ADVANCED WEB DEVELOPMENT

PHP And MySQL Database Continue

Safar M. Asaad

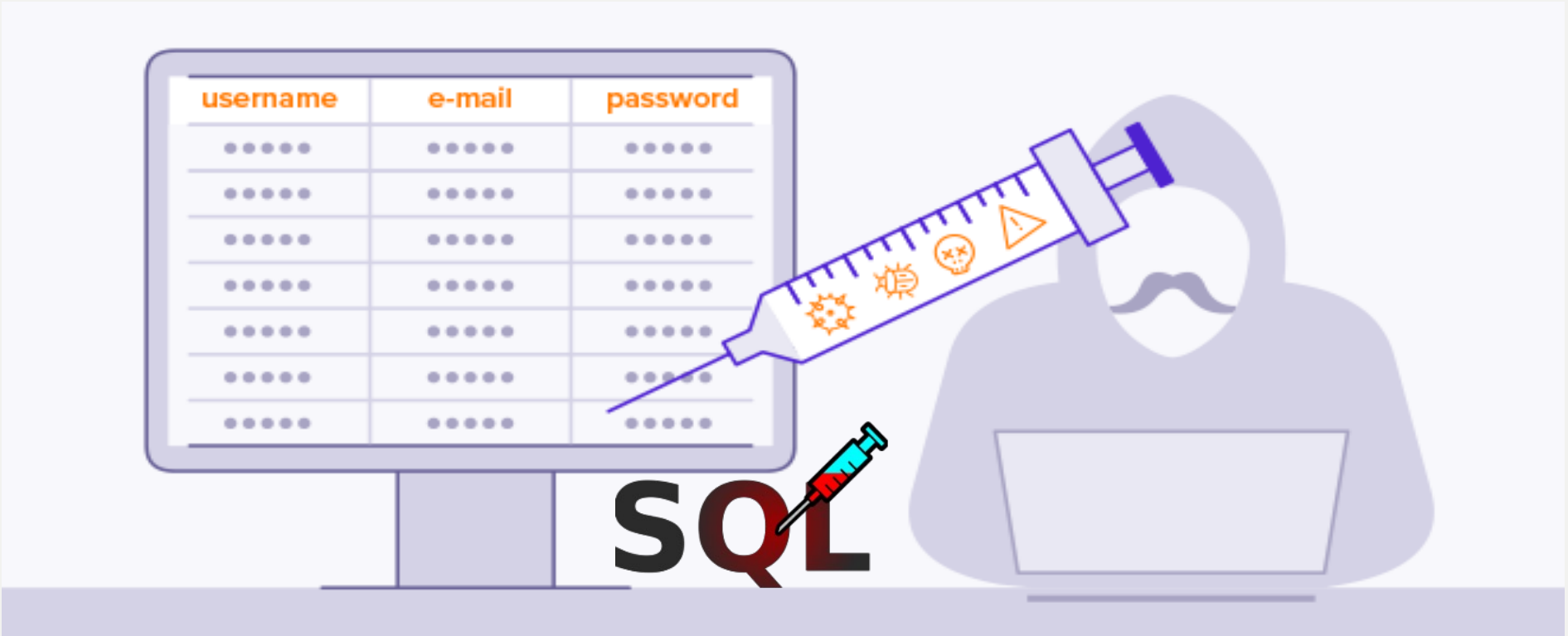
Koya University
Faculty of Engineering
Software Engineering Department

Outline

- Explanation of SQL Injection?
- The Role of prepared statement.
- Retrieving data from Database Tables.
- Deleting Table records.
- Update Data in MySQL Database.

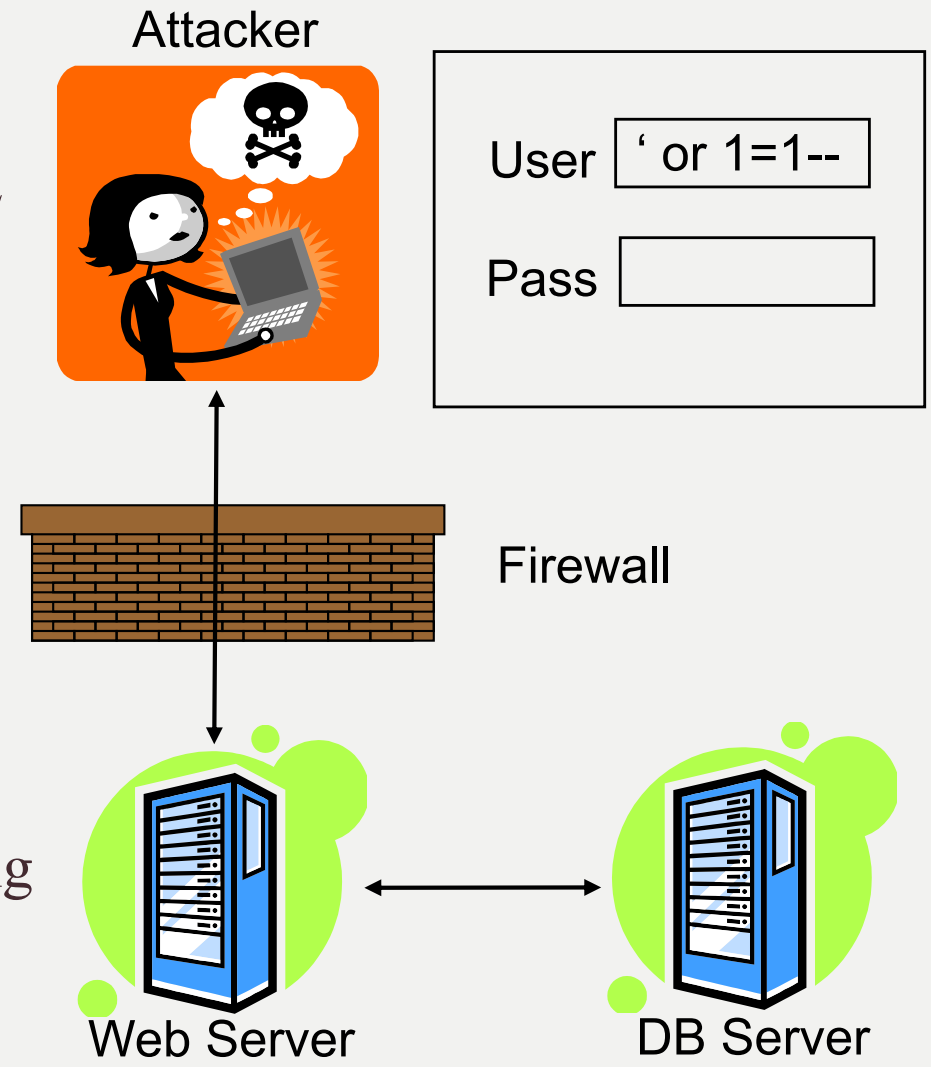
SQL Injection

- SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input.
- Injected SQL commands can alter SQL statement and compromise the security of a web application.
- An SQL Injection can destroy your database.

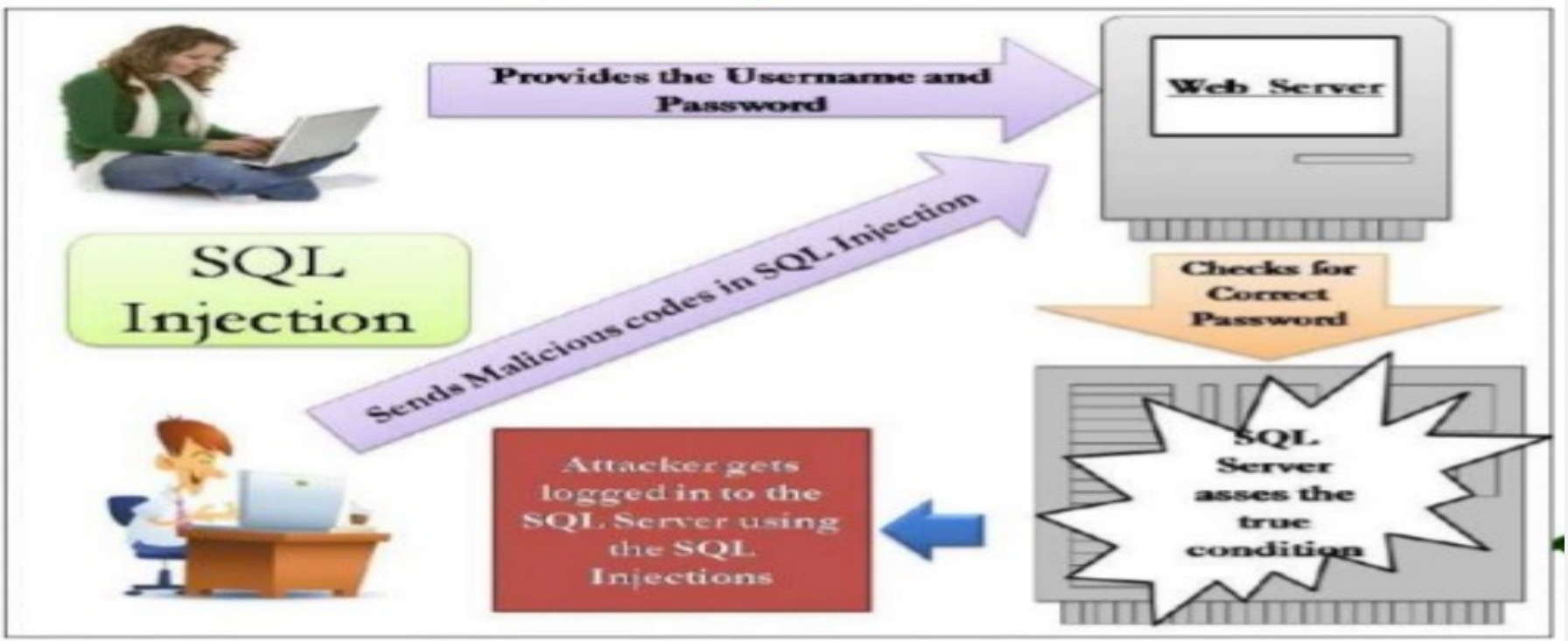


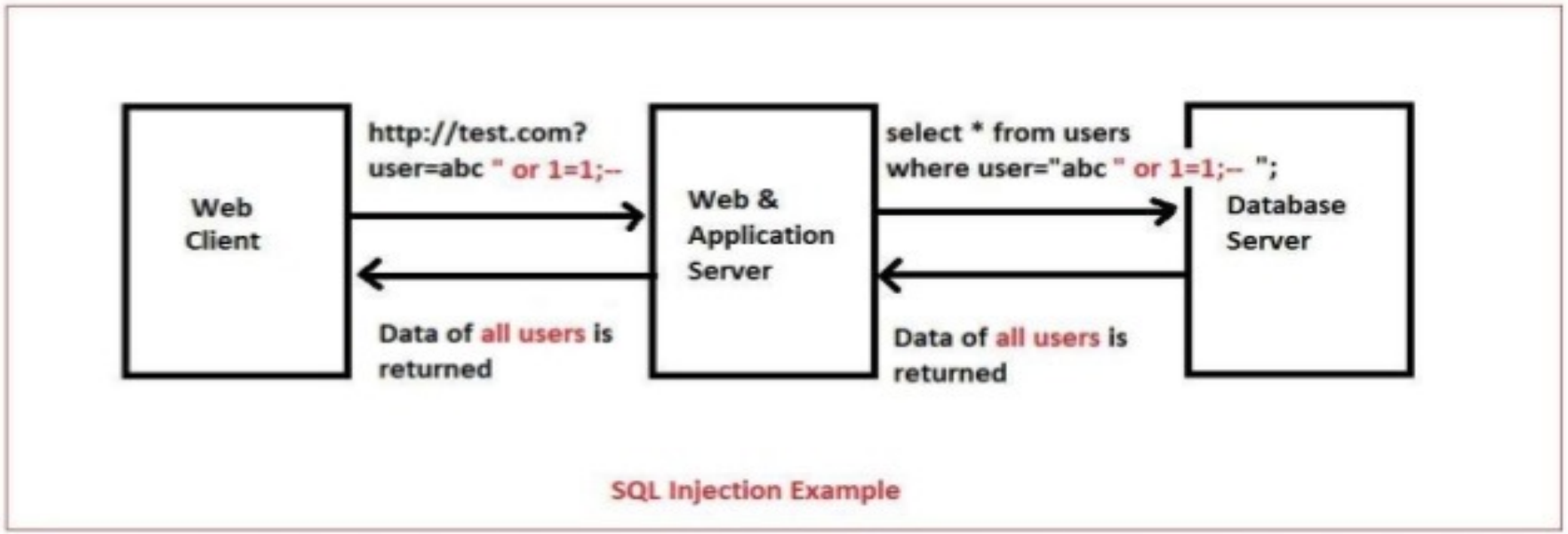
How SQL Injection Works

1. App sends form to user.
2. Attacker submit form with SQL exploit data
3. Application Build string with exploit data.
4. Application send SQL query to DBMS
5. DBMS executes query, including exploit data, sends data back to the application
6. Application returns data to user



How SQL Injection Works





Preventing SQL Injection Attacks

1. PHP mysqli_real_escape_string() Function or real_escape_string() function (OOP style)

2. PHP Prepared Statements

PHP real_escape_string() function

- It is used to escape all special characters for use in an SQL query.
- It is used before inserting a string in a database,
  - *it removes any special characters that may interfere with the query operations.*
  - *special characters like backslashes and apostrophes which are included (especially when they are getting data directly from a form where such data is entered).*
    - These are considered to be part of the query string and interfere with its normal functioning.

PHP real_escape_string() function

```php
<?php
// Escape special characters, if any
$firstname = $conn -> real_escape_string($_POST['firstname']);
$lastname = $conn -> real_escape_string($_POST['lastname']);
$age = $conn -> real_escape_string($_POST['age']);

$sql="INSERT INTO MyGuests (FirstName, LastName, Age) VALUES ('$firstname', '$lastname',
'$age')";

if (!$conn -> query($sql)) {
  echo("Row inserted.", $conn->affected_rows);
}
?>
```

PHP Prepared Statements

- It is a feature used to execute the same SQL statements repeatedly with high efficiency.
- Prepared statements basically work like this:
  1. *Prepare: The statement template is created by the application and sent to the DBMS.*
     - Certain values are left unspecified, called *parameters*, *placeholders* or *bind variables* (labelled "?").
     - Example: INSERT INTO MyGuests VALUES (?, ?, ?)
  2. *The DBMS parses, compiles, and performs query optimization on the statement template, and stores the result without executing it*
  3. *Execute: At a later time, the application binds the values to the parameters, and the database executes the statement.*
     - The application may execute the statement as many times as it wants with different values.

PHP Prepared Statements

- Compared to executing SQL statements directly, prepared statements have the following main advantages:
  - *Prepared statements reduces parsing time as the preparation on the query is done only once,*
    - although the statement is executed multiple times)
  - *Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query*
  - *Prepared statements are very useful against SQL injection.*

Prepared Statements Example

```
<?php                    $servername = "localhost";
                         $username = "root";
                         $password = "root";
                         $dbname = "myDB";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind

$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)

VALUES (?, ?, ?)");

$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
```

Continue >>

Prepared Statements Example

```
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully”;
$stmt->close();
$conn->close();   ?>
```

Select Data From MySQL

- The **SELECT statement** is used to select data from one or more tables:
- SELECT *column_name*, *column_name* FROM *table_name;*
- use the * character to select ALL columns from a table:
  - *SELECT * FROM table_name*

Select Data From MySQL

```
<?php
```

You need write the required code to establish the DB Connection just like the previous examples.

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
     echo (count($row));
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
    }
} else {
    echo "0 results";
}
$conn->close();                    ?>
```

SQL WHERE Clause

- The WHERE clause is used to filter records.
- **Syntax:**

```
SELECT column_name, column_name
FROM table_name
WHERE column_name operator value;
```

Operators in The WHERE Clause

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
IN	To specify multiple possible values for a column
LIKE	Search for a pattern

Example

//$sql = "SELECT id, firstname, lastname FROM MyGuests where id = 5";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where firstname ='".$x."'";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where firstname = 'San'";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where id >= 5";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where id between 5 and 10";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where firstname between 'A' and 'S'";

//$sql = "SELECT id, firstname, lastname FROM MyGuests where id in (3, 5, 6, 8, 9, 1, 100, 300, 0)";

Example Cont.

- **Examples:**

```
$result = $conn->query($sql);


if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. "
" . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
?>
```

SQL LIKE Operator

LIKE Operator	Description
WHERE firstName LIKE 'a%'	Finds any values that start with "a"
WHERE firstName LIKE '%a'	Finds any values that end with "a"
WHERE firstName LIKE '%or%'	Finds any values that have "or" in any position
WHERE firstName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE firstName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE firstName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE firstName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL AND & OR Operators

- The AND & OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if both the first condition AND the second condition are true.
- The OR operator displays a record if either the first condition OR the second condition is true.
- **AND Operator Example**
- $sql = "SELECT id, firstname, lastname FROM MyGuests where firstname = ‘Sardar’ and lastname = ‘Qadir’ ";
- **OR Operator Example**
- $sql = "SELECT id, firstname, lastname FROM MyGuests where firstname = ‘Sardar’ OR lastname = ‘Ali’ ";

ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set.
- **ORDER BY Syntax**
- SELECT *column_name*, *column_name*
- FROM *table_name*
- ORDER BY *column_name* ASC|DESC, *column_name* ASC|DESC;
- **Example**
- $sql = "SELECT id, firstname, lastname FROM MyGuests where firstname = San or lastname = Ahmed' ORDER By lastname DESC";

PHP Limit Data Selections From MySQL

- MySQL provides a LIMIT clause that is used to specify the number of records to return.
- The LIMIT clause makes it easy to code multi page results and is very useful on large tables. Returning a large number of records can impact on performance.
- Assume we wish to select all records from 1 - 30 (inclusive) from a table called " **MyGuests** ". The SQL query would then look like this:
- **$sql = "SELECT * FROM MyGuests LIMIT 30";**
  - *When the SQL query above is run, it will return the first 30 records.*
- What if we want to select records 16 - 25 (inclusive)?
  - *Mysql also provides a way to handle this: by using OFFSET.*

PHP Limit Data Selections From MySQL

- *The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":*
- ***$sql = "SELECT * FROM MyGuests LIMIT 10 OFFSET 15";***

- You could also use a shorter syntax to achieve the same result:
- **$sql = "SELECT * FROM MyGuests LIMIT 15, 10";**
- **Notice that the numbers are reversed when you use a comma.**

Data retrieve with prepare Statement

```
$sql = "SELECT * FROM myGuests WHERE id = ?";
$stmt = $conn->prepare($sql);
$id = 1;
$stmt->bind_param("i", $id);
// execute the prepared statement
$stmt->execute();
// get the result set from the executed statement
$result = $stmt->get_result();
// iterate through the result set and display each row
while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . "<br>";
        echo "Name: " . $row[”firstName"] . "<br>";
        echo "Email: " . $row["email"] . "<br>";
}
$stmt->close();
$conn->close(); ?>
```

Delete Data From MySQL

- The **DELETE statement** is used to delete records from a table.
  - *DELETE FROM table_name WHERE some_column = some_value*

```
<?php
```

You need write the required code to establish the DB Connection just like the previous examples.

```
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=44";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

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

Delete Data Using Prepared Stmt.

```
<?php
```
You need write the required code to establish the DB Connection just like the previous examples.
```
// sql to delete a record
$deletid= 42;
if($stmt= $conn->prepare("DELETE FROM MyGuests WHERE id=?"))
{
        $stmt->bind_param("i", $deletid);
        $result=$stmt -> execute();
 $rowsAffected = $stmt->affected_rows;
 if($rowsAffected > 0){
        echo $rowsAffected."Rows Affected (deleted) ";
 }else{
        echo 'Sorry, Please try after some time';
 }
$stmt -> close();
}
$conn->close(); ?>
```

Update Data in MySQL

- **UPDATE statement** is used to update existing records in a table:
  - *UPDATE table_name SET column1=value, column2=value2,...*

    WHERE some_column=some_value

```
<?php
```

You need write the required code to establish the DB Connection just like the previous examples.

```
$sql = "UPDATE MyGuests SET lastname='Ahmad' WHERE id=45";

if ($conn->query($sql) === TRUE) {
   echo "Record updated successfully";
} else {
   echo "Error updating record: " . $conn->error;
}

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

Update Data using Prepared Stmt.

```
<?php
```
You need write the required code to establish the DB Connection just like the previous examples.

```
$id=47;
$lname=”Hasan";
if($stmt = $conn->prepare("UPDATE MyGuests SET lastname =? WHERE id =?"))
{
$stmt->bind_param('si', $lname, $id);
$result=$stmt->execute();
$rowsAffected = $stmt->affected_rows;
 if($rowsAffected > 0){
     echo "Rows Affected: ".$rowsAffected;
 }else{
     echo 'Sorry, Please try after some time';
 }
$stmt->close();
}
$conn->close();
?>
```