

# **DATABASE MANAGEMENT SYSTEMS**

## **(MS SQL)**

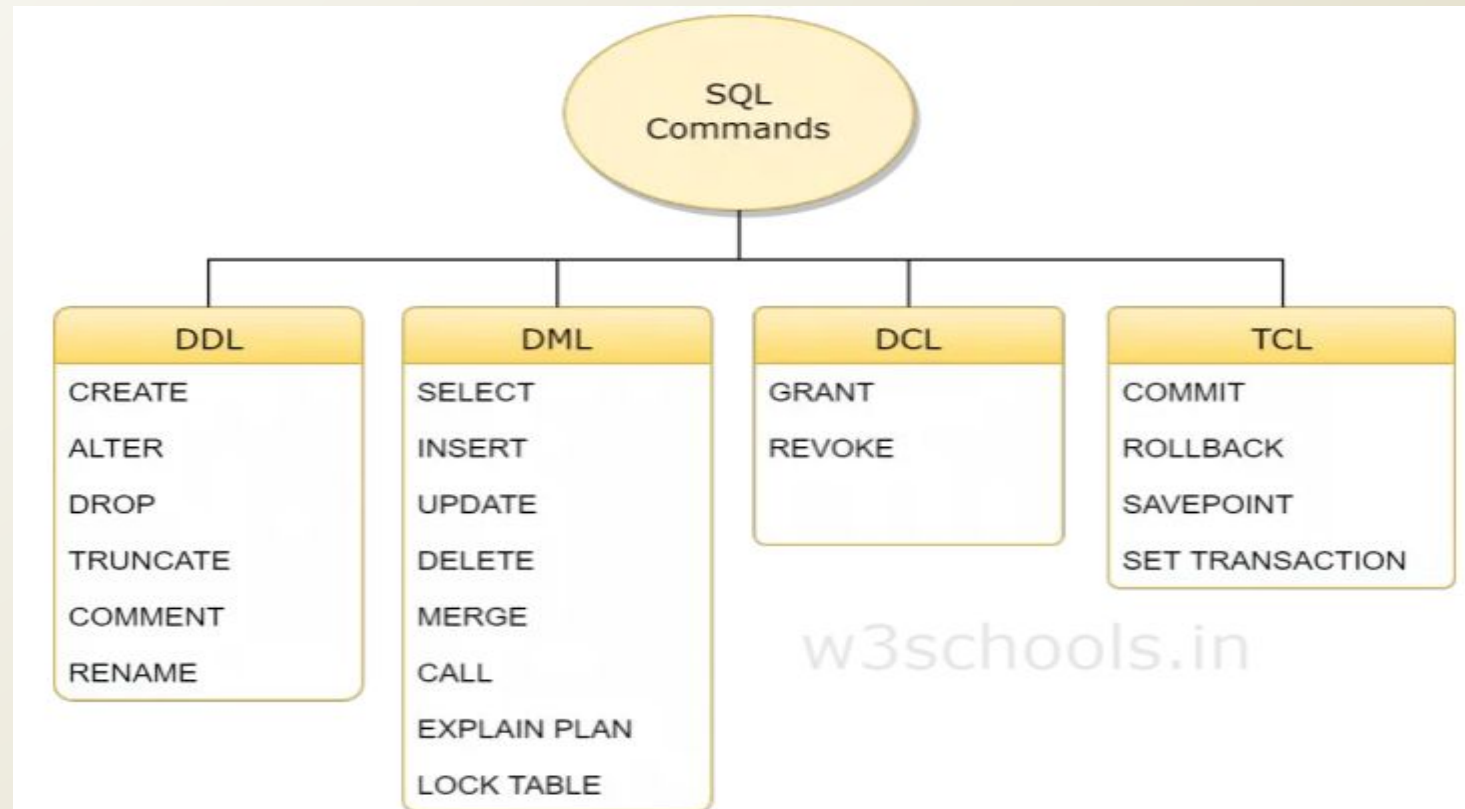
# CHAPTER - 2

## SQL Commands

## Database Languages in DBMS

- ✓ A DBMS has appropriate languages and interfaces to express database queries and updates.
- ✓ Database languages can be used to read, store and update the data in the database.

### Types of Database Languages



## **Data Definition Language (DDL)**

- ✓ DDL stands for Data Definition Language. It is used to define database structure or pattern.
- ✓ It is used to create schema, tables, indexes, constraints, etc. in the database.
- ✓ Using the DDL statements, you can create the skeleton of the database.
- ✓ Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- ✓ **Create:** It is used to create objects in the database.
- ✓ **Alter:** It is used to alter the structure of the database.
- ✓ **Drop:** It is used to delete objects from the database.
- ✓ **Truncate:** It is used to remove all records from a table.
- ✓ **Rename:** It is used to rename an object.
- ✓ **Comment:** It is used to comment on the data dictionary.
- ✓ These commands are used to update the database schema that's why they come under Data definition language.

## Data Manipulation Language (DML)

DML stands for Data Manipulation Language. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- ✓ **Select:** It is used to retrieve data from a database.
- ✓ **Insert:** It is used to insert data into a table.
- ✓ **Update:** It is used to update existing data within a table.
- ✓ **Delete:** It is used to delete all records from a table.
- ✓ **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- ✓ **Call:** It is used to call a structured query language or a Java subprogram.
- ✓ **Explain Plan:** It has the parameter of explaining data.
- ✓ **Lock Table:** It controls concurrency.

## Data Control Language (DCL)

DCL stands for Data Control Language. It is used to retrieve the stored or saved data.

The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- ✓ Grant: It is used to give user access privileges to a database.
- ✓ Revoke: It is used to take back permissions from the user.
- ✓ There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

## Transaction Control Language (TCL)

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

**Commit:** It is used to save the transaction on the database.

**Rollback:** It is used to restore the database to original since the last Commit.



## What is SQL?

- ✓ SQL is a database computer language designed for the retrieval and management of data in a relational database. SQL stands for Structured Query Language.
- ✓ SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.
- ✓ SQL is not a database system, but it is a query language.

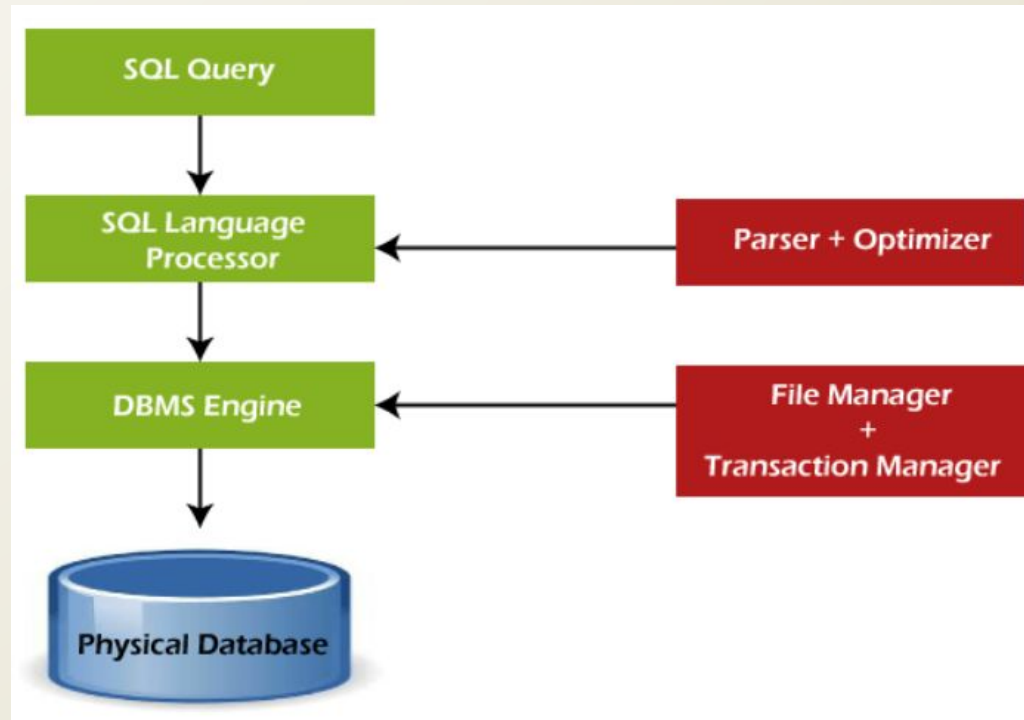
### Process of SQL

- ✓ When we are executing the command of SQL on any Relational database management system, then the system automatically finds the best routine to carry out our request, and the SQL engine determines how to interpret that particular command.

Structured Query Language contains the following four components in its process:

- ✓ Query Dispatcher
- ✓ Optimization Engines
- ✓ Classic Query Engine
- ✓ SQL Query Engine, etc.

A classic query engine allows data professionals and users to maintain non-SQL queries. The architecture of SQL is shown in the following diagram:



## Processing a SQL Statement

To process an SQL statement, a DBMS performs the following five steps:

- ✓ The DBMS first **parses** the SQL statement. It breaks the statement up into individual words, called tokens, makes sure that the statement has a valid verb and valid clauses, and so on. Syntax errors and misspellings can be detected in this step.
- ✓ The DBMS **validates** the statement. It checks the statement against the **system catalog**. Do all the tables named in the statement exist in the database? Do all of the columns exist and are the column names unambiguous? Does the user have the required privileges to execute the statement? Certain semantic errors can be detected in this step.
- ✓ The DBMS generates an **access plan** for the statement. The access plan is a binary representation of the steps that are required to carry out the statement; it is the DBMS equivalent of executable code.

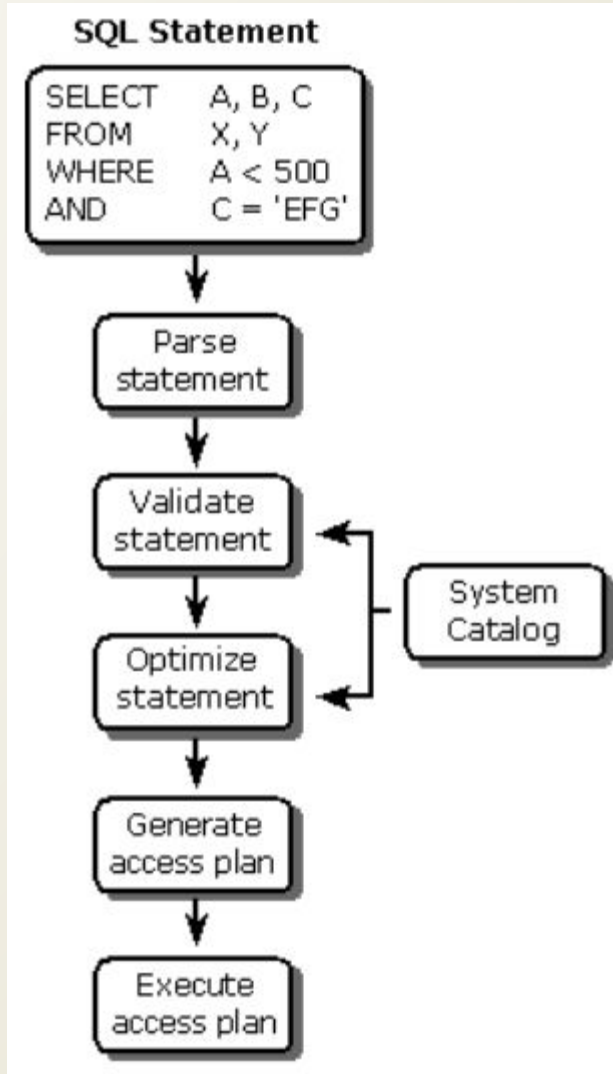
## Processing a SQL Statement

- ✓ The DBMS **optimizes the access plan**. It explores various ways to carry out the access plan. Can an index be used to speed a search? Should the DBMS first apply a search condition to Table A and then join it to Table B, or should it begin with the join and use the search condition afterward?
- ✓ The DBMS **executes** the statement by running the access plan.

The steps used to process an SQL statement vary in the amount of database access they require and the amount of time they take.

Parsing an SQL statement does not require access to the database and can be done very quickly. Optimization, on the other hand, is a very CPU-intensive process and requires access to the system catalog.

## Processing a SQL Statement



## Data Types:

- ✓ Each column in a database table is required to have a name and a data type.
- ✓ An SQL developer must decide what type of data that will be stored inside each column when creating a table.
- ✓ The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

In MySQL there are three main data types: string, numeric, and date and time.



## String Data Types:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters

## Numeric Data Types:

Data type	Description
BIT( <i>size</i> )	A bit-value type. The number of bits per value is specified in <i>size</i> . The <i>size</i> parameter can hold a value from 1 to 64. The default value for <i>size</i> is 1.
TINYINT( <i>size</i> )	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The <i>size</i> parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT( <i>size</i> )	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The <i>size</i> parameter specifies the maximum display width (which is 255)
MEDIUMINT( <i>size</i> )	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The <i>size</i> parameter specifies the maximum display width (which is 255)
INT( <i>size</i> )	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The <i>size</i> parameter specifies the maximum display width (which is 255)
INTEGER( <i>size</i> )	Equal to INT( <i>size</i> )



## Numeric Data Types:

BIGINT( <i>size</i> )	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The <i>size</i> parameter specifies the maximum display width (which is 255)
FLOAT( <i>size</i> , <i>d</i> )	A floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT( <i>p</i> )	A floating point number. MySQL uses the <i>p</i> value to determine whether to use FLOAT or DOUBLE for the resulting data type. If <i>p</i> is from 0 to 24, the data type becomes FLOAT(). If <i>p</i> is from 25 to 53, the data type becomes DOUBLE()
DOUBLE( <i>size</i> , <i>d</i> )	A normal-size floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter
DOUBLE PRECISION( <i>size</i> , <i>d</i> )	
DECIMAL( <i>size</i> , <i>d</i> )	An exact fixed-point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. The maximum number for <i>size</i> is 65. The maximum number for <i>d</i> is 30. The default value for <i>size</i> is 10. The default value for <i>d</i> is 0.
DEC( <i>size</i> , <i>d</i> )	Equal to DECIMAL( <i>size</i> , <i>d</i> )

## DateTime Data Types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME( <i>fsp</i> )	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP( <i>fsp</i> )	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME( <i>fsp</i> )	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

## Data Definition Language (DDL)

### **CREATE Statement:**

CREATE is a DDL command used to create databases, tables, triggers and other database objects.

This example describes how to create a new database using the CREATE DDL command.

### **Syntax to Create a Database:**

```
CREATE Database Database_Name;
```

Ex: Create Database Books;

This example describes how to create a new table using the CREATE DDL command.

**Syntax to create a new table:**

```
CREATE TABLE table_name  
(  
    column_Name1 data_type ( size of the column ) ,  
    column_Name2 data_type ( size of the column) ,  
    column_Name3 data_type ( size of the column) ,  
    ...  
    column_NameN data_type ( size of the column )  
);
```

### **Example:**

```
CREATE TABLE Student  
(  
Roll_No Int ,  
First_Name Varchar (20) ,  
Last_Name Varchar (20) ,  
Age Int ,  
Marks Int  
);
```



## Data Definition Language (DDL)

### **DROP Statement:**

DROP is a DDL command used to delete/remove the database objects from the SQL database.

We can easily remove the entire table, view, or index from the database using this DDL command.

This example describes how to remove a database from the SQL database.

### **Syntax to remove a database:**

```
DROP DATABASE Database_Name;
```

Example:

```
DROP DATABASE Books;
```

This example describes how to remove the existing table from the SQL database.

**Syntax to remove a table:**

DROP TABLE Table\_Name;

Example:

DROP TABLE Student;

## Data Definition Language (DDL)

### ALTER Statement:

#### 1. ADD

Syntax:

```
ALTER TABLE table_name
```

```
ADD column_name column_definition [FIRST|AFTER existing_column];
```

```
ALTER TABLE table_name
```

```
ADD column_name1 column_definition [FIRST|AFTER existing_column],
```

```
ADD column_name2 column_definition [FIRST|AFTER existing_column];
```

Example: Here, we are adding a new column to the existing table.

```
ALTER TABLE Student
```

```
ADD Mobile_Number varchar(10);
```



## Data Definition Language (DDL)

### ALTER Statement:

#### 2. MODIFY

Syntax:

```
ALTER TABLE table_name
```

```
MODIFY (fieldname datatype(...));
```

Example: Modify a datatype in an existing table.

```
ALTER TABLE Student
```

```
MODIFY FirstName VARCHAR(30);
```

#### 3. DESCRIBE TABLE

Syntax:

```
DESCRIBE TABLE NAME;
```

Example: This query is used to view the table.

```
DESCRIBE Student;
```

## Data Definition Language (DDL)

### ALTER Statement:

#### 4. DROP Column

ALTER TABLE table\_name DROP COLUMN column\_name;

First, we need to specify the table name from which we want to remove the column.

Next, after the DROP COLUMN clause, we have to specify the column name that we want to delete from the table. It is to note that the COLUMN keyword is optional in the DROP COLUMN clause.

If we want to remove multiple columns from the table, execute the following statements:

ALTER TABLE table\_name

DROP COLUMN column\_1,

DROP COLUMN column\_2, .....;

## Difference between Delete and Truncate Command:

- ✓ The difference between DELETE and TRUNCATE command is the most common part of an interview question.
- ✓ They are mainly used to delete data from the database.
- ✓ The main difference between them is that the delete statement deletes data without resetting a table's identity, whereas the truncate command resets a particular table identity

## What is a DELETE command?

- ✓ It is a DML or data manipulation command used to delete records from a table that is not required in the database.
- ✓ It removes the complete row from the table and produces the count of deleted rows.
- ✓ We need the delete permission on the target table to execute this command.
- ✓ It also allows us to filter and delete any specific records using the WHERE clause from the table.
- ✓ It clarifies that we have a backup of our database before executing this command because we cannot recover the deleted records using this query.
- ✓ Therefore, the database backups allow us to restore the data whenever we need it in the future.

The following syntax explains the DELETE command to remove data from the table:

DELETE FROM table\_name [WHERE condition].

## What is a TRUNCATE command?

- ✓ DDL or data definition language command used to removes complete data from the table without removing the table structure.
- ✓ We cannot use the WHERE clause with this command, so that filtering of records is not possible.
- ✓ After executing this command, we cannot rollback the deleted data because the log is not maintained while performing this operation.
- ✓ The truncate command deallocates the pages instead of rows and makes an entry for the deallocating pages instead of rows in transaction logs. This command locks the pages instead of rows; thus, it requires fewer locks and resources. Note that we cannot use the truncate statement when a table is referenced by a foreign key or participates in an indexed view.
- ✓ The following syntax explains the TRUNCATE command to remove data from the table
- ✓ `TRUNCATE TABLE table_name;`



## DELETE vs. TRUNCATE Comparison Chart

Comparison Basis	DELETE	TRUNCATE
Definition	The delete statement is used to remove single or multiple records from an existing table depending on the specified condition.	The truncate command removes the complete data from an existing table but not the table itself. It preserves the table structure or schema.
Language	It is a DML (Data Manipulation Language) command.	It is a DDL (Data Definition Language) command.
WHERE	It can use the WHERE clause to filter any specific row or data from the table.	It does not use the WHERE clause to filter records from the table.
Permission	We need to have DELETE permission to use this command.	We need to have ALTER permission to use this command.
Working	This command eliminates records one by one.	This command deletes the entire data page containing the records.

## DELETE vs. TRUNCATE Comparison Chart

<b>Lock</b>	It will lock the row before deletion.	It will lock the data page before deletion.
<b>Table Identity</b>	This command does not reset the table identity because it only deletes the data.	It always resets the table identity.
<b>Transaction</b>	It maintains transaction logs for each deleted record.	It does not maintain transaction logs for each deleted data page.
<b>Speed</b>	Its speed is slow because it maintained the log.	Its execution is fast because it deleted entire data at a time without maintaining transaction logs.
<b>Trigger</b>	This command can also activate the trigger applied on the table and causes them to fire.	This command does not activate the triggers applied on the table to fire.
<b>Restore</b>	It allows us to restore the deleted data by using the COMMIT or ROLLBACK statement.	We cannot restore the deleted data after using executing this command.
<b>Indexed view</b>	It can be used with indexed views.	It cannot be used with indexed views.
<b>Space</b>	The DELETE statement occupies more transaction space than truncate because it maintains a log for each deleted row.	The TRUNCATE statement occupies less transaction space because it maintains a transaction log for the entire data page instead of each row.

## **Data Manipulation Language (DML):**

### **MySQL INSERT Statement:**

MySQL INSERT statement is used to store or add data in MySQL table within the database.

We can perform insertion of records in two ways using a single query in MySQL:

Insert record in a single row

Insert record in multiple rows

Syntax: (Single Record)

```
INSERT INTO table_name ( field1, field2,...fieldN )
```

```
VALUES ( value1, value2,...valueN );
```

In the above syntax, we first have to specify the table name and list of comma-separated columns. Second, we provide the list of values corresponding to columns name after the VALUES clause.

NOTE: Field name is optional. If we want to specify partial values, the field name is mandatory.

It also ensures that the column name and values should be the same. Also, the position of columns and corresponding values must be the same.



## MySQL INSERT Statement:

If we want to insert multiple records within a single command, use the following statement:

```
INSERT INTO table_name VALUES
```

```
( value1, value2,...valueN )
```

```
( value1, value2,...valueN )
```

```
.....
```

```
( value1, value2,...valueN );
```

In the above syntax, all rows should be separated by commas in the value fields.

## MySQL INSERT Example:

Let us understand how INSERT statements work in MySQL with the help of multiple examples.

First, create a table "People" in the database using the following command:

```
CREATE TABLE People(  
    id int NOT NULL AUTO_INCREMENT,  
    name varchar(45) NOT NULL,  
    occupation varchar(35) NOT NULL,  
    age int,  
    PRIMARY KEY (id)  
);
```

1. If we want to store single records for all fields, use the syntax as follows:

```
INSERT INTO People (id, name, occupation, age)  
VALUES (101, 'Peter', 'Engineer', 32);
```

## MySQL INSERT Example:

2. If we want to store multiple records, use the following statements where we can either specify all field names or don't specify any field.

```
INSERT INTO People VALUES
```

```
(102, 'Joseph', 'Developer', 30),
```

```
(103, 'Mike', 'Leader', 28),
```

```
(104, 'Stephen', 'Scientist', 45);
```

3. If we want to store records without giving all fields, we use the following partial field statements. In such case, it is mandatory to specify field names.

```
INSERT INTO People (name, occupation)
```

```
VALUES ('Stephen', 'Scientist'), ('Bob', 'Actor');
```

## Inserting Date in MySQL Table:

We can also use the INSERT STATEMENT to add the date in MySQL table.

MySQL provides several data types for storing dates such as DATE, TIMESTAMP, DATETIME, and YEAR. The default format of the date in MySQL is YYYY-MM-DD.

This format has the below descriptions:

YYYY: It represents the four-digit year, like 2020.

MM: It represents the two-digit month, like 01, 02, 03, and 12.

DD: It represents the two-digit day, like 01, 02, 03, and 31.

Following is the basic syntax to insert date in MySQL table:

```
INSERT INTO table_name (column_name, column_date) VALUES ('DATE: Manual Date', '2008-7-04');
```

## Data Manipulation Language (DML):

### SELECT Statement:

The SELECT statement in MySQL is used to fetch data from one or more tables.

We can retrieve records of all fields or specified fields that match specified criteria using this statement.

Syntax:

SELECT field\_name1, field\_name 2,... field\_nameN

FROM table\_name1, table\_name2...

[WHERE condition]

[GROUP BY field\_name(s)]

[HAVING condition]

[ORDER BY field\_name(s)]

[OFFSET M ][LIMIT N];

## SELECT Statement:

Parameter Name	Descriptions
field_name(s) or *	It is used to specify one or more columns to returns in the result set. The asterisk (*) returns all fields of a table.
table_name(s)	It is the name of tables from which we want to fetch data.
WHERE	It is an optional clause. It specifies the condition that returned the matched records in the result set.
GROUP BY	It is optional. It collects data from multiple records and grouped them by one or more columns.
HAVING	It is optional. It works with the GROUP BY clause and returns only those rows whose condition is TRUE.
ORDER BY	It is optional. It is used for sorting the records in the result set.
OFFSET	It is optional. It specifies to which row returns first. By default, It starts with zero.
LIMIT	It is optional. It is used to limit the number of returned records in the result set.



## MySQL SELECT Statement Example:

Let us understand how SELECT command works in MySQL with the help of various examples. Suppose we have a table named **employee\_detail** that contains the following data:

ID	Name	Email	Phone	City	Working_hours
1	Peter	peter@javatpoint.com	49562959223	Texas	12
2	Suzi	suzi@javatpoint.com	70679834522	California	10
3	Joseph	joseph@javatpoint.com	09896765374	Alaska	14
4	Alex	alex@javatpoint.com	97335737548	Los Angeles	9
5	Mark	mark@javatpoint.con	78765645643	Washington	12
6	Stephen	stephen@javatpoint.com	986345793248	New York	10

## **SELECT Statement:**

1. If we want to retrieve a single column from the table, we need to execute the below query:

```
SELECT Name FROM employee_detail;
```

2. If we want to query multiple columns from the table, we need to execute the below query:

```
SELECT Name, Email, City FROM employee_detail;
```

3. If we want to fetch data from all columns of the table, we need to use all column's names with the select statement. Specifying all column names is not convenient to the user, so MySQL uses an asterisk (\*) to retrieve all column data as follows:

```
SELECT * FROM employee_detail;
```



## **WHERE Clause:**

MySQL WHERE Clause is used with SELECT, INSERT, UPDATE and DELETE clause to filter the results. It specifies a specific position where you have to do the operation.

Syntax:

WHERE conditions;

Parameter:

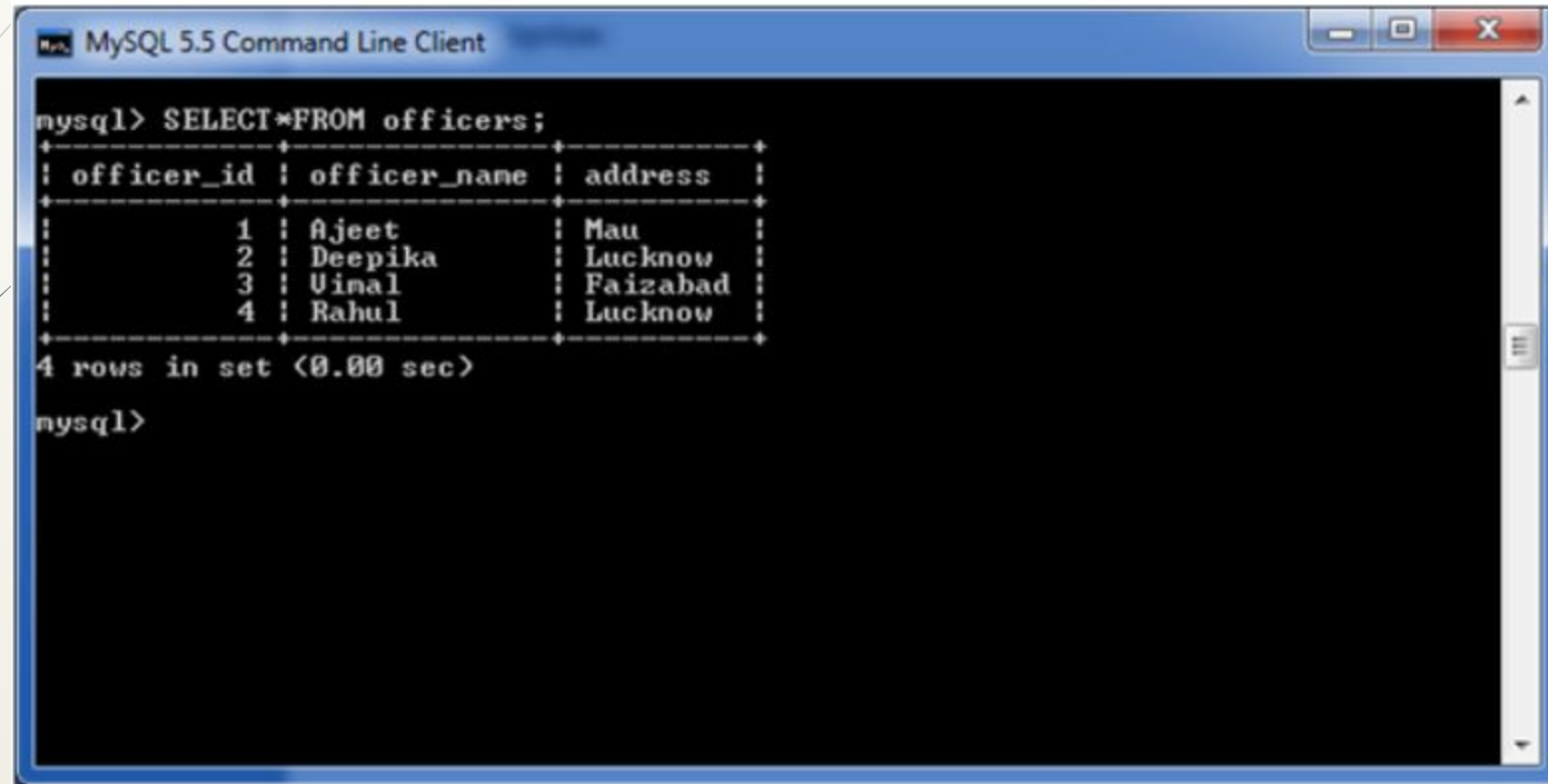
conditions: It specifies the conditions that must be fulfilled for records to be selected.

### **MySQL WHERE Clause with single condition**

Let's take an example to retrieve data from a table "officers".

## WHERE Clause:

Table structure:



The screenshot shows a MySQL 5.5 Command Line Client window. The command prompt shows the command `mysql> SELECT * FROM officers;` and the output is a table with 4 rows. The columns are `officer_id`, `officer_name`, and `address`. The data rows are: (1, Ajeet, Mau), (2, Deepika, Lucknow), (3, Uimal, Faizabad), and (4, Rahul, Lucknow). Below the table, it says "4 rows in set (0.00 sec)".

```
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 (0.00 sec)
mysql>
```

```
SELECT *  
FROM officers  
WHERE address = 'Mau';
```

### **MySQL WHERE Clause with AND condition**

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
AND officer_id < 5;
```

## **WHERE Clause with OR condition**

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
OR address = 'Mau';
```

## **MySQL WHERE Clause with combination of AND & OR conditions**

```
SELECT *  
FROM officers  
WHERE (address = 'Mau' AND officer_name = 'Ajeet')  
OR (officer_id < 5);
```

## GROUP BY Clause

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

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

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



## Parameters

**expression1, expression2, ... expression\_n:** It specifies the expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

**aggregate\_function:** It specifies a function such as SUM, COUNT, MIN, MAX, or AVG etc.  
**tables:** It specifies the tables, from where you want to retrieve the 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.

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

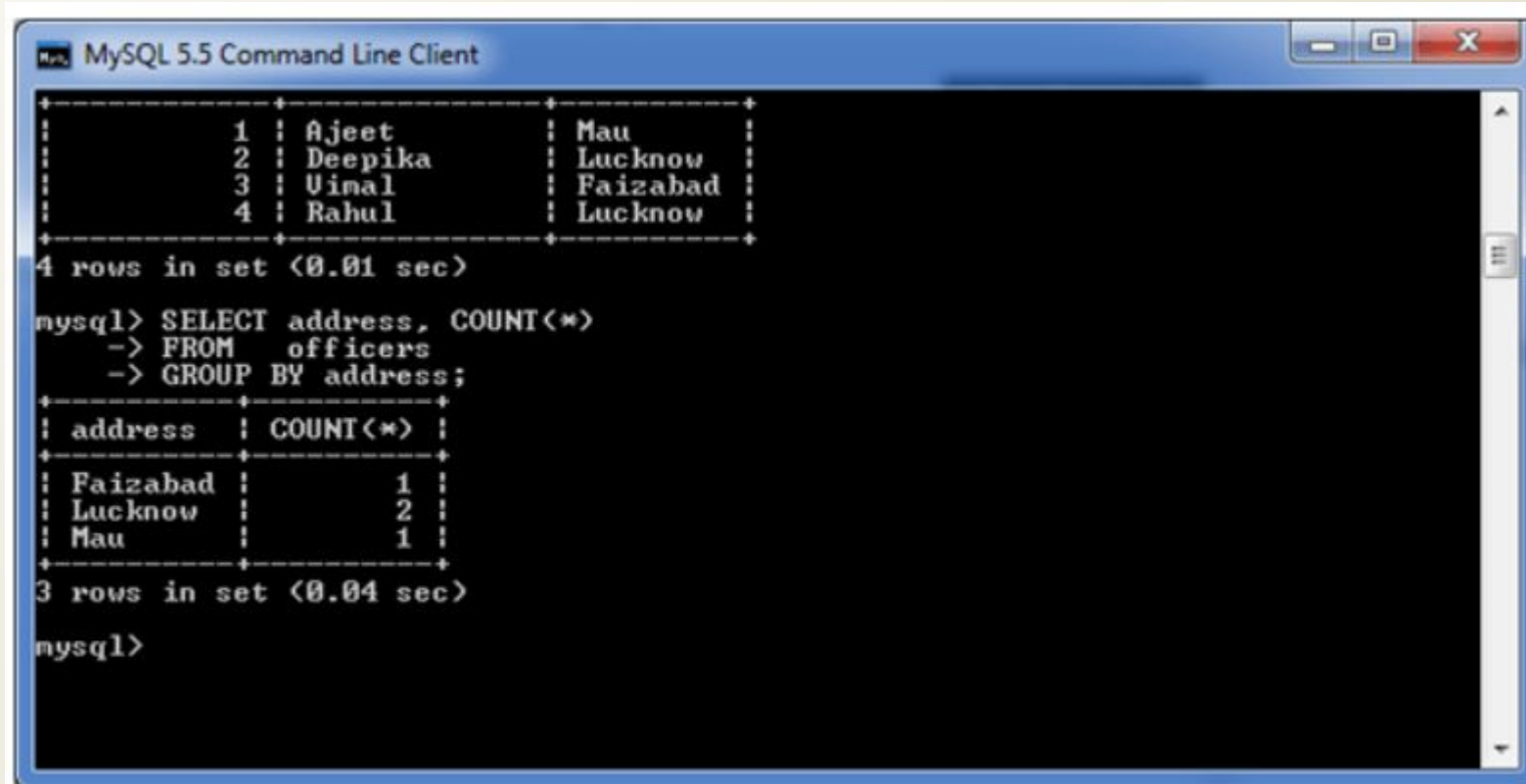
Consider a table named "officers" table, having the following records.

## Parameters

SELECT address, COUNT(\*)

FROM officers

GROUP BY address;



The screenshot shows a MySQL 5.5 Command Line Client window. It displays the results of a query and the execution of a new query. The first query shows 4 rows in a set, and the second query shows 3 rows in a set.

```
MySQL 5.5 Command Line Client
```

	id	name	address
1	1	Ajeet	Mau
2	2	Deepika	Lucknow
3	3	Uimal	Faizabad
4	4	Rahul	Lucknow

4 rows in set (0.01 sec)

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

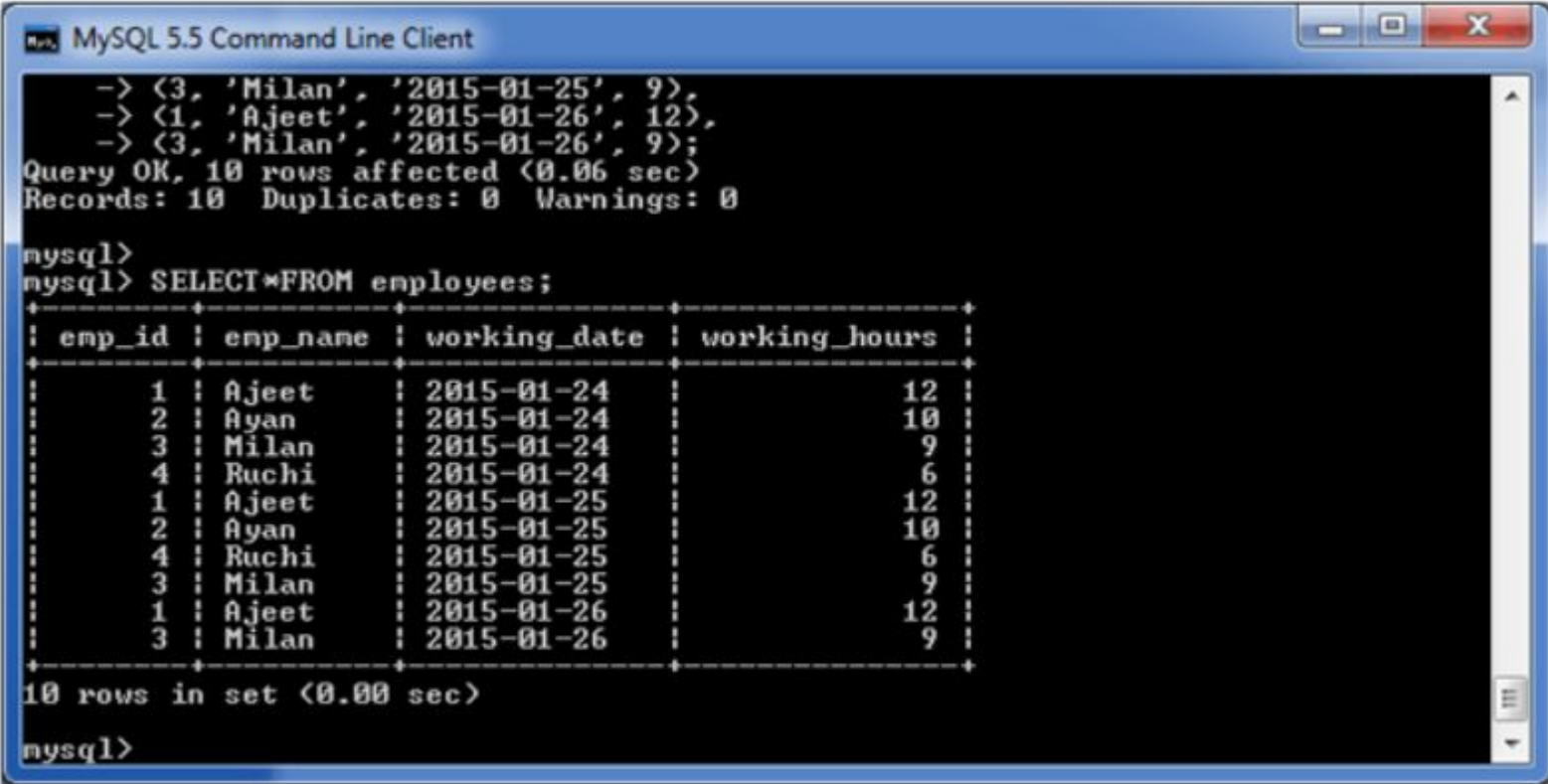
address	COUNT(*)
Faizabad	1
Lucknow	2
Mau	1

3 rows in set (0.04 sec)

```
mysql>
```

## 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)

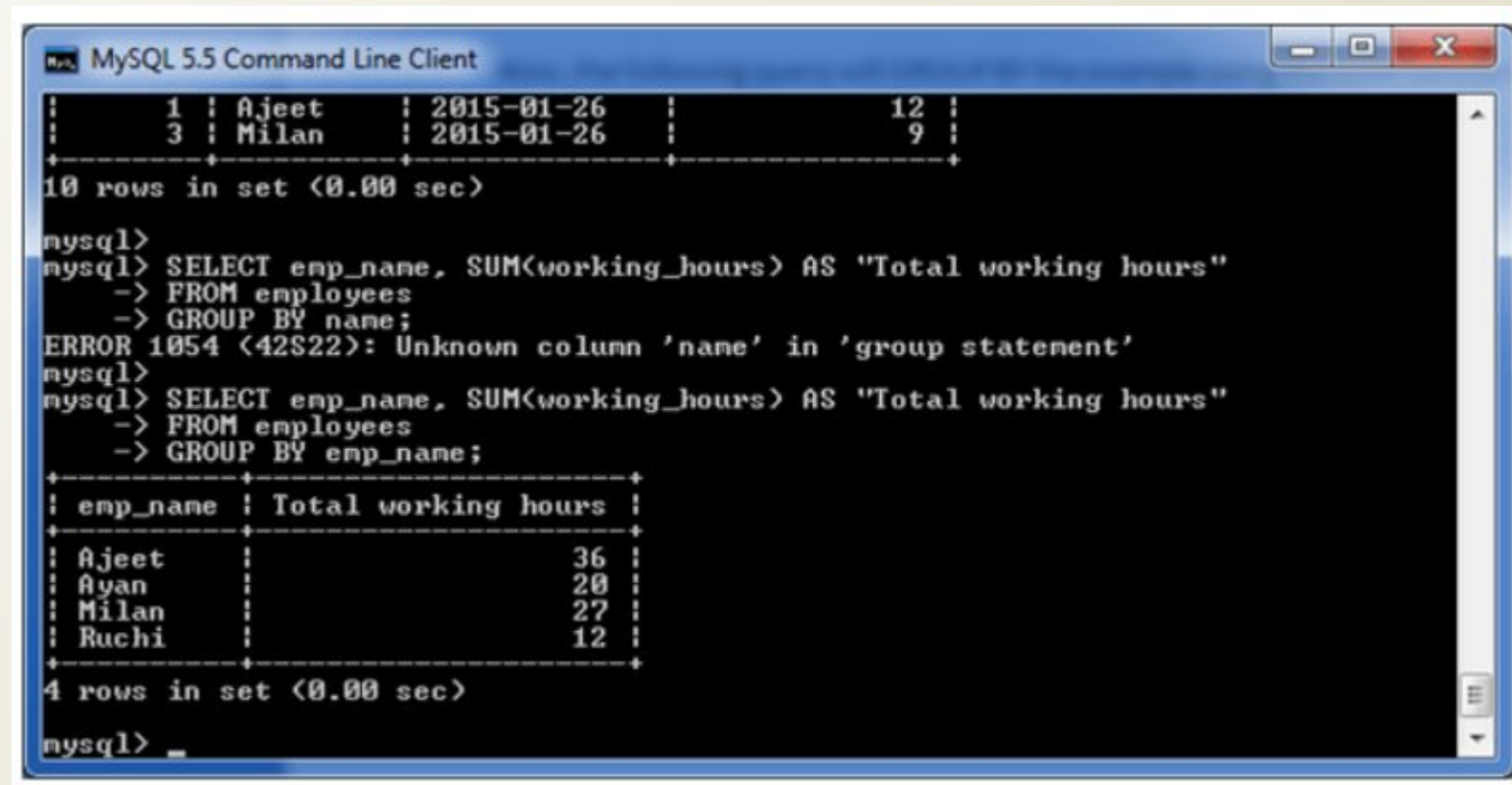
mysql>
```

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.

```
SELECT emp_name, SUM(working_hours) AS "Total working hours"
```

```
FROM employees
```

```
GROUP BY emp_name;
```



The screenshot shows a MySQL 5.5 Command Line Client window. It 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;`. The results show 4 rows in the set, with columns `emp_name` and `Total working hours`. The data is as follows:

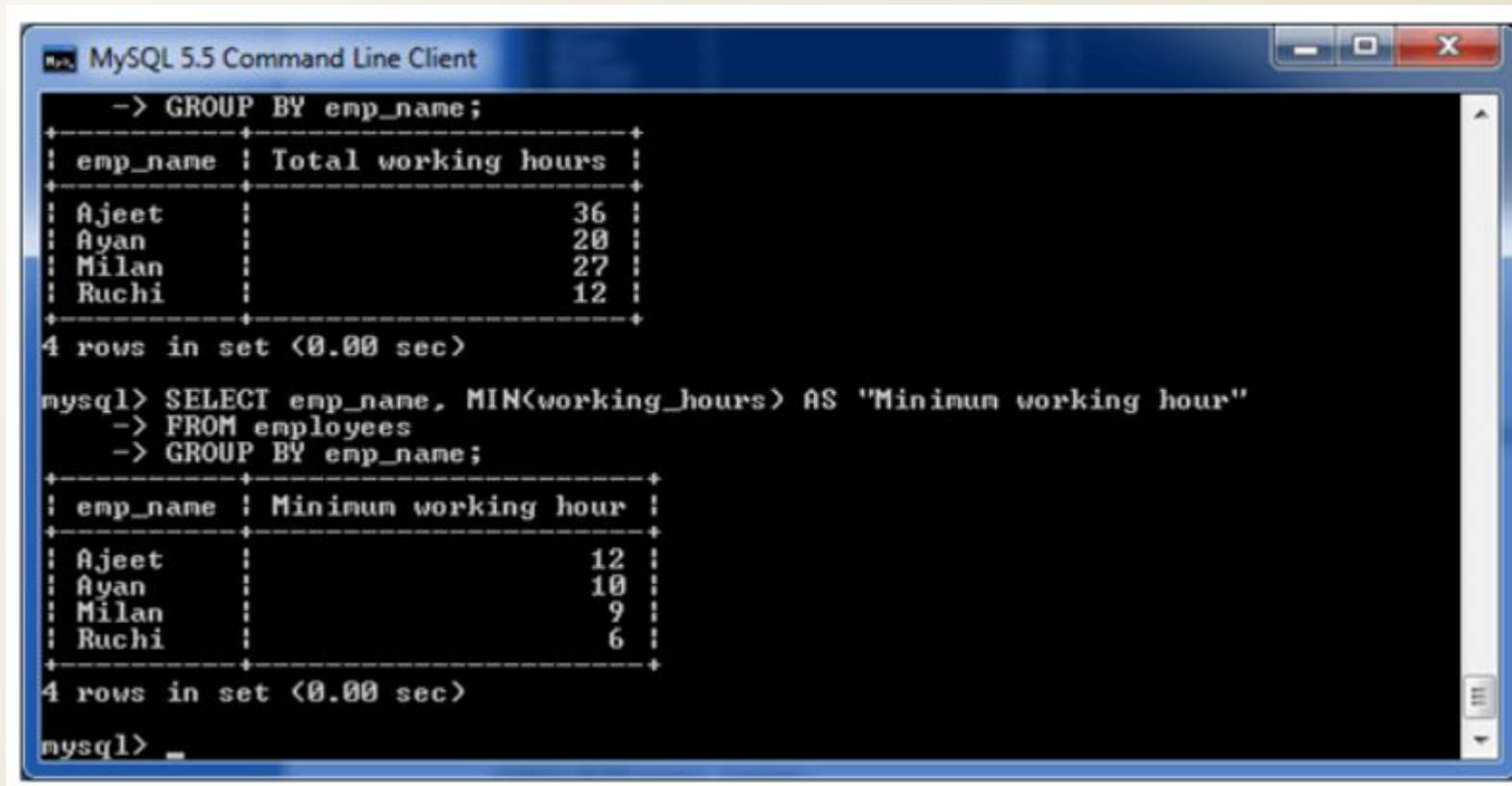
emp_name	Total working hours
Ajeet	36
Ayan	20
Milan	27
Ruchi	12

The client also shows an error message: `ERROR 1054 (42S22): Unknown column 'name' in 'group statement'` when the query was first executed with `GROUP BY name;`.

## GROUP BY Clause with MIN function

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

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



The screenshot shows the MySQL 5.5 Command Line Client interface. It displays two SQL queries and their corresponding results. The first query is a simple GROUP BY statement, and the second query uses the MIN function to find the minimum working hours for each employee.

```
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



## **GROUP BY Clause with MAX function**

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

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

## **GROUP BY Clause with AVG function**

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

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

## HAVING Clause

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;
```

## Parameters

**aggregate\_function:** It specifies any one of the aggregate function such as SUM, COUNT, MIN, MAX, or AVG.

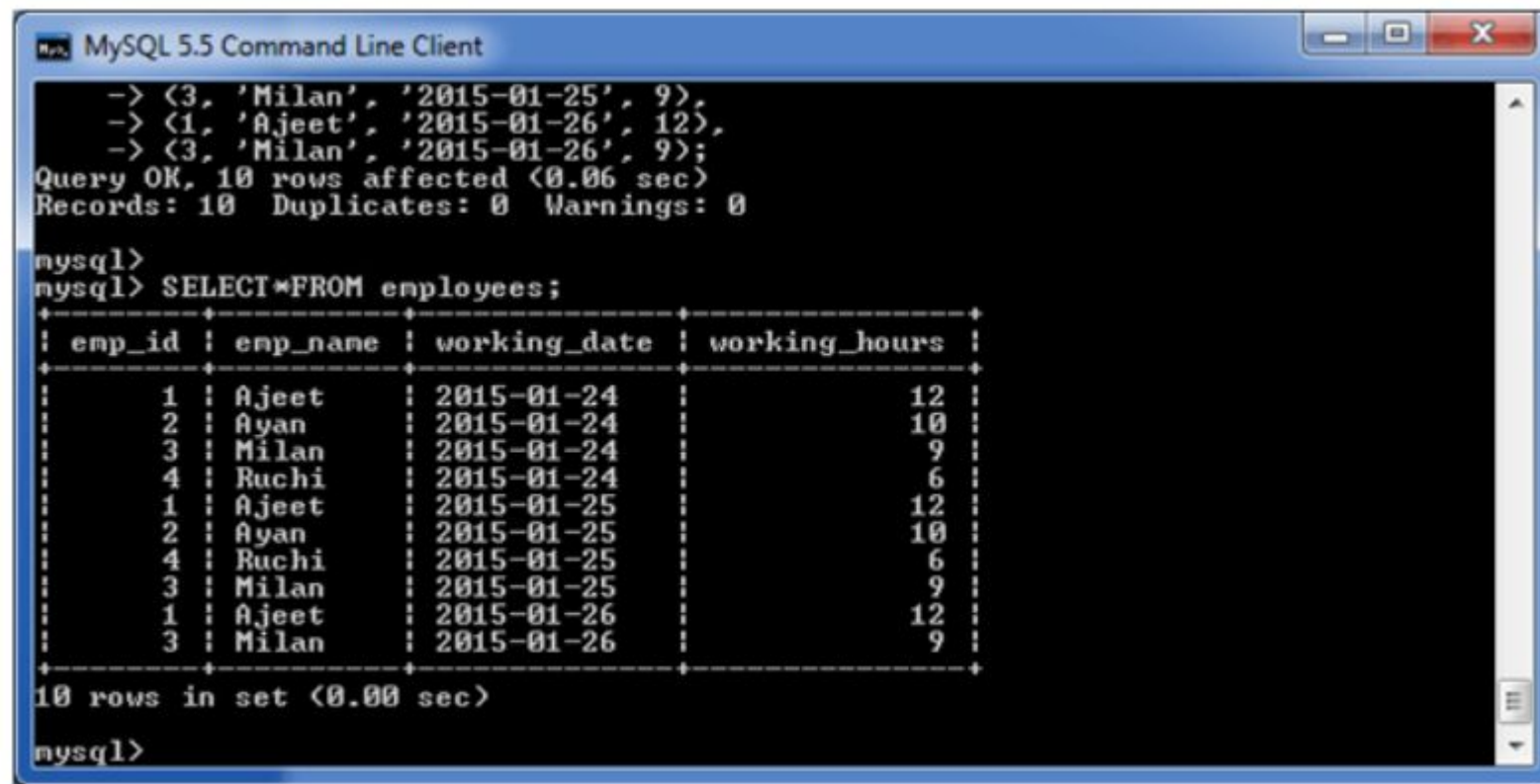
**expression1, expression2, ... expression\_n:** It specifies the expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

**WHERE conditions:** It is optional. It specifies the conditions for the records to be selected.

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

## 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>
```

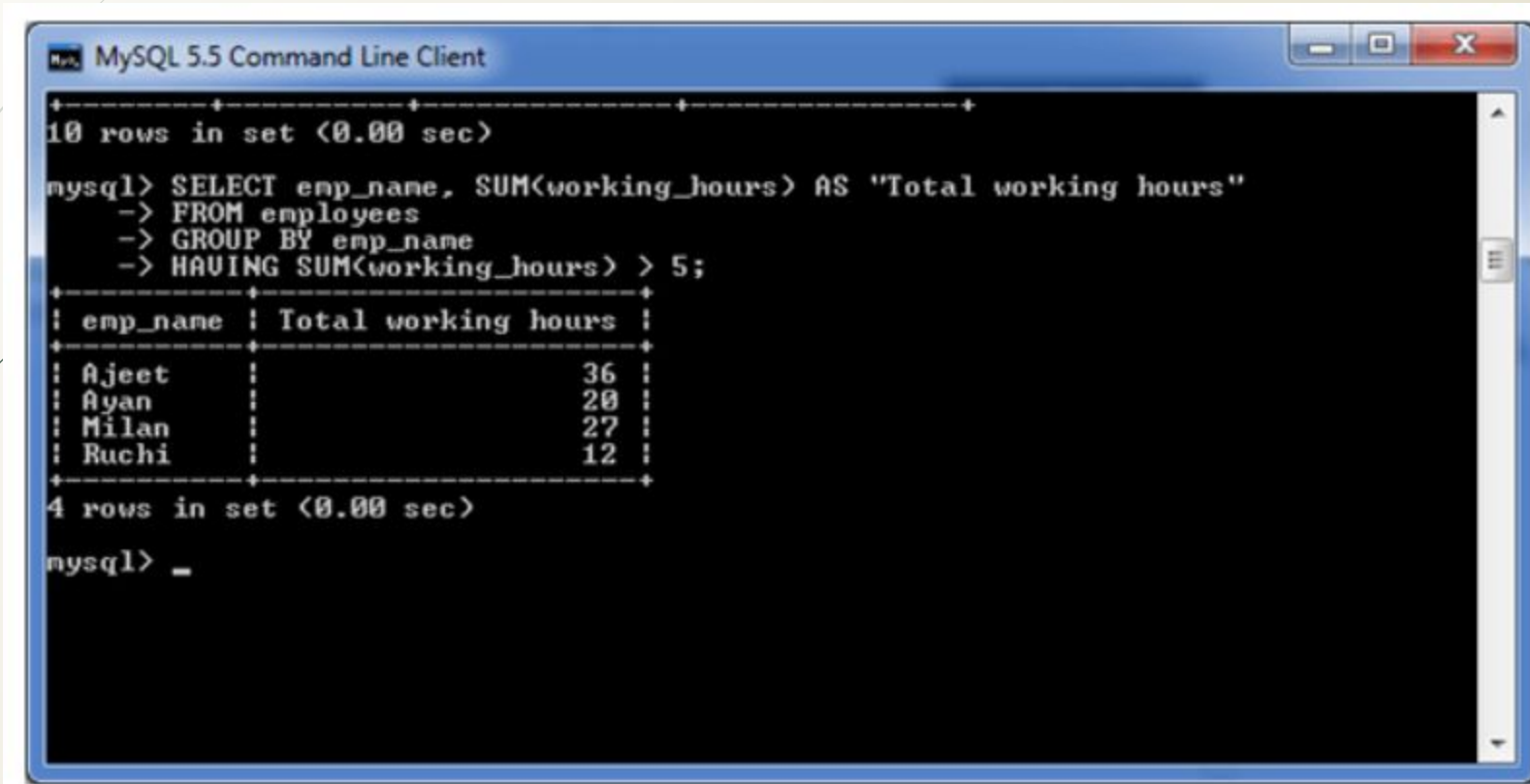
## HAVING Clause with SUM function

Here, we use the SUM function with the HAVING Clause to return the emp\_name and sum of their working hours.

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



## HAVING Clause with SUM function



```
MySQL 5.5 Command Line Client
10 rows in set (0.00 sec)

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

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

4 rows in set (0.00 sec)

mysql> _
```

## **ORDER BY Clause**

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

SELECT expressions

FROM tables

[WHERE conditions]

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 provided).

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

## **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.

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

## **ORDER BY: with ASC attribute**

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

### **ORDER BY: with DESC attribute**

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

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

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

## **UPDATE Statement:**

MySQL UPDATE query is a DML statement used to modify the data of the MySQL table within the database.

The UPDATE statement is used with the SET and WHERE clauses. The SET clause is used to change the values of the specified column. We can update single or multiple columns at a time.

### **Syntax**

UPDATE table\_name

SET column\_name1 = new-value1,  
    column\_name2=new-value2, ...

[WHERE Clause]



## Parameter Descriptions

**table\_name** - It is the name of a table in which we want to perform updation.

**column\_name** - It is the name of a column in which we want to perform updation with the new value using the SET clause. If there is a need to update multiple columns, separate the columns with a comma operator by specifying the value in each column.

**WHERE Clause** It is optional. It is used to specify the row name in which we are going to perform updation. If we omit this clause, MySQL updates all rows.

### Note:

This statement can update values in a single table at a time.

We can update single or multiple columns all together with this statement.

Any condition can be specified by using the WHERE clause.

WHERE clause is very important because sometimes we want to update only a single row, and if we omit this clause, it accidentally updates all rows of the table.

The UPDATE command supports these modifiers in MySQL:

**LOW\_PRIORITY:** This modifier instructs the statement to delay the UPDATE command's execution until no other clients reading from the table. It takes effects only for the storage engines that use only table-level locking.

**IGNORE:** This modifier allows the statement to do not abort the execution even if errors occurred. If it finds duplicate-key conflicts, the rows are not updated.

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name  
    SET column_assignment_list  
    [WHERE condition]
```

**Example:**

Let us understand the UPDATE statement with the help of various examples. Suppose we have a table "trainer" within the "testdb" database. We are going to update the data within the "trainer" table.

## Update Single Column

This query will update the email id of Java course with the new id as follows:

```
UPDATE trainer
```

```
SET email = 'mike@tutorialandexamples.com'
```

```
WHERE course_name = 'Java';
```

## Update Multiple Columns

The UPDATE statement can also be used to update multiple columns by specifying a comma-separated list of columns. Suppose we have a table as below:

This statement explains will update the name and occupation whose id = 105 in the People table as follows:

```
UPDATE People
```

```
SET name = 'Mary', occupation = 'Content Writer'
```

```
WHERE id = 105;
```

We can verify the output below:

### **UPDATE Statement to Replace String**

We can also use the UPDATE statement in MySQL to change the string name in the particular column. The following example updates the domain parts of emails of Android course:

```
UPDATE Trainer_table
```

```
SET email = REPLACE(email, '@javatpoint.com', '@tutorialandexample.com')
```

```
WHERE course_name = 'Testing';
```