NOTES AND ASSIGNMENT FOR CLASS XII

SUBJECT: COMPUTER SCIENCE

NOTES

History of MySQL

MySQL is an open source database product that was created by MySQL AB, a company founded in 1995 in Sweden. In 2008, MySQL AB announced that it had agreed to be acquired by Sun Microsystems for approximately $1 billion.

A Data Type specifies a particular type of data, like integer, floating points, Boolean etc. It also identifies the possible values for that type, the operations that can be performed on that type and the way the values of that type are stored.

MySQL supports a lot number of SQL standard data types in various categories. It uses many different data types broken into mainly three categories: numeric, date and time, and string types.

MySQL Features

- **Relational Database Management System (RDBMS):** MySQL is a relational database management system.
- **Easy to use:** MySQL is easy to use. You have to get only the basic knowledge of SQL. You can build and interact with MySQL with only a few simple SQL statements.
- **It is secure:** MySQL consist of a solid data security layer that protects sensitive data from intruders. Passwords are encrypted in MySQL.
- **Client/ Server Architecture:** MySQL follows a client /server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc.
- **Free to download:** MySQL is free to use and you can download it from MySQL official website.
- **It is scalable:** MySQL can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, you can increase this number to a theoretical limit of 8 TB of data.
- **Compatibale on many operating systems:** MySQL is compatible to run on many operating systems, like Novell NetWare, Windows* Linux*, many varieties of UNIX* (such as Sun* Solaris*, AIX, and DEC* UNIX), OS/2, FreeBSD*, and others. MySQL also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).
- **Allows roll-back:** MySQL allows transactions to be rolled back, commit and crash recovery.
- **High Performance:** MySQL is faster, more reliable and cheaper because of its unique storage engine architecture.
- **High Flexibility:** MySQL supports a large number of embedded applications which makes MySQL very flexible.
- **High Productivity:** MySQL uses Triggers, Stored procedures and views which allows the developer to give a higher productivity.

Disadvantages / Drawback of MySQL:

Following are the few disadvantages of MySQL:

- MySQL version less than 5.0 doesn't support ROLE, COMMIT and stored procedure.
- MySQL does not support a very large database size as efficiently.
- MySQL doesn't handle transactions very efficiently and it is prone to data corruption.
- MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.
- MySQL doesn't support SQL check constraints.

Numeric Data Type

Data Type	Description

Syntax	
INT	A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2 allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
TINYINT	A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsig from 0 to 255. You can specify a width of up to 4 digits.
SMALLINT	A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsig from 0 to 65535. You can specify a width of up to 5 digits.
MEDIUMINT	A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 838 allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
BIGINT	A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 t unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
FLOAT(m,d)	A floating-point number that cannot be unsigned. You can define the display length (m) and the number of deci and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decim go to 24 places for a float.
DOUBLE(m,d)	A double precision floating-point number that cannot be unsigned. You can define the display length (m) and th This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 synonym for double.
DECIMAL(m,d)	An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds t display length (m) and the number of decimals (d) is required. Numeric is a synonym for decimal.

Date and Time Data Type:

Data Type Syntax	Maximum Size	Explanation
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'yy
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'yy
TIMESTAMP(m)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC.	Displayed as 'YY
TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH
YEAR[(2|4)]	Year value as 2 digits or 4 digits.	Default is 4 digit

String Data Types:

Data Type Syntax	Maximum Size	Explanation
CHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Fixed-length st to equal size characters.
VARCHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length
TINYTEXT(size)	Maximum size of 255 characters.	Where size is the number of characters to store.
TEXT(size)	Maximum size of 65,535 characters.	Where size is the number of characters to store.
MEDIUMTEXT(size)	Maximum size of 16,777,215 characters.	Where size is the number of characters to store.
LONGTEXT(size)	Maximum size of 4GB or 4,294,967,295 characters.	Where size is the number of characters to store.
BINARY(size)	Maximum size of 255 characters.	Where size is the number of binary characters to store. Fixed-le on right to equal size characters. (introduced in MySQL 4.1.2)

VARBINARY(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length (introduced in MySQL 4.1.2)

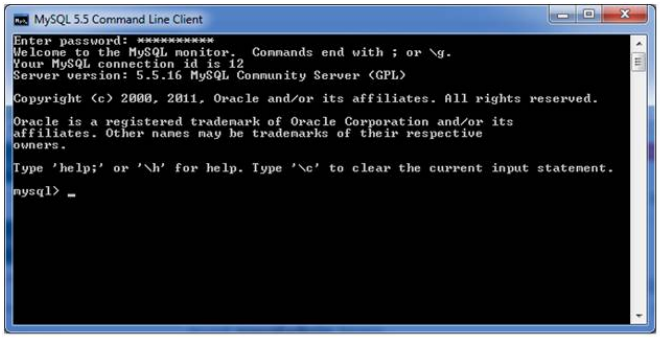
Large Object Data Types (LOB) Data Types:

Data Type Syntax	Maximum Size
TINYBLOB	Maximum size of 255 bytes.
BLOB(size)	Maximum size of 65,535 bytes.
MEDIUMBLOB	Maximum size of 16,777,215 bytes.
LONGTEXT	Maximum size of 4gb or 4,294,967,295 characters.

MySQL Create Database

You can create a MySQL database by using MySQL Command Line Client.

Open the MySQL console and write down password, if you set one while installation. You will get the following:



Now you are ready to create database.

Syntax:

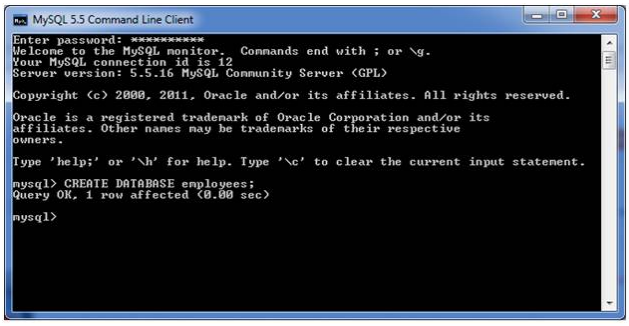
1. **CREATE DATABASE** database_name;

Example:

Let's take an example to create a database name "employees"

1. **CREATE DATABASE** employees;

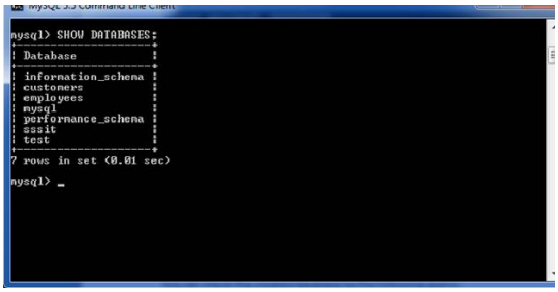
It will look like this:



You can check the created database by the following query:

1. SHOW DATABASES;

Output



Here, you can see the all created databases.

MySQL CREATE TABLE

The MySQL CREATE TABLE command is used to create a new table into the database. A table creation command requires three things:

- Name of the table
- Names of fields
- Definitions for each field

Syntax:

Following is a generic syntax for creating a MySQL table in the database.

```
1. CREATE TABLE table_name (column_name column_type...);
```

Example:

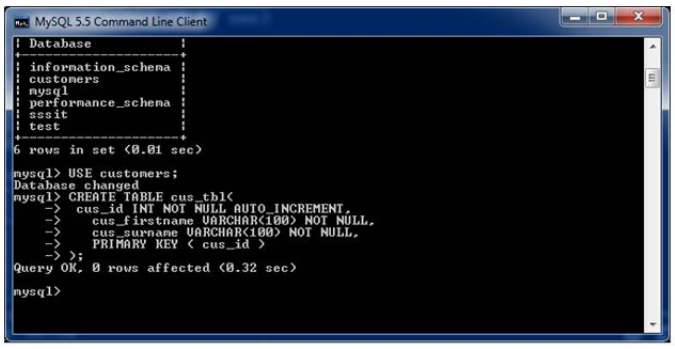
Here, we will create a table named "cus_tbl" in the database "customers".

```
1. CREATE TABLE cus_tbl(
2.    cus_id INT NOT NULL AUTO_INCREMENT,
3.    cus_firstname VARCHAR(100) NOT NULL,
4.    cus_surname VARCHAR(100) NOT NULL,
5.    PRIMARY KEY ( cus_id )
6. );
```

Note:

1. Here, NOT NULL is a field attribute and it is used because we don't want this field to be NULL. If you will try to create a record with NULL value, then MySQL will raise an error.
2. The field attribute AUTO_INCREMENT specifies MySQL to go ahead and add the next available number to the id field.PRIMARY KEY is used to define a column as primary key. You can use multiple columns separated by comma to define a primary key.

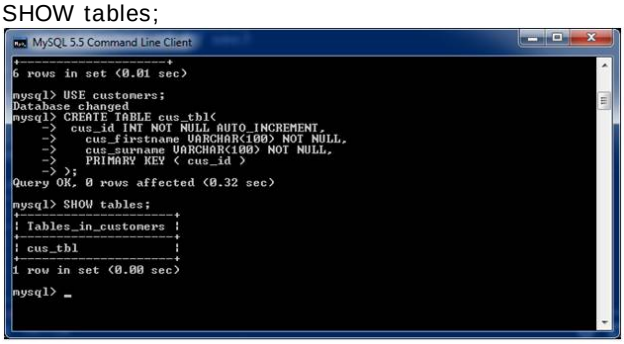
Visual representation of creating a MySQL table:



See the created table:

Use the following command to see the table already created:

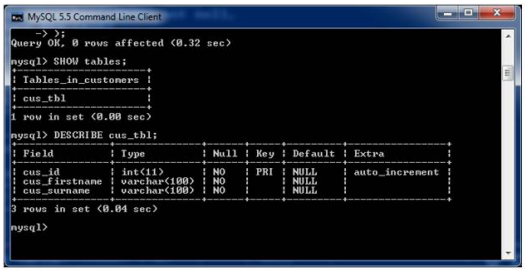
1. SHOW tables;



See the table structure:

Use the following command to see the table already created:

1. DESCRIBE cus_tbl;



MySQL INSERT Statement

MySQL INSERT statement is used to insert data in MySQL table within the database. We can insert single or multiple records using a single query in MySQL.

Syntax:

The SQL INSERT INTO command is used to insert data in MySQL table. Following is a generic syntax:

1. **INSERT INTO** table_name (field1, field2,...fieldN)
2. **VALUES**
3. (value1, value2,...valueN);

Field name is optional. If you want to specify partial values, field name is mandatory.

Syntax for all fields:

1. **INSERT INTO** table_name **VALUES** (value1, value2,...valueN);

MySQL INSERT Example 1: for all fields

If you have to store all the field values, either specify all field name or don't specify any field.

Example:

1. **INSERT INTO** emp **VALUES** (7, 'Sonoo', 40000);

Or,

1. **INSERT INTO** emp(id,**name**,salary) **VALUES** (7, 'Sonoo', 40000);

MySQL INSERT Example 2: for partial fields

In such case, it is mandatory to specify field names.

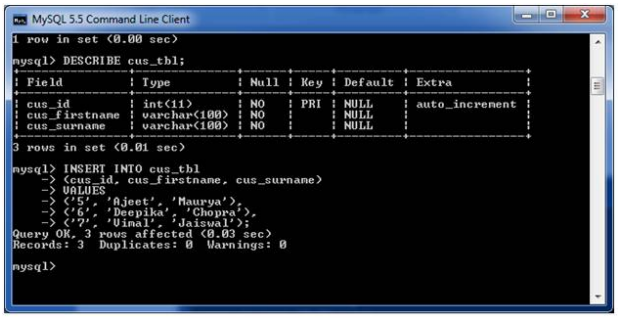
```
1. INSERT INTO emp(id,name) VALUES (7, 'Sonoo');
```

MySQL INSERT Example 3: inserting multiple records

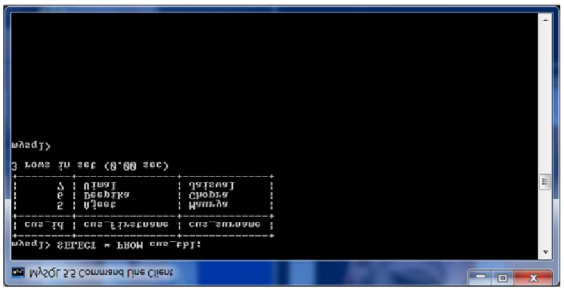
Here, we are going to insert record in the "cus_tbl" table of "customers" database.

```
1.  INSERT INTO cus_tbl
2. (cus_id, cus_firstname, cus_surname)
3. VALUES
4. (5, 'Ajeet', 'Maurya'),
5. (6, 'Deepika', 'Chopra'),
6. (7, 'Vimal', 'Jaiswal');
```

Visual Representation:



See the data within the table by using the SELECT command:



MySQL SELECT Statement

The MySQL SELECT statement is used to fetch data from the one or more tables in MySQL. We can retrieve records of all fields or specified fields.

Syntax for specified fields:

```
1. SELECT expressions
2. FROM tables
3. [WHERE conditions];
```

Syntax for all fields:

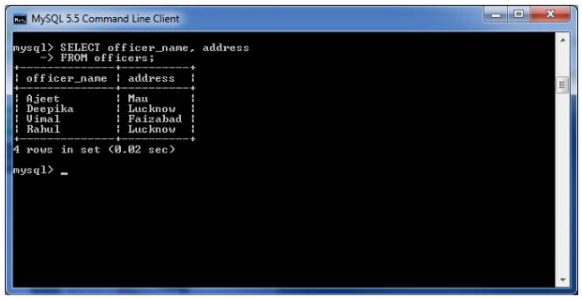
```
1. SELECT * FROM tables [WHERE conditions];
```

MySQL SELECT Example 1: for specified fields

In such case, it is mandatory to specify field names.

Example:

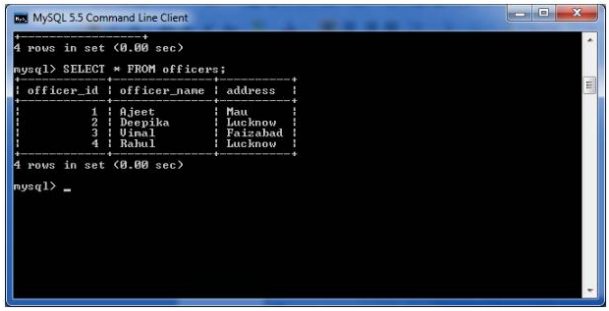
```
1. SELECT officer_name, address
2. FROM officers
```



MySQL SELECT Example 2: for all fields

In such case, we can specify either all fields or * (asterisk) symbol.

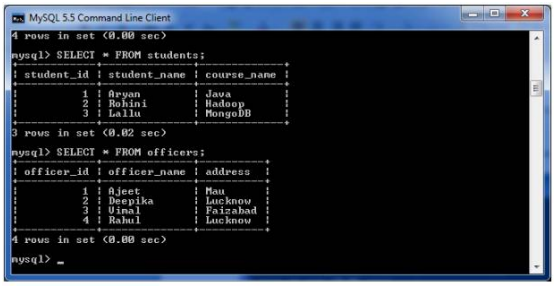
1. **SELECT** * **FROM** officers



MySQL SELECT Example 3: from multiple tables

MySQL SELECT statement can also be used to retrieve records from multiple tables by using JOIN statement.

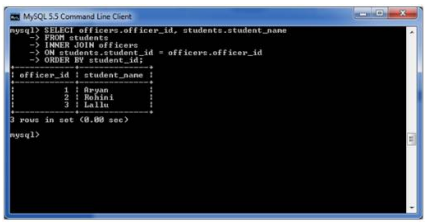
Let's take two tables "students" and "officers", having the following data.



Execute the following query:

1. **SELECT** officers.officer_id, students.student_name
2. **FROM** students
3. **INNER** JOIN officers
4. **ON** students.student_id = officers.officer_id
5. **ORDER BY** student_id;

Output:



MySQL Distinct Clause

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:

1. **SELECT DISTINCT** expressions
2. **FROM** tables
3. [**WHERE** conditions];

Parameters

expressions: specify the columns or calculations that you want to retrieve.

tables: specify the name of the tables from where you retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be met for the records to be selected.

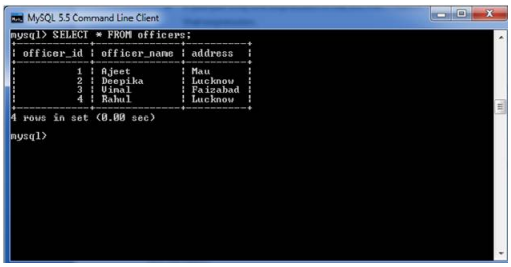
Note:

- If you put only one expression in the DISTINCT clause, the query will return the unique values for that expression.
- If you put more than one expression in the DISTINCT clause, the query will retrieve unique combinations for the expressions listed.
- In MySQL, the DISTINCT clause doesn't ignore NULL values. So if you are using the DISTINCT clause in your SQL statement, your result set will include NULL as a distinct value.

MySQL DISTINCT Clause with single expression

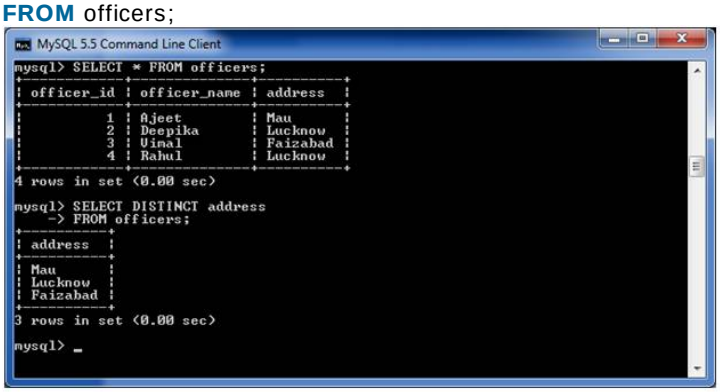
If you use a single expression then the MySQL DISTINCT clause will return a single field with unique records (no duplicate record).

See the table:



Use the following query:

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



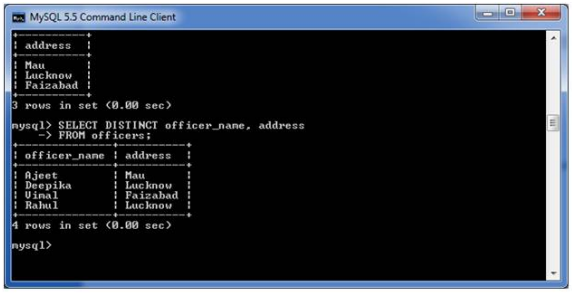
MySQL DISTINCT Clause with multiple expressions

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:

1. **SELECT DISTINCT** officer_name, address

2. **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**];

Parameters

expressions: It specifies the columns that you want to retrieve.

tables: It specifies the tables, from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies conditions that must be fulfilled for the records to be selected.

ASC: It is optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).

DESC: It is also optional. It sorts the result set in descending order by expression.

Note: You can use MySQL ORDER BY clause in a SELECT statement, SELECT LIMIT statement, and DELETE LIMIT statement.

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:

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

Output:

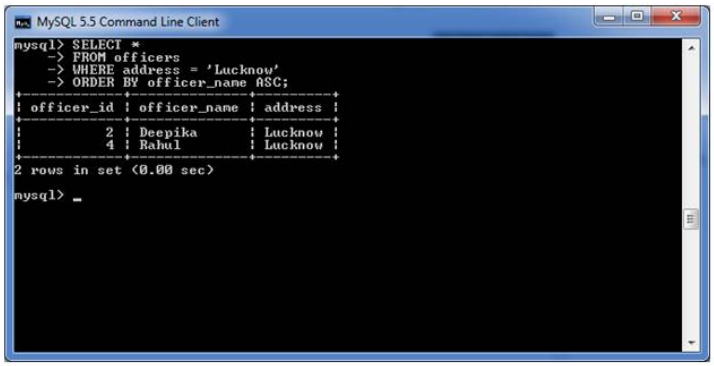
MySQL ORDER BY: with ASC attribute

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 ORDER BY: with DESC attribute

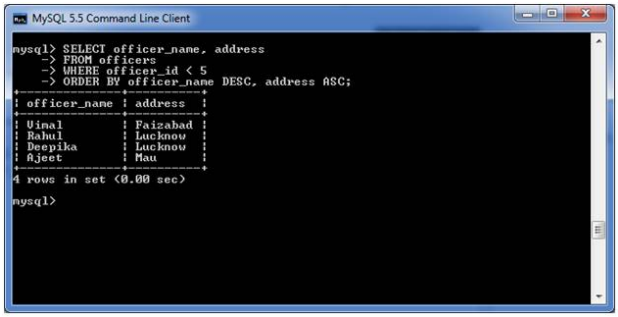
```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name DESC;
```

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 BETWEEN Condition

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:

1. expression BETWEEN value1 AND value2;

Parameters

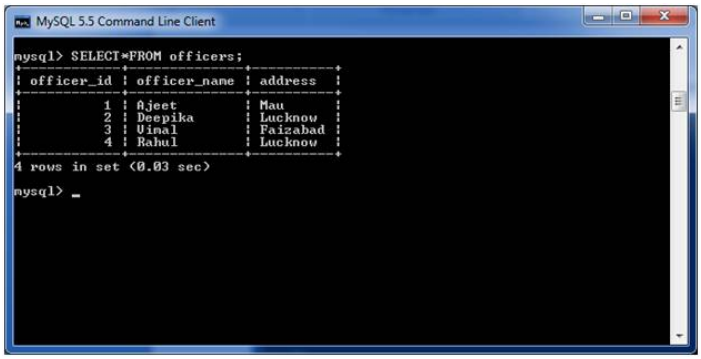
expression: It specifies a column.

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:

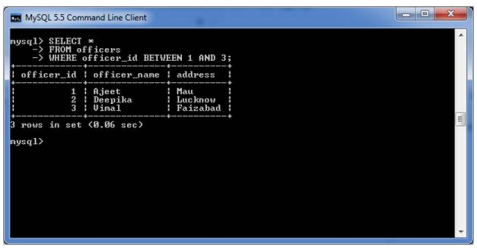
Consider a table "officers" having the following data.



Execute the following query:

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

Output:



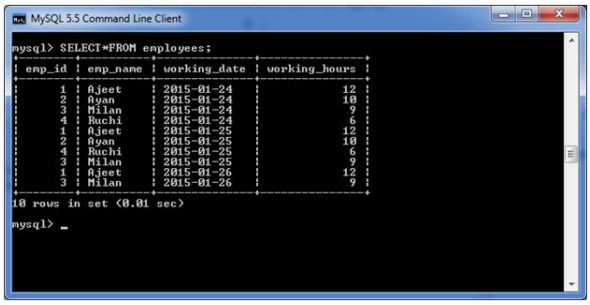
Note: In the above example, you can see that only three rows are returned between 1 and 3.

(ii) MySQL BETWEEN condition with date:

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

See this example:

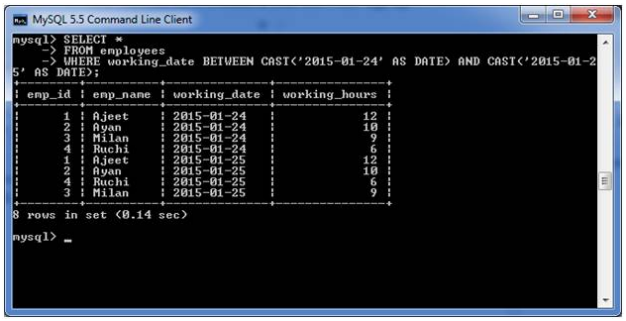
Consider a table "employees", having the following data.



Execute the following query:

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

Output:



Note: In the above example you can see that only data between specific dates are shown.

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.

Syntax:

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

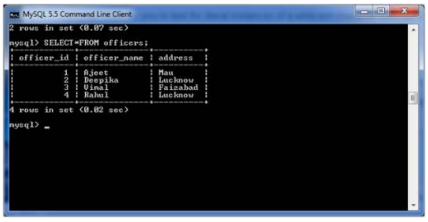
Parameters

expression: It specifies a value to test.

value1, value2, ... or value_n: These are the values to test against expression. If any of these values matches expression, then the IN condition will evaluate to true. This is a quick method to test if any one of the values matches expression.

MySQL IN Example

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

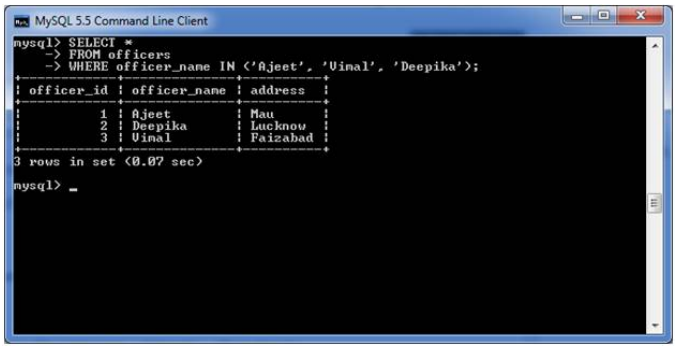


Execute the following query:

```
1. SELECT *
2. FROM officers
```

3. **WHERE** officer_name IN ('Ajeet', 'Vimal', 'Deepika');

Output:

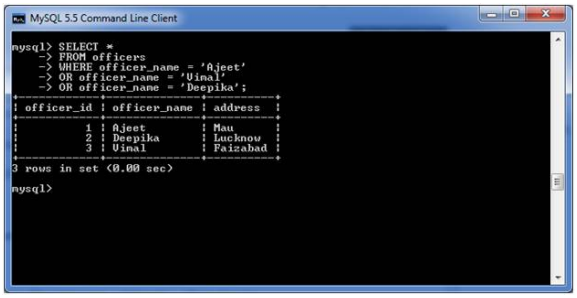


Let's see why it is preferred over OR condition:

Execute the following query:

1. **SELECT** *
2. **FROM** officers
3. **WHERE** officer_name = 'Ajeet'
4. OR officer_name = 'Vimal'
5. OR officer_name = 'Deepika';

Output:



It also produces the same result. So IN condition is preferred over OR condition because it has minimum number of codes.

MySQL LIKE condition

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.

Syntax:

1. expression LIKE pattern [**ESCAPE** 'escape_character']

Parameters

expression: It specifies a column or field.

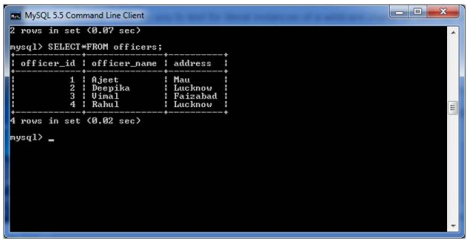
pattern: It is a character expression that contains pattern matching.

escape_character: It is optional. It allows you to test for literal instances of a wildcard character such as % or _. If you do not provide the escape_character, MySQL assumes that "\" is the escape_character.

MySQL LIKE Examples

1) Using % (percent) Wildcard:

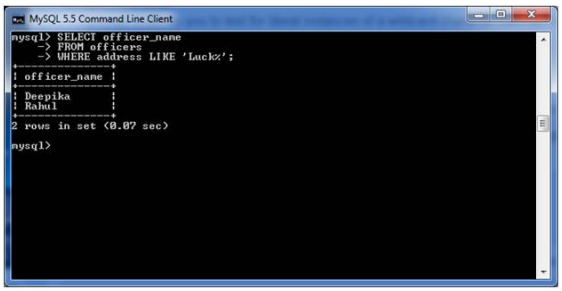
Consider a table "officers" having the following data.



Execute the following query:

```
1.  SELECT officer_name
2. FROM officers
3. WHERE address LIKE 'Luck%';
```

Output:



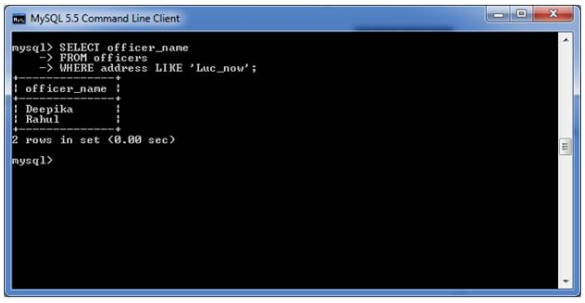
2) Using _ (Underscore) Wildcard:

We are using the same table "officers" in this example too.

Execute the following query:

```
1.
2. SELECT officer_name
3. FROM officers
4. WHERE address LIKE 'Luc_now';
```

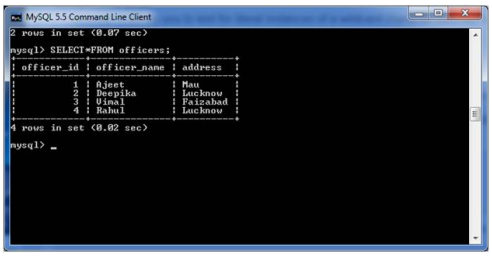
Output:



3) Using NOT Operator:

You can also use NOT operator with MySQL LIKE condition. This example shows the use of % wildcard with the NOT Operator.

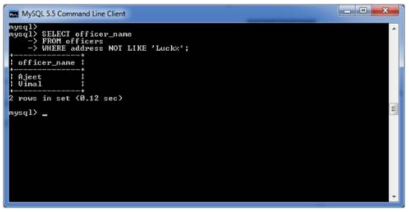
Consider a table "officers" having the following data.



Execute the following query:

```
1. SELECT officer_name
2. FROM officers
3. WHERE address NOT LIKE 'Luck%';
```

Output:



Aggregate Functions

MySQL Count() Function

MySQL count() function is used to returns the count of an expression. It allows us to count all rows or only some rows of the table that matches a specified condition. It is a type of aggregate function whose return type is BIGINT. This function returns 0 if it does not find any matching rows.

We can use the count function in three forms, which are explained below:

- Count (*)
- Count (expression)
- Count (distinct)

Let us discuss each in detail.

COUNT(*) Function: This function uses the SELECT statement to returns the count of rows in a result set. The result set contains all Non-Null, Null, and duplicates rows.

COUNT(expression) Function: This function returns the result set without containing Null rows as the result of an expression.

COUNT(distinct expression) Function: This function returns the count of distinct rows without containing NULL values as the result of the expression.

Syntax

The following are the syntax of the COUNT() function:

```
1. SELECT COUNT (aggregate_expression)
2. FROM table_name
3. [WHERE conditions];
```

Parameter explanation

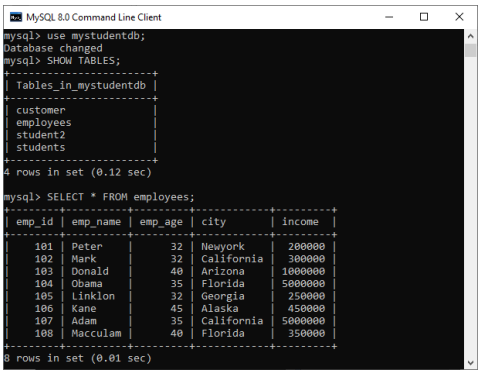
aggregate_expression: It specifies the column or expression whose NON-NULL values will be counted.

table_name: It specifies the tables from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL count() function example

Consider a table named "employees" that contains the following data.



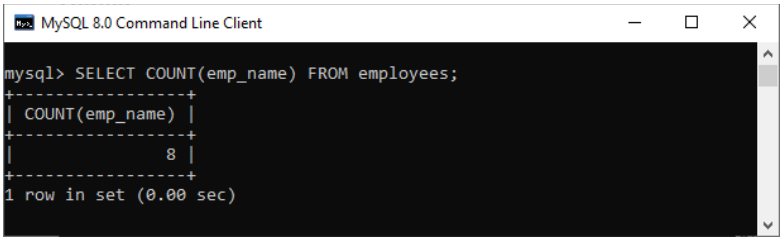
Let us understand how count() functions work in MySQL.

Example1

Execute the following query that uses the COUNT(expression) function to calculates the total number of employees name available in the table:

1. mysql> **SELECT** COUNT(emp_name) **FROM** employees;

Output:

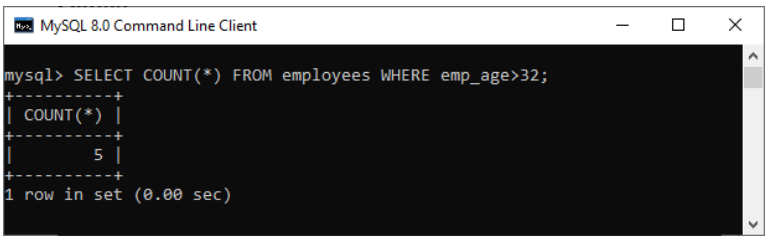


Example2

Execute the following statement that returns all rows from the employee table and WHERE clause specifies the rows whose value in the column emp_age is greater than 32:

1. mysql> **SELECT** COUNT(*) **FROM** employees **WHERE** emp_age>32;

Output:

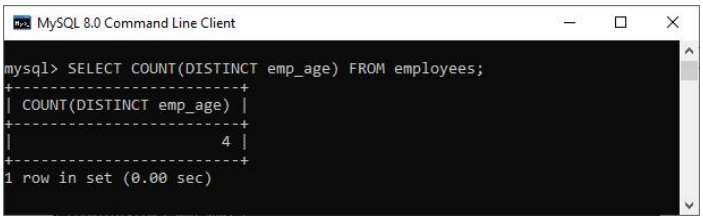


Example3

This statement uses the COUNT(distinct expression) function that counts the Non-Null and distinct rows in the column emp_age:

1. mysql> **SELECT** COUNT(**DISTINCT** emp_age) **FROM** employees;

Output:

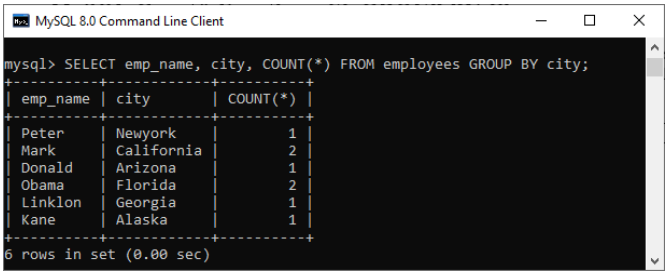


MySQL Count() Function with GROUP BY Clause

We can also use the count() function with the GROUP BY clause that returns the count of the element in each group. For example, the following statement returns the number of employee in each city:

```
1. mysql> SELECT emp_name, city, COUNT(*) FROM employees GROUP BY city;
```

After the successful execution, we will get the result as below:

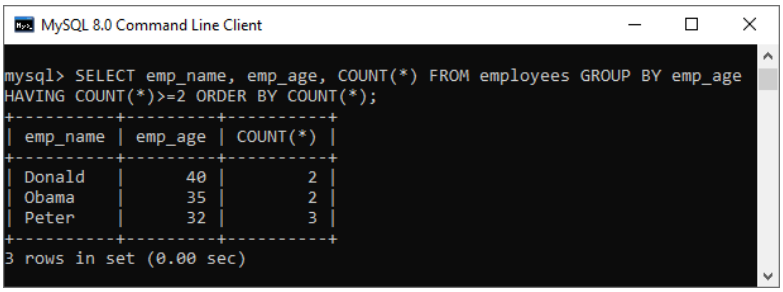


MySQL Count() Function with HAVING and ORDER BY Clause

Let us see another clause that uses ORDER BY and Having clause with the count() function. Execute the following statement that gives the employee name who has at least two age same and sorts them based on the count result:

```
1. mysql> SELECT emp_name, emp_age, COUNT(*) FROM employees
2. GROUP BY emp_age
3. HAVING COUNT(*)>=2
4. ORDER BY COUNT(*);
```

This statement will give the output as below:



MySQL sum() function

The MySQL sum() function is used to return the total summed value of an expression. It returns **NULL** if the result set does not have any rows. It is one of the kinds of aggregate functions in MySQL.

Syntax

Following are the syntax of sum() function in MySQL:

```
1. SELECT SUM(aggregate_expression)
2. FROM tables
3. [WHERE conditions];
```

Parameter Explanation

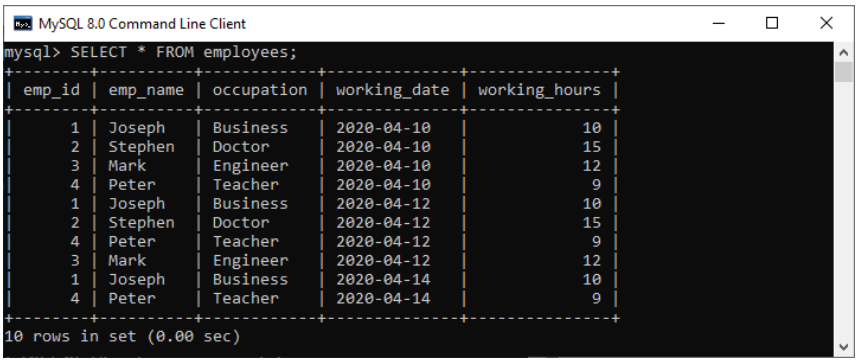
aggregate_expression: It specifies the column or expression that we are going to calculate the sum.

table_name: It specifies the tables from where we want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL sum() function example

Consider our database has a table named **employees**, having the following data. Now, we are going to understand this function with various examples:



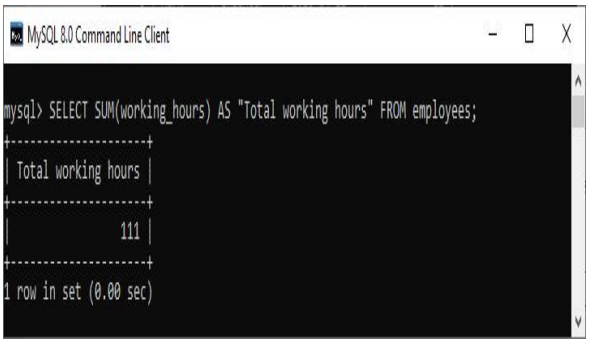
1. Basic Example

Execute the following query that calculates the total number of working hours of all employees in the table:

1. mysql> **SELECT** SUM(working_hours) **AS** "Total working hours" **FROM** employees;

Output:

We will get the result as below:



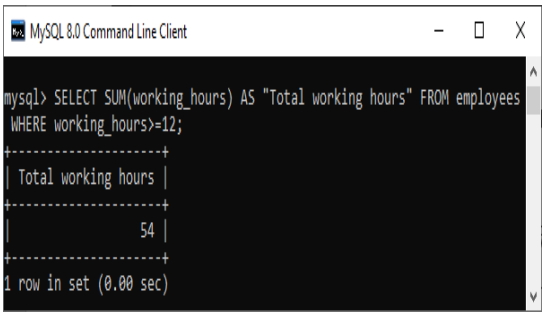
2. MySQL sum() function with WHERE clause

This example is used to return the result based on the condition specified in the WHERE clause. Execute the following query to calculate the total working hours of employees whose **working_hours>= 12**.

1. mysql> **SELECT** SUM(working_hours) **AS** "Total working hours" **FROM** employees **WHERE** working_hours>=12;

Output:

This statement will give the output as below:



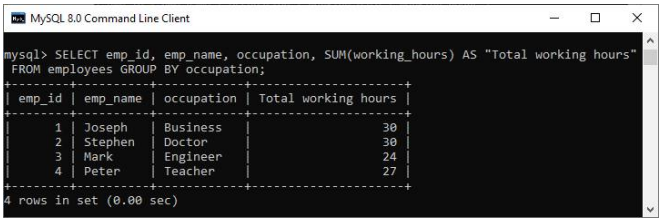
3. MySQL sum() function with GROUP BY clause

We can also use the SUM() function with the GROUP BY clause to return the total summed value for each group. For example, this statement calculates the total working hours of each employee by using the SUM() function with the GROUP BY clause, as shown in the following query:

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

Output:

Here, we can see that the total working hours of each employee calculates by grouping them based on their occupation.

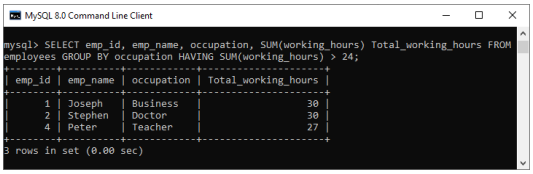


4. MySQL sum() function with HAVING clause

The HAVING clause is used to **filter** the group with the sum() function in MySQL. Execute the following statement that calculates the working hours of all employees, grouping them based on their occupation and returns the result whose Total_working_hours>24.

```
mysql> SELECT emp_id, emp_name, occupation,
SUM(working_hours) Total_working_hours
FROM employees
GROUP BY occupation
HAVING SUM(working_hours)>24;
```

Output:



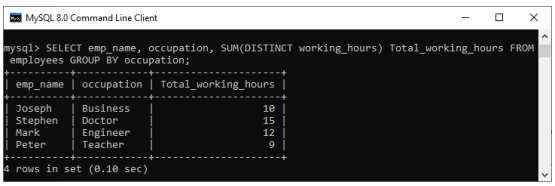
5. MySQL sum() function with DISTINCT clause

MySQL uses the DISTINCT keyword to remove the **duplicate** rows from the column name. This clause can also be used with sum() function to return the total summed value of a Unique number of records present in the table.

Execute the following query that removes the duplicate records in the working_hours column of the employee table and then calculates the sum:

```
mysql> SELECT emp_name, occupation,
SUM(DISTINCT working_hours) Total_working_hours
FROM employees
GROUP BY occupation;
```

Output:



MySQL Count() Function

MySQL count() function is used to returns the count of an expression. It allows us to count all rows or only some rows of the table that matches a specified condition. It is a type of aggregate function whose return type is BIGINT. This function returns 0 if it does not find any matching rows.

We can use the count function in three forms, which are explained below:

- Count (*)
- Count (expression)
- Count (distinct)

Let us discuss each in detail.

COUNT(*) Function: This function uses the SELECT statement to returns the count of rows in a result set. The result set contains all Non-Null, Null, and duplicates rows.

COUNT(expression) Function: This function returns the result set without containing Null rows as the result of an expression.

COUNT(distinct expression) Function: This function returns the count of distinct rows without containing NULL values as the result of the expression.

Syntax

The following are the syntax of the COUNT() function:

```
1. SELECT COUNT (aggregate_expression)
2. FROM table_name
3. [WHERE conditions];
```

Parameter explanation

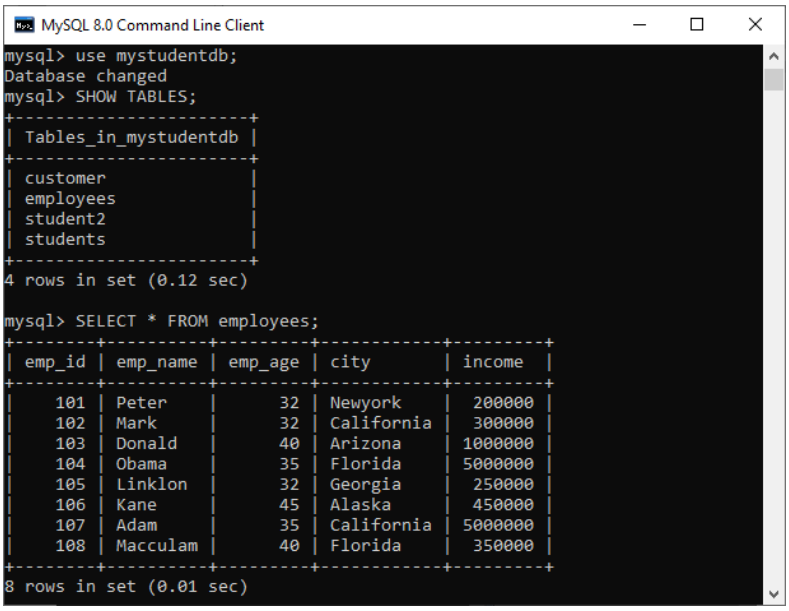
aggregate_expression: It specifies the column or expression whose NON-NULL values will be counted.

table_name: It specifies the tables from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL count() function example

Consider a table named "employees" that contains the following data.



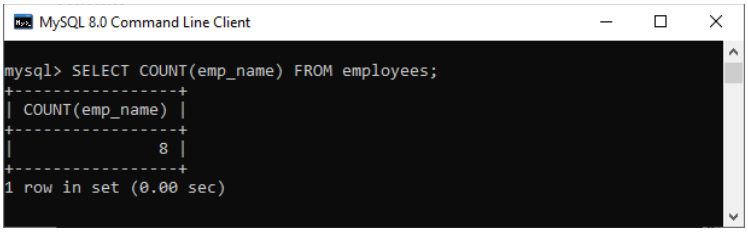
Let us understand how count() functions work in MySQL.

Example1

Execute the following query that uses the COUNT(expression) function to calculates the total number of employees name available in the table:

```
1. mysql> SELECT COUNT(emp_name) FROM employees;
```

Output:

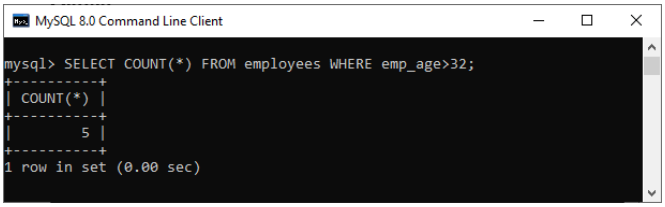


Example2

Execute the following statement that returns all rows from the employee table and WHERE clause specifies the rows whose value in the column emp_age is greater than 32:

```
1. mysql> SELECT COUNT(*) FROM employees WHERE emp_age>32;
```

Output:

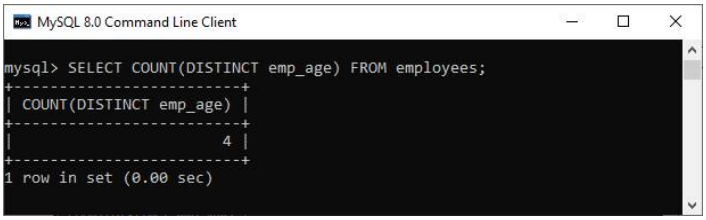


Example3

This statement uses the COUNT(distinct expression) function that counts the Non-Null and distinct rows in the column emp_age:

```
1. mysql> SELECT COUNT(DISTINCT emp_age) FROM employees;
```

Output:

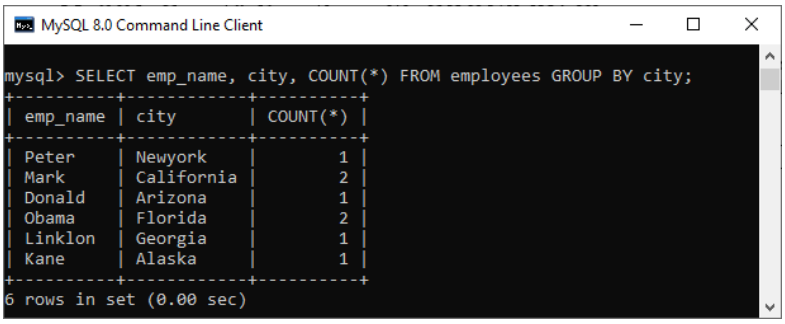


MySQL Count() Function with GROUP BY Clause

We can also use the count() function with the GROUP BY clause that returns the count of the element in each group. For example, the following statement returns the number of employee in each city:

```
1. mysql> SELECT emp_name, city, COUNT(*) FROM employees GROUP BY city;
```

After the successful execution, we will get the result as below:

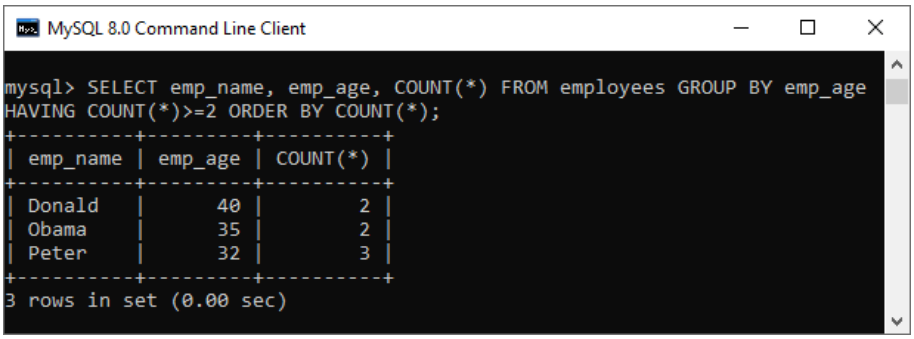


MySQL Count() Function with HAVING and ORDER BY Clause

Let us see another clause that uses ORDER BY and Having clause with the count() function. Execute the following statement that gives the employee name who has at least two age same and sorts them based on the count result:

```
mysql> SELECT emp_name, emp_age, COUNT(*) FROM employees
GROUP BY emp_age
HAVING COUNT(*)>=2
ORDER BY COUNT(*);
```

This statement will give the output as below:



MySQL min() function

The MySQL min() function is used to return the minimum value from the table.

Syntax:

```
SELECT MIN (aggregate_expression)
FROM tables
[WHERE conditions];
```

Parameter explanation

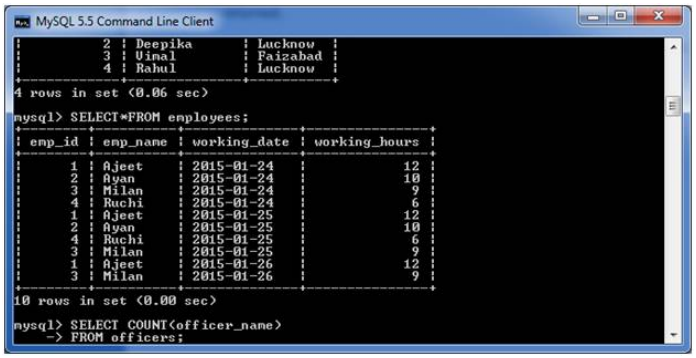
aggregate_expression: It specifies the column or expression, from which the minimum value will be returned.

table_name: It specifies the tables, from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL min() function example

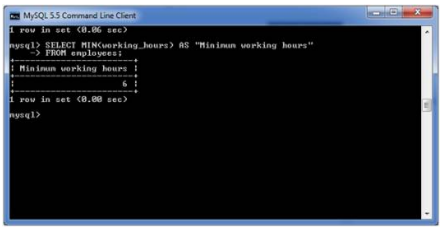
Consider a table named "employees", having the following data.



Execute the following query:

```
1. SELECT MIN (working_hours) AS "Minimum working hours"
2. FROM employees;
```

Output:



MySQL max() function

The MySQL max() function is used to return the maximum value of an expression. It is used when you need to get the maximum value from your table.

Syntax:

```
1. SELECT MAX(aggregate_expression)
2. FROM tables
3. [WHERE conditions];
```

Parameter explanation

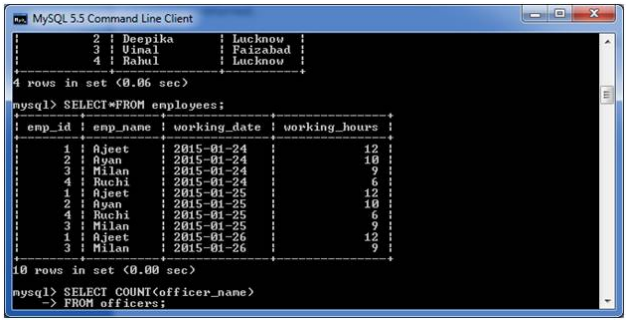
aggregate_expression: It specifies the column or expression, from which the maximum value will be returned.

table_name: It specifies the tables, from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL max() function example

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

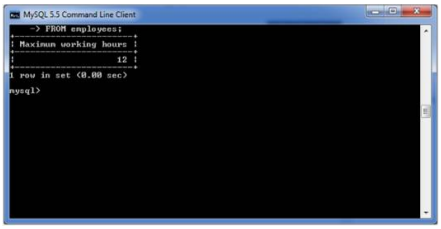


Execute the following query:

```
1. SELECT MAX (working_hours) AS "Maximum working hours"
```

2. **FROM** employees;

Output:



MySQL avg() function

The MySQL avg() is an aggregate function used to return the average value of an expression in various records.

Syntax

The following are the basic syntax an avg() function in MySQL:

1. **SELECT** AVG(aggregate_expression)
2. **FROM** tables
3. [**WHERE** conditions];

Parameter explanation

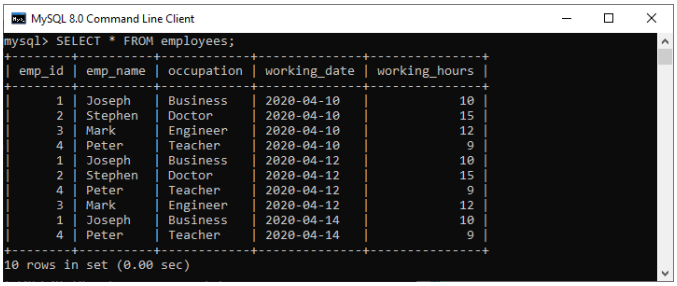
aggregate_expression: It specifies the column or expression that we are going to find the average result.

table_name: It specifies the tables from where we want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

MySQL avg() function example

Consider our database has a table named **employees**, having the following data. Now, we are going to understand this function with various examples:



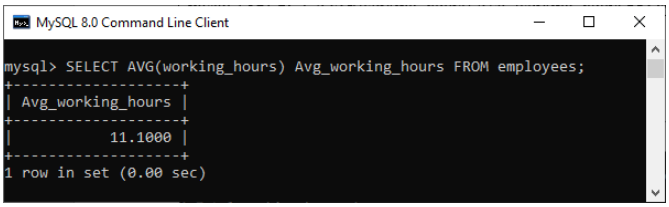
1. Basic Example

Execute the following query that calculates the **average working hours** of all employees in the table:

1. mysql> **SELECT** AVG(working_hours) Avg_working_hours **FROM** employees;

Output:

We will get the result as below:



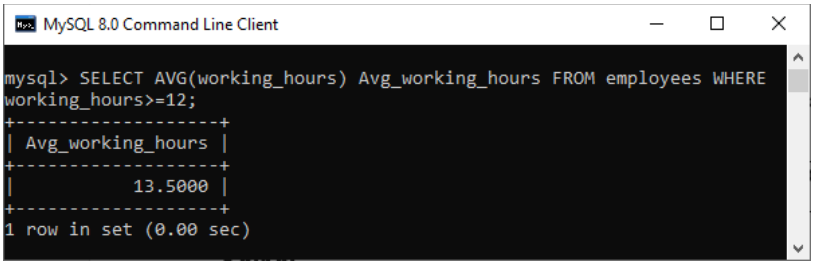
2. MySQL AVG() function with WHERE clause

The WHERE clause specifies the conditions that must be fulfilled for the selected records. Execute the following query to calculate the total average working hours of employees whose **working_hours>= 12**.

1. mysql> **SELECT** AVG(working_hours) Avg_working_hours **FROM** employees **WHERE** working_hours>=12;

Output:

It will give the following output:



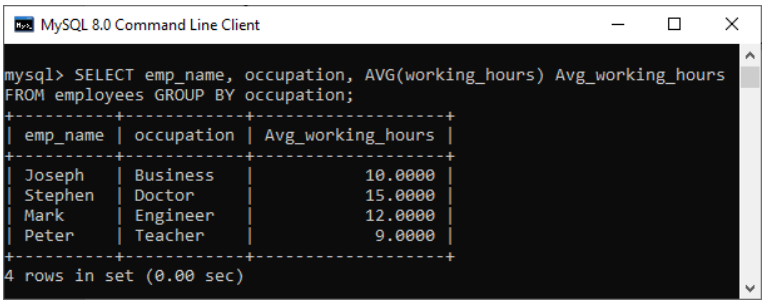
3. MySQL AVG() function with GROUP BY clause

The GROUP BY clause is used to return the result for each group by one or more columns. For example, this statement calculates the average working hours of each employee using the AVG() function and then group the result with the GROUP BY clause:

1. mysql> **SELECT** emp_name, occupation, AVG(working_hours) Avg_working_hours **FROM** employees **GROUP BY** occupation;

Output:

Here, we can see that the total working hours of each employee calculates by grouping them based on their **occupation**.



ASSIGNMENT

Table : Stud

Field Name : Roll Name Stipend Subject average Div

Write the SQL query :

1. To display all records of a table stud.
2. To display student name and subject of all records of a table stud.
3. To display Roll number and average of all records of a table stud.
4. To display all records of a table stud whose division is 1.
5. To display student name of all records of a table stud whose subject is accounts.
6. To display Roll No, student name of all records whose stipend is more than 400 and stipend is less than 500.
7. To display sum of stipend.
8. To display average of stipend.
9. To display minimum of average .

10. To display Maximum of average .
11. To insert a new record (11,"Sajal",300,"English",75,1)
12. To count the number of records of a table stud.
13. To count the number of records of a table stud whose subject is "Mathematics'.
14. To display all the records whose Name start with character 'D'.
15. To display all the records whose Name contains a character 'o'.
16. To display all the records whose Name ends with a character 'a'.
17. To display the unique names of subject available.
18. To display how many subject are there in a table stud.
19. To increase the stipend of each student by Rs 50.
20. To increase the stipend by 10% whose division is 1.
21. To display student name from table stud whose subject is "Mathematics" and stipend is less than 400.
22. To display all the records in order of name.
23. To display all the records in order of division.
24. To display all the records in descending order of stipend .
25. To display the sum of stipend whose subject is "Mathematics"
26. To display the average of stipend whose division is 2.
27. Modify subject as "Accounts" whose name is "John"
28. To Delete all the records whose average marks is less than 60.
29. To Delete all the records whose subject is either "English " or "Accounts"
30. To display the name of all the student whose subject is either "Mathematics" or "Informatics Practices".