PHP Database

PHP Database Connection Types

- MySQLi (Object-Oriented) (MySQL database only)

  Ex: $mysqli = new mysqli("localhost", "root", "", "example_db");

- MySQLi (Procedural) (MySQL database only)

  Ex: $conn = mysqli_connect("localhost", "root", "", "example_db");

- PDO (PHP Data Objects) (Multiple database support)
  - PDO is a more flexible approach that supports multiple database types (MySQL, PostgreSQL, SQLite, etc.).

  Ex: $pdo = new PDO("mysql:host=localhost;dbname=example_db", "root", "");

Database Types Comparison

Comparison Table

Feature	MySQLi (OO & Procedural)	PDO	Legacy MySQL
Supports MySQL?	✅ Yes	✅ Yes	✅ Yes
Supports Multiple Databases?	❌ No (Only MySQL)	✅ Yes (MySQL, PostgreSQL, SQLite, etc.)	❌ No
Supports Prepared Statements?	✅ Yes	✅ Yes	❌ No
Object-Oriented Support?	✅ Yes	✅ Yes	❌ No
Future-Proof?	✅ Yes	✅ Yes	❌ No (Deprecated)

PHP Connect to the MySQL Server

Use the PHP mysqli () function to open a new connection to the MySQL server.

Syntax: new mysqli (host, username, password, dbname);

Parameter	Description
host	Optional. Either a host name or an IP address
username	Optional. The MySQL user name
password	Optional. The password to log in with
dbname	Optional. The default database to be used when performing queries

Note: There are more available parameters, but the ones listed above are the most important. In the following example we store the connection in a variable ($con) for later use in the script:

PHP Connect to the MySQL Server ...

```
<?php
   $servername = "localhost";
   $username = "root";
   $password = "";
   $dbname = "example_db";

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

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

Close a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysqli_close() function:

```
<?php
    $servername = " localhost ";
    $username = "root";
    $password = "";
    $dbname = "example_db";

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

    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    echo "Connected successfully";
    // Close connection (optional)
    $conn->close();
?>
```

PHP Create Database

- A database holds one or more tables.
- The CREATE DATABASE statement is used to create a database in MySQL.
- We must add the CREATE DATABASE statement to the query() function to execute the command.
- The following example creates a database named " example_db“:

PHP Create Database...

```
<?php
    // Connection details declared directly to fit the code into the slide.
    $mysqli = new mysqli("localhost", "root", "");

    // Check connection
    if ($mysqli->connect_error) {
        die("Failed to connect to MySQL: " . $mysqli->connect_error);
    }

    // Create database
    $sql = "CREATE DATABASE example_db";
    if ($mysqli->query($sql) === TRUE) {
        echo "Database 'my_db' created successfully";
    } else {
        echo "Error creating database: " . $mysqli->error;
    }

    // Close connection
    $mysqli->close();
?>
```

Create a Table

- The CREATE TABLE statement is used to create a table in MySQL.
- We must add the CREATE TABLE statement to the mysqli_query() function to execute the command.
- The following example creates a table named "Persons", with three columns: "FirstName", "LastName" and "Age":

Create a Table

```
<?php
// Create a connection
$mysqli = new mysqli("localhost", "root", "", "example_db");

// Check connection
if ($mysqli->connect_error) {
    die("Failed to connect to MySQL: " . $mysqli->connect_error);
}

// Create table
$sql = "CREATE TABLE persons (
    first_name CHAR(30),
    last_name CHAR(30),
    age INT
)";

// Execute query
if ($mysqli->query($sql) === TRUE) {
    echo "Table 'Persons' created successfully";
} else {
    echo "Error creating table: " . $mysqli->error;
}
?>
```

Primary Keys and Auto Increment Fields ...

- Each table in a database should have a primary key field.
- A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.
- AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added.
- Each primary key value must be unique within the table.
- To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field:

Primary Keys, Auto Increment & Alter Table

```
<?php
$mysqli = new mysqli("localhost", "root", "", "example_db");
if ($mysqli->connect_error) die("Connection failed: " . $mysqli->connect_error);

$sql = "ALTER TABLE persons ADD pid INT NOT NULL AUTO_INCREMENT PRIMARY KEY";
echo $mysqli->query($sql) ? "Table 'persons' altered successfully" : "Error: " . $mysqli->error;
?>
```

Insert Data Into a Database Table

- The INSERT INTO statement is used to add new records to a database table. Syntax
- It is possible to write the INSERT INTO statement in two forms.
- The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name VALUES (value1, value2, value3,…)
```

- The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,…)
VALUES (value1, value2, value3,…)
```

- To get PHP to execute the statements above we must use the mysqli_query() function. This function is used to send a query or command to a MySQL connection.

Insert Data Into a Database Table ...

- The following example adds two new records to the "Persons" table:

```
<?php
$mysqli = new mysqli("localhost", "root", "", "example_db");
if ($mysqli->connect_error) die("Connection failed: " . $mysqli->connect_error);

$sql = "INSERT INTO persons (first_name, last_name, age) VALUES
   ('Peter', 'Griffin', 35),
   ('Glenn', 'Quagmire', 33),
   ('Lois', 'Albert', 34),
   ('Stewie', 'Nonis', 2),
   ('Brian', 'Andrew', 7),
   ('Meg', 'Griffin', 16)";

echo $mysqli->query($sql) ? "Records inserted" : "Error: " . $mysqli->error;
?>
```

Insert Data From a Form Into a Database

```
<body>
   <h2>Insert User Data</h2>
   <form action="insert_form_action.php" method="post">
      <label for="first_name">First Name:</label>
      <input type="text" id="first_name" name="first_name" maxlength="30"
required><br><br>

      <label for="last_name">Last Name:</label>
      <input type="text" id="last_name" name="last_name" maxlength="30"
required><br><br>

      <label for="age">Age:</label>
      <input type="number" id="age" name="age" required><br><br>

      <input type="submit" value="Submit">
   </form>
</body>
```

Insert Data From a Form Into a Database

- When a user clicks the submit button in the HTML form, in the example above, the form data is sent to "insert_form_action.php".

- The " insert_form_action.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables.

```
$conn = new mysqli($servername, $username, $password, $dbname);

// Check if form data is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Collect form data
    $first_name = $_POST['first_name'];
    $last_name = $_POST['last_name'];
    $age = $_POST['age'];

    // Prepare and bind SQL statement
    $stmt = $conn->prepare("INSERT INTO your_table_name (first_name,
last_name, age) VALUES (?, ?, ?)");
    $stmt->bind_param("ssi", $first_name, $last_name, $age);

    // Execute the statement
    if ($stmt->execute()) {
        echo "New record inserted successfully!";
    } else {
        echo "Error: " . $stmt->error;
    }

    // Close the statement
    $stmt->close();
}
```

What are prepared statements and SQL injection ?

- **Prepared Statements**: A prepared statement is a feature used in database management systems to execute SQL queries in a more secure and efficient manner. When you use a prepared statement, the SQL query is first prepared and compiled by the database, and then parameters (e.g., user input) are bound to the query before execution. This separates the query structure from the data, making it less vulnerable to SQL injection attacks.

- **SQL Injection**: SQL injection is a type of attack that allows attackers to execute arbitrary SQL code on a database by manipulating user input. This can lead to unauthorized access, data corruption, or even deletion of data. SQL injection typically occurs when user input is not properly sanitized and is inserted directly into SQL queries.

- Prepared statements help protect against SQL injection because the SQL query and user input are handled separately, reducing the risk of malicious input being treated as part of the query itself.

prepared statement example

```
$stmt = $conn->prepare("INSERT INTO your_table_name (first_name,
last_name, age) VALUES (?, ?, ?)");

$stmt->bind_param("ssi", $first_name, $last_name, $age);
```

Select Data From a Database Table

- The SELECT statement is used to select data from a database.

```
SELECT column_name(s)
FROM table_name
```

Select Data From a Database Table …

```
<?php
// Connection details declared directly to fit into the slide.
$conn = new mysqli("localhost", "root", "", "example_db");

// Check connection
if ($conn->connect_error) { die("Failed to connect to MySQL: " . $conn->connect_error); }

$sql = "SELECT * FROM persons";
$result = $conn->query($sql);
// Check if the query was successful
if (!$result) die("Error executing query: " . $conn->error);

// Fetch and display data
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo $row['first_name'] . " " . $row['last_name']. " " . $row['age'];
        echo "<br>";
    }
} else { echo "No records found in the 'persons' table."; }
?>
```

Display the Result in an HTML Table

```
// Query the database
$result = $con->query("SELECT * FROM persons");

// Check if the query was successful
if ($result->num_rows > 0) {
    // Start the table
    echo "<table border='1'>
        <tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Age</th>
        </tr>";
    while ($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row['first_name'] . "</td>";
        echo "<td>" . $row['last_name'] . "</td>";
        echo "<td>" . $row['age'] . "</td>";
        echo "</tr>";
    }
    // Close the table
    echo "</table>";
} else {
    echo "No records found";
}
```

The WHERE clause

- The WHERE clause is used to extract only those records that fulfill a specified criterion.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

The WHERE clause ...

- The following example selects all rows from the “persons" table where “first_name='Peter'“.
- Next example is showing the age is greater than 30.
- Also can combine the criteria also.

- $result = $con->query("SELECT * FROM persons WHERE first_name = 'Peter'");
- $result = $con->query("SELECT * FROM persons WHERE age > 30");
- $result = $con->query("SELECT * FROM persons
  WHERE last_name = 'Griffin' AND age > 30");

The ORDER BY Keyword

- The ORDER BY keyword is used to sort the data in a record set.
- The ORDER BY keyword sort the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

Update Data In a Database

- The UPDATE statement is used to update existing records in a table.

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

- **Note:** Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Update Data In a Database ...

The following example updates some data in the “persons" table:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

```
// Update query
$query = "UPDATE persons SET age = 36  WHERE first_name = 'Peter'
AND last_name = 'Griffin'";

// Execute the query
if ($con->query($query) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $con->error;        }
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Update Data In a Database ...

After the update, the “persons" table will look like this:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

```
DELETE FROM table_name
WHERE some_column = some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Delete Data In a Database …

Look at the following “persons" table:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

```
// Delete query
$query = "DELETE FROM persons WHERE last_name = 'Griffin'";

// Execute the query
if ($con->query($query) === TRUE) {
    echo "Record(s) deleted successfully";
} else {
    echo "Error deleting record: " . $con->error;
}
```

Delete Data In a Database …

After the deletion, the table will look like this:

FirstName	LastName	Age
Glenn	Quagmire	33

SQL Joins

SQL joins are used to combine rows from two or more tables.

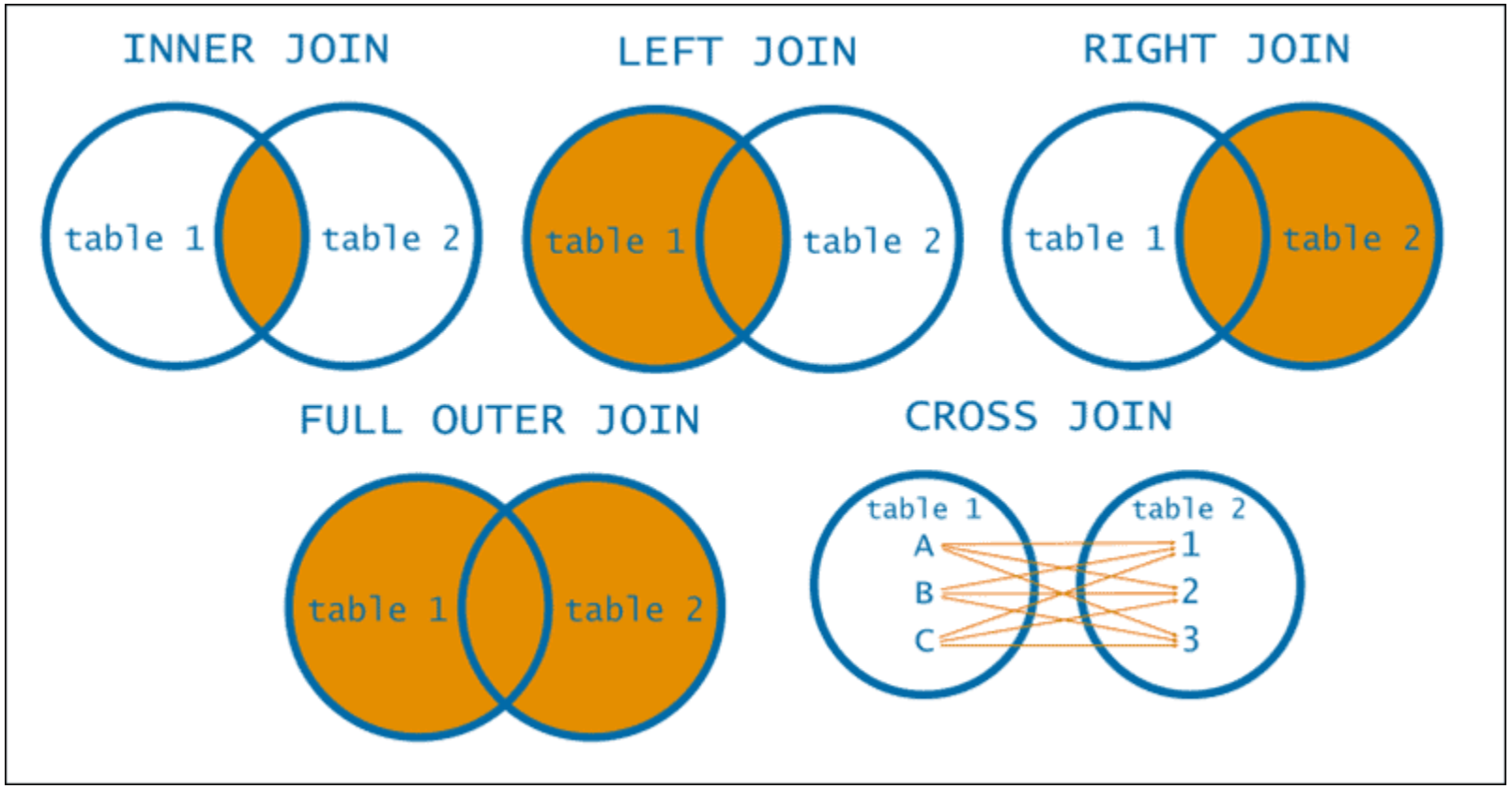
SQL JOIN

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

Let's look at a selection from the "Orders" table:

Join Types


INNER JOIN
table 1
table 2
LEFT JOIN
table 1
table 2
RIGHT JOIN
table 1
table 2
FULL OUTER JOIN
table 1
table 2
CROSS JOIN
table 1
A
B
C
table 2
1
2
3

SQL Joins...

"orders" table

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

"customers" table

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados yhelados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

SQL Joins ...

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

- INNER JOIN only returns matching records from both tables.
- If there is no match, the row is excluded from the result.
- Use INNER JOIN when you only want records that exist in both tables.