid, Ename, Address FROM Employee;
```

Eid	Ename	Address
101	Hari Aryal	kalanki
102	Dipesh Giri	sorahakhutte
103	Manzil Gautam	tripureshor
104	Prabesh Aryal	lolang
105	Basnet Sameer	kalanki
106	Bishal Acharya	kalanki
107	Hari Khatri	thamel
NULL	NULL	NULL

CONCLUSION:

Hence, specific columns were selected using MYSQL SELECT command.

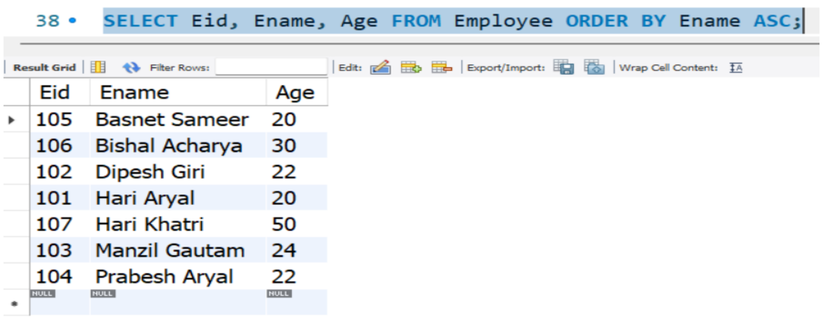
Lab 04

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:

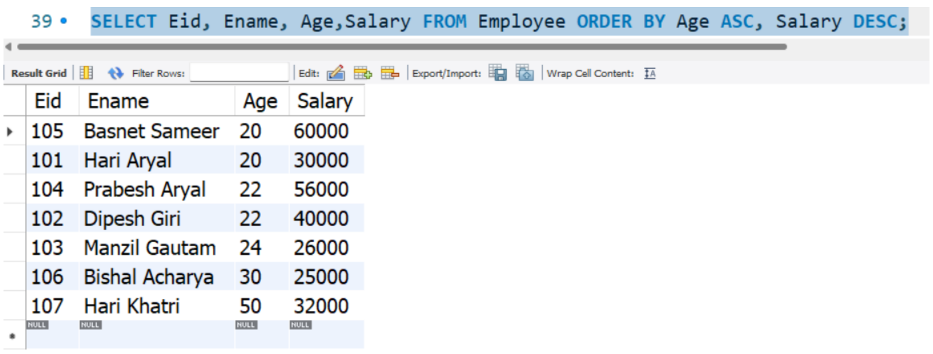


```
SELECT Eid, Ename, Age FROM Employee ORDER BY Ename ASC;
```

Eid	Ename	Age
105	Basnet Sameer	20
106	Bishal Acharya	30
102	Dipesh Giri	22
101	Hari Aryal	20
107	Hari Khatri	50
103	Manzil Gautam	24
104	Prabesh Aryal	22
NULL	NULL	NULL

QUERY2
SELECT Eid, Ename, Age, Salary
FROM Employee
ORDER BY Age ASC, Salary DESC;

OUTPUT2:



```
SELECT Eid, Ename, Age,Salary FROM Employee ORDER BY Age ASC, Salary DESC;
```

Eid	Ename	Age	Salary
105	Basnet Sameer	20	60000
101	Hari Aryal	20	30000
104	Prabesh Aryal	22	56000
102	Dipesh Giri	22	40000
103	Manzil Gautam	24	26000
106	Bishal Acharya	30	25000
107	Hari Khatri	50	32000
NULL	NULL	NULL	NULL

CONCLUSION:

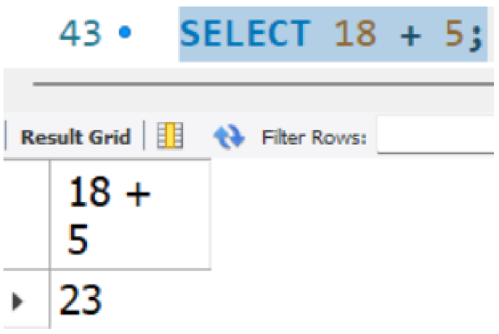
Hence, the working of ORDER BY clause was shown in MYSQL by sequencing the tuples in decreasing order of name.

Lab 05

OBJECTIVE: Illustration of Arithmetic Operators

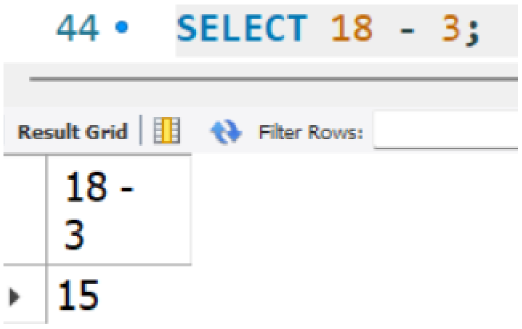
SYNTAX:
SELECT <operand> OPERATOR (+, -, *, /, %) <operand>
QUERY:
SELECT 17 + 5;
SELECT 17 - 5;
SELECT 17 * 5;
SELECT 17 / 5;
SELECT 17 % 5;

OUTPUT:



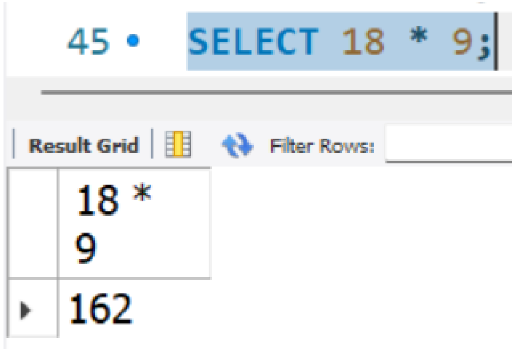
```
43 • SELECT 18 + 5;
```

```
18 +
5
23
```



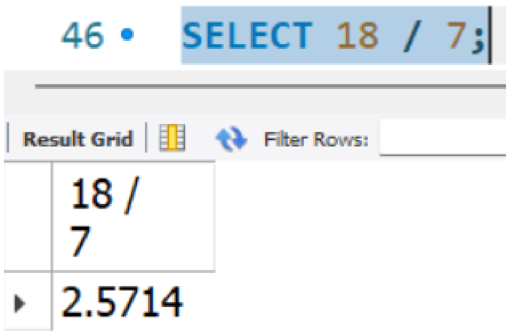
```
44 • SELECT 18 - 3;
```

```
18 -
3
15
```



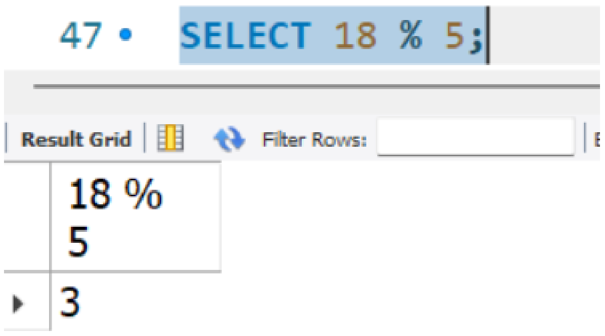
```
45 • SELECT 18 * 9;
```

```
18 *
9
162
```



```
46 • SELECT 18 / 7;
```

```
18 /
7
2.5714
```



```
47 • SELECT 18 % 5;
```

```
18 %
5
3
```

CONCLUSION:

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

Lab 06

OBJECTIVE: Illustration of Operator Precedence in Arithmetic expression

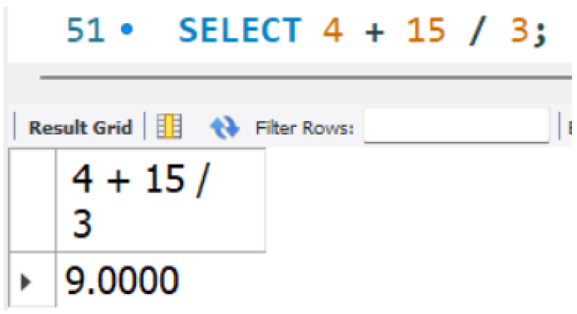
THEORY:
Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

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 (=) or as an assignment operator (=). 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.

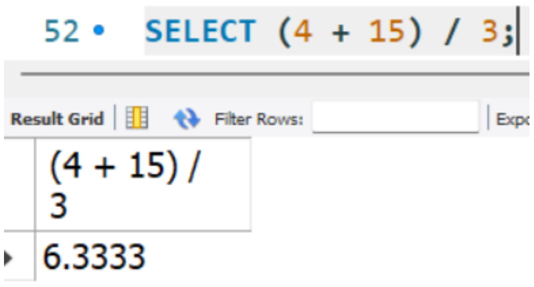
QUERY :
SELECT 4 + 5 * 4;
SELECT (4 + 5) * 4;

OUTPUT:



```
SELECT 4 + 15 / 3;
```

```
4 + 15 / 3
9.0000
```



```
SELECT (4 + 15) / 3;
```

```
(4 + 15) / 3
6.3333
```

CONCLUSION:

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

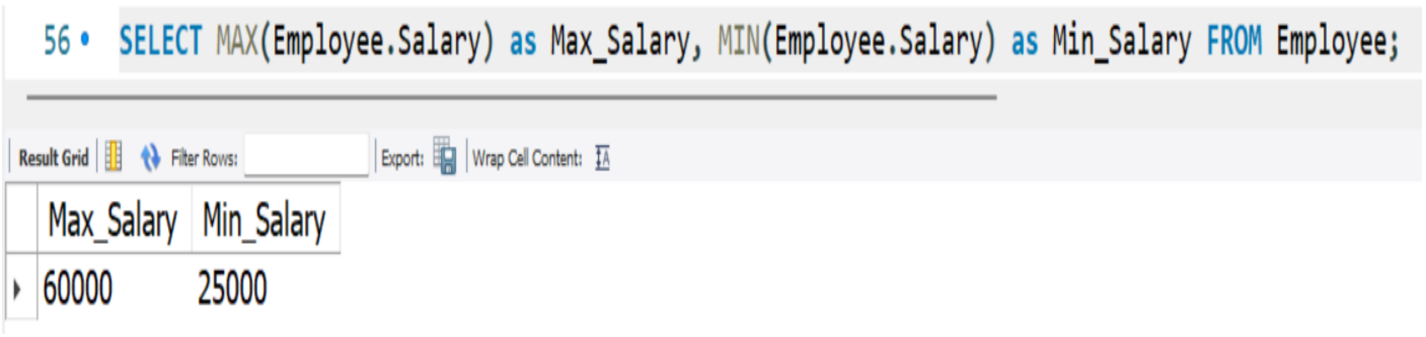
Lab 07

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:



```
SELECT MAX(Employee.Salary) as Max_Salary, MIN(Employee.Salary) as Min_Salary FROM Employee;
```

```
Max_Salary  Min_Salary
60000       25000
```

CONCLUSION:

Hence, an aggregate function used in MYSQL command.

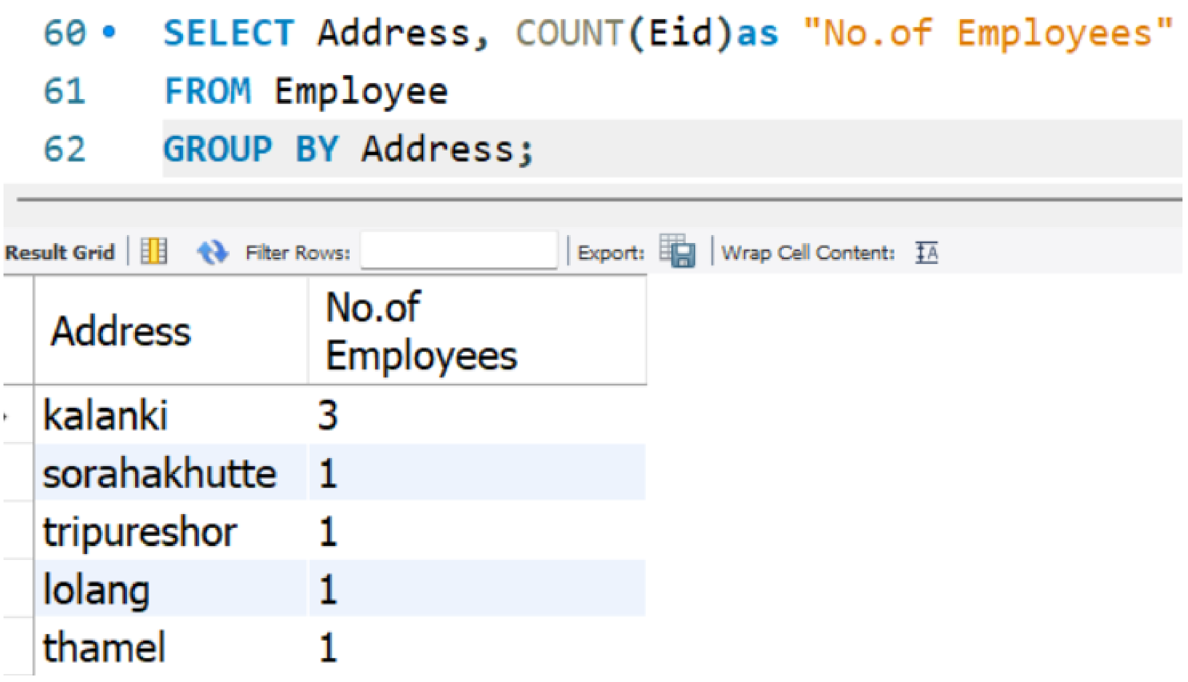
Lab 08

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:



```
SELECT Address, COUNT(Eid)as "No.of Employees"
FROM Employee
GROUP BY Address;
```

Address	No.of Employees
kalanki	3
sorahakhutte	1
tripureshor	1
lolang	1
thamel	1

CONCLUSION:

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

Lab 09

OBJECTIVE: Illustration of Restricting Group Results with the HAVING Clause

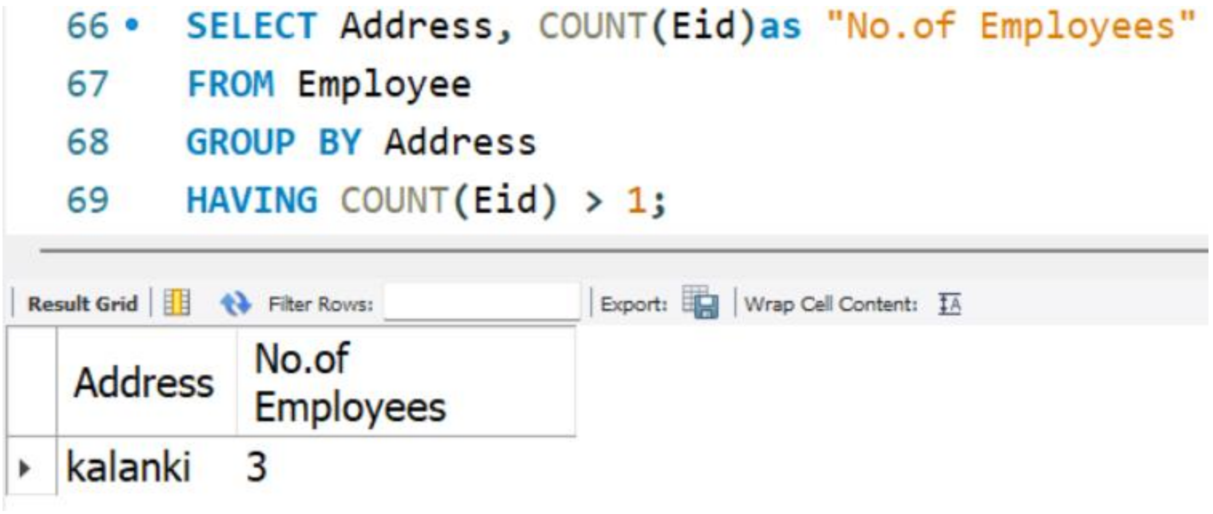
SYNTAX:

```
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:



```
SELECT Address, COUNT(Eid)as "No.of Employees"
FROM Employee
GROUP BY Address
HAVING COUNT(Eid) > 1;
```

Address	No.of Employees
kalanki	3

CONCLUSION:

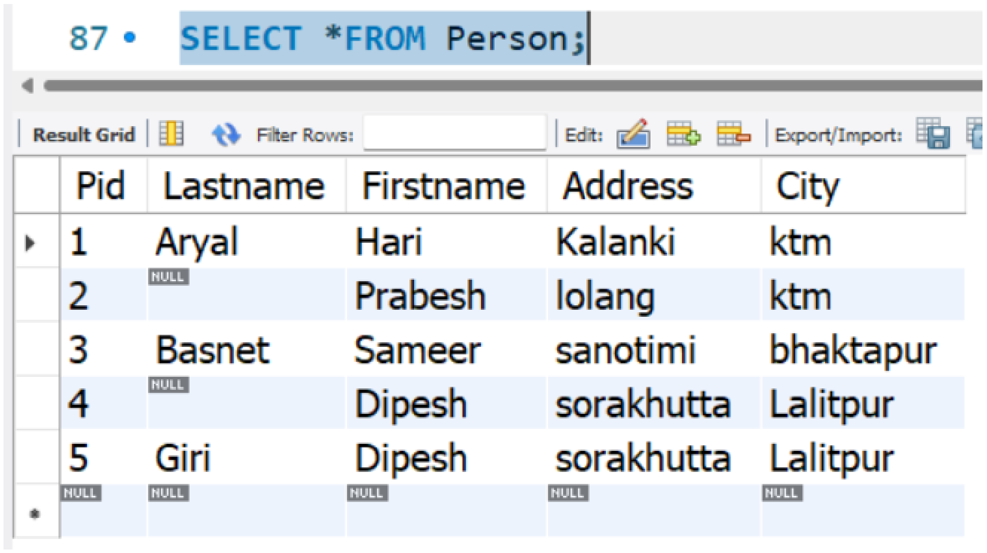
Hence, the group result was restricted by introducing HAVING clause condition.

Lab 10

OBJECTIVE: Illustration of defining a NULL value

SYNTAX:
SELECT <column_name>
FROM <table_name>
WHERE <column _name> IS NULL;

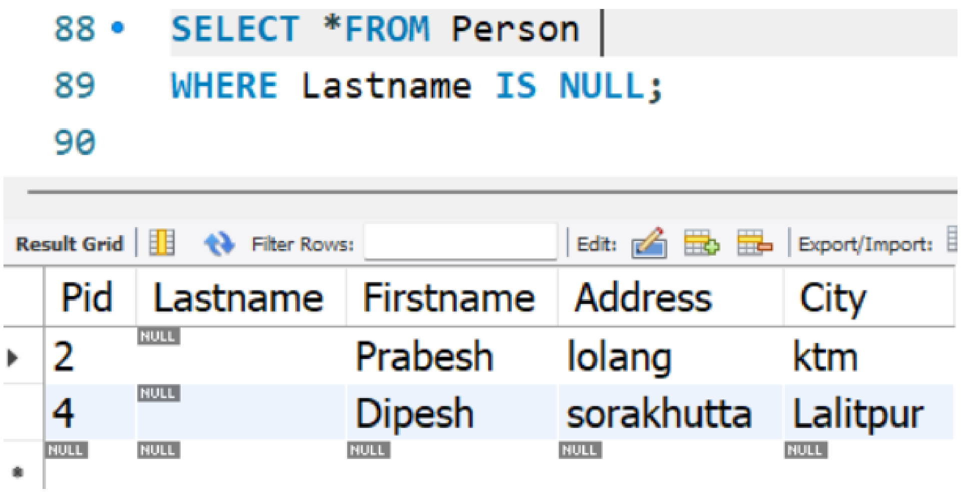
QUERY:
SELECT *FROM Person
WHERE Lastname IS NULL;
OUTPUT:



```
SELECT *FROM Person;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	Pid	Lastname	Firstname	Address	City
▸	1	Aryal	Hari	Kalanki	ktm
	2	NULL	Prabesh	lolang	ktm
	3	Basnet	Sameer	sanotimi	bhaktapur
	4	NULL	Dipesh	sorakhutta	Lalitpur
	5	Giri	Dipesh	sorakhutta	Lalitpur
*	NULL	NULL	NULL	NULL	NULL



```
SELECT *FROM Person
WHERE Lastname IS NULL;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	Pid	Lastname	Firstname	Address	City
▸	2	NULL	Prabesh	lolang	ktm
	4	NULL	Dipesh	sorakhutta	Lalitpur
*	NULL	NULL	NULL	NULL	NULL

CONCLUSION:

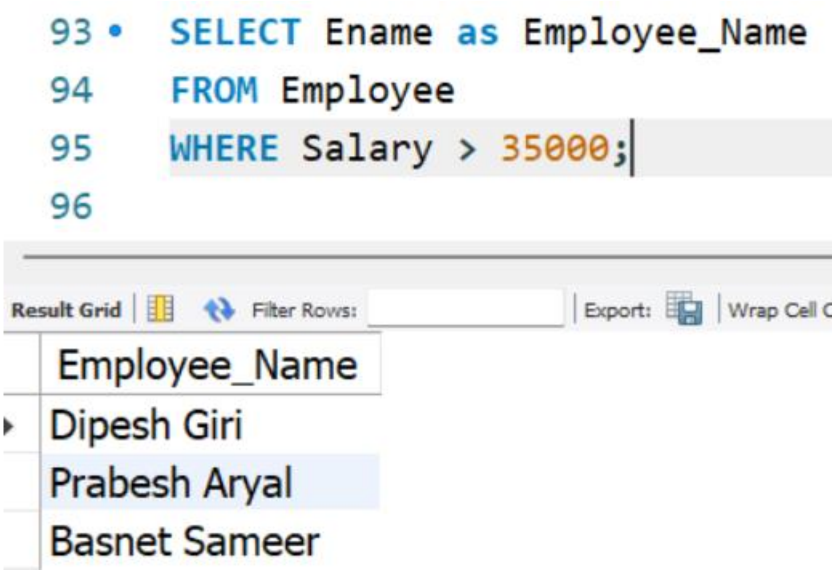
Hence, NULL values were defined and tuples having NULL values were accessed.

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:



```
SELECT Ename as Employee_Name
FROM Employee
WHERE Salary > 35000;
```

```
Employee_Name
Dipesh Giri
Prabesh Aryal
Basnet Sameer
```

CONCLUSION:

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

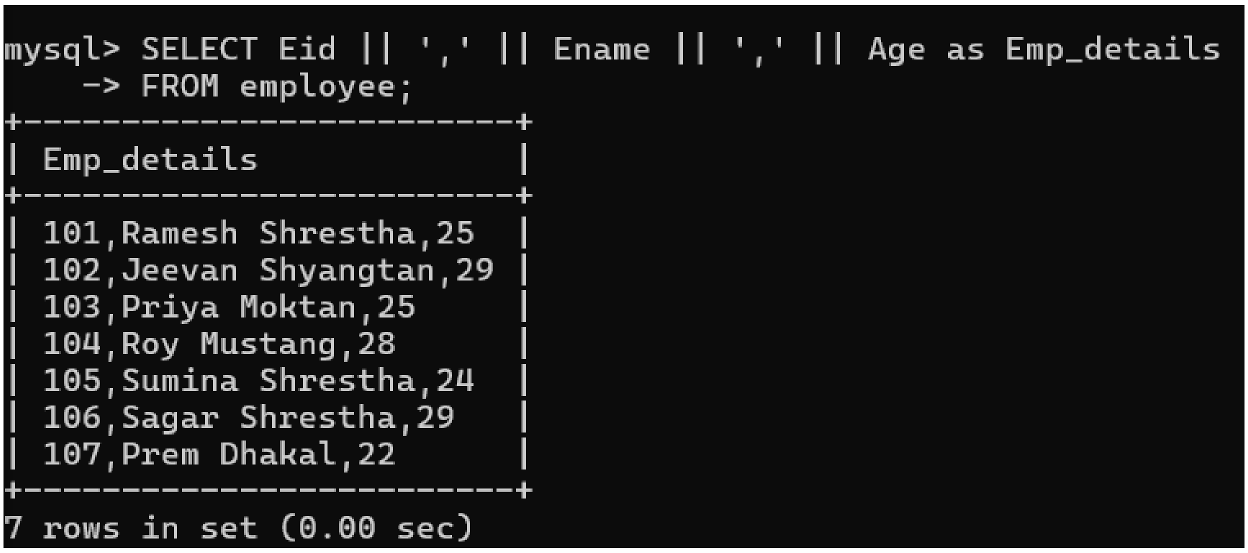
Lab 12

OBJECTIVE: Illustration of using Concatenation Operator

SYNTAX:
SELECT expression1 || expression2 || …
FROM <table_name>

QUERY
SELECT Eid || ',' || Ename || ',' || Age as Emp_details
FROM employee;

OUTPUT:



```
mysql> SELECT Eid || ',' || Ename || ',' || Age as Emp_details
    -> FROM employee;
+-------------------------+
| Emp_details             |
+-------------------------+
| 101,Ramesh Shrestha,25  |
| 102,Jeevan Shyangtan,29 |
| 103,Priya Moktan,25     |
| 104,Roy Mustang,28      |
| 105,Sumina Shrestha,24  |
| 106,Sagar Shrestha,29   |
| 107,Prem Dhakal,22      |
+-------------------------+
7 rows in set (0.00 sec)
```

CONCLUSION:

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

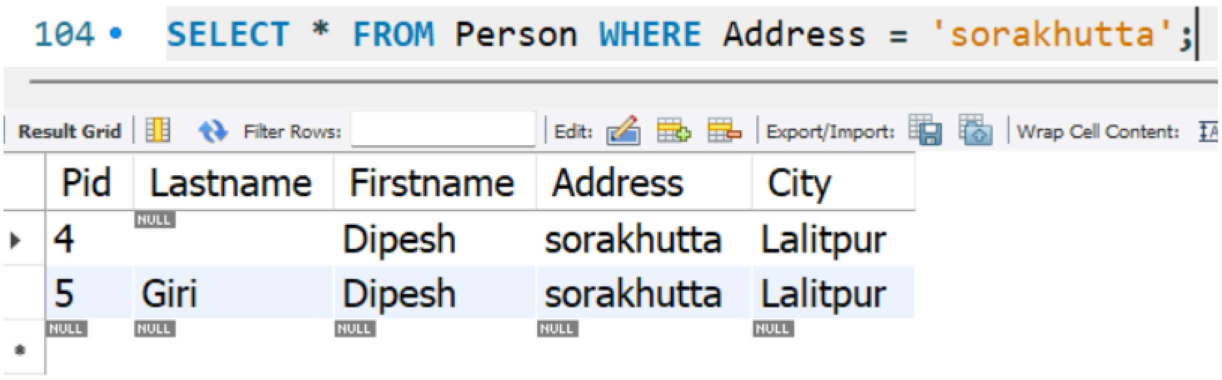
Lab 13

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:



```
SELECT * FROM Person WHERE Address = 'sorakhutta';
```

Pid	Lastname	Firstname	Address	City
4	NULL	Dipesh	sorakhutta	Lalitpur
5	Giri	Dipesh	sorakhutta	Lalitpur
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

Hence, the literal character string was implemented.

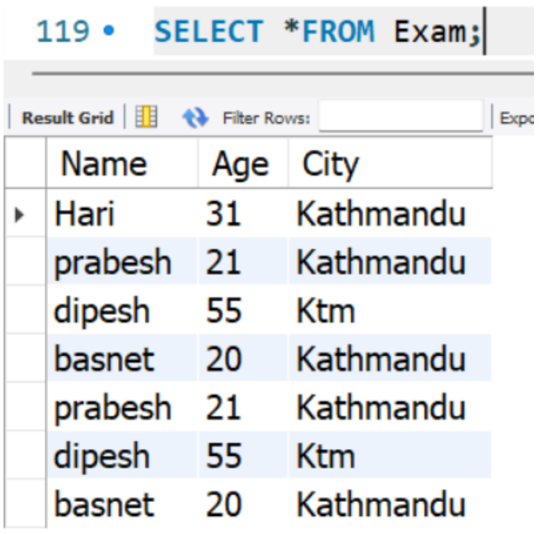
Lab 14

OBJECTIVE: Illustration of Displaying Distinct Rows

SYNTAX:
SELECT DISTINCT column1, column2, ………………column n
FROM <table name>
WHERE <condition>

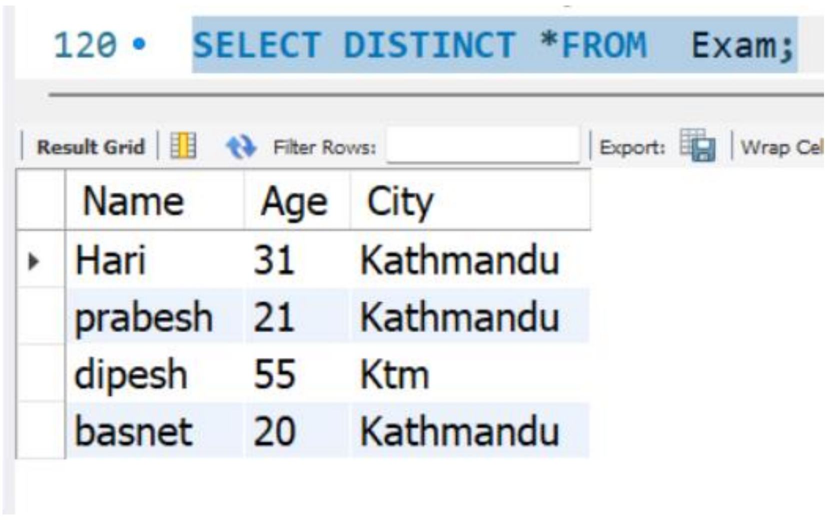
QUERY:
SELECT DISTINCT *
FROM Exam;

OUTPUT:



```
SELECT *FROM Exam;
```

Name	Age	City
Hari	31	Kathmandu
prabesh	21	Kathmandu
dipesh	55	Ktm
basnet	20	Kathmandu
prabesh	21	Kathmandu
dipesh	55	Ktm
basnet	20	Kathmandu



```
SELECT DISTINCT *FROM  Exam;
```

Name	Age	City
Hari	31	Kathmandu
prabesh	21	Kathmandu
dipesh	55	Ktm
basnet	20	Kathmandu

CONCLUSION:

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

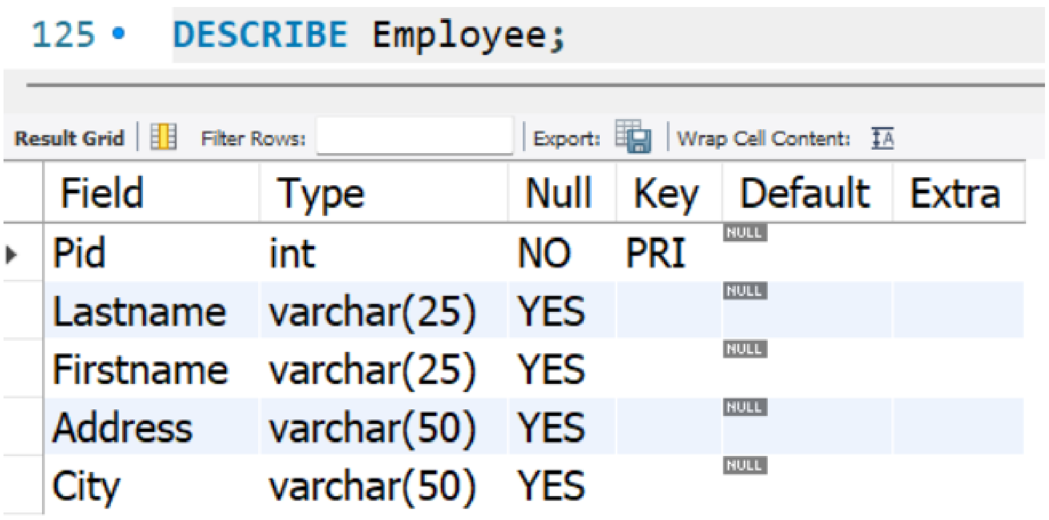
Lab 15

OBJECTIVE: Illustration of Displaying Table Structures

SYNTAX:
DESCRIBE <table_name>;

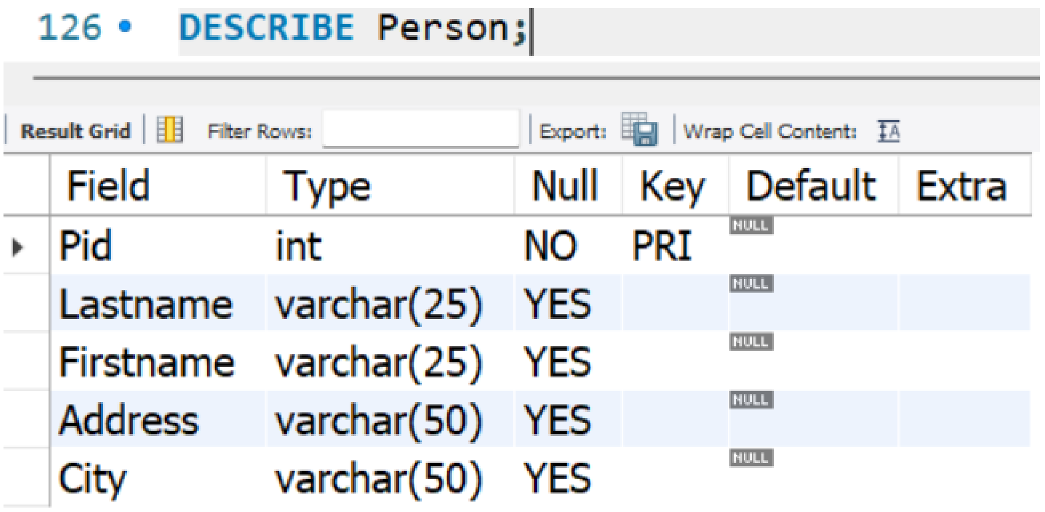
QUERY
DESCRIBE employee;
DESCRIBE PERSONS;

OUTPUT:



```
125 • DESCRIBE Employee;
```

Field	Type	Null	Key	Default	Extra
Pid	int	NO	PRI	NULL	
Lastname	varchar(25)	YES		NULL	
Firstname	varchar(25)	YES		NULL	
Address	varchar(50)	YES		NULL	
City	varchar(50)	YES		NULL	



```
126 • DESCRIBE Person;
```

Field	Type	Null	Key	Default	Extra
Pid	int	NO	PRI	NULL	
Lastname	varchar(25)	YES		NULL	
Firstname	varchar(25)	YES		NULL	
Address	varchar(50)	YES		NULL	
City	varchar(50)	YES		NULL	

CONCLUSION:

Hence, table structure was displayed using keyword DESCRIBE.

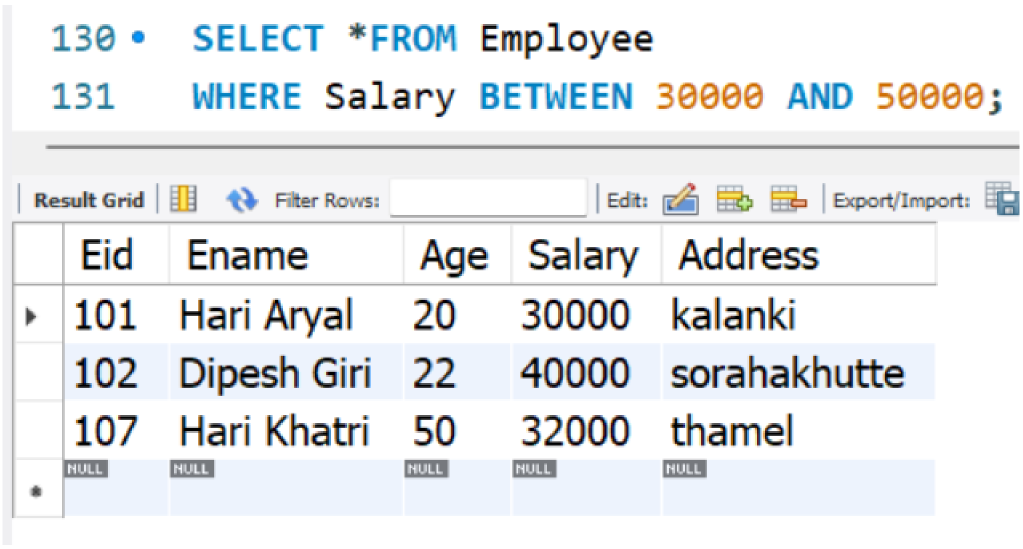
Lab 16

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:



```
SELECT *FROM Employee
WHERE Salary BETWEEN 30000 AND 50000;
```

Eid	Ename	Age	Salary	Address
101	Hari Aryal	20	30000	kalanki
102	Dipesh Giri	22	40000	sorahakhutte
107	Hari Khatri	50	32000	thamel
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

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

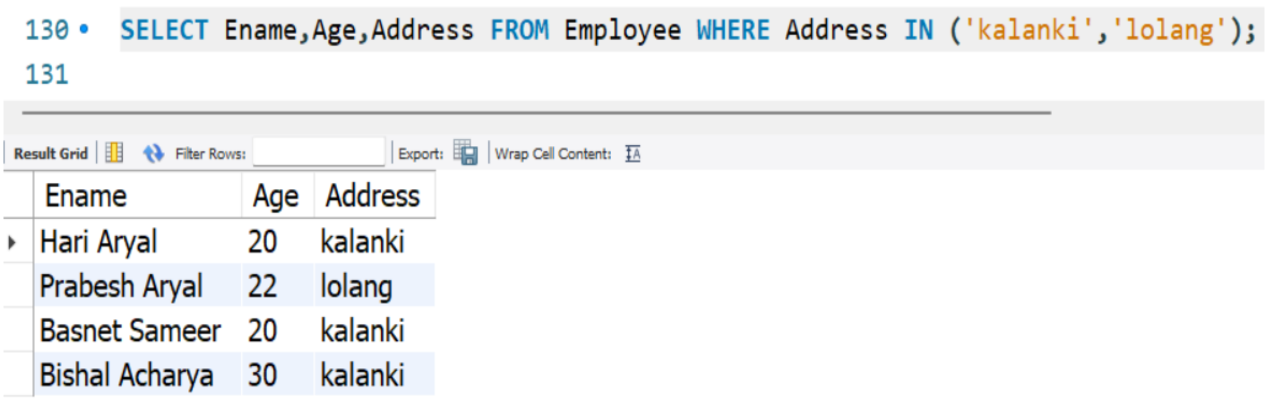
Lab 17

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:



```
SELECT Ename,Age,Address FROM Employee WHERE Address IN ('kalanki','lolang');
```

Ename	Age	Address
Hari Aryal	20	kalanki
Prabesh Aryal	22	lolang
Basnet Sameer	20	kalanki
Bishal Acharya	30	kalanki

CONCLUSION:

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

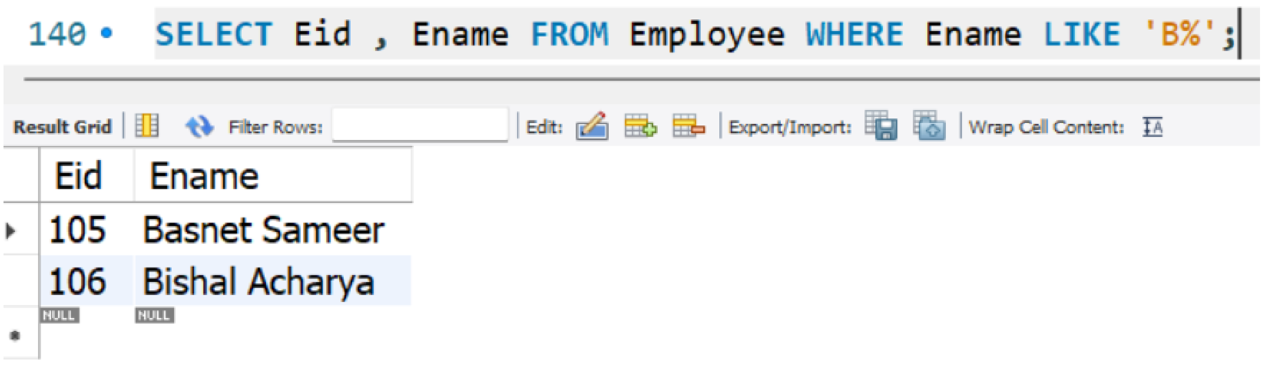
Lab 18

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:



```
SELECT Eid , Ename FROM Employee WHERE Ename LIKE 'B%';
```

Eid	Ename
105	Basnet Sameer
106	Bishal Acharya
NULL	NULL

CONCLUSION:

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

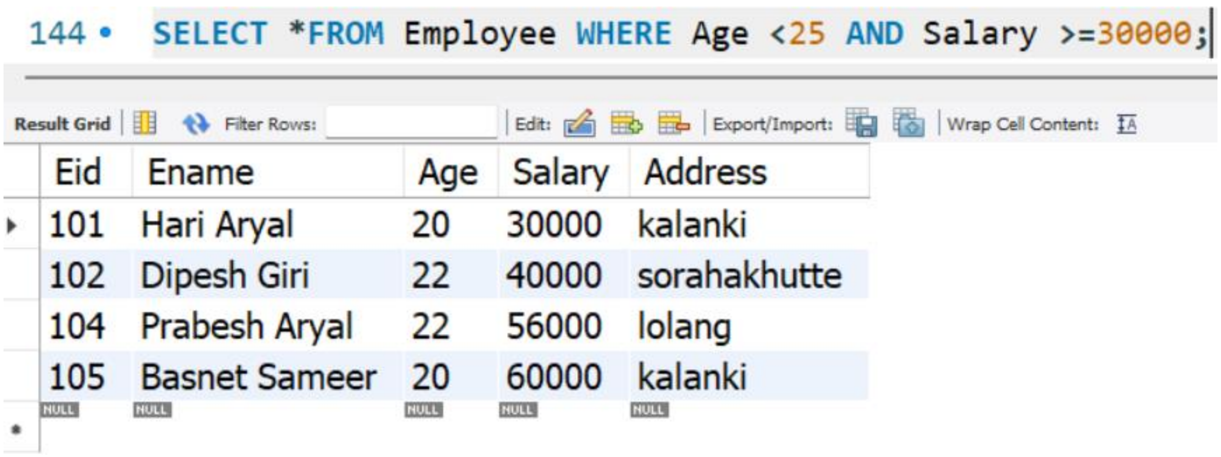
Lab 19

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:



```
SELECT *FROM Employee WHERE Age <25 AND Salary >=30000;
```

Eid	Ename	Age	Salary	Address
101	Hari Aryal	20	30000	kalanki
102	Dipesh Giri	22	40000	sorahakhutte
104	Prabesh Aryal	22	56000	lolang
105	Basnet Sameer	20	60000	kalanki
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

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

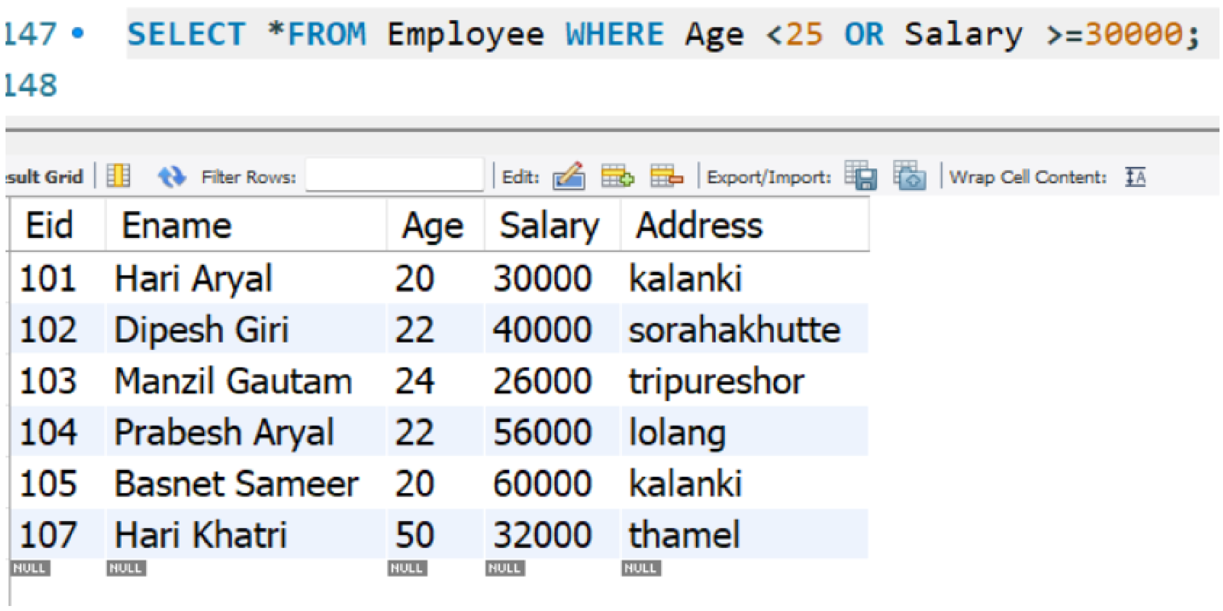
Lab 20

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:



```
SELECT *FROM Employee WHERE Age <25 OR Salary >=30000;
```

Eid	Ename	Age	Salary	Address
101	Hari Aryal	20	30000	kalanki
102	Dipesh Giri	22	40000	sorahakhutte
103	Manzil Gautam	24	26000	tripureshor
104	Prabesh Aryal	22	56000	lolang
105	Basnet Sameer	20	60000	kalanki
107	Hari Khatri	50	32000	thamel
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

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

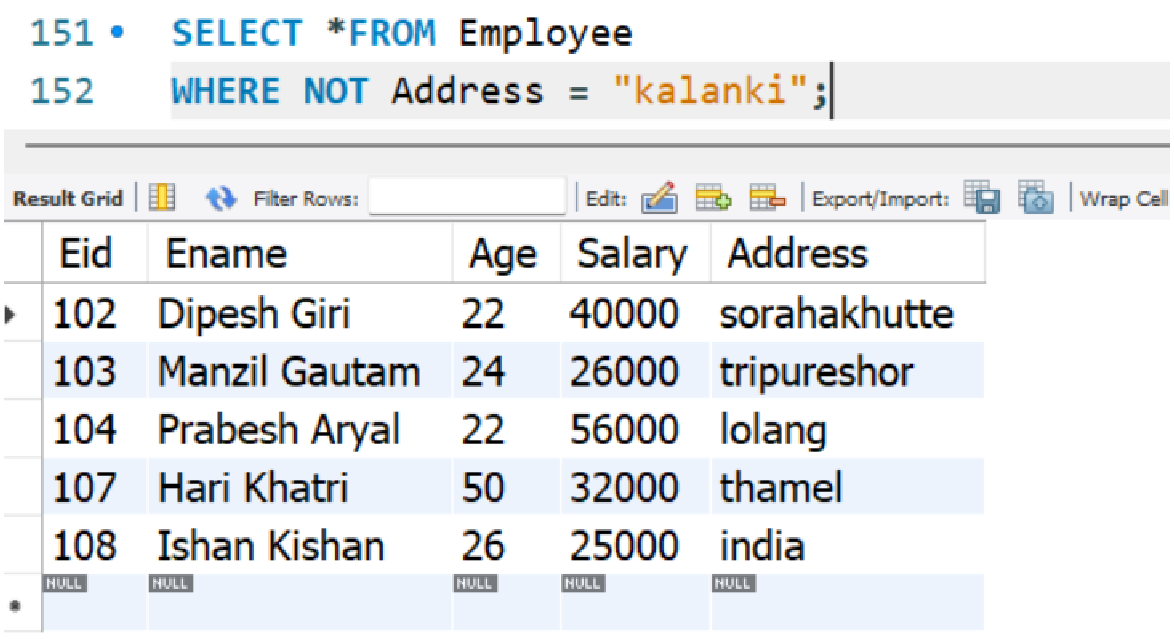
Lab 21

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:



```
SELECT *FROM Employee
WHERE NOT Address = "kalanki";
```

Eid	Ename	Age	Salary	Address
102	Dipesh Giri	22	40000	sorahakhutte
103	Manzil Gautam	24	26000	tripureshor
104	Prabesh Aryal	22	56000	lolang
107	Hari Khatri	50	32000	thamel
108	Ishan Kishan	26	25000	india
NULL	NULL	NULL	NULL	NULL

CONCLUSION:

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

Lab 22

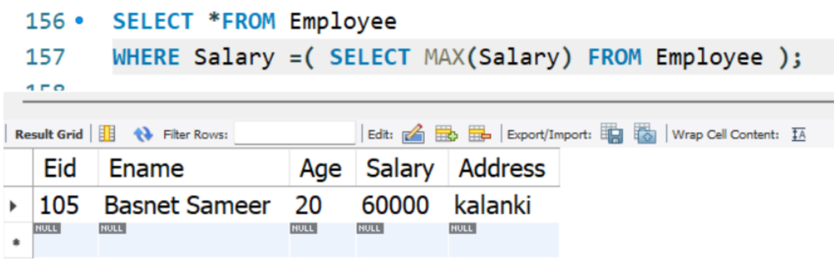
OBJECTIVE: Illustration of Subquery

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);

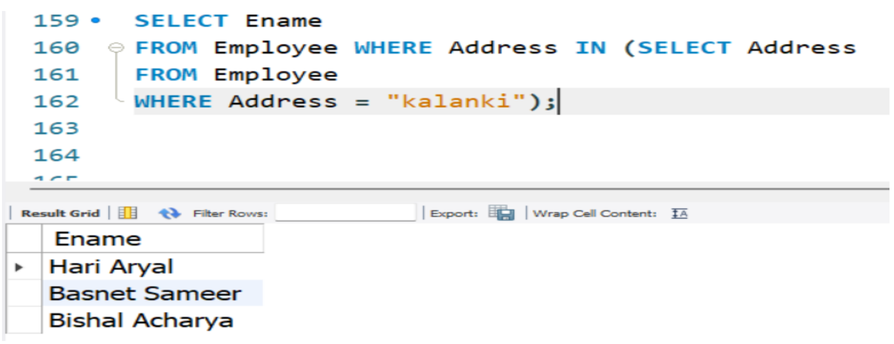
Multiple Rows Sub Query
SELECT Ename
FROM employee
WHERE Address IN (SELECT Address
FROM employee
WHERE Address =”kalanki”);

OUTPUT:



```
SELECT *FROM Employee
WHERE Salary =( SELECT MAX(Salary) FROM Employee );
```

Eid	Ename	Age	Salary	Address
105	Basnet Sameer	20	60000	kalanki
NULL	NULL	NULL	NULL	NULL



```
SELECT Ename
FROM Employee WHERE Address IN (SELECT Address
FROM Employee
WHERE Address = "kalanki");
```

Ename
Hari Aryal
Basnet Sameer
Bishal Acharya

CONCLUSION:

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

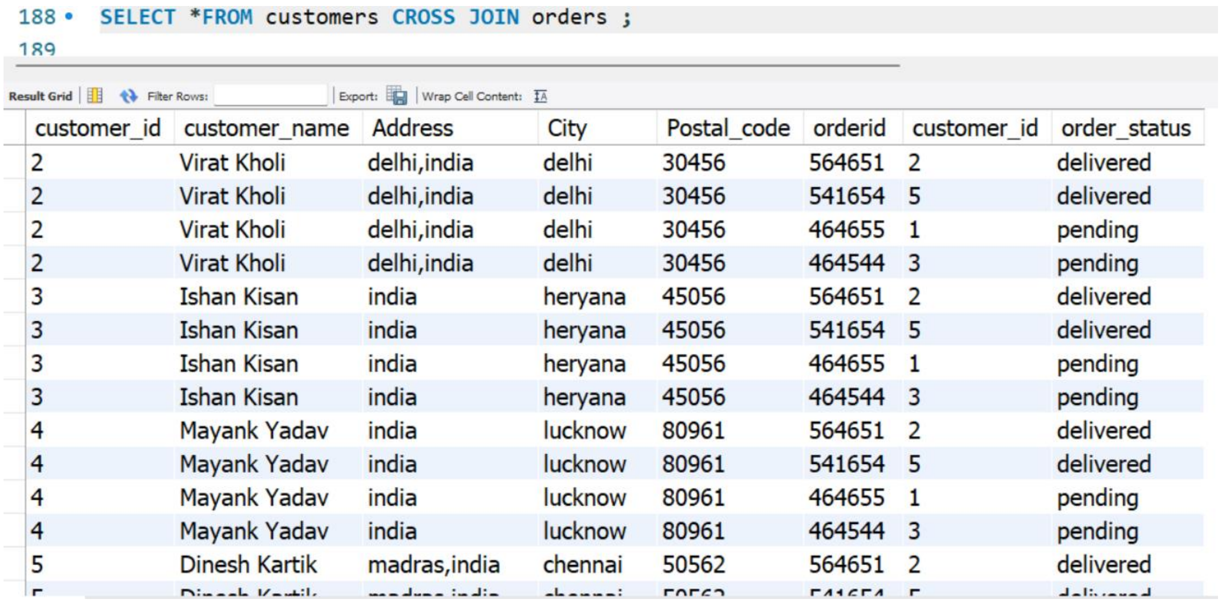
Lab 23

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:



```
SELECT *FROM customers CROSS JOIN orders ;
```

customer_id	customer_name	Address	City	Postal_code	orderid	customer_id	order_status
2	Virat Kholi	delhi,india	delhi	30456	564651	2	delivered
2	Virat Kholi	delhi,india	delhi	30456	541654	5	delivered
2	Virat Kholi	delhi,india	delhi	30456	464655	1	pending
2	Virat Kholi	delhi,india	delhi	30456	464544	3	pending
3	Ishan Kisan	india	heryana	45056	564651	2	delivered
3	Ishan Kisan	india	heryana	45056	541654	5	delivered
3	Ishan Kisan	india	heryana	45056	464655	1	pending
3	Ishan Kisan	india	heryana	45056	464544	3	pending
4	Mayank Yadav	india	lucknow	80961	564651	2	delivered
4	Mayank Yadav	india	lucknow	80961	541654	5	delivered
4	Mayank Yadav	india	lucknow	80961	464655	1	pending
4	Mayank Yadav	india	lucknow	80961	464544	3	pending
5	Dinesh Kartik	madras,india	chennai	50562	564651	2	delivered

CONCLUSION:

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

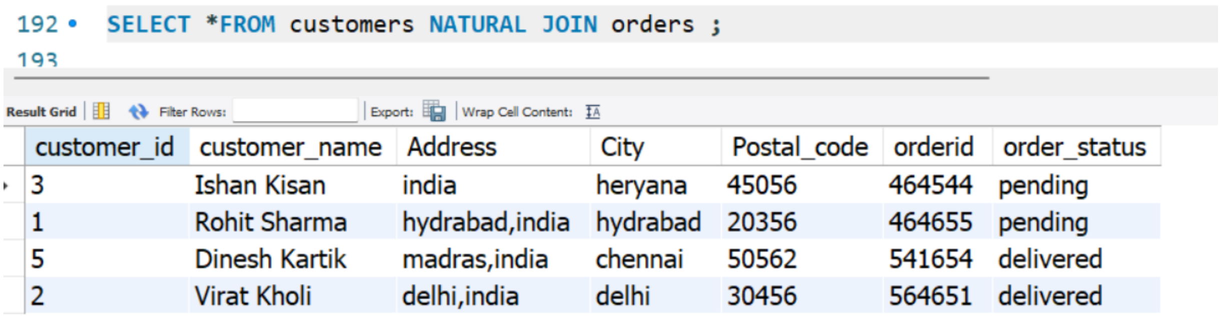
Lab 24

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:



```
SELECT *FROM customers NATURAL JOIN orders ;
```

customer_id	customer_name	Address	City	Postal_code	orderid	order_status
3	Ishan Kisan	india	heryana	45056	464544	pending
1	Rohit Sharma	hydrabad,india	hydrabad	20356	464655	pending
5	Dinesh Kartik	madras,india	chennai	50562	541654	delivered
2	Virat Kholi	delhi,india	delhi	30456	564651	delivered

CONCLUSION:

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

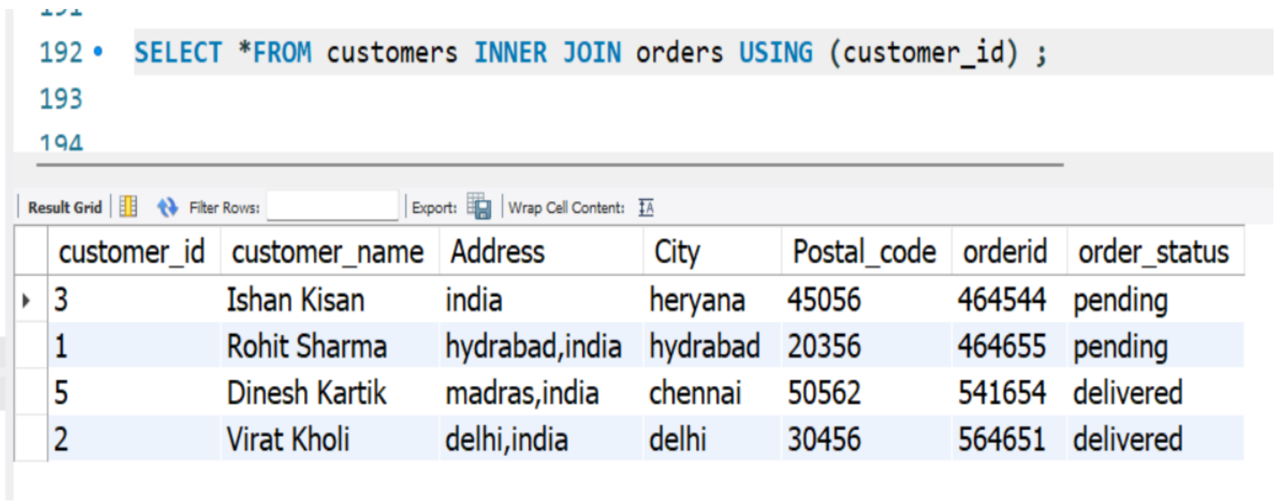
Lab 25

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:



```
SELECT *FROM customers INNER JOIN orders USING (customer_id) ;
```

customer_id	customer_name	Address	City	Postal_code	orderid	order_status
3	Ishan Kisan	india	heryana	45056	464544	pending
1	Rohit Sharma	hydrabad,india	hydrabad	20356	464655	pending
5	Dinesh Kartik	madras,india	chennai	50562	541654	delivered
2	Virat Kholi	delhi,india	delhi	30456	564651	delivered

CONCLUSION:

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

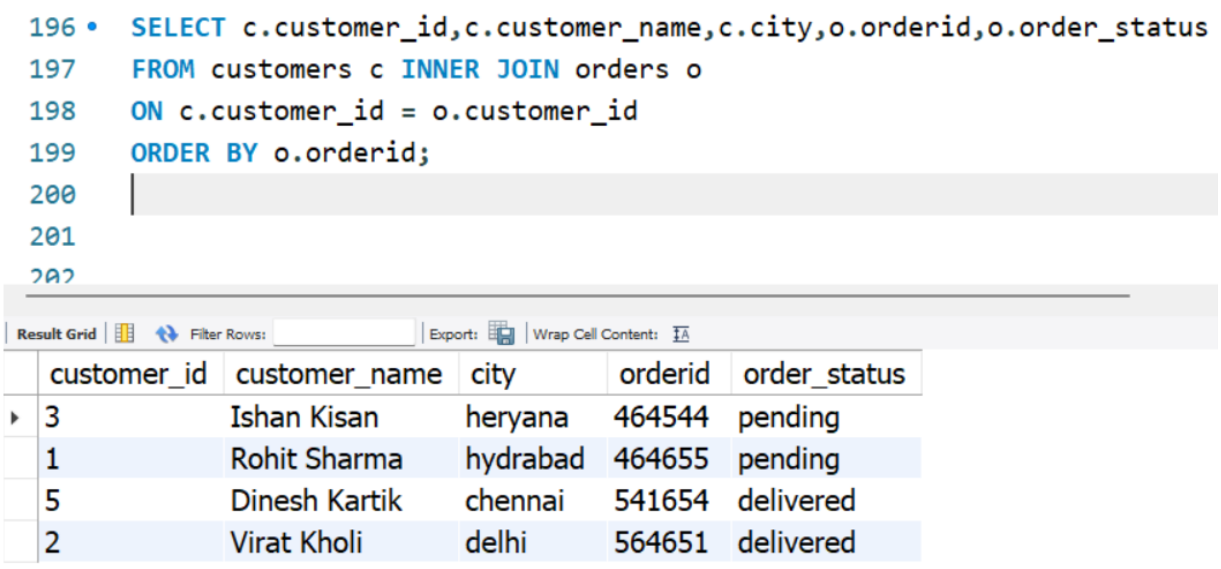
Lab 26

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:



```
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;
```

customer_id	customer_name	city	orderid	order_status
3	Ishan Kisan	heryana	464544	pending
1	Rohit Sharma	hydrabad	464655	pending
5	Dinesh Kartik	chennai	541654	delivered
2	Virat Kholi	delhi	564651	delivered

CONCLUSION:

Hence, two tables can be combined using ON clause.

Lab 27

OBJECTIVE: Illustration of LEFT OUTER JOIN

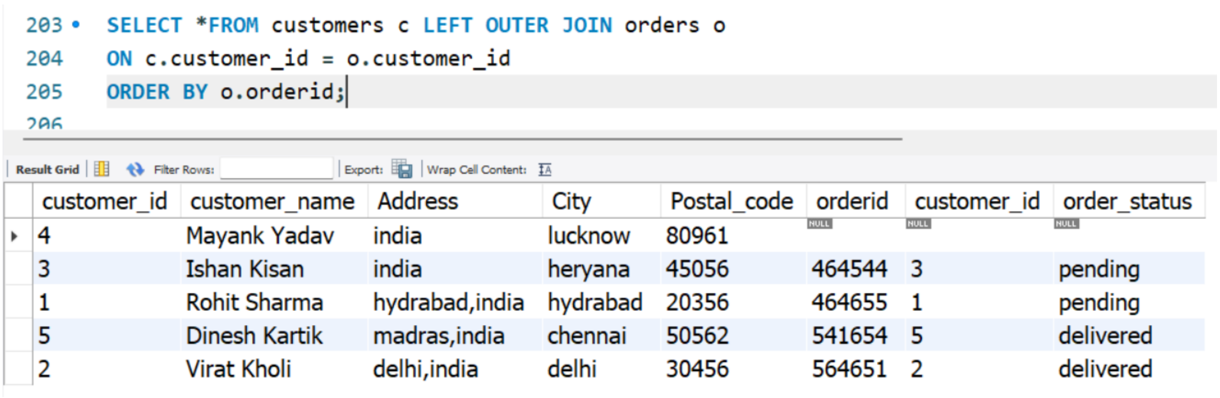
SYNTAX:

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:



```
SELECT *FROM customers c LEFT OUTER JOIN orders o
ON c.customer_id = o.customer_id
ORDER BY o.orderid;
```

customer_id	customer_name	Address	City	Postal_code	orderid	customer_id	order_status
4	Mayank Yadav	india	lucknow	80961	NULL	NULL	NULL
3	Ishan Kisan	india	heryana	45056	464544	3	pending
1	Rohit Sharma	hydrabad,india	hydrabad	20356	464655	1	pending
5	Dinesh Kartik	madras,india	chennai	50562	541654	5	delivered
2	Virat Kholi	delhi,india	delhi	30456	564651	2	delivered

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.

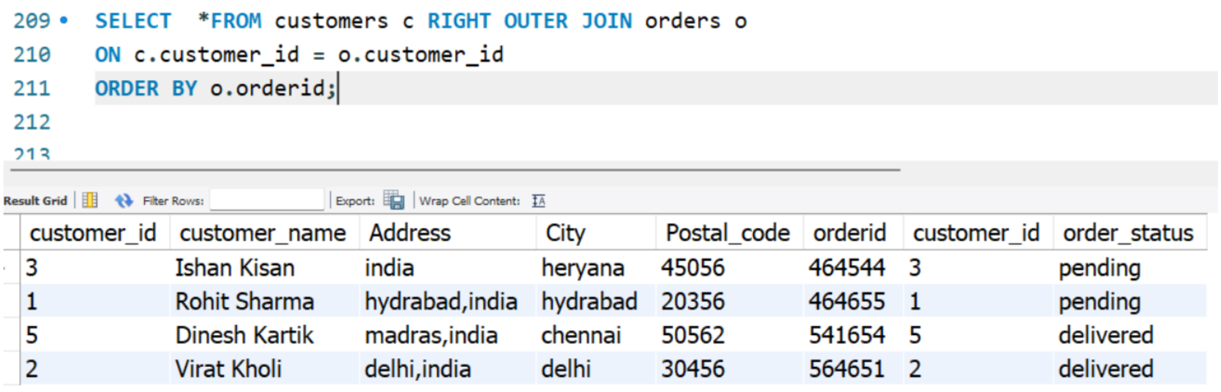
Lab 28

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:


```
SELECT  *FROM customers c RIGHT OUTER JOIN orders o
ON c.customer_id = o.customer_id
ORDER BY o.orderid;
```

customer_id	customer_name	Address	City	Postal_code	orderid	customer_id	order_status
3	Ishan Kisan	india	heryana	45056	464544	3	pending
1	Rohit Sharma	hydrabad,india	hydrabad	20356	464655	1	pending
5	Dinesh Kartik	madras,india	chennai	50562	541654	5	delivered
2	Virat Kholi	delhi,india	delhi	30456	564651	2	delivered

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;

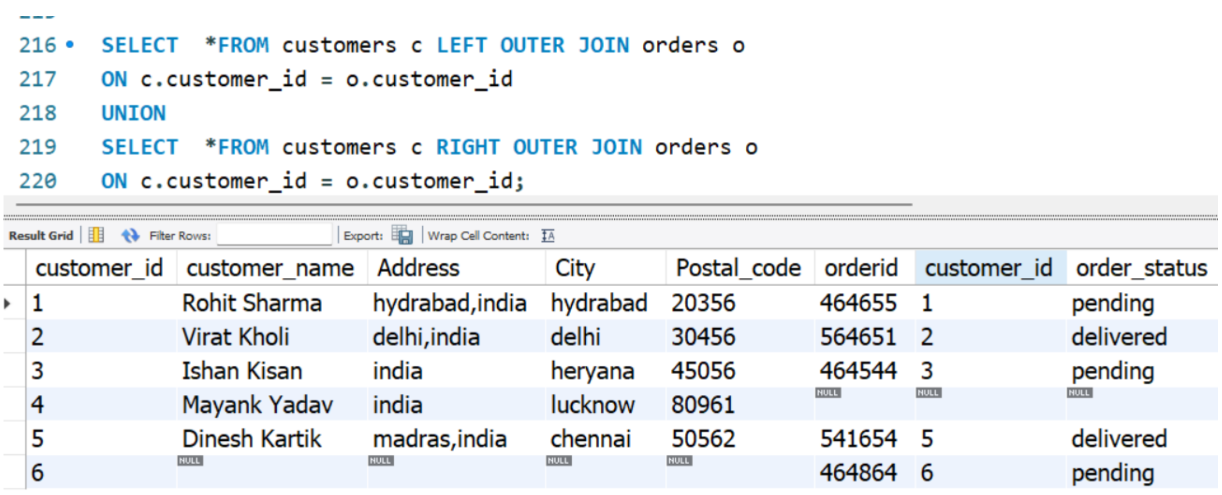
OR

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:



```
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;
```

customer_id	customer_name	Address	City	Postal_code	orderid	customer_id	order_status
1	Rohit Sharma	hydrabad,india	hydrabad	20356	464655	1	pending
2	Virat Kholi	delhi,india	delhi	30456	564651	2	delivered
3	Ishan Kisan	india	heryana	45056	464544	3	pending
4	Mayank Yadav	india	lucknow	80961	NULL	NULL	NULL
5	Dinesh Kartik	madras,india	chennai	50562	541654	5	delivered
6	NULL	NULL	NULL	NULL	464864	6	pending

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.

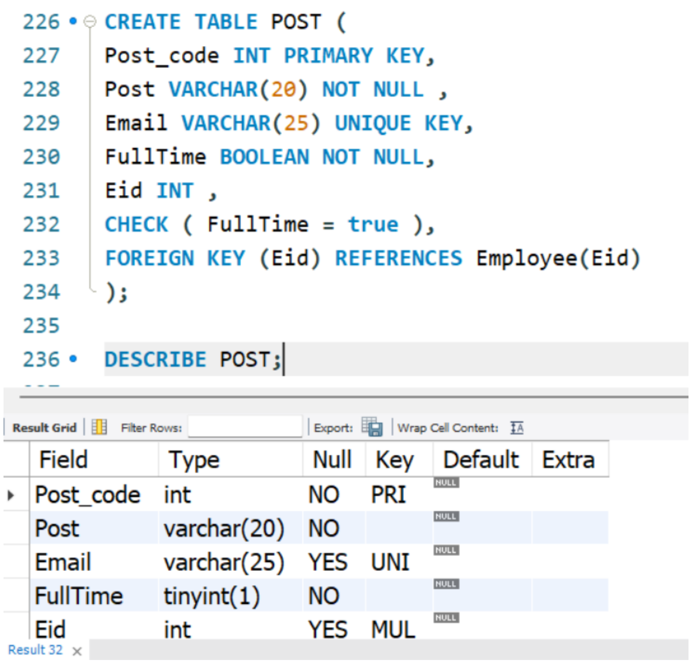
SYNTAX:

```
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:



```
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)
);

DESCRIBE POST;
```

Field	Type	Null	Key	Default	Extra
Post_code	int	NO	PRI	NULL	
Post	varchar(20)	NO		NULL	
Email	varchar(25)	YES	UNI	NULL	
FullTime	tinyint(1)	NO		NULL	
Eid	int	YES	MUL	NULL	

CONCLUSION:

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