

## **SQL Clauses(Continued...)**

# DISTINCT

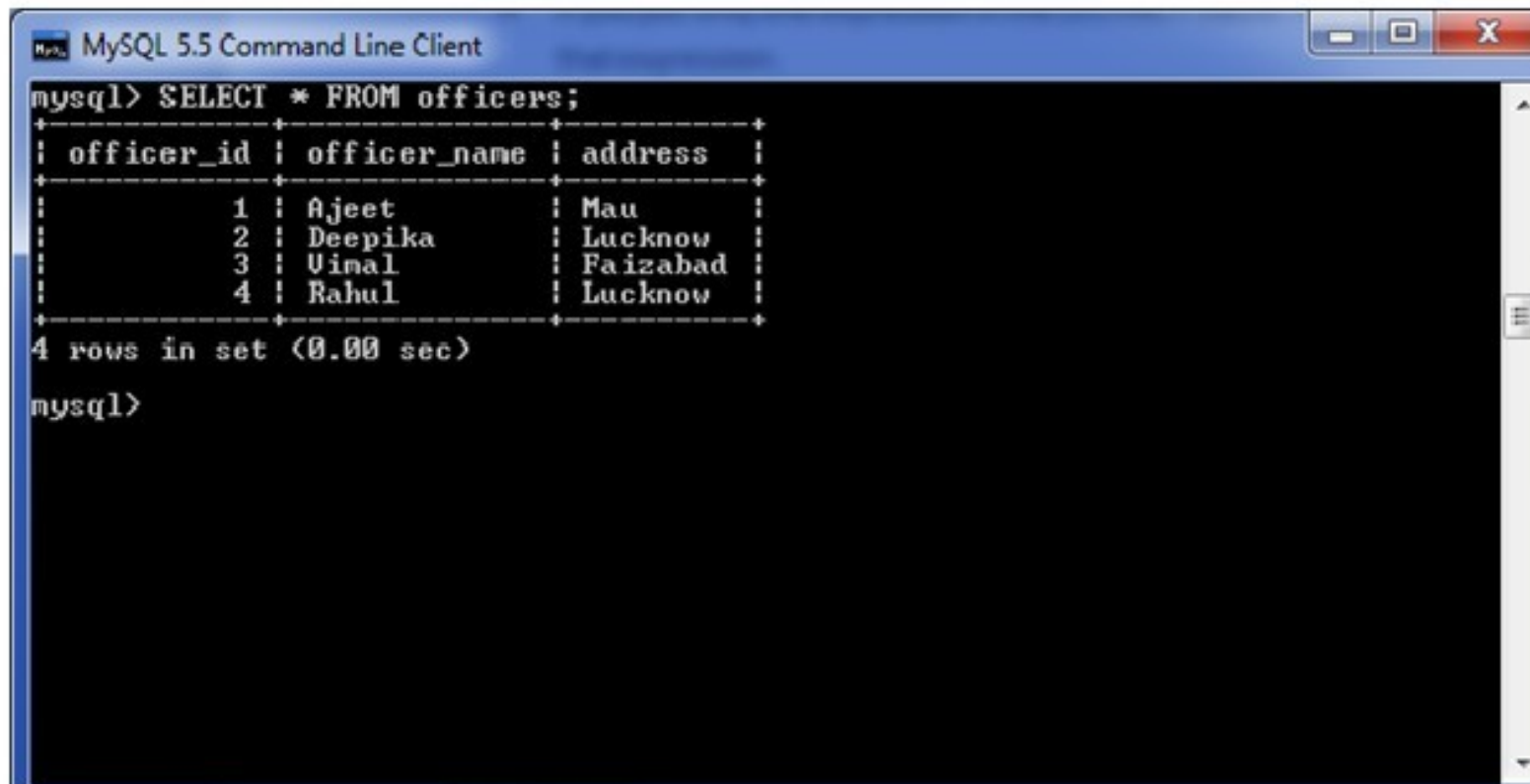
Syntax:

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

# MySQL DISTINCT Clause with single expression

If you use a single expression then the MySQL DISTINCT clause will return a single field with unique records (no duplicate record).

See the table:



```
mysql> SELECT * FROM officers;
```

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uinal	Faizabad
4	Rahul	Lucknow

```
4 rows in set (0.00 sec)
```

```
mysql>
```

**SELECT DISTINCT** address  
**FROM** officers;



```
mysql> SELECT * FROM officers;
```

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Vinal	Faizabad
4	Rahul	Lucknow

```
4 rows in set (0.00 sec)
```

```
mysql> SELECT DISTINCT address  
-> FROM officers;
```

address
Mau
Lucknow
Faizabad

```
3 rows in set (0.00 sec)
```

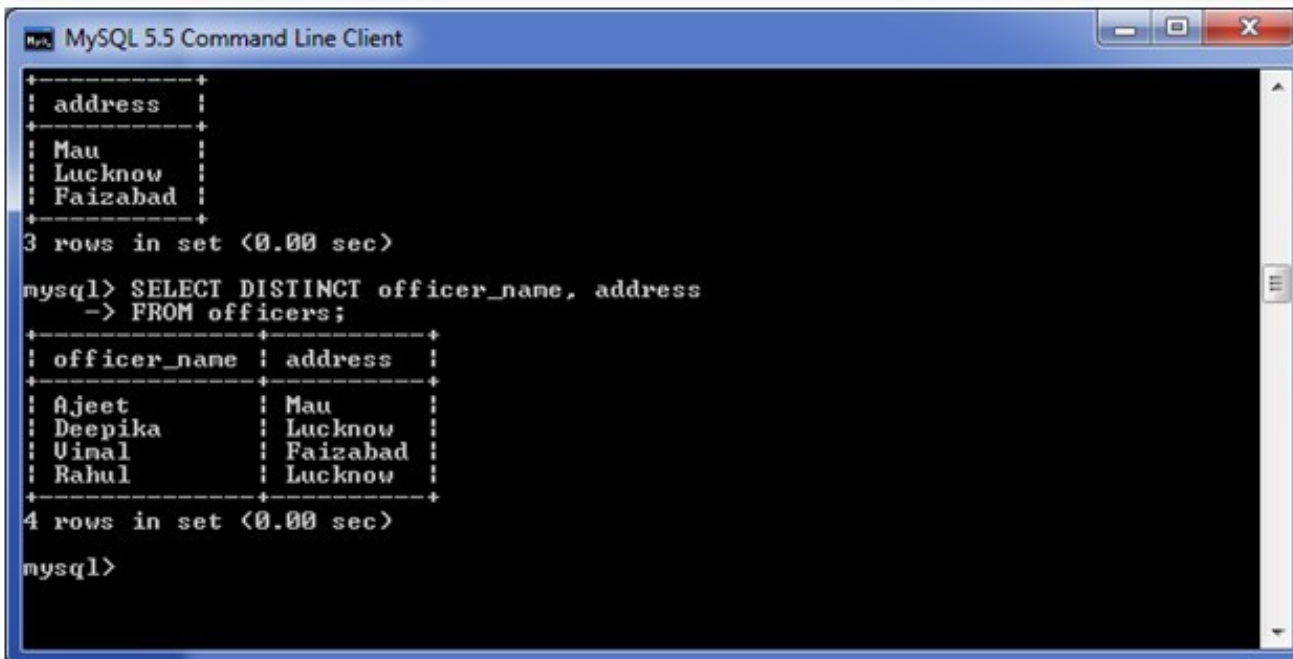
```
mysql> _
```

## MySQL DISTINCT Clause with multiple expressions

If you use multiple expressions with DISTINCT Clause then MySQL DISTINCT clause will remove duplicates from more than one field in your SELECT statement.

Use the following query:

```
SELECT DISTINCT officer_name, address  
FROM officers;
```



The screenshot shows a MySQL 5.5 Command Line Client window. It displays the results of a query that selects distinct combinations of officer names and addresses. The first part of the output shows a table with one column, 'address', containing three rows: 'Mau', 'Lucknow', and 'Faizabad'. Below this, it states '3 rows in set (0.00 sec)'. The second part shows the execution of the query 'SELECT DISTINCT officer\_name, address FROM officers;'. The output is a table with two columns: 'officer\_name' and 'address'. It contains four rows: 'Ajeet' from 'Mau', 'Deepika' from 'Lucknow', 'Uinal' from 'Faizabad', and 'Rahul' from 'Lucknow'. Below this, it states '4 rows in set (0.00 sec)'. The prompt 'mysql>' is visible at the bottom.

```
MySQL 5.5 Command Line Client  
+-----+  
| address |  
+-----+  
| Mau    |  
| Lucknow|  
| Faizabad|  
+-----+  
3 rows in set (0.00 sec)  
mysql> SELECT DISTINCT officer_name, address  
-> FROM officers;  
+-----+-----+  
| officer_name | address |  
+-----+-----+  
| Ajeet        | Mau     |  
| Deepika      | Lucknow |  
| Uinal        | Faizabad|  
| Rahul        | Lucknow |  
+-----+-----+  
4 rows in set (0.00 sec)  
mysql>
```

# MySQL ORDER BY Clause

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

**Syntax:**

```
SELECT expressions  
FROM tables  
[WHERE conditions]  
ORDER BY expression [ ASC | DESC ];
```

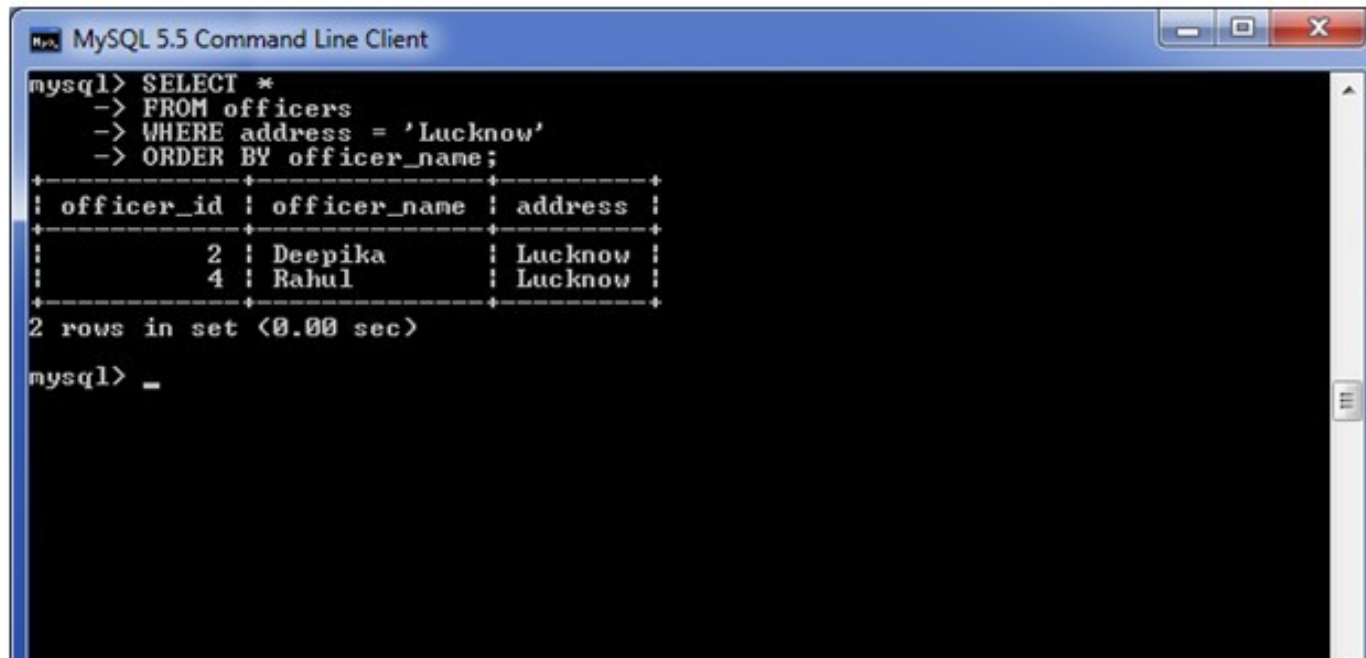
# MySQL ORDER BY: without using ASC/DESC attribute

If you use MySQL ORDER BY clause without specifying the ASC and DESC modifier then by default you will get the result in ascending order.

Execute the following query:

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
ORDER BY officer_name;
```

Output:



```
MySQL 5.5 Command Line Client  
mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Lucknow'  
-> ORDER BY officer_name;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 2 | Deepika | Lucknow |  
| 4 | Rahul | Lucknow |  
+-----+-----+-----+  
2 rows in set (0.00 sec)  
mysql> _
```

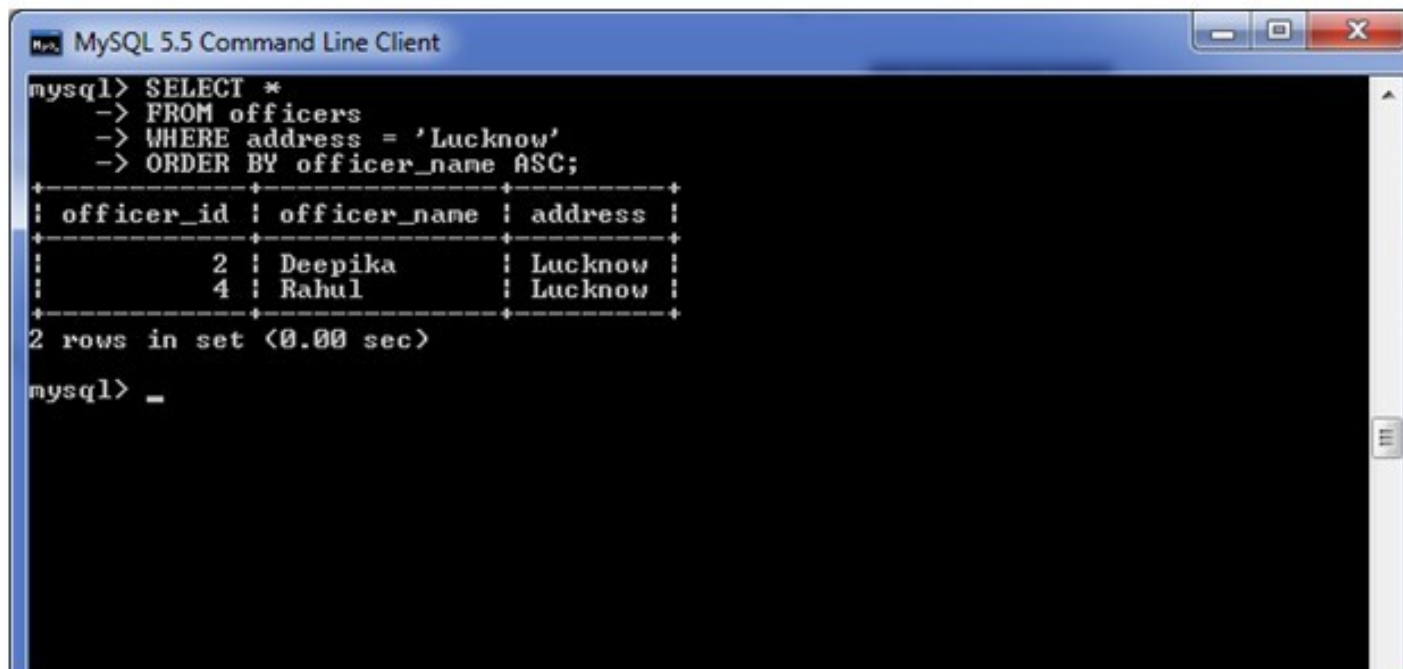
# MySQL ORDER BY: with ASC attribute

Let's take an example to retrieve the data in ascending order.

Execute the following query:

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
ORDER BY officer_name ASC;
```

Output:

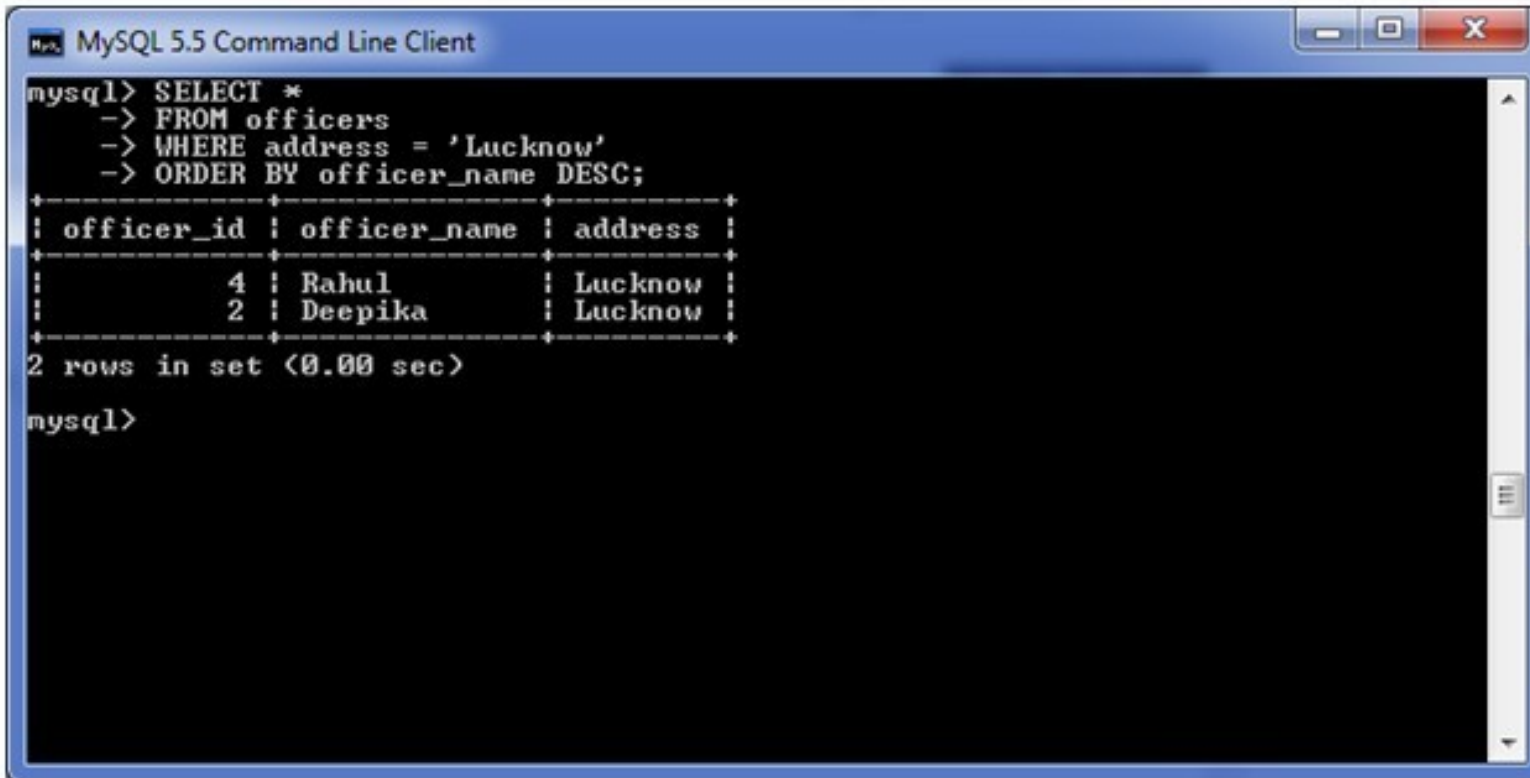


```
MySQL 5.5 Command Line Client  
mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Lucknow'  
-> ORDER BY officer_name ASC;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
|          2 | Deepika      | Lucknow |  
|          4 | Rahul        | Lucknow |  
+-----+-----+-----+  
2 rows in set (0.00 sec)  
mysql> _
```



# MySQL ORDER BY: with DESC attribute

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
ORDER BY officer_name DESC;
```



The screenshot shows the MySQL 5.5 Command Line Client window. The query entered is: `mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Lucknow'  
-> ORDER BY officer_name DESC;` The output displays a table with 3 columns: `officer_id`, `officer_name`, and `address`. The results are ordered by `officer_name` in descending order, showing two rows: `4 | Rahul | Lucknow` and `2 | Deepika | Lucknow`. Below the table, it says `2 rows in set (0.00 sec)`. The prompt `mysql>` is visible at the bottom.

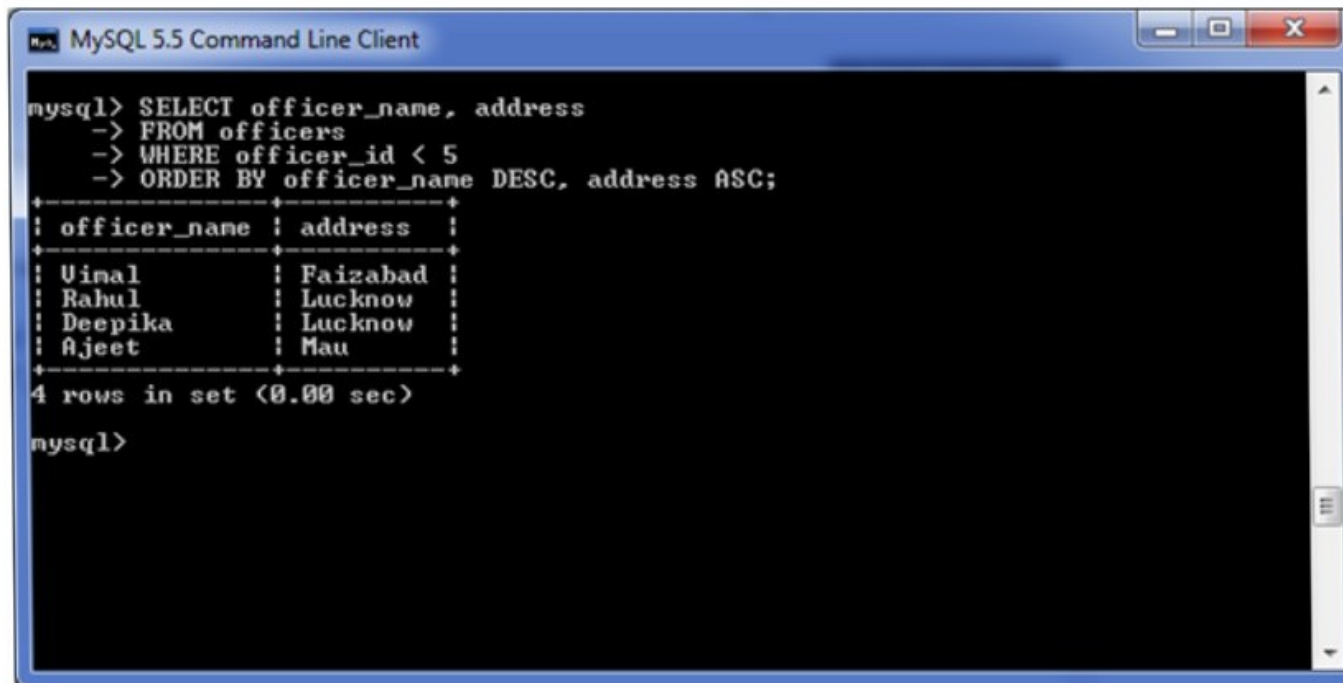
```
mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Lucknow'  
-> ORDER BY officer_name DESC;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 4 | Rahul | Lucknow |  
| 2 | Deepika | Lucknow |  
+-----+-----+-----+  
2 rows in set (0.00 sec)  
mysql>
```

## MySQL ORDER BY: using both ASC and DESC attributes

Execute the following query:

```
SELECT officer_name, address
FROM officers
WHERE officer_id < 5
ORDER BY officer_name DESC, address ASC;
```

Output:



```
mysql> SELECT officer_name, address
-> FROM officers
-> WHERE officer_id < 5
-> ORDER BY officer_name DESC, address ASC;
+-----+-----+
| officer_name | address |
+-----+-----+
| Uinal        | Faizabad |
| Rahul        | Lucknow  |
| Deepika      | Lucknow  |
| Ajeet        | Mau      |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

# MySQL GROUP BY Clause

[< prev](#)[next >](#)

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.

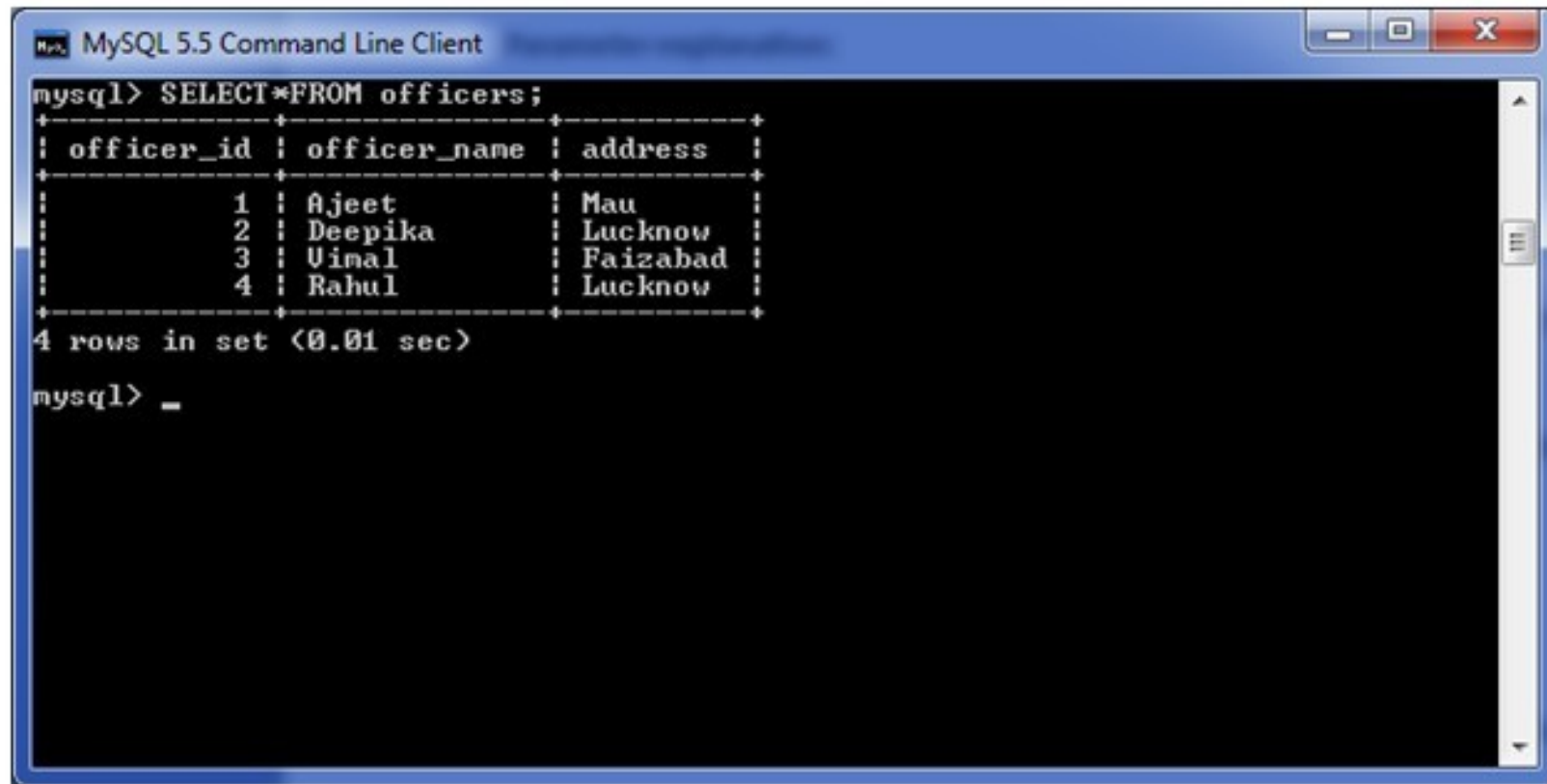
You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.

## Syntax:

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

## (i) MySQL GROUP BY Clause with COUNT function

Consider a table named "officers" table, having the following records.



```
mysql> SELECT * FROM officers;
```

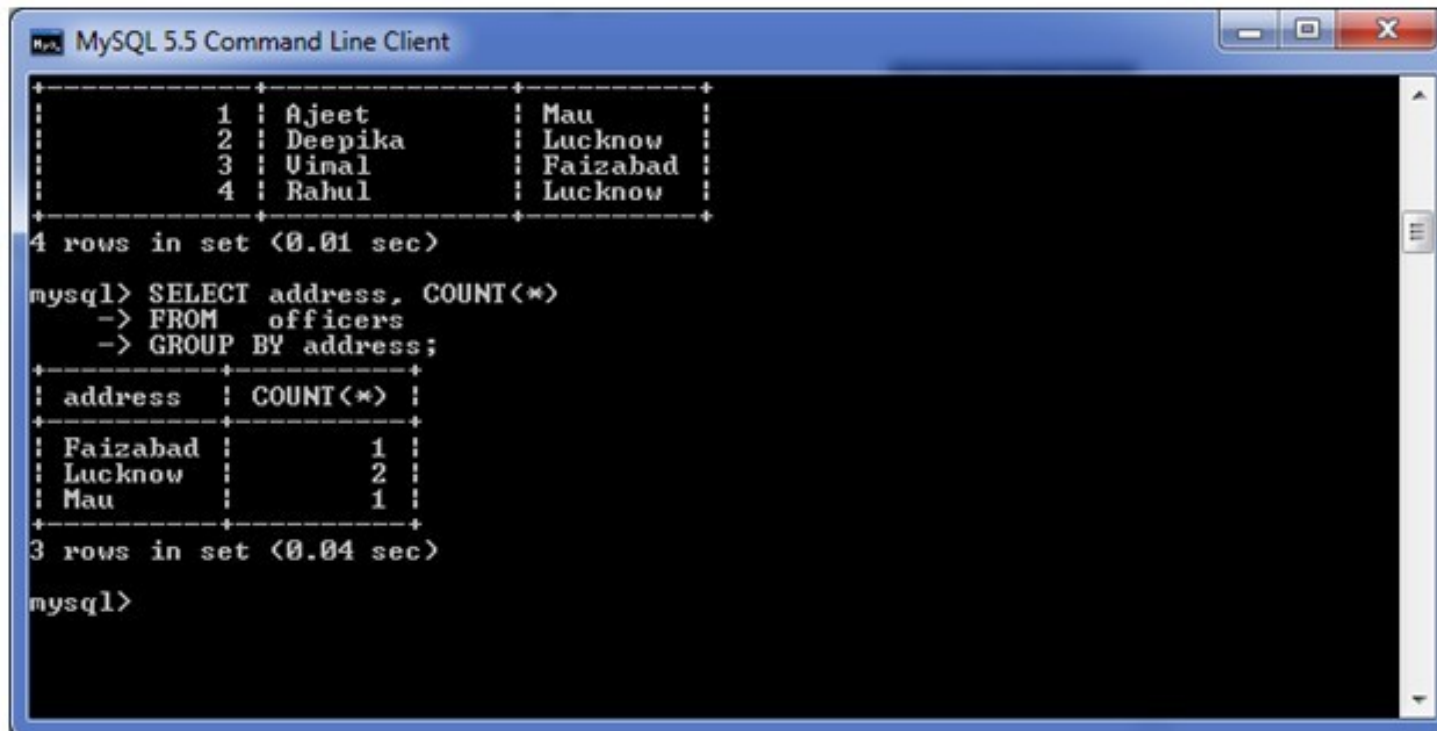
officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Vinal	Faizabad
4	Rahul	Lucknow

```
4 rows in set (0.01 sec)  
mysql> _
```

Execute the following query:

```
SELECT address, COUNT(*)  
FROM officers  
GROUP BY address;
```

Output:

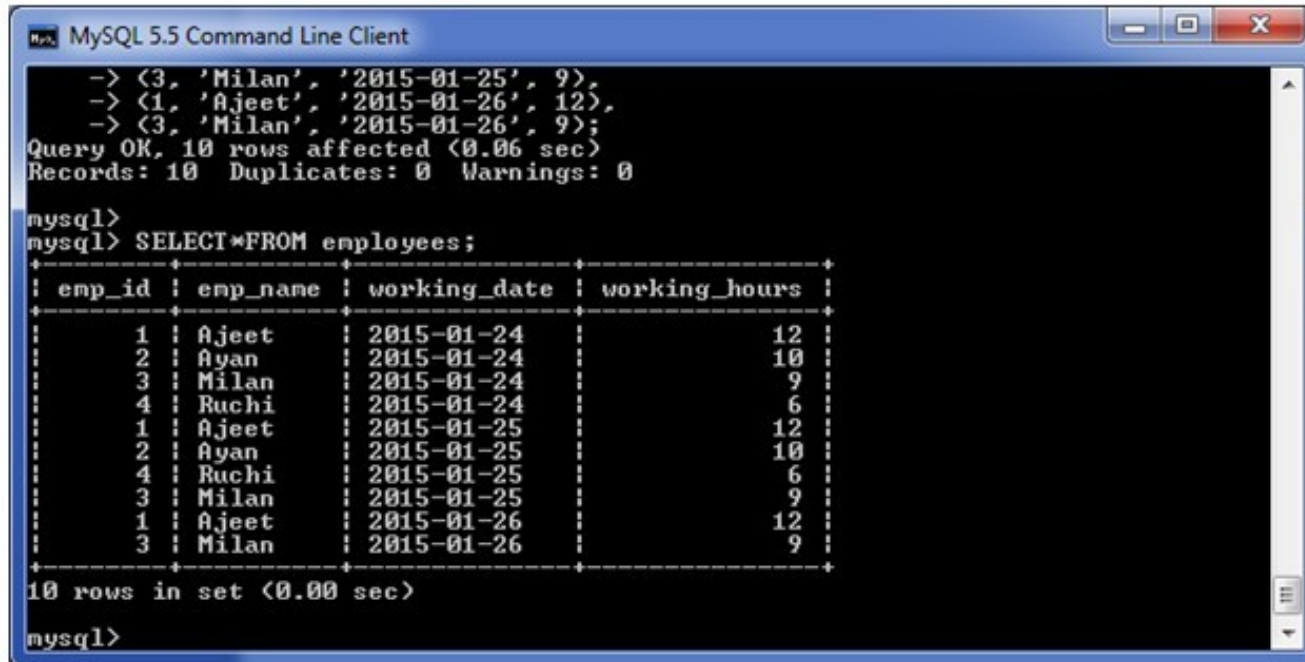


The screenshot shows a MySQL 5.5 Command Line Client window. It displays the results of a query on the 'officers' table, followed by the execution of a new query that counts the number of officers for each address.

```
MySQL 5.5 Command Line Client  
+-----+  
| 1 | Ajeet | Mau |  
| 2 | Deepika | Lucknow |  
| 3 | Vimal | Faizabad |  
| 4 | Rahul | Lucknow |  
+-----+  
4 rows in set (0.01 sec)  
  
mysql> SELECT address, COUNT(*)  
-> FROM officers  
-> GROUP BY address;  
+-----+  
| address | COUNT(*) |  
+-----+  
| Faizabad | 1 |  
| Lucknow | 2 |  
| Mau | 1 |  
+-----+  
3 rows in set (0.04 sec)  
  
mysql>
```

## (ii) MySQL GROUP BY Clause with SUM function

Let's take a table "employees" table, having the following data.



```
MySQL 5.5 Command Line Client
-> <3, 'Milan', '2015-01-25', 9>,
-> <1, 'Ajeet', '2015-01-26', 12>,
-> <3, 'Milan', '2015-01-26', 9>;
Query OK, 10 rows affected (0.06 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql>
mysql> SELECT * FROM employees;
+----+-----+-----+-----+
| emp_id | emp_name | working_date | working_hours |
+----+-----+-----+-----+
| 1 | Ajeet | 2015-01-24 | 12 |
| 2 | Ayan | 2015-01-24 | 10 |
| 3 | Milan | 2015-01-24 | 9 |
| 4 | Ruchi | 2015-01-24 | 6 |
| 1 | Ajeet | 2015-01-25 | 12 |
| 2 | Ayan | 2015-01-25 | 10 |
| 4 | Ruchi | 2015-01-25 | 6 |
| 3 | Milan | 2015-01-25 | 9 |
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

Now, the following query will GROUP BY the example using the SUM function and return the emp\_name and total working hours of each employee.

Execute the following query:

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



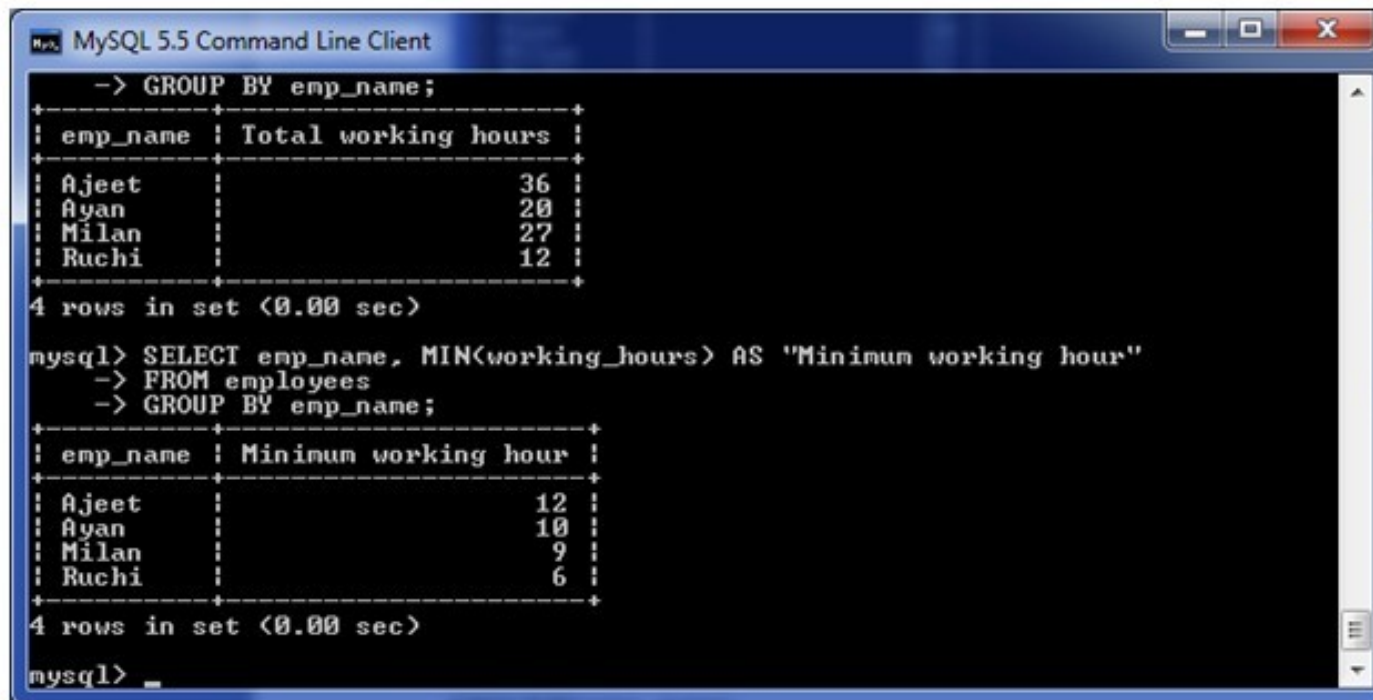
### (iii) MySQL GROUP BY Clause with MIN function

The following example specifies the minimum working hours of the employees from the table "employees".

Execute the following query:

```
SELECT emp_name, MIN(working_hours) AS "Minimum working hour"
FROM employees
GROUP BY emp_name;
```

Output:



```
MySQL 5.5 Command Line Client
-> GROUP BY emp_name;
+-----+-----+
| emp_name | Total working hours |
+-----+-----+
| Ajeet    | 36                  |
| Ayan     | 20                  |
| Milan    | 27                  |
| Ruchi     | 12                  |
+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT emp_name, MIN(working_hours) AS "Minimum working hour"
-> FROM employees
-> GROUP BY emp_name;
+-----+-----+
| emp_name | Minimum working hour |
+-----+-----+
| Ajeet    | 12                   |
| Ayan     | 10                   |
| Milan    | 9                    |
| Ruchi     | 6                    |
+-----+-----+
4 rows in set (0.00 sec)

mysql> _
```

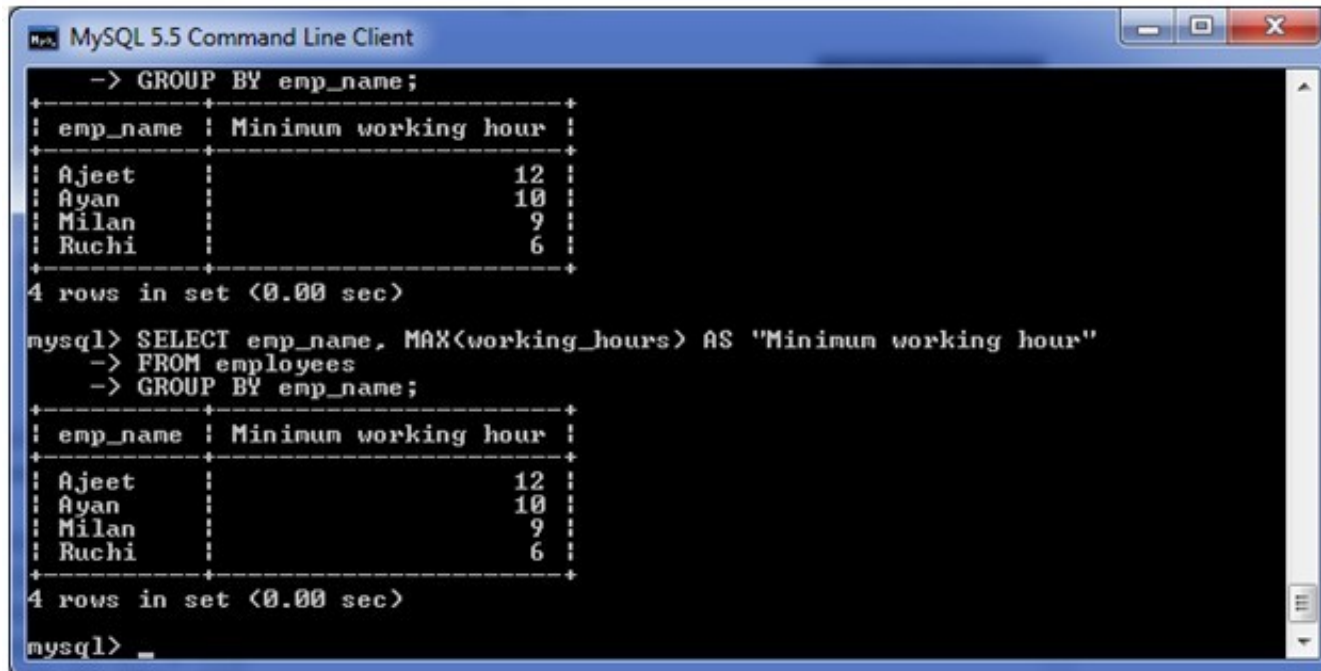
## (iv) MySQL GROUP BY Clause with MAX function

The following example specifies the maximum working hours of the employees from the table "employees".

Execute the following query:

```
SELECT emp_name, MAX (working_hours) AS "Minimum working hour"  
FROM employees  
GROUP BY emp_name;
```

Output:



```
MySQL 5.5 Command Line Client  
-> GROUP BY emp_name;  
+-----+-----+  
| emp_name | Minimum working hour |  
+-----+-----+  
| Ajeet    | 12                    |  
| Ayan     | 10                    |  
| Milan    | 9                    |  
| Ruchi    | 6                    |  
+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> SELECT emp_name, MAX(working_hours) AS "Minimum working hour"  
-> FROM employees  
-> GROUP BY emp_name;  
+-----+-----+  
| emp_name | Minimum working hour |  
+-----+-----+  
| Ajeet    | 12                    |  
| Ayan     | 10                    |  
| Milan    | 9                    |  
| Ruchi    | 6                    |  
+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> _
```



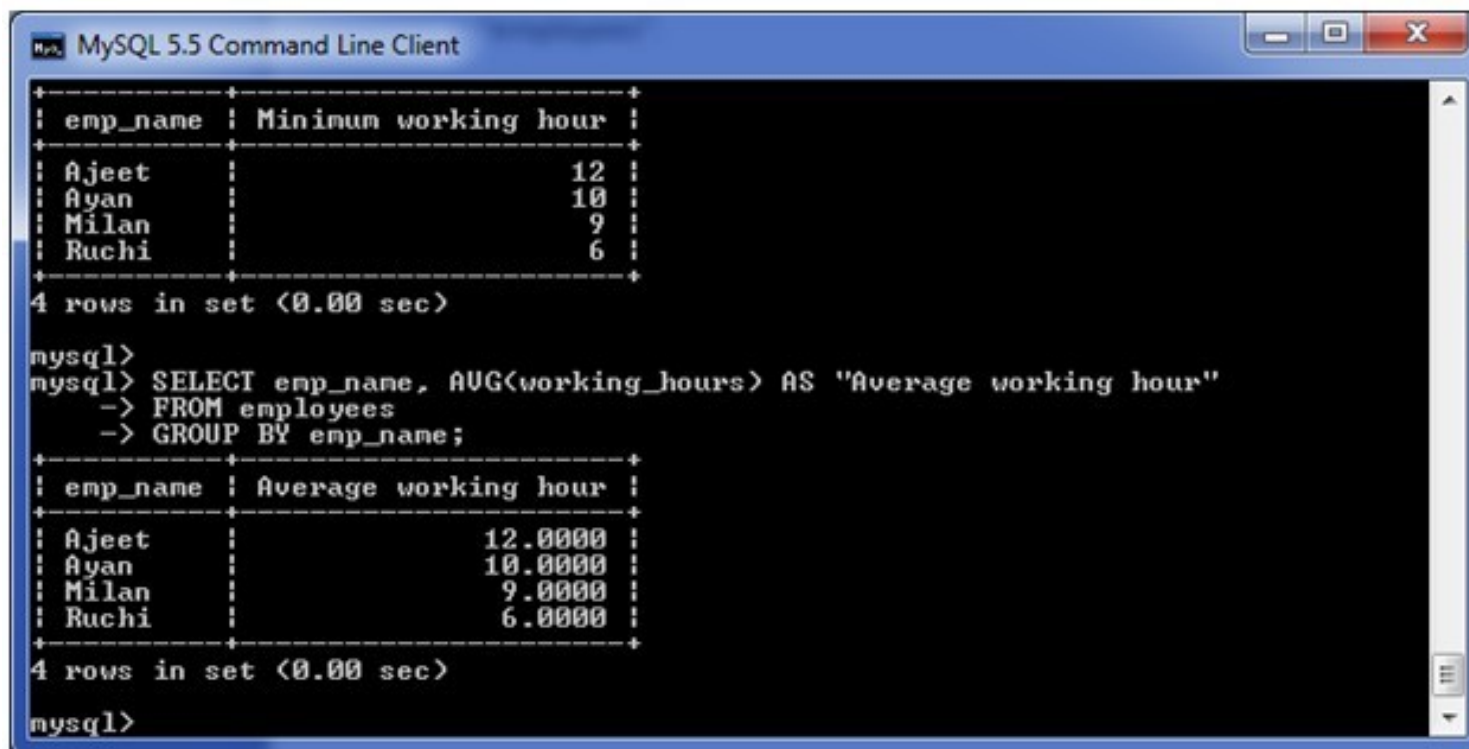
## (v) MySQL GROUP BY Clause with AVG function

The following example specifies the average working hours of the employees form the table "employees".

Execute the following query:

```
SELECT emp_name, AVG(working_hours) AS "Average working hour"
FROM employees
GROUP BY emp_name;
```

Output:



```
MySQL 5.5 Command Line Client
+-----+
| emp_name | Minimum working hour |
+-----+
| Ajeet    | 12                    |
| Ayan     | 10                    |
| Milan    | 9                     |
| Ruchi    | 6                     |
+-----+
4 rows in set (0.00 sec)

mysql>
mysql> SELECT emp_name, AVG(working_hours) AS "Average working hour"
-> FROM employees
-> GROUP BY emp_name;
+-----+
| emp_name | Average working hour |
+-----+
| Ajeet    | 12.0000              |
| Ayan     | 10.0000              |
| Milan    | 9.0000               |
| Ruchi    | 6.0000               |
+-----+
4 rows in set (0.00 sec)

mysql>
```

# MySQL HAVING Clause

[← prev](#)[next →](#)

MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.

**Syntax:**

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

# HAVING Clause with SUM function

Consider a table "employees" table having the following data.

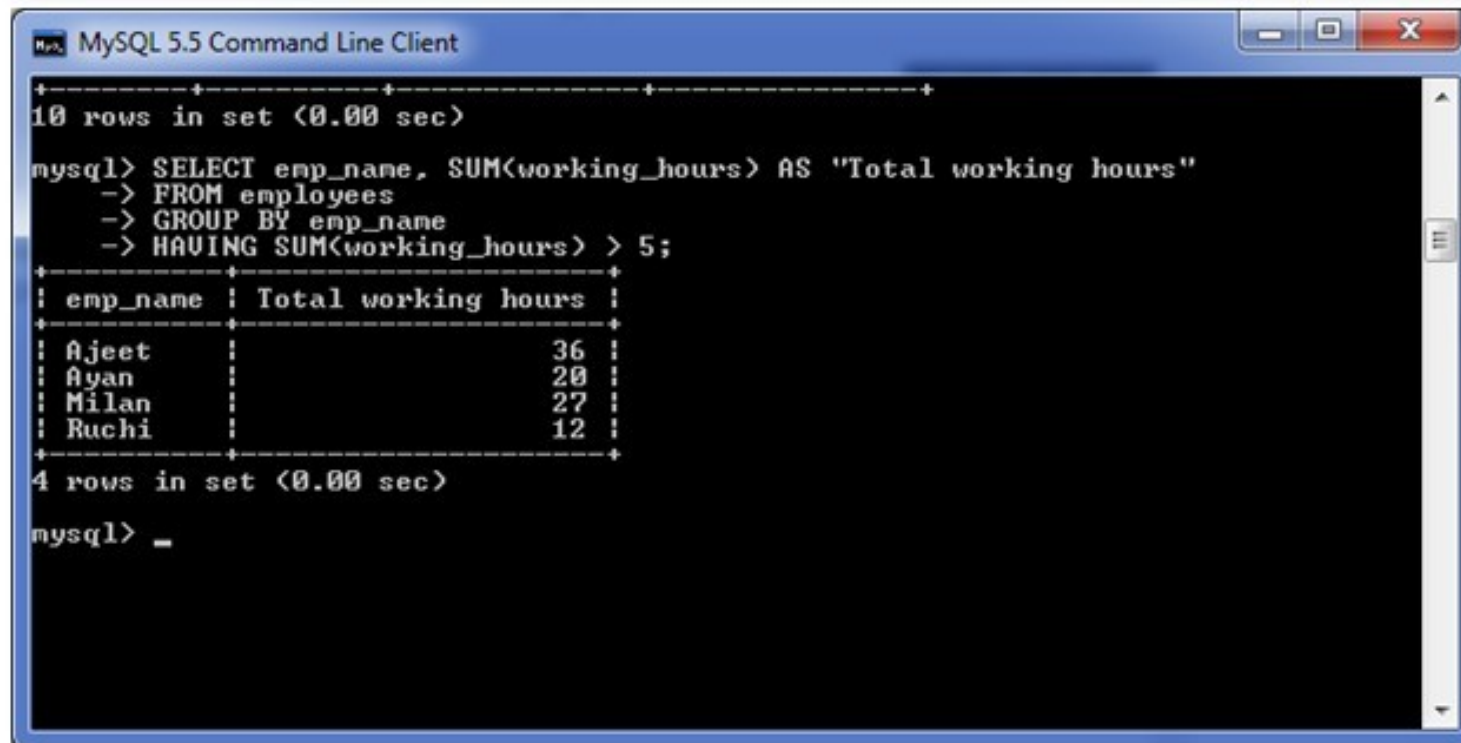
```
MySQL 5.5 Command Line Client
-> (3, 'Milan', '2015-01-25', 9),
-> (1, 'Ajeet', '2015-01-26', 12),
-> (3, 'Milan', '2015-01-26', 9);
Query OK, 10 rows affected (0.06 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql>
mysql> SELECT * FROM employees;
+-----+-----+-----+-----+
| emp_id | emp_name | working_date | working_hours |
+-----+-----+-----+-----+
| 1 | Ajeet | 2015-01-24 | 12 |
| 2 | Ayan | 2015-01-24 | 10 |
| 3 | Milan | 2015-01-24 | 9 |
| 4 | Ruchi | 2015-01-24 | 6 |
| 1 | Ajeet | 2015-01-25 | 12 |
| 2 | Ayan | 2015-01-25 | 10 |
| 4 | Ruchi | 2015-01-25 | 6 |
| 3 | Milan | 2015-01-25 | 9 |
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

Execute the following query:

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



The screenshot shows a MySQL 5.5 Command Line Client window. The terminal displays the execution of a SQL query and its results. The query is: `SELECT emp_name, SUM(working_hours) AS "Total working hours" FROM employees GROUP BY emp_name HAVING SUM(working_hours) > 5;`. The results show 4 rows in the set, with columns `emp_name` and `Total working hours`. The data is as follows:

emp_name	Total working hours
Ajeet	36
Ayan	20
Milan	27
Ruchi	12

The terminal also shows the prompt `mysql>` and the command `_` at the bottom.

# MySQL IS NULL Condition

[< prev](#)[next >](#)

MySQL IS NULL condition is used to check if there is a NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statement.

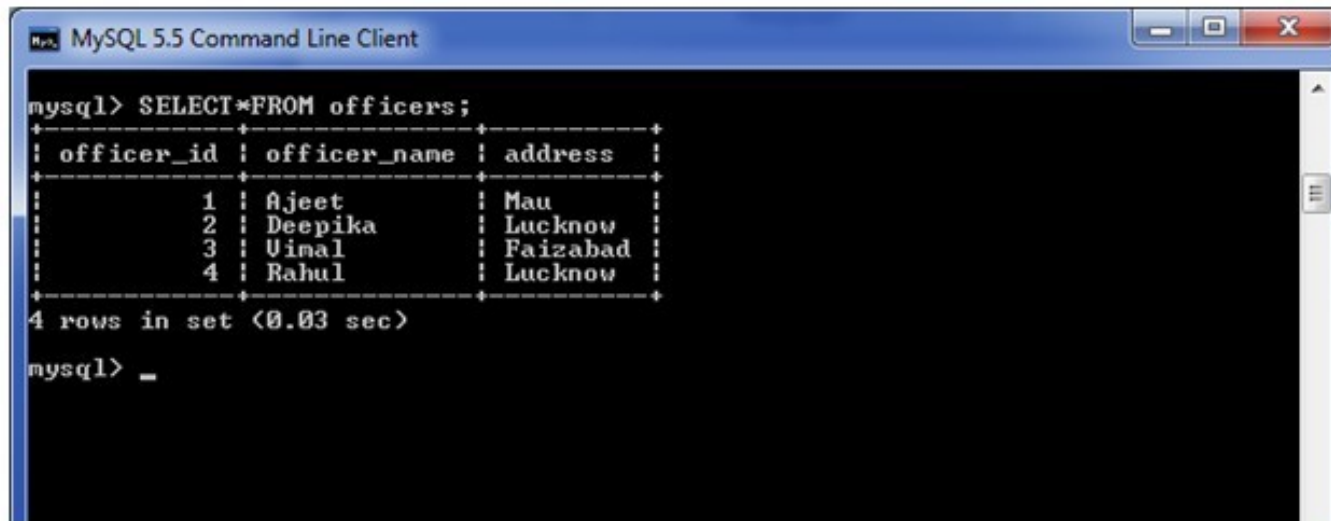
**Syntax:**

```
expression IS NULL
```

## Parameter

**expression:** It specifies a value to test if it is NULL value.

Consider a table "officers" having the following data.



```
mysql> SELECT * FROM officers;
```

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uinal	Faizabad
4	Rahul	Lucknow

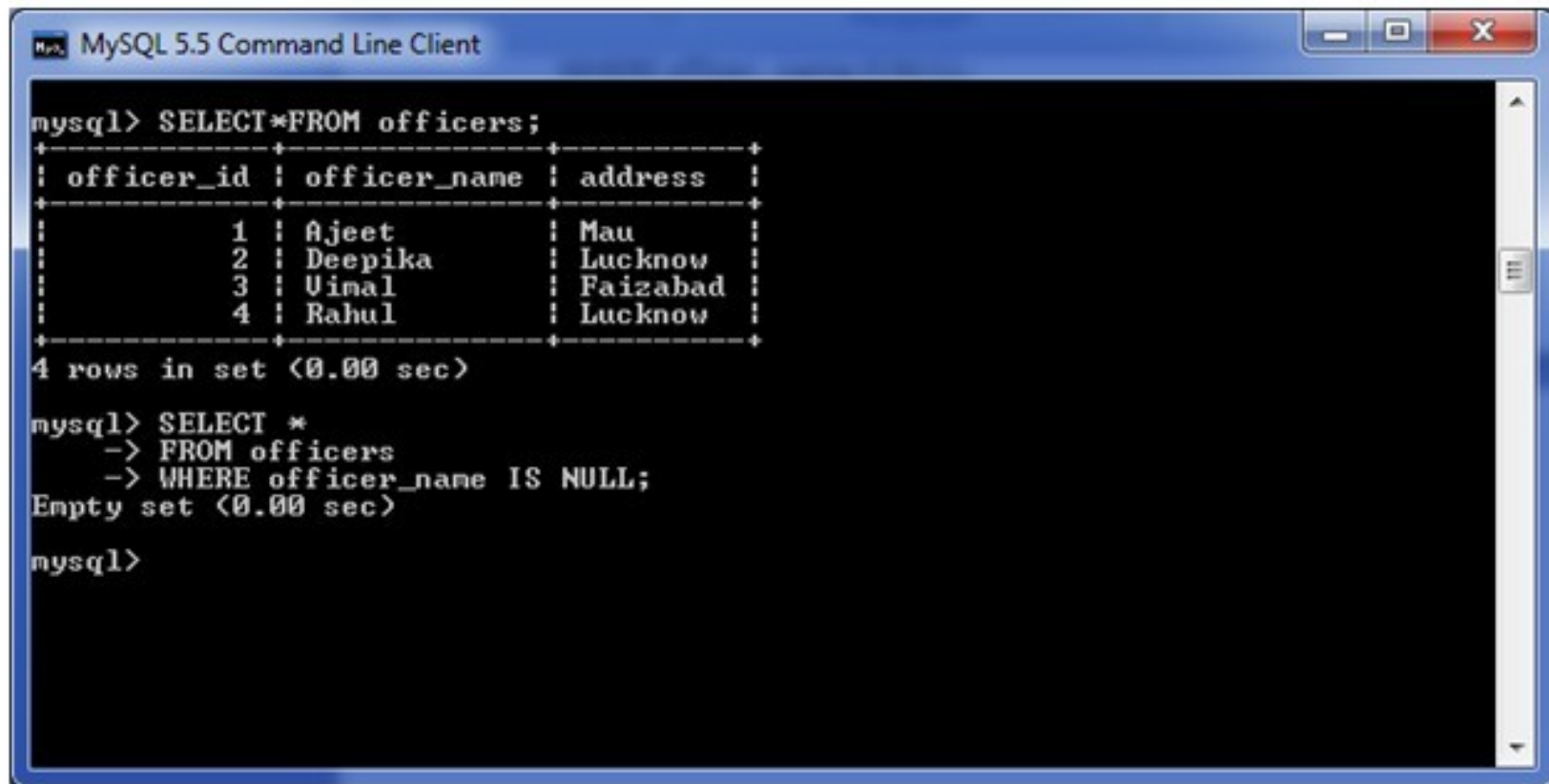
```
4 rows in set (0.03 sec)

mysql> _
```

Execute the following query:

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

Output:



The screenshot shows a MySQL 5.5 Command Line Client window. The first query executed is `mysql> SELECT*FROM officers;`, which returns a table with 4 rows. The second query is `mysql> SELECT *  
-> FROM officers  
-> WHERE officer_name IS NULL;`, which returns an empty set.

```
mysql> SELECT*FROM officers;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 1 | Ajeet | Mau |  
| 2 | Deepika | Lucknow |  
| 3 | Vinal | Faizabad |  
| 4 | Rahul | Lucknow |  
+-----+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> SELECT *  
-> FROM officers  
-> WHERE officer_name IS NULL;  
Empty set (0.00 sec)  
  
mysql>
```

# MySQL IN Condition

[← prev](#)[next →](#)

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

## Syntax:

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

# MySQL IN Example

Consider a table "officers", having the following data.



```
MySQL 5.5 Command Line Client
2 rows in set (0.07 sec)
mysql> SELECT * FROM officers;
+-----+-----+-----+
| officer_id | officer_name | address |
+-----+-----+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uinal | Faizabad |
| 4 | Rahul | Lucknow |
+-----+-----+-----+
4 rows in set (0.02 sec)
mysql> _
```

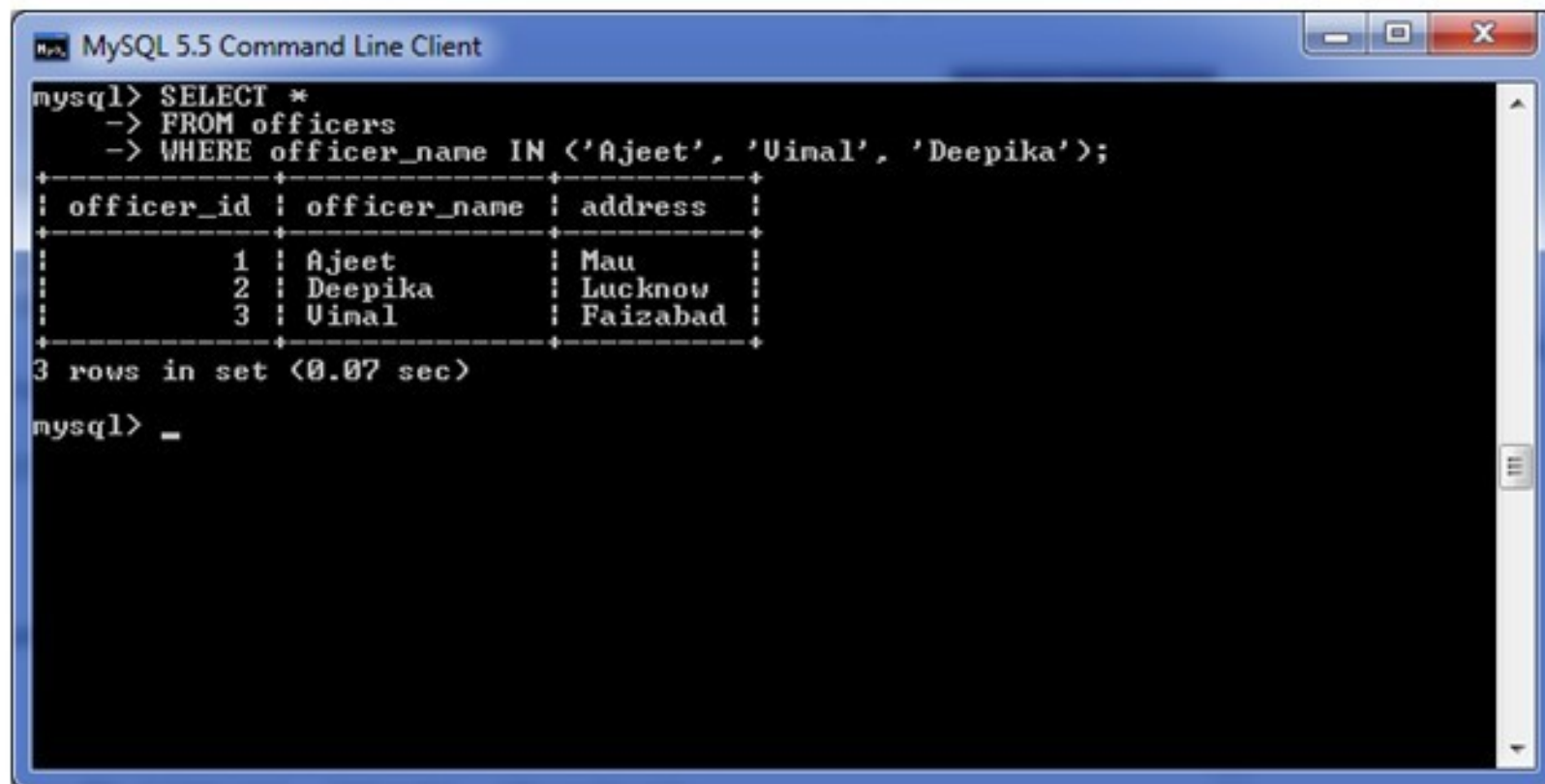
officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uinal	Faizabad
4	Rahul	Lucknow



Execute the following query:

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

Output:



The screenshot shows a MySQL 5.5 Command Line Client window. The user has entered the following SQL query:

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

The output of the query is displayed in a table format:

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Vimal	Faizabad

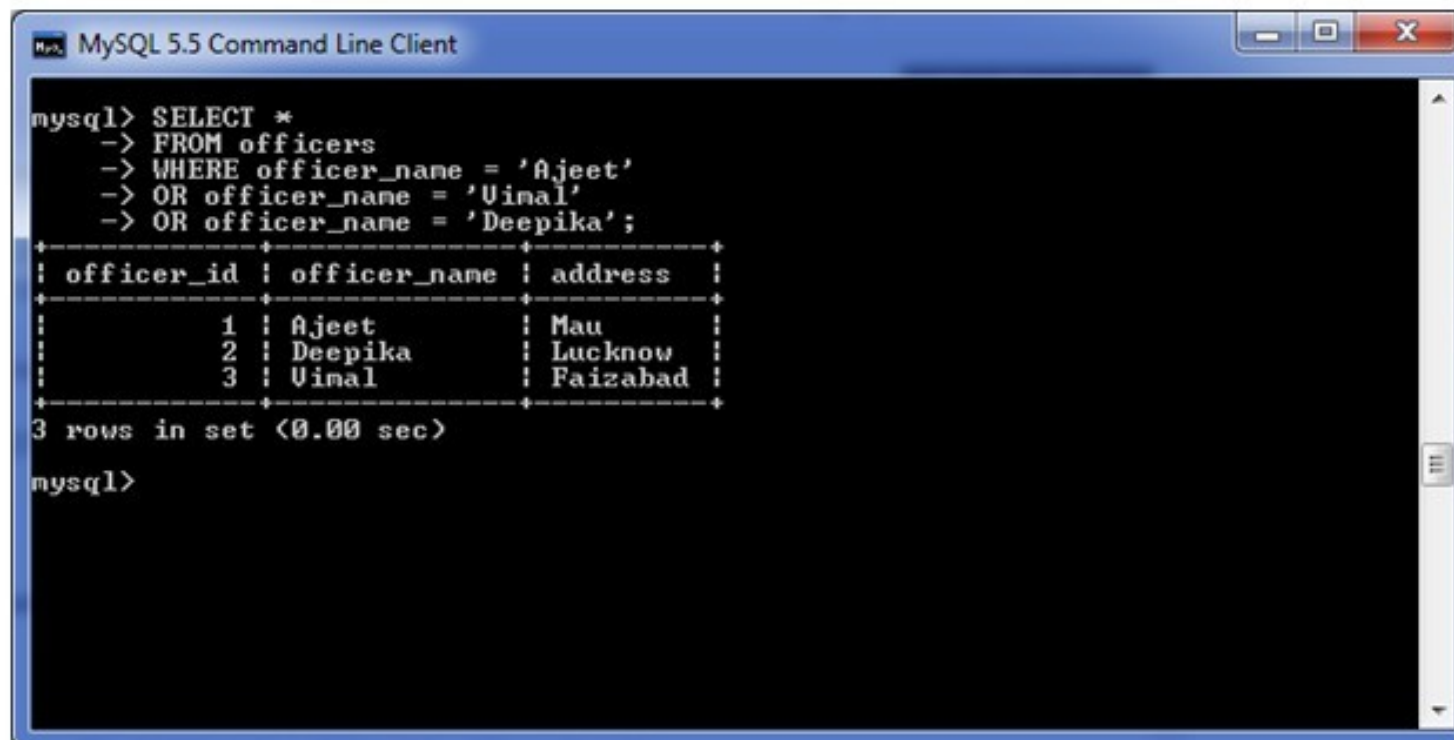
Below the table, the client reports: 3 rows in set (0.07 sec)

The prompt mysql> \_ is visible at the bottom of the window.

Execute the following query:

```
SELECT *  
FROM officers  
WHERE officer_name = 'Ajeet'  
OR officer_name = 'Vimal'  
OR officer_name = 'Deepika';
```

Output:



```
mysql> SELECT *  
-> FROM officers  
-> WHERE officer_name = 'Ajeet'  
-> OR officer_name = 'Vimal'  
-> OR officer_name = 'Deepika';  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 1 | Ajeet | Mau |  
| 2 | Deepika | Lucknow |  
| 3 | Vimal | Faizabad |  
+-----+-----+-----+  
3 rows in set (0.00 sec)  
  
mysql>
```

# MySQL IS NOT NULL Condition

[< prev](#)[next >](#)

MySQL IS NOT NULL condition is used to check the NOT NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statements.

**Syntax:**

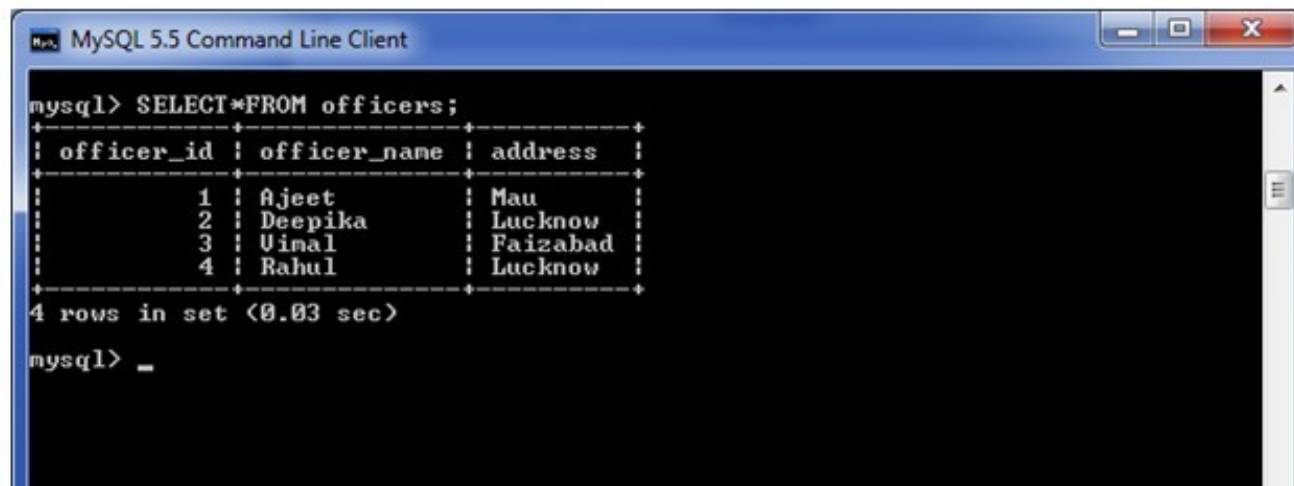
```
expression IS NOT NULL
```

## Parameter

**expression:** It specifies a value to test if it is not NULL value.

## MySQL IS NOT NULL Example

Consider a table "officers" having the following data.




```
MySQL 5.5 Command Line Client
mysql> SELECT * FROM officers;
+-----+-----+-----+
| officer_id | officer_name | address |
+-----+-----+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uimal | Faizabad |
| 4 | Rahul | Lucknow |
+-----+-----+-----+
4 rows in set (0.03 sec)

mysql> _
```

Execute the following query:

```
SELECT *  
FROM officers  
WHERE officer_name IS NOT NULL;
```

Output:



The screenshot shows a MySQL 5.5 Command Line Client window. The user has entered the following SQL query:

```
mysql> SELECT *  
-> FROM officers  
-> WHERE officer_name IS NOT NULL;
```

The output is displayed as a table with three columns: officer\_id, officer\_name, and address. The table contains four rows of data:

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Vinal	Faizabad
4	Rahul	Lucknow

Below the table, the client reports: 4 rows in set (0.00 sec). The prompt mysql> \_ is shown at the bottom.

# MySQL BETWEEN Condition

[← prev](#)[next →](#)

The MYSQL BETWEEN condition specifies how to retrieve values from an expression within a specific range. It is used with SELECT, INSERT, UPDATE and DELETE statement.

## Syntax:

```
expression BETWEEN value1 AND value2;
```

## Parameters

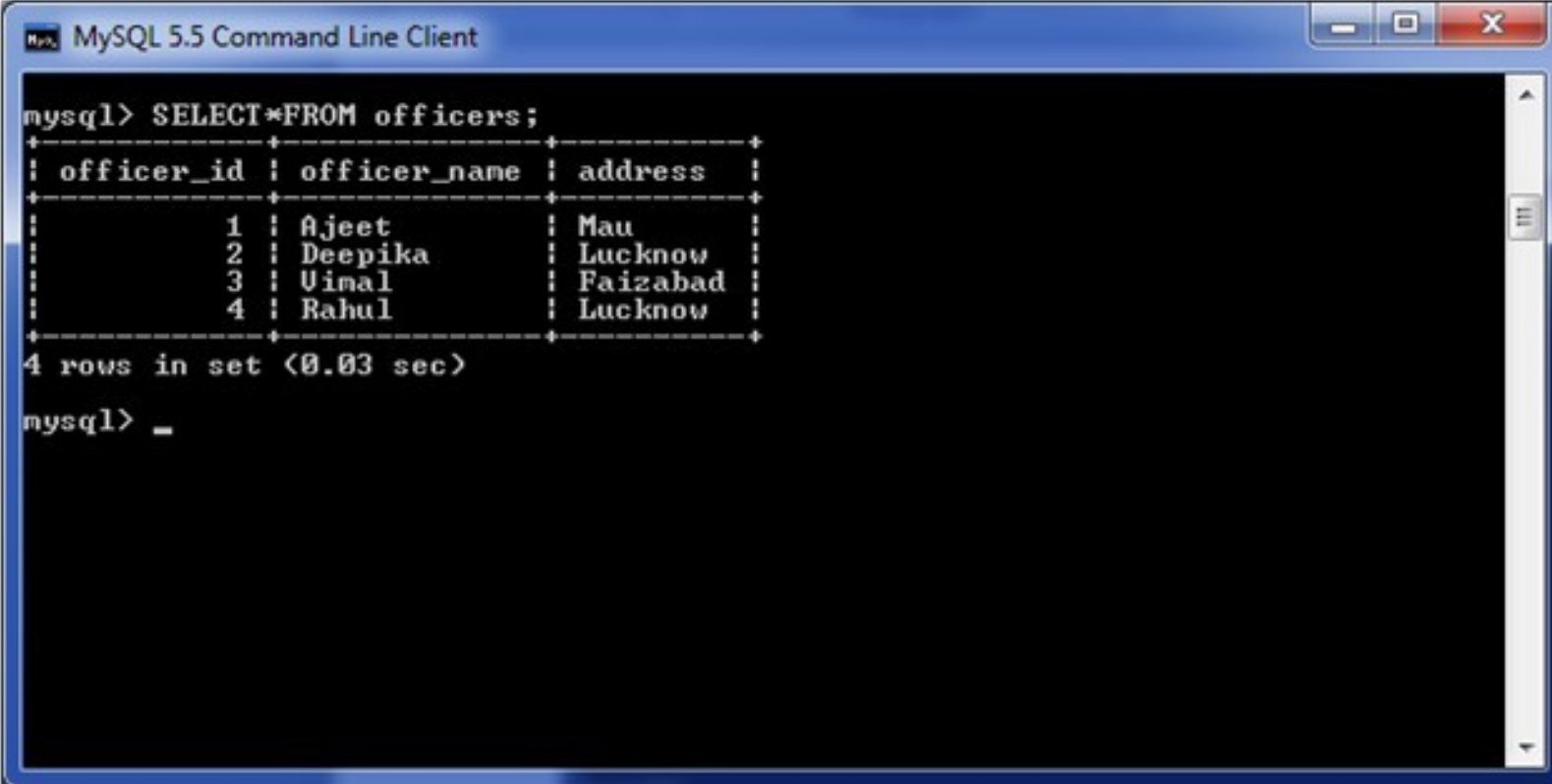
**expression:** It specifies a column.

**value1 and value2:** These values define an inclusive range that expression is compared to.

Let's take some examples.

## (i) MySQL BETWEEN condition with numeric value:

Consider a table "officers" having the following data.



```
mysql> SELECT * FROM officers;
```

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Vinal	Faizabad
4	Rahul	Lucknow

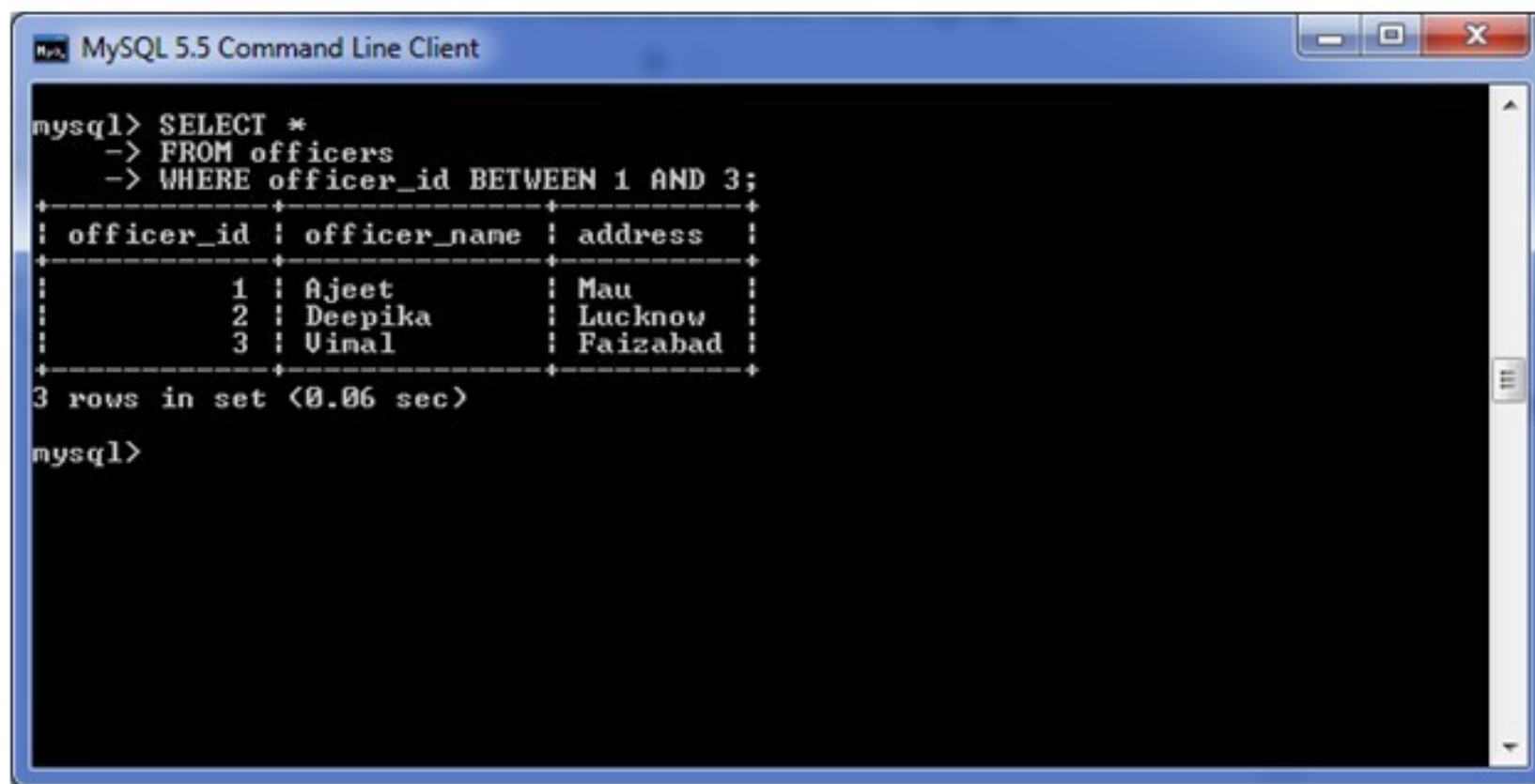
```
4 rows in set (0.03 sec)  
  
mysql> _
```

The screenshot shows a MySQL 5.5 Command Line Client window. The title bar reads "MySQL 5.5 Command Line Client". The command prompt shows the command `mysql> SELECT * FROM officers;` being executed. The output is a table with three columns: `officer_id`, `officer_name`, and `address`. The table contains four rows of data. Below the table, it says "4 rows in set (0.03 sec)". The prompt then shows `mysql> _`.

Execute the following query:

```
SELECT *  
FROM officers  
WHERE officer_id BETWEEN 1 AND 3;
```

Output:



The screenshot shows a MySQL 5.5 Command Line Client window. The user has entered the following SQL query:

```
mysql> SELECT *  
-> FROM officers  
-> WHERE officer_id BETWEEN 1 AND 3;
```

The output of the query is displayed in a table format:

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uinal	Faizabad

Below the table, the client reports: 3 rows in set (0.06 sec)

The prompt mysql> is shown at the bottom of the window.

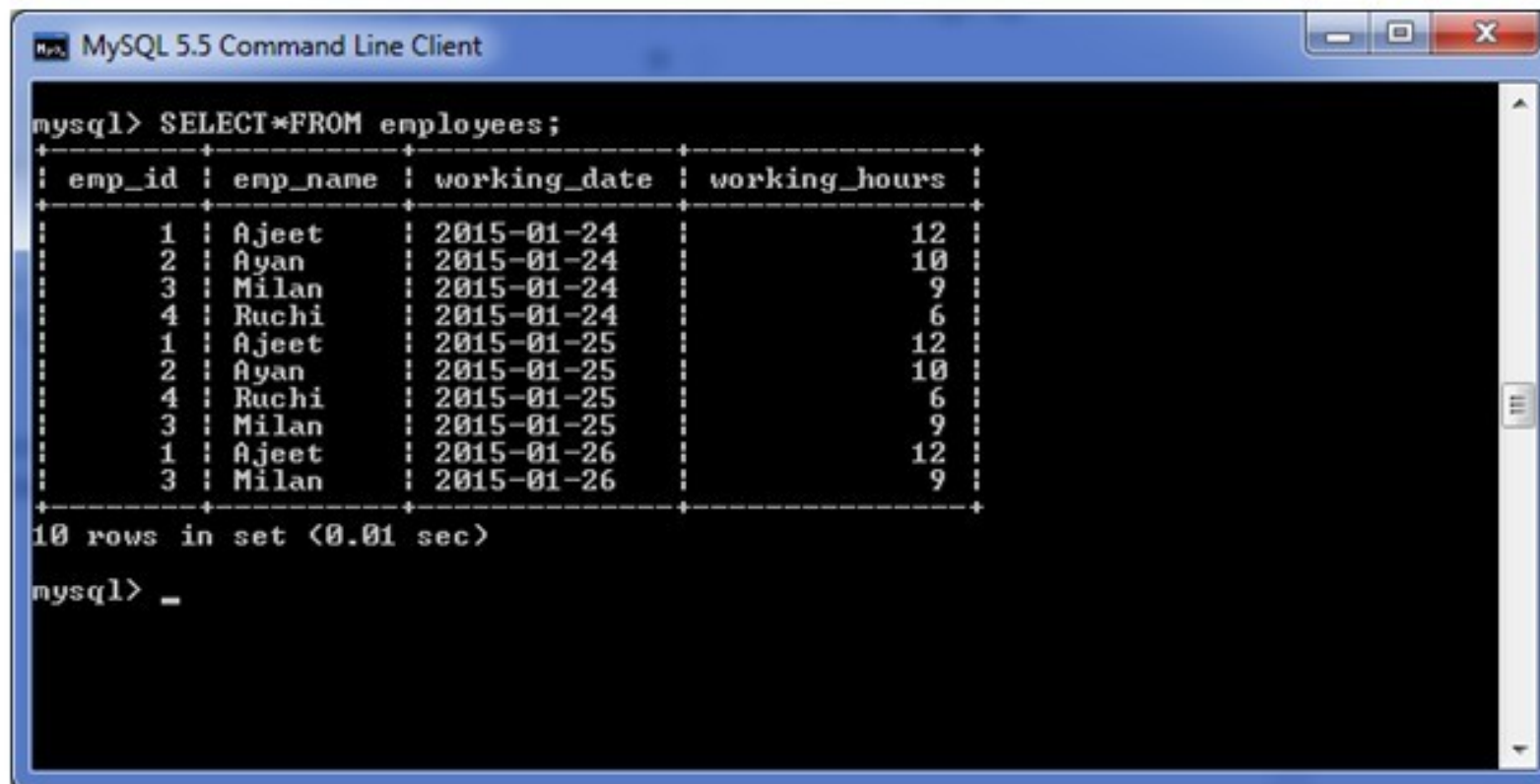


## (ii) MySQL BETWEEN condition with date:

MySQL BETWEEN condition also facilitates you to retrieve records according to date.

See this example:

Consider a table "employees", having the following data.



```
mysql> SELECT * FROM employees;
```

emp_id	emp_name	working_date	working_hours
1	Ajeet	2015-01-24	12
2	Ayan	2015-01-24	10
3	Milan	2015-01-24	9
4	Ruchi	2015-01-24	6
1	Ajeet	2015-01-25	12
2	Ayan	2015-01-25	10
4	Ruchi	2015-01-25	6
3	Milan	2015-01-25	9
1	Ajeet	2015-01-26	12
3	Milan	2015-01-26	9

```
10 rows in set (0.01 sec)
```

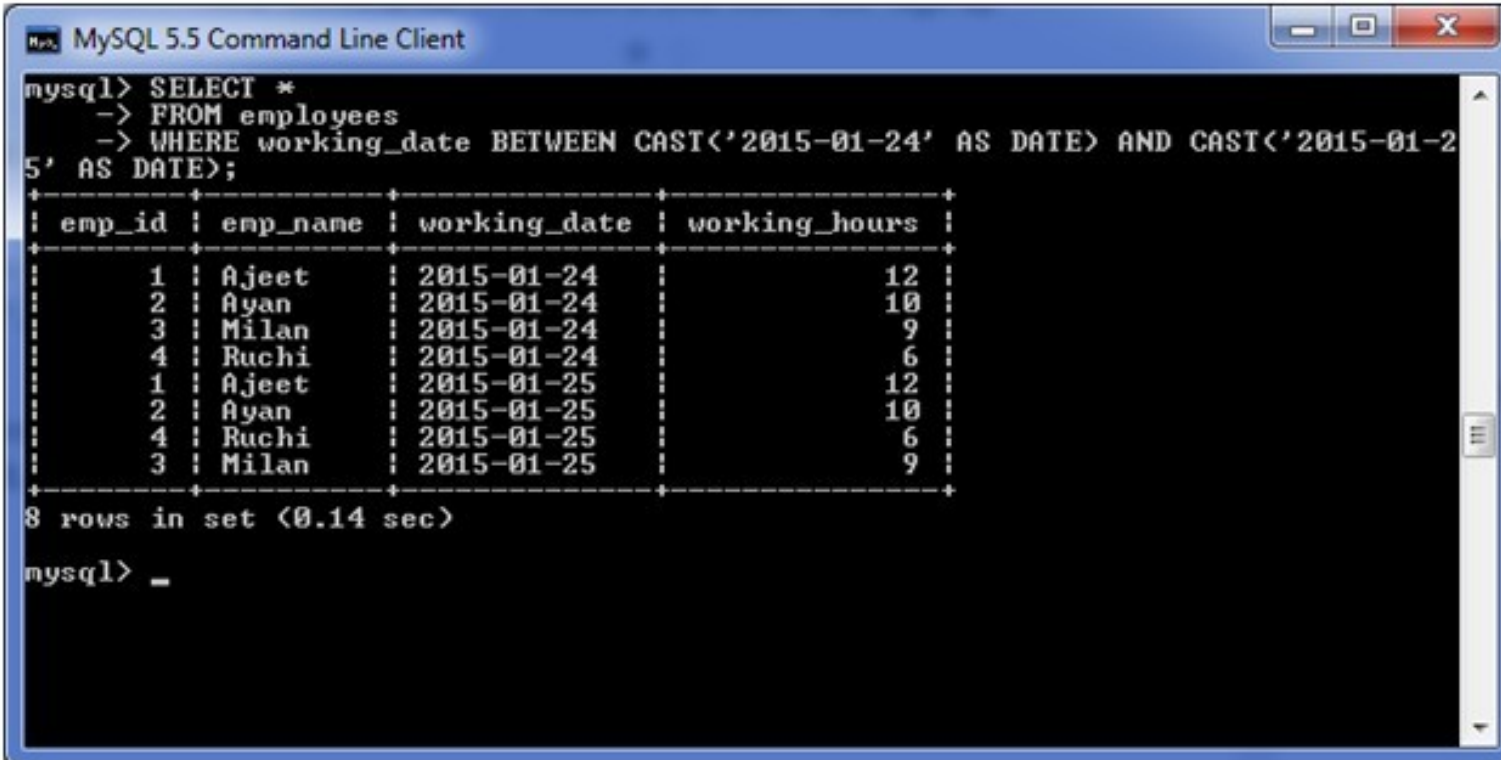
```
mysql> _
```



Execute the following query:

```
SELECT *  
FROM employees  
WHERE working_date BETWEEN CAST ('2015-01-24' AS DATE) AND CAST ('2015-01-25' AS DATE);
```

Output:



```
mysql> SELECT *  
-> FROM employees  
-> WHERE working_date BETWEEN CAST('2015-01-24' AS DATE) AND CAST('2015-01-25' AS DATE);
```

emp_id	emp_name	working_date	working_hours
1	Ajeet	2015-01-24	12
2	Ayan	2015-01-24	10
3	Milan	2015-01-24	9
4	Ruchi	2015-01-24	6
1	Ajeet	2015-01-25	12
2	Ayan	2015-01-25	10
4	Ruchi	2015-01-25	6
3	Milan	2015-01-25	9

```
8 rows in set (0.14 sec)  
mysql> _
```

# MySQL JOINS

[< prev](#)[next >](#)

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)
- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

## MySQL Inner JOIN (Simple Join)

The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

**Syntax:**

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

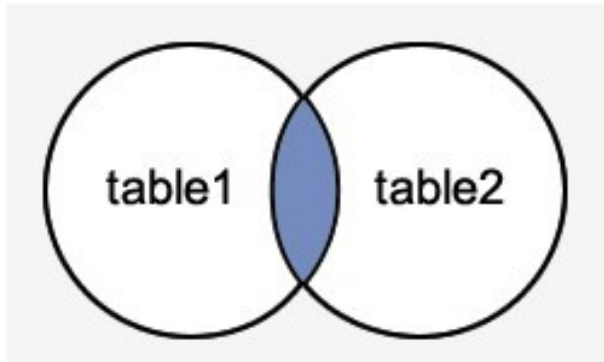
# MySQL Inner JOIN (Simple Join)

The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

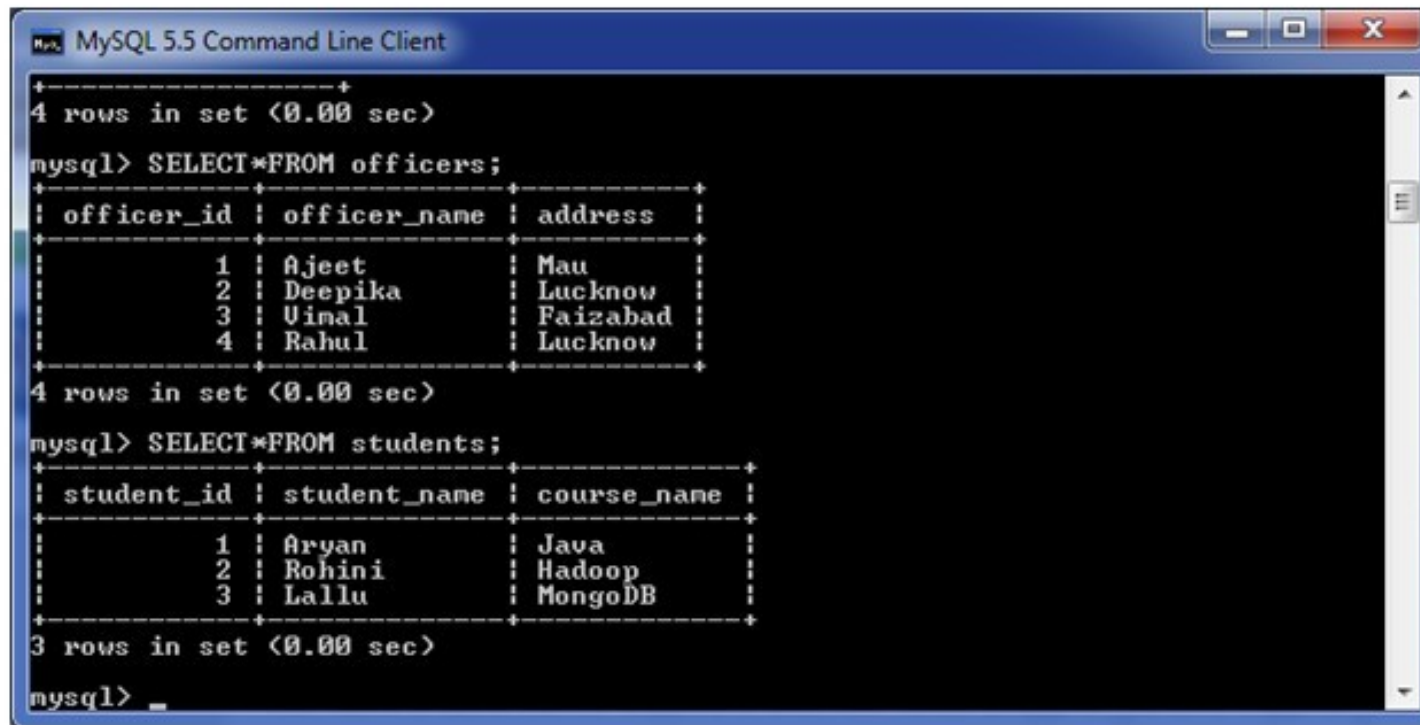
Syntax:

```
SELECT columns  
FROM table1  
INNER JOIN table2  
ON table1.column = table2.column;
```

Image representation:



Consider two tables "officers" and "students", having the following data.



```
MySQL 5.5 Command Line Client
+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM officers;
+-----+
| officer_id | officer_name | address |
+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uinal | Faizabad |
| 4 | Rahul | Lucknow |
+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM students;
+-----+
| student_id | student_name | course_name |
+-----+
| 1 | Aryan | Java |
| 2 | Rohini | Hadoop |
| 3 | Lallu | MongoDB |
+-----+
3 rows in set (0.00 sec)

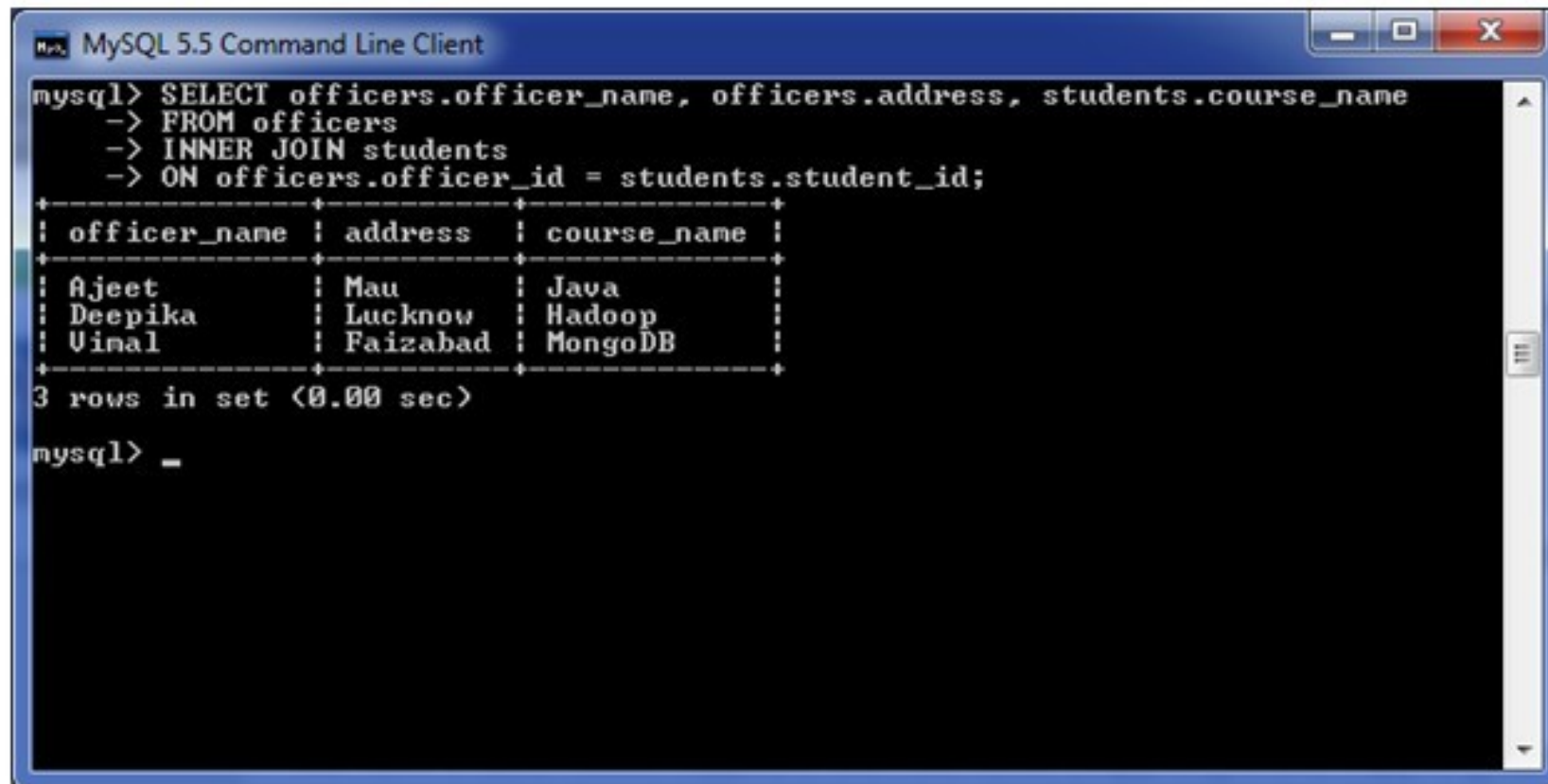
mysql> _
```

Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;
```

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;
```

Output:



The screenshot shows a terminal window titled "MySQL 5.5 Command Line Client". The user has entered the following SQL query:

```
mysql> SELECT officers.officer_name, officers.address, students.course_name
-> FROM officers
-> INNER JOIN students
-> ON officers.officer_id = students.student_id;
```

The output is displayed in a table format with three columns: officer\_name, address, and course\_name. The data is as follows:

officer_name	address	course_name
Ajeet	Mau	Java
Deepika	Lucknow	Hadoop
Uinal	Faizabad	MongoDB

Below the table, the text "3 rows in set (0.00 sec)" is displayed. The prompt "mysql> \_" is visible at the bottom of the terminal.

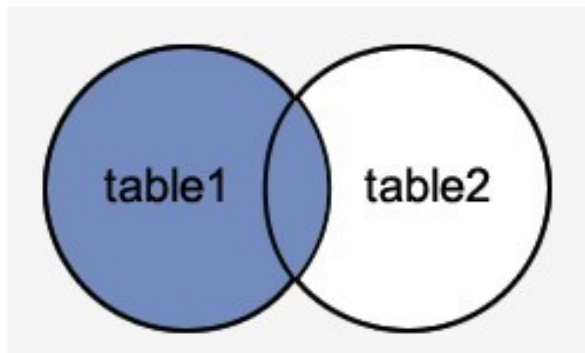
# MySQL Left Outer Join

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

**Syntax:**

```
SELECT columns  
FROM table1  
LEFT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

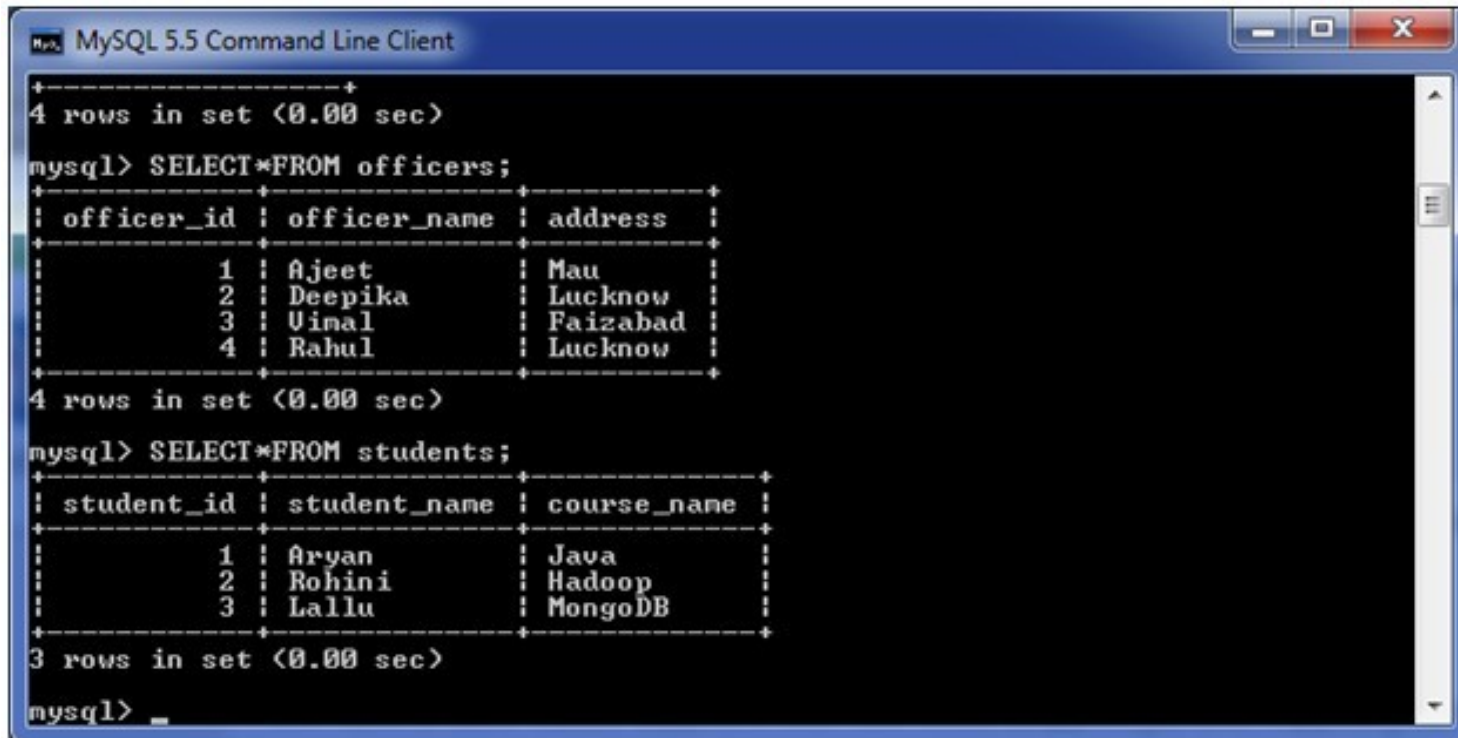
**Image representation:**





Let's take an example:

Consider two tables "officers" and "students", having the following data.



```
MySQL 5.5 Command Line Client
+-----+
4 rows in set (0.00 sec)
mysql> SELECT * FROM officers;
+-----+
| officer_id | officer_name | address |
+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Vinal | Faizabad |
| 4 | Rahul | Lucknow |
+-----+
4 rows in set (0.00 sec)
mysql> SELECT * FROM students;
+-----+
| student_id | student_name | course_name |
+-----+
| 1 | Aryan | Java |
| 2 | Rohini | Hadoop |
| 3 | Lallu | MongoDB |
+-----+
3 rows in set (0.00 sec)
mysql> _
```

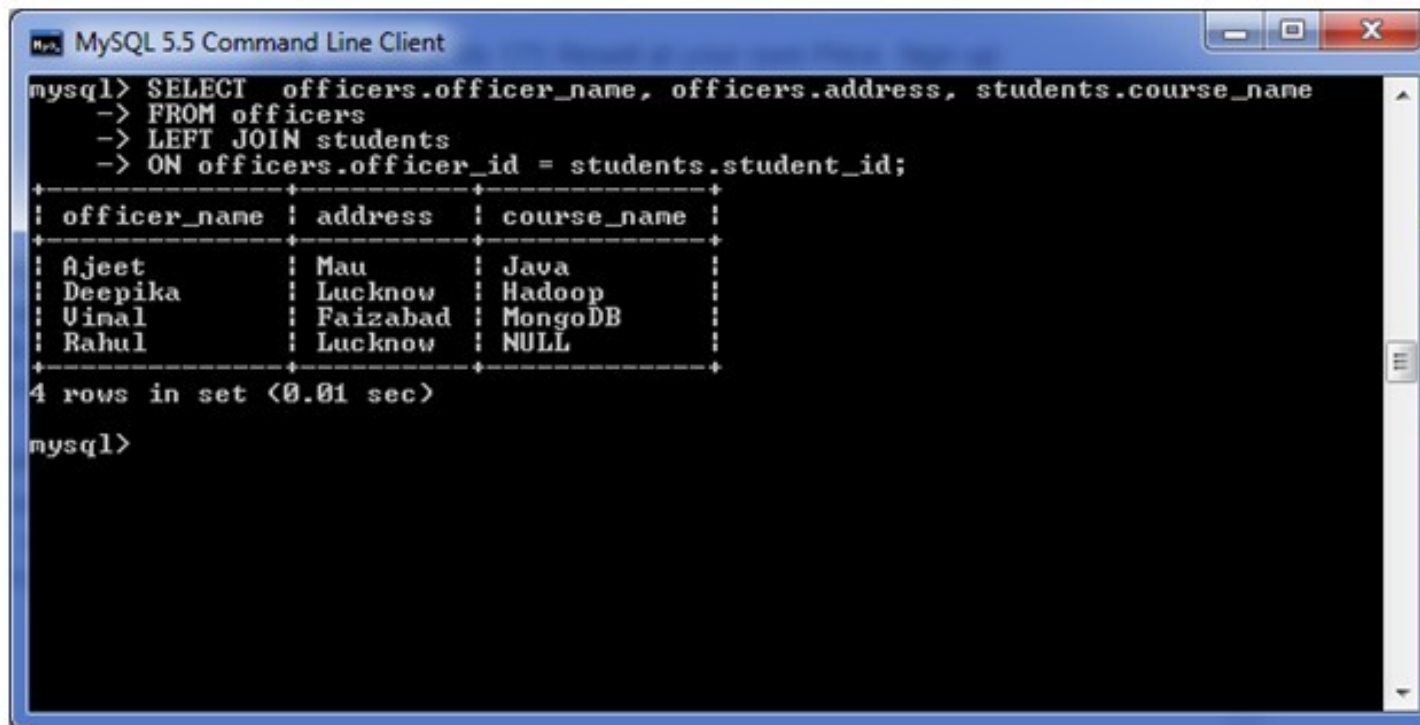
Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

Output:



The screenshot shows a MySQL 5.5 Command Line Client window. The user has entered a SQL query to select officer names, addresses, and course names from the officers and students tables. The output is displayed in a table format with 4 rows. The first three rows contain data, and the fourth row shows NULL for the course\_name.

```
mysql> SELECT officers.officer_name, officers.address, students.course_name
-> FROM officers
-> LEFT JOIN students
-> ON officers.officer_id = students.student_id;
+-----+-----+-----+
| officer_name | address | course_name |
+-----+-----+-----+
| Ajeet       | Mau    | Java       |
| Deepika     | Lucknow | Hadoop     |
| Uinal       | Faizabad | MongoDB   |
| Rahul       | Lucknow | NULL       |
+-----+-----+-----+
4 rows in set (0.01 sec)

mysql>
```



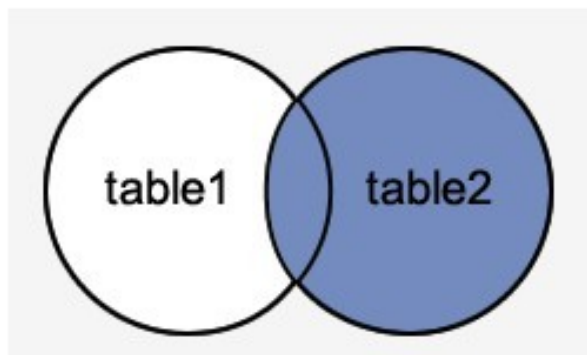
# MySQL Right Outer Join

The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

**Syntax:**

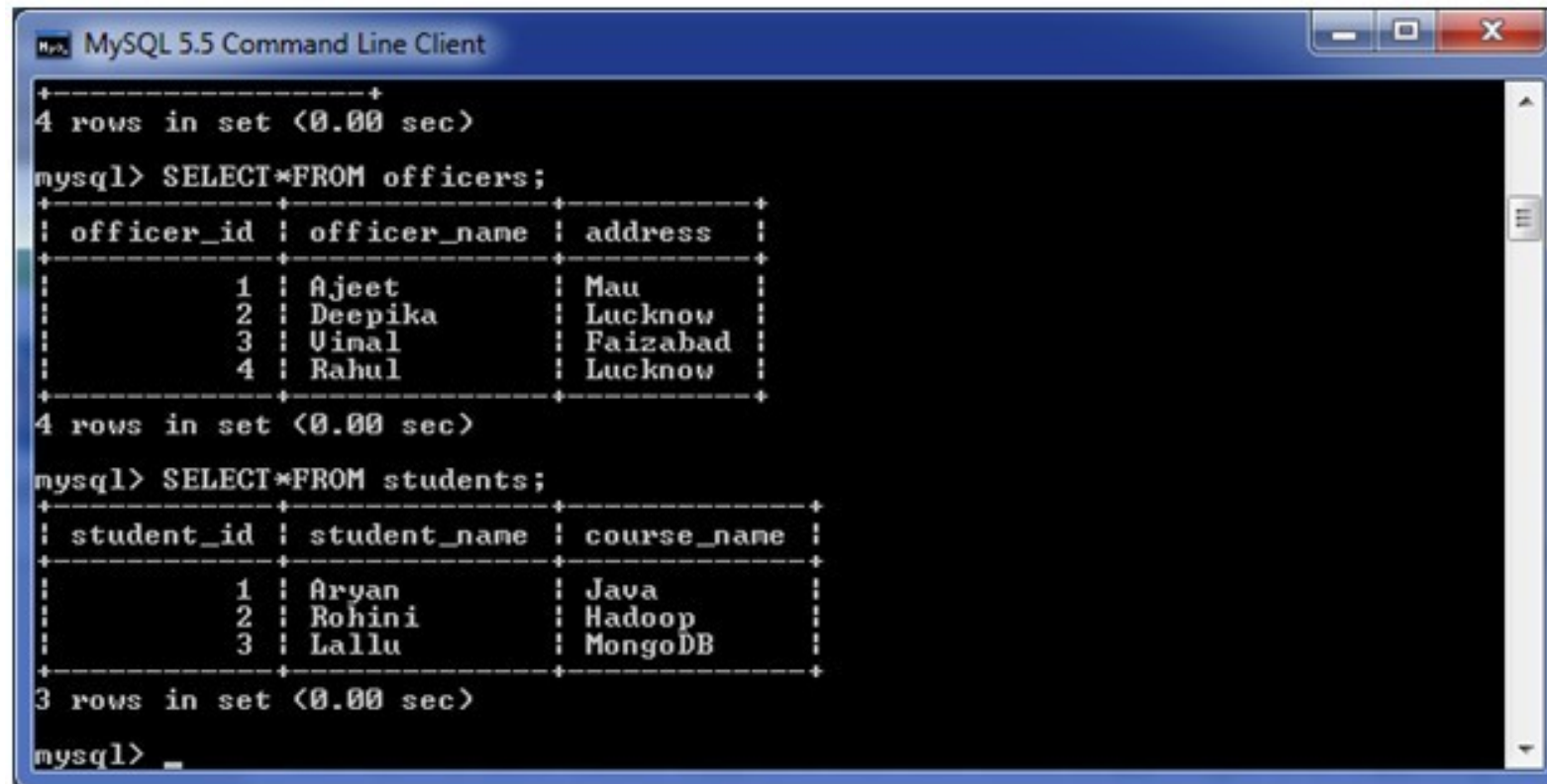
```
SELECT columns  
FROM table1  
RIGHT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

**Image representation:**



Let's take an example:

Consider two tables "officers" and "students", having the following data.



```
MySQL 5.5 Command Line Client
+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM officers;
+-----+
| officer_id | officer_name | address |
+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uinal | Faizabad |
| 4 | Rahul | Lucknow |
+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM students;
+-----+
| student_id | student_name | course_name |
+-----+
| 1 | Aryan | Java |
| 2 | Rohini | Hadoop |
| 3 | Lallu | MongoDB |
+-----+
3 rows in set (0.00 sec)

mysql> _
```

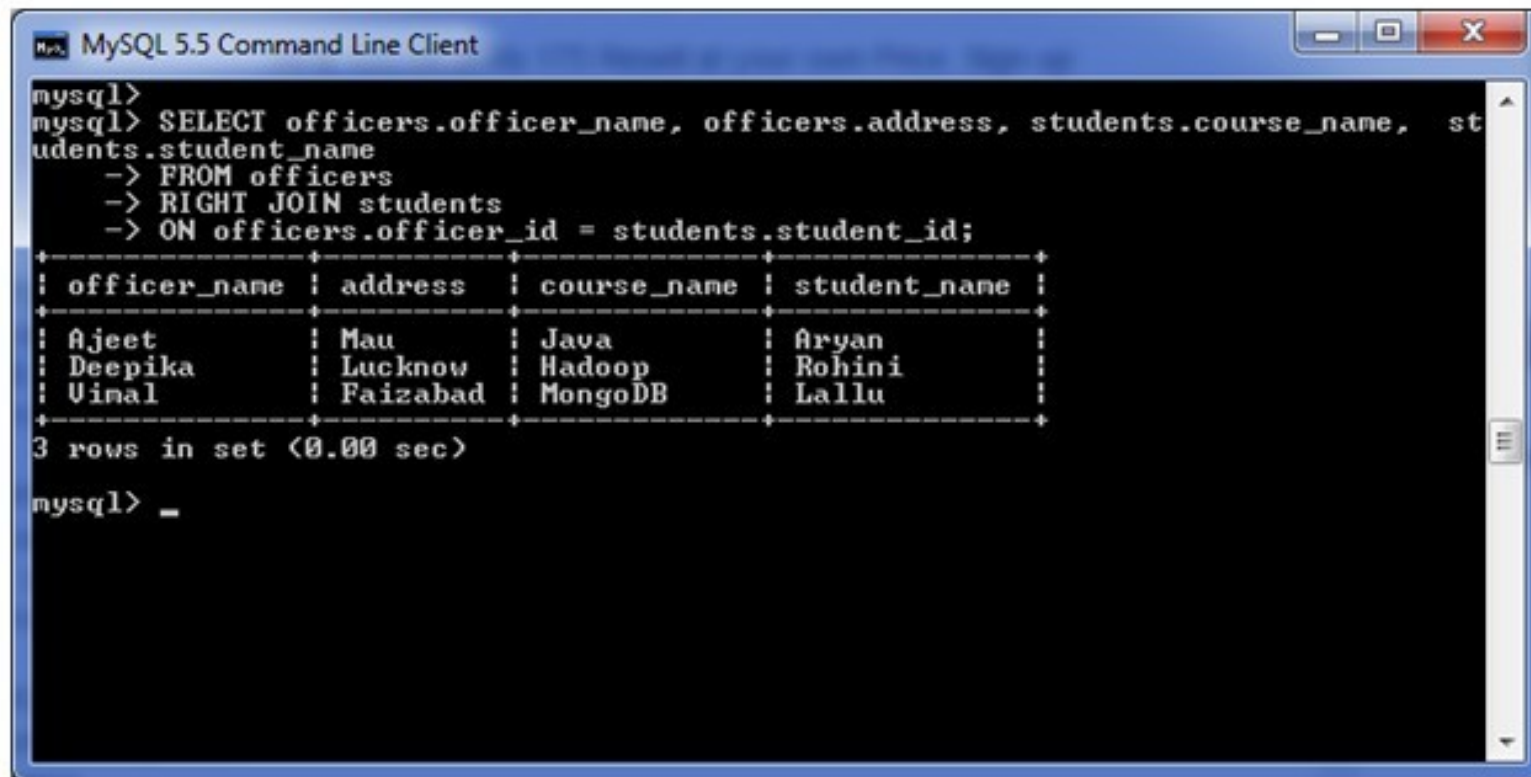
Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name, students.student_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name, students.student_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

Output:



```
mysql>
mysql> SELECT officers.officer_name, officers.address, students.course_name, st
udents.student_name
-> FROM officers
-> RIGHT JOIN students
-> ON officers.officer_id = students.student_id;
+-----+-----+-----+-----+
| officer_name | address | course_name | student_name |
+-----+-----+-----+-----+
| Ajeet       | Mau    | Java      | Aryan       |
| Deepika     | Lucknow | Hadoop    | Rohini      |
| Uinal       | Faizabad | MongoDB  | Lallu       |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```