

UNIT - 4

- PHP with MySQL
- Creating Connection
- Selecting Database
- Perform Database (query)
- Use returned data,
- close connections
- file handling in PHP – reading and writing from and to FILE.
- Using MySQL from PHP
- Self Learning: Introduction to MySQL,
- CRUD - Select statements,
- Creating Database/Tables,
- Inserting values, updating and Deleting.

PHP | MySQL Database Introduction

What is MySQL?

[MySQL](#) is an open-source relational database management system (RDBMS). It is the most popular database system used with PHP.

MySQL is developed, distributed, and supported by Oracle Corporation.

- The data in a MySQL database are stored in tables which consists of columns and rows.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use database system. It uses standard SQL
- MySQL compiles on a number of platforms.

How to connect PHP with MySQL Database?

PHP 5 and later can work with a MySQL database using:

- MySQLi extension.
- PDO (PHP Data Objects).

Difference Between MySQLi and PDO

- PDO works on 12 different database systems, whereas MySQLi works only with MySQL databases.
- Both PDO and MySQLi are object-oriented, but MySQLi also offers a procedural API.
- If at some point of development phase, the user or the development team wants to change the database then it is easy to that in PDO than MySQLi as PDO supports 12 different database systems. He would have to only change the connection string and a few queries. With MySQLi, he will need to rewrite the entire code including the queries.

There are three ways of working with MySQL and PHP

- MySQLi (object-oriented)
- MySQLi (procedural)
- PDO

Connecting to MySQL database using PHP

There are 3 ways in which we can connect to MySQL from PHP as listed above and described below:

- Using MySQLi object-oriented procedure: We can use the MySQLi object-oriented procedure to establish a connection to MySQL database from a PHP script.

Syntax:

```
<?php
$servername = "localhost";
$username = "username";
```

```

$password = "password";

// Creating connection
$conn = new mysqli($servername, $username, $password);

// Checking connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>

```

Output:

Connected successfully

Explanation: We can create an instance of the mysqli class providing all the necessary details required to establish the connection such as host, username, password etc. If the instance is created successfully then the connection is successful otherwise there is some error in establishing connection.

Using MySQLi procedural procedure : There is also a procedural approach of MySQLi to establish a connection to MySQL database from a PHP script as described below.

Syntax:

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Creating connection
$conn = mysqli_connect($servername, $username, $password);

// Checking connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";

```

?>

Output:

Connected successfully

Explanation: In MySQLi procedural approach instead of creating an instance we can use the `mysqli_connect()` function available in PHP to establish a connection. This function takes the information as arguments such as host, username , password , database name etc. This function returns MySQL link identifier on successful connection or FALSE when failed to establish a connection.

Using PDO procedure: PDO stands for PHP Data Objects. That is, in this method we connect to the database using data objects in PHP as described below:

Syntax:

```
<?php
```

```
$servername = "localhost";
```

```
$username = "username";
```

```
$password = "password";
```

```
try {
```

```
    $conn = new PDO("mysql:host=$servername;dbname=myDB",  
$username, $password);
```

```
    // setting the PDO error mode to exception
```

```
    $conn->setAttribute(PDO::ATTR_ERRMODE,  
PDO::ERRMODE_EXCEPTION);
```

```
    echo "Connected successfully";
```

```
}
```

```
catch(PDOException $e)
```

```
{
```

```
    echo "Connection failed: " . $e->getMessage();
```

```
}
```

```
?>
```

Output:

Connected successfully

Explanation:The exception class in PDO is used to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Closing A Connection:

When we establish a connection to MySQL database from a PHP script , we should also disconnect or close the connection when our work is finished. Here we have described the syntax of closing the connection to a MySQL database in all 3 methods described above. We have assumed that the reference to the connection is stored in \$conn variable.

Using MySQLi object oriented procedure

Syntax:

```
$conn->close();
```

Using MySQLi procedural procedure

Syntax

```
mysqli_close($conn);
```

Using PDO procedure

Syntax

```
$conn = null;
```

PHP - Function MySQLi Select DB

Syntax:

```
mysqli_select_db(connection,dbname);
```

Definition and Usage:

This function changes the default database.

Return Values:

It returns true on success or false on failure

Parameters:

Sr.No	Parameters & Description
-------	--------------------------

1	connection It specifies the connection to use
2	dbname It specifies about the database to use.

Example

Try out the following example

```
<?php
```

```
    $connection_mysql =  
    mysqli_connect("localhost","user","password","db");
```

```
    if (mysqli_connect_errno($connection_mysql)){  
        echo "Failed to connect to MySQL: " . mysqli_connect_error();  
    }
```

```
    mysqli_select_db($connection_mysql,"test");  
    mysqli_close($connection_mysql);
```

```
?>
```

Performing Queries from PHP

Almost every Web application has a need to somehow store data. This data can be shopping cart information, quotes, or just about anything else that you can imagine.

basic steps of accessing a MySQL database from within a PHP script:

1. Connect to the MySQL database.
2. Select the database to use.
3. Perform the desired SQL queries.
4. Retrieve any data returned from the SQL queries.
5. Close the connection to the database.

Connecting to the Database

The first step in any database transaction from PHP is connecting to the database. When you're using MySQL directly, this is analogous to executing the MySQL client application; however, in PHP this is done by using the `mysqli_connect()` function. The syntax for this function is as follows:

```
mysqli_connect([$hostname [, $username [, $password  
[, $dbname [, $port [, $socket]]]]]);
```

As you can see, the `mysqli_connect()` function takes a number of parameters, all of which are optional and most of which should be fairly self-explanatory. Starting from the left, the first parameter is the `$hostname` parameter, which represents the hostname where the server is located. Because it is often the case that the MySQL server is located on the same machine as your Web server, this often is the "localhost" value. The next two parameters, `$username` and `$password`, represent the username and password to use when authenticating to the MySQL server. The third parameter, `$dbname`, is a string representing the name of the database to use by default. The last two parameters, `$port` and `$socket`, are used in conjunction with the `$hostname` parameter to specify a specific port or socket to be used for the connection.

When executed, the `mysqli_connect()` function will attempt to connect to the MySQL server using the parameters provided. Upon success, this function will return a resource representing a connection to the database or `false` on failure.

In general, at a minimum this function requires three parameters (`$hostname`, `$username`, and `$password`); however, it is not uncommon to see the fourth parameter, `$dbname`, also used to select a default database.

Selecting a Database

While we are on the topic of selecting a database to use, it is appropriate to point out that a database does not necessarily have to be selected through the `mysqli_connect()` function. After a connection has been created, the `mysqli_select_db()` function can be used to select the current database in the same way the `USE` SQL command was used

from the client. The syntax for the `mysqli_select_db()` function is as follows:

```
mysqli_select_db($link, $dbname);
```

`$link` is the database connection resource returned from the `mysqli_connect()` function, and `$dbname` is a string representing the name of the database to select. This function will return a Boolean `TRUE` if the new database was successfully selected or `false` on failure.

Performing a Basic Query

Now that we know how to connect to a MySQL server and select a database to use, it's time to start performing SQL queries against it. To perform queries, we will use the `mysqli_query()` function with the following syntax:

```
mysqli_query($link, $query [, $resultmode]);
```

`$link` is the database connection and `$query` is a string containing a single SQL query without a terminating semicolon (;) or (\g). The third optional parameter, `$resultmode`, determines how the resultset from the query will be transferred back to PHP. This parameter is either the `MYSQLI_USE_RESULT` or `MYSQLI_STORE_RESULT` (the default value) constants. This parameter is used to indicate whether the MySQLi extension should copy the entire resultset to memory after a query (`MYSQLI_STORE_RESULT`), or only the current row in the resultset. In practical terms, storing the entire resultset in memory allows your scripts to access any row within the resultset arbitrarily when otherwise each row could be read only sequentially. Unless you are working with particularly large data sets, it is generally acceptable to ignore this parameter. Upon success, `mysqli_query()` will return a resource representing the result of the query or a Boolean `false` on failure.

Fetching Resultset Rows

When performing queries that modify the tables within a database, such as INSERT and UPDATE, generally there is no data to return from the

database. However, queries such as SELECT, which retrieve data from the database, require that data must somehow be accessed from within PHP. When returning the results from a resultset into PHP, various options are available, each providing a unique usefulness depending on the circumstance.

The most general of the result-retrieving functions is the `mysqli_fetch_array()` function. The syntax for this function is as follows:

```
mysqli_fetch_array($result [, $array_type])
```

`$result` is the resource representing the query result returned from a call to the `mysqli_query()` function, and the optional parameter `$array_type` is one of the following constants:

MYSQLI_ASSOC Return an associative array.

MYSQLI_NUM Return an enumerated array.

MYSQLI_BOTH Return both an enumerated and associative array.

If no value is provided, the **MYSQLI_BOTH** constant is the default value.

Applying this function to your scripts is a fairly simple task. After a query has been performed using the `mysqli_query()` function, the result of that query is passed to a function such as the `mysqli_fetch_array()` function, which will return a single row of the result table.

For every subsequent call, `mysqli_fetch_array()` will return another row from the result table until there are no more rows available, in which case `mysqli_fetch_array()` will return a Boolean **false**.

As its name implies, the `mysqli_fetch_array()` function will return each row as an array. The details of how that array is formatted, however, depend on the optional `$array_type` parameter.

If the **MYSQLI_ASSOC** constant is used, the `mysqli_fetch_array()` function will return an associative array whose keys represent the column names of the resultset and whose values represent the appropriate value for each column for the current row.

On the other hand, if the `MYSQLI_NUM` constant is used, `mysqli_fetch_array()` will return an enumerated array representing the current row, where index zero is the first column, 1 is the second, and so on. The final (and default) constant `MYSQLI_BOTH`, as its name implies, returns an array that contains both associative and enumerated keys and values for the current resultset.

NOTE

Along with the `mysqli_fetch_array()` function, PHP also provides the `mysqli_fetch_row()` and `mysqli_fetch_assoc()` functions. These functions take a single parameter the result resource returned from `mysqli_query()` and returns either an enumerated or associative array. Functionally, the `mysqli_fetch_row()` function is identical to calling the `mysqli_fetch_array()` function using the `MYSQLI_NUM` constant for the `$array_type` parameter. Likewise, the `mysqli_fetch_assoc()` function is identical to calling `mysqli_fetch_array()` using the `MYSQLI_ASSOC` parameter.

To illustrate the use of the `mysqli_fetch_array()` function, consider the following example, which will retrieve all the results from a hypothetical query into a simple HTML table (see [Listing 24.4](#)):

Listing 24.4. Using the `mysqli_fetch_array()` Function

```
<?php

$link = mysqli_connect("localhost", "username", "password");
if(!$link) {
    trigger_error("Could not connect to the database",
E_USER_ERROR);
}

mysqli_select_db($link, "unleashed");
$result = mysqli_query("SELECT first, last FROM members");

if(!$result) {
    trigger_error("Could not perform the specified query",
E_USER_ERROR);
}
```

```

    echo "<TABLE><TR><TD>First Name</TD><TD>Last
Name</TD></TR>";
    while($row = mysqli_fetch_array($result)) {
        echo "<TR>";
        echo "<TD>{$row['first']}</TD><TD>{$row['last']}</TD>";
        echo "</TR>";
    }
    echo "</TABLE>";
    mysqli_close($link);

?>

```

As you can see in [Listing 24.4](#), we start by connecting to our MySQL database using the `mysqli_connect()` function and then proceed to select the database using the `mysqli_select_db()` function. After the database has been selected, a **SELECT** query can be performed (in this case to retrieve the first and last name from the hypothetical table `members`) using the `mysqli_query()` function.

At this point, the `$result` variable contains a resource representing the resultset of the query **SELECT first, last FROM members**. To retrieve the data from that resultset, we use the `mysqli_fetch_array()` function to return the first row of the resultset and store it as an array in the `$row` variable. The results for that row are then printed and the process continues until the entire resultset has been traversed.

Under certain circumstances, it may be beneficial to be able to randomly access any particular row within a resultset instead of sequentially, as has been introduced so far. To accomplish this, the "current" row that MySQLi will retrieve using a function such as `mysqli_fetch_array()` must be adjusted using the `mysqli_data_seek()` function. The syntax for this function is as follows:

```
mysqli_data_seek($result, $row_num);
```

`$result` is the MySQLi resultset resource and `$row_num` is the zero-indexed row number to move to.

NOTE

As previously indicated, the `mysqli_data_seek()` function is available only when the resultset was stored in memory by passing the `MYSQLI_STORE_RESULT` constant to `mysqli_query()` or the equivalent.

Counting Rows and Columns

Often, it is useful to know how many rows or columns exist for a given resultset. For these purposes, MySQLi provides the `mysql_num_rows()` and `mysqli_num_fields()` functions, whose syntax follows:

```
mysqli_num_rows($result)
mysqli_num_fields($result)
```

In each function, `$result` is the resource representing the resultset. Each of these functions will return a total count of its respective rows or columns for the provided resultset. If no rows were returned or an error occurred, these functions will return a Boolean `false`.

Freeing Results

After a query is performed, the resultset for that query is stored in memory until the PHP script that performed the query is terminated. Although generally this is acceptable, when you are working with large resultsets, it becomes important to free the resultset in memory. This can be done using the `mysqli_free_result()` function as follows:

```
mysqli_free_result($result)
```

`$result` is the resource representing the resultset from a query. As I stated previously, although it is not always necessary to free the memory from a resultset, it is considered best practice to do so when a resultset is no longer needed.

NOTE

Using the `unset()` function will not free the result! You must use the `mysqli_free_result()` function.

Retrieving Error Messages

Now that we have gone over some of the fundamentals of using the MySQLi extension, before we go any further it's important to discuss the tools available when something goes wrong (when an error occurs).

The first step in dealing with errors is to realize that an error has occurred. Although at times this can be obvious (a visible error message is displayed), in most cases the actual function where the error occurred does not produce any visible warning or error. Instead, the MySQLi function in question will return a Boolean `false` instead of the desired resource or expected value. When such a situation occurs, the MySQLi extension will provide the script with an integer error code as well as a string describing the error. To retrieve these values, we use the `mysqli_errno()` and `mysqli_error()` functions. The syntax for these functions is as follows:

```
mysqli_errno($link);  
mysqli_error($link);
```

In both situations, `$link` is the resource representing the database connection. These two functions will return the most recent error code and description associated with the provided database connection.

NOTE

As you may have noticed, the error-reporting functionality in the MySQLi extension requires that a database connection actually exist to work. Hence, failures that occur during a call to the `mysqli_connect()` cannot use these error-reporting functions to determine the cause of the error. Rather, use the `mysqli_connect_errno()` and `mysqli_connect_error()` functions that accept no parameters.

To illustrate the use of the MySQLi extension's error-reporting functions, consider an example of their use in [Listing 24.5](#):

Listing 24.5. Using `mysqli_error()` and `mysqli_errno()`

```
<?php
```

```
    $link = mysqli_connect("hostname", "username", "password",  
"mydatabase");
```

```
    if(!$link) {  
        $mysql_error = "Connection error: ".mysqli_connect_error();  
        die($mysql_error);  
    }
```

```
    $result = mysqli_query($link, "SELECT * FROM foo");
```

```
    if(!$result) {  
        $errno = mysqli_errno($link);  
        $error = mysqli_error($link);  
        die("Query Error: $error (code: $errno)");  
    }
```

```
?>
```

In the example found in [Listing 24.2](#), note that there are two potential places where an error can occur. The first is during the call to the `mysqli_connect()` function. Because an error at that stage would leave the script without a valid database resource, the `mysqli_connect_error()` and `mysqli_connect_errno()` must be used. In the second error check for the `mysqli_query()` function, however, the error-reporting functions can be used to determine why the query failed.

Closing the Database Connections

Although PHP will automatically take care of closing any outstanding database connections (as appropriate), a connection to a database can be explicitly closed using the `mysqli_close()` function with the following syntax:

```
mysqli_close($link)
```

`$link` is the resource representing the database connection to close.

Executing Multiple Queries

One of the features most lacking in the old MySQL extension is the capability to perform multiple queries from a single PHP command. In MySQLi, such an action is now possible through the use of the `mysqli_multi_query()` function. The syntax of this function is as follows:

```
mysqli_multi_query($link, $queries);
```

`$link` is a valid MySQLi database connection and `$queries` is one or more SQL queries separated by a semicolon character. When executed, the `mysqli_multi_query()` function returns a Boolean indicating the success of the operation.

Unlike the `mysqli_query()` function, note that the `mysqli_multi_query()` function does not return a result directly. To retrieve the first resultset of a multiquery operation, either the `mysqli_store_result()` or `mysqli_use_result()` functions must be used. As was the case with the `$resultmode` parameter of the `mysqli_query()` function, these functions determine how the MySQL client will access the data contained within the resultset. The syntax of these functions follows:

```
mysqli_store_result($link);  
mysqli_use_result($link);
```

`$link` is the MySQLi database link. When executed, this function will return a result resource for the resultset. This resultset may then be

used in the normal fashion using the MySQLi API to retrieve the individual rows.

Because the `mysqli_multi_query()` function executes multiple different queries sequentially, one or more resultsets may be retrieved. The first of these resultsets will be available immediately after the `mysqli_multi_query()` function is executed. To advance to the next resultset, two functions are provided to assist you. The first of these functions is the `mysqli_more_results()` function with the following syntax:

```
mysqli_more_results($link);
```

`$link` is the MySQLi database link. This function will return a Boolean value indicating whether more resultsets are awaiting processing. To access the next available resultset, the `mysqli_next_result()` function is used:

```
mysqli_next_result($link);
```

`$link` is again the MySQLi database link. This function will advance the current resultset to the next available, which can then be retrieved using the `mysqli_store_result()` or `mysqli_use_result()` function.

To demonstrate the execution of multiple queries, consider [Listing 24.6](#):

Listing 24.6. Multiple Queries Using MySQLi

<?php

```
$mysqli = new mysqli("localhost", "username", "password",  
                    "mydatabase", 3306);  
$queries = "SELECT * FROM mytable; SELECT * FROM  
anothertable";
```

```
if(mysql_multi_query($mysqli, $queries)) {  
    do {  
        if($result = mysqli_store_result($mysqli)) {  
            while($row = mysqli_fetch_row($result)) {  
                foreach($row as $key => $value) {
```



```
        echo "$key => $value<BR/>\n";
    }
}
mysqli_free_result($result);

}

if(mysqli_more_results($mysqli)) {
    echo "<BR/>\nNext result set<BR/>\n";
}

} while(mysqli_next_result($mysqli));

}
mysqli_close($mysqli);

?>
```

PHP / Basics of File Handlin

Q. What is file handling?

File Handling is the storing of data in a file using a program. In C programming language, the programs store results, and other data of the program to a file using file handling in C. Also, we can extract/fetch data from a file to work with it in the program.

File handling is needed for any application. For some tasks to be done file needs to be processed.

PHP File System allows us to create file, read file line by line, read file character by character, write file, append file, delete file and close file.

File handling in PHP is similar as file handling is done by using any programming language like C. PHP has many functions to work with normal files. Those functions are:

1) fopen() – PHP fopen() function is used to open a file. First parameter of fopen() contains name of the file which is to be opened and second parameter tells about mode in which file needs to be opened, e.g.,

```
<?php
```

```
$file = fopen("demo.txt", 'w');
```

```
?>
```

Files can be opened in any of the following modes :

“w” – Opens a file for write only. If file not exist then new file is created and if file already exists then contents of file is erased.

“r” – File is opened for read only.

“a” – File is opened for write only. File pointer points to end of file. Existing data in file is preserved.

“w+” – Opens file for read and write. If file not exist then new file is created and if file already exists then contents of file is erased.

“r+” – File is opened for read/write.

“a+” – File is opened for write/read. File pointer points to end of file. Existing data in file is preserved. If file is not there then new file is created.

“x” – New file is created for write only.

2) fread() — After file is opened using fopen() the contents of data are read using fread(). It takes two arguments. One is file pointer and another is file size in bytes, e.g.,

```
<?php
```

```
$filename = "demo.txt";
```

```
$file = fopen( $filename, 'r' );
```

```
$size = filesize( $filename );  
$filedata = fread( $file, $size );  
?>
```

3) fwrite() – New file can be created or text can be appended to an existing file using fwrite() function. Arguments for fwrite() function are file pointer and text that is to be written to file. It can contain optional third argument where length of text to be written is specified, e.g.,

```
<?php  
$file = fopen("demo.txt", 'w');  
$text = "Hello world\n";  
fwrite($file, $text);  
?>
```

4) fclose() – file is closed using fclose() function. Its argument is file which needs to be closed, e.g.,

```
<?php  
$file = fopen("demo.txt", 'r');  
//some code to be executed  
fclose($file);  
?>
```

Introduction to MYSQL:

What is a Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as **Foreign Keys**.

A **Relational DataBase Management System (RDBMS)** is a software that –

- Enables you to implement a database with tables, columns and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

RDBMS Terminology

Before we proceed to explain the MySQL database system, let us revise a few definitions related to the database.

- **Database** – A database is a collection of tables, with related data.
- **Table** – A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- **Column** – One column (data element) contains data of one and the same kind, for example the column postcode.
- **Row** – A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
- **Redundancy** – Storing data twice, redundantly to make the system faster.
- **Primary Key** – A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
- **Foreign Key** – A foreign key is the linking pin between two tables.

- **Compound Key** – A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index** – An index in a database resembles an index at the back of a book.
- **Referential Integrity** – Referential Integrity makes sure that a foreign key value always points to an existing row.

MySQL Database

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons –

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

CRUD Operations in SQL

As we know, CRUD operations act as the foundation of any computer programming language or technology. one must be proficient in working on its CRUD operations. This same rule applies to databases as well.



1. Create:

In CRUD operations, 'C' is an acronym for create, which means to add or insert data into the SQL table. So, firstly we will create a table using CREATE command and then we will use the INSERT INTO command to insert rows in the created table.

Syntax for table creation:

CREATE TABLE Table_Name (ColumnName1 Datatype, ColumnName2 Datatype,..., ColumnNameN Datatype);

where,

Table_Name is the name that we want to assign to the table.

Column_Name is the attributes under which we want to store data of the table.

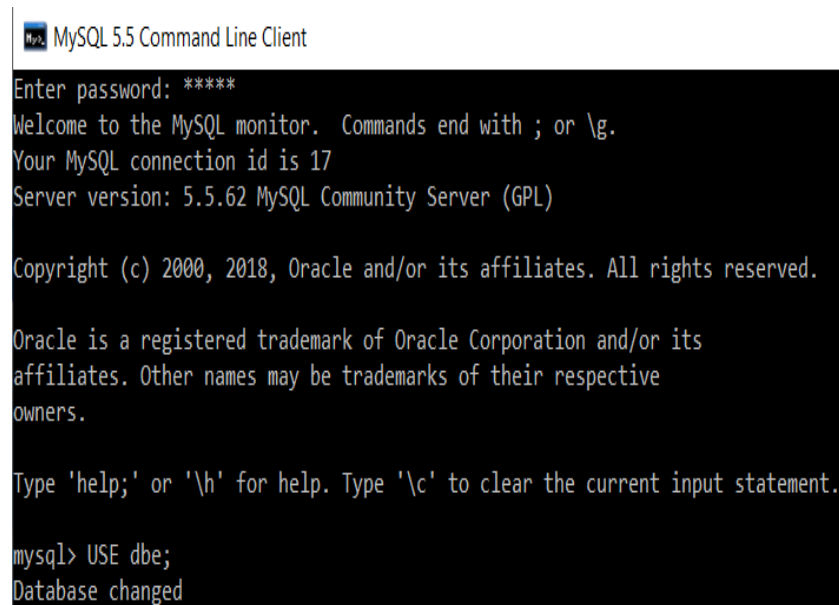
Datatype is assigned to each column. Datatype decides the type of data that will be stored in the respective column.

Syntax for insertion of data in table:

INSERT INTO Table_Name (ColumnName1,..., ColumnNameN) VALUES (Value 1,...,Value N),,,,,, (Value 1,...,Value N);

Prior to the creation of a table in SQL, we need to create a database or select an existing database. Since we already had a database, we will select the database with the USE command.

```
mysql> USE dbe;
```



```
MySQL 5.5 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.5.62 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE dbe;
Database changed
```

Now, we will write a query to create a table named employee in the database named dbe.

```
mysql> CREATE TABLE employee(ID INT PRIMARY KEY, First_Name
VARCHAR(20), Last_Name VARCHAR(20), Salary INT, Email_Id VARC
HAR(40));
```

To ensure that the table is created as per the column names, data types and sizes which we have assigned during table creation, we will execute the following query:

```
mysql> DESC employee;
```

You will get the following output:

| Field | Type | Null | Key | Default | Extra |
|-------|------|------|-----|---------|-------|
|-------|------|------|-----|---------|-------|

| | | | | | |
|------------|-------------|-----|-----|------|--|
| ID | int(11) | NO | PRI | NULL | |
| First_Name | varchar(20) | YES | | NULL | |
| Last_Name | varchar(20) | YES | | NULL | |
| Salary | int(11) | YES | | NULL | |
| Email_Id | varchar(40) | YES | | NULL | |

The above results verify that the table is created successfully as per the requirements.

We will execute the following query to insert multiple records in the employee table:

```
INSERT INTO employee(ID, First_Name, Last_Name, Salary, Email_Id)
VALUES(1, "Neeta", "Korade", 59000, "neetak12@gmail.com"), (2, "Sus
hma", "Singh", 62000, "sushsingh67@gmail.com"), (3, "Kavita", "Rathod"
, 27000, "kavitar09@gmail.com"), (4, "Mrunalini", "Deshmukh", 88000, "
mrunald78@gmail.com"), (5, "Swati", "Patel", 34000, "swatip67@gmail.c
om"), (6, "Laxmi", "Kadam", 44000, "laxmik14@gmail.com"), (7, "Lalita",
"Shah", 66000, "lalita45@gmail.com"), (8, "Savita", "Kulkarni", 31000, "s
avitak56@gmail.com"), (9, "Shravani", "Jaiswal", 38000, "shravanij39@g
mail.com"), (10, "Shweta", "Wagh", 20000, "shwetaw03@gmail.com");
```

2. Read:

In CRUD operations, 'R' is an acronym for read, which means retrieving or fetching the data from the SQL table. So, we will use the SELECT command to fetch the inserted records from the SQL table. We can retrieve all the records from a table using an asterisk (*) in a SELECT query. There is also an option of retrieving only those records which

satisfy a particular condition by using the WHERE clause in a SELECT query.

Syntax to fetch all the records:

SELECT *FROM TableName;

Syntax to fetch records according to the condition:

SELECT *FROM TableName WHERE CONDITION;

Example 1:

Write a query to fetch all the records stored in the employee table.

Query:

mysql> SELECT *FROM employee;

Here, an asterisk is used in a SELECT query. This means all the column values for every record will be retrieved.

You will get the following output after executing the above query:

| ID | First_Name | Last_Name | Salary | Email_Id |
|----|------------|-----------|--------|-----------------------|
| 1 | Neeta | Korade | 59000 | neetak12@gmail.com |
| 2 | Sushma | Singh | 62000 | sushsingh67@gmail.com |
| 3 | Kavita | Rathod | 27000 | kavitar09@gmail.com |
| 4 | Mrunalini | Deshmukh | 88000 | mrunald78@gmail.com |
| 5 | Swati | Patel | 34000 | swatip67@gmail.com |

| | | | | |
|----|----------|----------|-------|-----------------------|
| 6 | Laxmi | Kadam | 44000 | laxmik14@gmail.com |
| 7 | Lalita | Shah | 66000 | lalita45@gmail.com |
| 8 | Savita | Kulkarni | 31000 | savitak56@gmail.com |
| 9 | Shravani | Jaiswal | 38000 | shravanij39@gmail.com |
| 10 | Shweta | Wagh | 20000 | shwetaw03@gmail.com |

All the records are successfully retrieved from the employee table.

Example 2:

Write a query to fetch only those records from the employee table whose salary is above 35000.

Query:

mysql> SELECT *FROM employee WHERE Salary > 35000;

Here, an asterisk is used in a SELECT query. This means all the column values for every record will be retrieved. We have applied the WHERE clause on Salary, which means the records will be filtered based on salary.

You will get the output as follows:

| ID | First_Name | Last_Name | Salary | Email_Id |
|----|------------|-----------|--------|-----------------------|
| 1 | Neeta | Korade | 59000 | neetak12@gmail.com |
| 2 | Sushma | Rathod | 62000 | sushsingh67@gmail.com |
| 4 | Mrunalini | Deshmukh | 88000 | mrunald78@gmail.com |

| | | | | |
|---|----------|---------|-------|-----------------------|
| 6 | Laxmi | Kadam | 44000 | laxmik14@gmail.com |
| 7 | Lalita | Shah | 66000 | lalita45@gmail.com |
| 9 | Shravani | Jaiswal | 38000 | shravanij39@gmail.com |

There are six records in the employee table whose salary is above 35000.

3. Update:

In CRUD operations, 'U' is an acronym for the update, which means making updates to the records present in the SQL tables. So, we will use the UPDATE command to make changes in the data present in tables.

Syntax:

UPDATE Table_Name SET ColumnName = Value WHERE CONDITION;

Example 1:

Write a query to update an employee's last name as 'Bose', whose employee id is 6.

Query:

```
mysql> UPDATE employee SET Last_Name = "Bose" WHERE ID = 6;
```

Here in the SELECT query, we have used the SET keyword to update an employee's last name as 'Bose'. We want to update an employee's last name only for the employee with id 6, so we have specified this condition using the WHERE clause.

To ensure that an employee's last name with employee id 6 is updated successfully, we will execute the SELECT query.

```
mysql> SELECT *FROM employee;
```

| ID | First_Name | Last_Name | Salary | Email_Id |
|----|------------|-----------|--------|-----------------------|
| 1 | Neeta | Korade | 59000 | neetak12@gmail.com |
| 2 | Sushma | Singh | 62000 | sushsingh67@gmail.com |
| 3 | Kavita | Rathod | 27000 | kavitar09@gmail.com |
| 4 | Mrunalini | Deshmukh | 88000 | mrunald78@gmail.com |
| 5 | Swati | Patel | 34000 | swatip67@gmail.com |
| 6 | Laxmi | Bose | 44000 | laxmik14@gmail.com |
| 7 | Lalita | Shah | 66000 | lalita45@gmail.com |
| 8 | Savita | Kulkarni | 31000 | savitak56@gmail.com |
| 9 | Shravani | Jaiswal | 38000 | shravanij39@gmail.com |
| 10 | Shweta | Wagh | 20000 | shwetaw03@gmail.com |

The results above verify that an employee's last name with employee id 6 is now changed to 'Bose'.

Example 2:

Write a query to update the salary and email id of an employee as '35000' and 'shwetawagh03@gmail.com', respectively, whose employee id is 10.

Query:

```
mysql> UPDATE employee SET Salary = "35000", Email_Id= "shwetawagh03@gmail.com" WHERE ID = 10;
```

Here in the UPDATE query, we have used the SET keyword to update an employee's salary as '35000' and the email id as 'shwetawagh03@gmail.com'. We want to update the salary and email id of an employee only for the employee with id 10, so we have specified this condition using the WHERE clause.

To ensure that the salary and email id of an employee with employee id 10 is updated successfully, we will execute the SELECT query.

```
mysql> SELECT *FROM employee;
```

| ID | First_Name | Last_Name | Salary | Email_Id |
|----|------------|-----------|--------|-----------------------|
| 1 | Neeta | Korade | 59000 | neetak12@gmail.com |
| 2 | Sushma | Singh | 62000 | sushsingh67@gmail.com |
| 3 | Kavita | Rathod | 27000 | kavitar09@gmail.com |
| 4 | Mrunalini | Deshmukh | 88000 | mrunald78@gmail.com |
| 5 | Swati | Patel | 34000 | swatip67@gmail.com |
| 6 | Laxmi | Bose | 44000 | laxmik14@gmail.com |
| 7 | Lalita | Shah | 66000 | lalita45@gmail.com |
| 8 | Savita | Kulkarni | 31000 | savitak56@gmail.com |

| | | | | |
|----|----------|---------|-------|-----------------------|
| 9 | Shravani | Jaiswal | 38000 | shravanij39@gmail.com |
| 10 | Shweta | Wagh | 35000 | shwetaw03@gmail.com |

The results above verify that the salary and email id of an employee with employee id 10 is now changed to '35000' and 'shwetawagh03@gmail.com', respectively.

4. Delete:

In CRUD operations, 'D' is an acronym for delete, which means removing or deleting the records from the SQL tables. We can delete all the rows from the SQL tables using the DELETE query. There is also an option to remove only the specific records that satisfy a particular condition by using the WHERE clause in a DELETE query.

Syntax to delete all the records:

DELETE FROM TableName;

Syntax to delete records according to the condition:

DELETE FROM TableName WHERE CONDITION;

Example 1:

Write a query to delete the employee record from the employee table whose salary is above 34000.

Query:

mysql> DELETE FROM employee WHERE Salary = 34000;

Here we have applied the DELETE query on the employee table. We want to delete only the employee record whose salary is 34000, so we have specified this condition using the WHERE clause.

We will execute the SELECT query to ensure that the employee record with salary as 34000 is deleted successfully.

| ID | First_Name | Last_Name | Salary | Email_Id |
|----|------------|-----------|--------|-----------------------|
| 1 | Neeta | Korade | 59000 | neetak12@gmail.com |
| 2 | Sushma | Singh | 62000 | sushsingh67@gmail.com |
| 3 | Kavita | Rathod | 27000 | kavitar09@gmail.com |
| 4 | Mrunalini | Deshmukh | 88000 | mrunald78@gmail.com |
| 6 | Laxmi | Bose | 44000 | laxmik14@gmail.com |
| 7 | Lalita | Shah | 66000 | lalita45@gmail.com |
| 8 | Savita | Kulkarni | 31000 | savitak56@gmail.com |
| 9 | Shravani | Jaiswal | 38000 | shravanij39@gmail.com |
| 10 | Shweta | Wagh | 35000 | shwetaw03@gmail.com |

```
mysql> SELECT *FROM employee;
```

The results above verify that the employee with a salary of 34000 no longer exists in the employee table.

Example 2:

Write a query to delete all the records from the employee table.

First, let us see the employee table, which is available currently.

```
mysql> SELECT *FROM employee;
```

To remove all the records from the employee table, we will execute the DELETE query on the employee table.

```
mysql> DELETE FROM employee;
```

We will execute the SELECT query to ensure that all the records are deleted successfully from the employee table.

```
mysql> SELECT *FROM employee;
```

The results above verify that the employee table does not contain any record now.