WEB 306 – PHP: Databases and Framework

Day 6 – Databases and SQL

1. Intro to Databases

1.1 What Is a Database?

> "A database is a collection of information that is organized so that it can be easily accessed, managed and updated.
>
> Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it."
>
> — From database (DB), TechTarget

1.2 Types of Databases

There are different types of databases.

1. **Relational**
2. **Non-Relational**
3. **Operational**

Database Resources

1. database (TechTarget)
2. relational database (TechTarget)
3. non-relational database (MongoDB)
4. operational database (odb) (Techopedia)
5. NoSQL Meets Bitcoin and Brings Down Two Exchanges: The Story of Flexcoin and Poloniex

2. Relational Databases

2.1 About Relational Databases

> "A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables."
>
> — From relational database, TechTarget

Basically, relational databases are databases which use tables with rows and columns to store data which can relate to data in other tables in complex ways.

2.2 Tables

If a Facebook user leaves a comment on a post then that comment is related to both the user's account and the post which the comment is on.

This relationship would be created by using a table for users, a table for posts and a table for comments. Each of these tables would have a column for an ID number, known as a "primary key" and the other tables would be able to reference this ID number as a "foreign key" to form a relationship.

id 🔑	user	post
1	5	6
2	7	5
3	4	8

2.2 The ACID Principle

Relational databases follow the ACID principle.

2.3.1 Atomicity

> "In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are."
>
> — From ACID (atomicity, consistency, isolation, and durability), TechTarget

2.3.2 Consistency

> "A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started."
>
> — From ACID (atomicity, consistency, isolation, and durability), TechTarget

2.3.3 Isolation

> "A transaction in process and not yet committed must remain isolated from any other transaction."
>
> — From ACID (atomicity, consistency, isolation, and durability), TechTarget

2.3.4 Durability

> "Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state."
>
> — From ACID (atomicity, consistency, isolation, and durability), TechTarget

3. Non-Relational Databases

3.1 About Non-Relational Databases

> "A non-relational database is any database that does not follow the relational model provided by traditional relational database management systems. This category of databases, also referred to as NoSQL databases, has seen steady adoption growth in recent years with the rise of Big Data applications."
>
> — From What Is A Non Relational Database, MongoDB

Non-relational databases can not contain data with complex relations but they have many other advantages over relational databases. The main advantage of non-relational databases and the simplest one, is that they can be faster than relational databases.

The structure of an average non-relational database follows the same basic structure as a PHP associative array but most non-relational databases store data in the form of JSON (JavaScript Object Notation) which is a way of storing data using JavaScript object literals that looks like this.

```
[
  {
        "name": "Jason",
        "age": 23
  },
  {
        "name": "Jaye Sohn",
        "age": 36
  },
  {
        "name": "Jaye's Son",
        "age": 16
  }
]
```

Another example of a non-relational database is a serialized PHP object.

```
O:3:"Car":7:{s:4:"name";s:8:"Some Car";s:6:"driver";s:8:"Some
Guy";s:10:" Car image";s:7:"car.gif";s:9:" Car
honk";s:8:"honk.mp3";s:9:"top_speed";s:3:"170";s:13:"current_
speed";i:0;s:3:"gas";i:100;}
```

However, technically any database which is not, relational (i.e. not in the form of a table) is a non-relational database.

3.2 Concurrent Processing

> "Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means something that happens at the same time as something else. Tasks are broken

> **down into subtasks that are then assigned to separate processors to perform simultaneously, instead of sequentially as they would have to be carried out by a single processor. Concurrent processing is sometimes said to be synonymous with parallel processing."**
>
> **— From concurrent processing, TechTarget**

Non-relational databases do not necessarily apply the ACID principle and allow for concurrent processing which allows them to retrieve data faster in many scenarios but due to the lack of isolation, can lead to actions being repeated for the same set of data. For this reason, you should be very careful about which database type you choose depending on the type of data you are handling.

4. Operational Databases

> **"An operational database is a database that is used to manage and store data in real time. An operational database is the source for a data warehouse. Elements in an operational database can be added and removed on the fly. These databases can be either SQL or NoSQL-based, where the latter is geared toward real-time operations."**
>
> **— From Operational Database (ODB), Technopedia**

5. Intro to SQL

5.1 About SQL

SQL stands for ***Structured Query Language***.

> **SQL (Structured Query Language) is a standardized programming language used for managing relational databases and performing various operations on the data in them.**
>
> **— From SQL (Structured Query Language), TechTarget**

SQL is for managing relational databases so it's designed to work with tables, rows and columns. Practically all relational databases use SQL.

5.2 Tables

Each table in an SQL database represents an object or subject which is being stored in the database. For instance, if you had a database of animals you could have an `elephants` table, a `dolphins` table and a `seals` table.

5.3 Rows/Records

Each "row" or "record" in an SQL database represents one entry in a table. For example, each row in our Dolphin table would represent a specific instance of a dolphin.

5.4 Columns/Fields

Each "column" or "field" in a database row represents an attribute of one instance of an entry. In a dolphin row, you could have a column for the dolphin ID, a column for the dolphin's name and a column for the dolphin's age.

id 🔑	**name**	**age**
1	**Flipper**	**3**
2	**Splashy**	**5**
3	**Dorsal**	**3**

6. MySQL/MariaDB

MySQL and MariaDB are Relational Database Management Systems (RDBMS). Relational Database Management Systems are a way of creating, updating and administrating a relational database. XAMPP used to come with MySQL but more recent versions use MariaDB instead.

There is very little difference between them as they both use SQL to manage databases and MariaDB is actually based on MySQL and is created by the creator of MySQL. However, MariaDB can export data as a non-relational database, in JSON format.

MySQL Resources

1. MySQL Website
2. MySQL Documentation
3. XAMPP
4. W3Schools
5. Stack Overflow

7. PHPMyAdmin

7.1 About PHPMyAdmin

PHPMyAdmin is a graphic user interface which can be used to operate MySQL, MariaDB and to simply write SQL statements. XAMPP also includes and installs PHPMyAdmin. To access PHPMyAdmin using XAMPP, turn on Apache and MySQL/MariaDB in your XAMPP control panel and enter the address ***localhost/phpmyadmin*** into your browser.

MariaDB Resources

1. MariaDB Website
2. MariaDB Documentation
3. XAMPP
4. Stack Overflow

7.2 Creating Databases and Tables

To create a new database using PHPMyAdmin click on "New" in the sidebar on the left, above the list of existing databases and then give your database a name and click "Create". Once you have created a database, create a table name, set a number of columns and click "Go".

7.3 Creating Columns/Fields

7.3.1 Column Names

Once you have created a table, you will be prompted to set the properties for your columns. The first property which you would set would be the Name of the column field – this should be text with no spaces or special characters.

PHPMyAdmin Resources

1. PHPMyAdmin Website
2. XAMPP
3. PHPMyAdmin Documentation
4. PHPMyAdmin Support
5. PHPMyAdmin Tutorials
6. Stack Overflow

7.3.2 SQL Data Types

The next property to set is the data type of this column field. There are many available data types but the primary ones you would be likely to use are:

1. **INT – This is for integers i.e. 1**
2. **FLOAT – This is for single-precision floating-point numbers i.e 1.2**
3. **DOUBLE – This is for double-precision floating-point numbers i.e 1.20**
4. **VARCHAR – This is for most short character strings**
5. **TINYTEXT – A character string with a maximum length of 255 characters**
6. **TEXT – A character string with a maximum length of 65535 characters**
7. **MEDIUMTEXT – A character string with a maximum length of 16777215 characters**
8. **LONGTEXT – A character string with a maximum length of 4294967295 characters**
9. **TINYBLOB – A binary large object column with a maximum length of 255 characters**
10. **BLOB – A binary large object column with a maximum length of 65535 characters**
11. **MEDIUMBLOB – A binary large object column with a maximum length of 16777215 characters**
12. **LONGBLOB – A binary large object column with a maximum length of 4294967295 characters**

Aside from **INT**, **FLOAT** and **DOUBLE** which are numeric types, the rest of the data types are all string types. The primary difference between **VARCHAR** and **TEXT** is that **VARCHAR** is stored inline in the table row while **TINYTEXT**, **TEXT**, **MEDIUMTEXT** and **LONGTEXT** are stored outside of the database itself and are referenced by the database. Because **VARCHAR** is stored inside the database itself it processes faster if the data is short but if the data has the potential to be long then a **TEXT** type would be faster.

BLOB, **TINYBLOB**, **MEDIUMBLOB** and **LONGBLOB** are binary large objects which can hold variable amounts of data. They are stored outside of the database itself like **TEXT**. **BLOB** types are treated as binary/byte strings. **TEXT** types are treated as non-binary/character strings. This means that the data in **TEXT** types is sorted and compared by the type of characters, while the data of **BLOB**

types is sorted and compared numerically by the bytes. **BLOB** types can be used to store binary image or video file data in a database. It is also recommended to use **BLOB** types for storing non-relational databases such as serialized PHP objects or JSON, although more recent versions of SQL include a dedicated JSON type. You can read more about **BLOB** versus **TEXT** types here.

7.3.3 Column Length

The next column field property to set is the Length/Values. This is the number of characters which are allowed for the data in this field. It's conventional to set **INT** types to 11 and **VARCHAR** types to 255 unless there is a specific amount of characters needed. **TEXT** and **BLOB** types do not need to have lengths set as they already have preset lengths.

7.3.4 Primary Key Column

Every row MUST have the first field set to an INT which will act as an ID, or "primary key". This first field must also have its Index set to "PRIMARY" and have the A_I (auto-increment) checkbox checked in order for your rows to each have a unique ID.

7.3.5 NULL Fields

For most of our fields, we can skip filling out the other properties but check the NULL checkbox if you want a given field to be optional. Once you have filled out all of the properties for your field, click "Save" and your database table will be ready to use.

8. SQL Code

8.1 SQL Files

You don't have to edit your databases using PHPMyAdmin. SQL is just like any other programming language. You can create an `.sql` file in your text editor and write your SQL code from scratch. You can also export an SQL file with the raw code for your database from PHPMyAdmin.

To do so, select your database in the left hand column and click "Export" in the menu at the top. You can then edit any of the code in the exported SQL file and import it back into PHPMyAdmin to update your database.

8.2 SQL Comments

Like most other programming languages, SQL files allow you to leave comments.

```
-- This is an inline SQL comment

/* Block SQL comments are
the same as PHP/CSS */
```

8.3 SQL Statements

8.3.1 SQL Statement Syntax

To make changes to our database using SQL, we would use SQL statements. SQL statements are normally written in ALL CAPS but are not case-sensitive.

There are multiple versions of SQL with slightly different syntaxes but they are all fairly similar. This statement will select all rows from the `dolphins` table.

```
SELECT * FROM `dolphins`;
```

The quote-like symbols which are wrapping the table name are not single quotes and that is intentional. They are ***backticks***. The backtick key is at the top left of your keyboard under the escape key. Database names, table names and column names MUST be wrapped in backticks or the SQL query will not work. Like most programming languages, most versions of SQL require a semicolon at the end of an SQL statement to end the line of code.

8.3.2 Common SQL Statements

These are the most common SQL statements but there are more:

1. **`SELECT` - extracts data from a database**
2. **`UPDATE` - updates data in a database**
3. **`DELETE` - deletes data from a database**
4. **`INSERT INTO` - inserts new data into a database**
5. **`CREATE DATABASE` - creates a new database**
6. **`ALTER DATABASE` - modifies a database**
7. **`CREATE TABLE` - creates a new table**
8. **`ALTER TABLE` - modifies a table**
9. **`DROP TABLE` - deletes a table**
10. **`CREATE INDEX` - creates an index (search key)**
11. **`DROP INDEX` - deletes an index**

SQL Resources

1. XAMPP
2. W3Schools
3. Scotch.io
4. Stack Overflow

8.3.3 SQL SELECT Statement

The **`SELECT`** statement selects data from a database table. Be careful to use ***backticks*** and not single quotes for the table name. Select everything in the `dolphins` table.

```
SELECT * FROM `dolphins`;
```

Select name and age from the `dolphins` table.

```
SELECT name, age FROM `dolphins`;
```

8.3.4 SQL WHERE Clause

The **`WHERE`** clause filters rows which meet specific conditions. It works like an if statement. Be careful to use ***backticks*** and not single quotes for the table name. Select all from `dolphins` table where age is 3.

```
SELECT * FROM `dolphins` WHERE age=3;
```

8.3.5 SQL Evaluators

These evaluators work with the `WHERE` clause:

1. **`=` (equal)**
2. **`<>` (not equal, may be written as `!=` in some SQL versions)**
3. **`>` (greater than)**
4. **`<` (less than)**
5. **`>=` (greater than or equal)**
6. **`<=` (less than or equal)**
7. **`BETWEEN` (between an inclusive range)**
8. **`LIKE` (search for a pattern)**
9. **`IN` (specify multiple possible values for a column)**

8.3.6 SQL AND, OR and NOT operators

The `AND`, `OR` and `NOT` operators can be used in conjuntion with the `WHERE` clause. The `AND` operator requires more than one condition be met to display a row.

```
SELECT * FROM `dolphins` WHERE name='Flipper' AND age=3;
```

The OR operator specifies alternate perameters which could be met to display a row.

```
SELECT * FROM `dolphins` WHERE name='Flipper' OR name='Splashy';
```

The NOT operator excludes a row from being displayed if a condition is met. Sorry Dorsal!

```
SELECT * FROM `dolphins` WHERE NOT name='Dorsal';
```

8.3.7 NULL Values

Like PHP variables, it is possible for an SQL field to have a value of null, meaning that it has no value. A table field can only be allowed to be null if it is an optional field. You can check if a field is null by using the `IS NULL` and `IS NOT NULL` statements. Display rows if column is NULL.

```
SELECT * FROM `dolphins` WHERE age IS NULL;
```

Display rows if column IS NOT NULL.

```
SELECT * FROM `dolphins` WHERE age IS NOT NULL;
```

8.3.8 SQL INSERT INTO Statement

The `INSERT INTO` statement inserts new rows, or "instances" of an item into a table. `INSERT INTO` is used in combination with the `VALUES` statement to define column values for the new row. Be careful to use ***backticks*** and not single quotes for the table and column names. However, column ***values*** are wrapped in regular ***single quotes***. Insert a new row into a table.

```
INSERT INTO `dolphins` (`id`, `name`, `age`) VALUES (4, 'Porpy', 6);
```

8.3.9 SQL UPDATE

The `UPDATE` statement is used to update rows in a table. The `UPDATE` statement is used in conjunction with the **SET** statement to set new values for fields.

```
UPDATE `dolphins` SET age=4 WHERE name='Flipper';
```

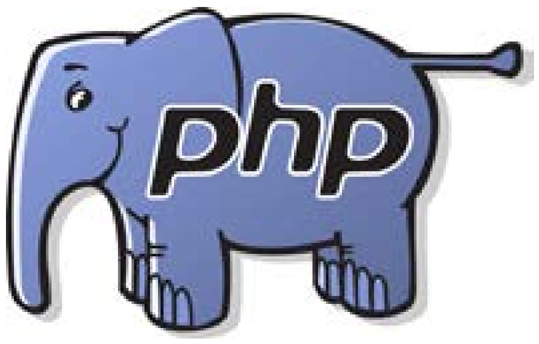
8.3.10 SQL DELETE

The DELETE statement deletes existing rows in a table. Delete from table where this condition is met.

```
DELETE FROM `dolphins` WHERE name='Dorsal';
```

Sorry again Dorsal!

9. MySQLi



9.1 MySQL vs. MySQLi

We are going to use PHP to connect to our MySQL database. In order to do this, we will need to import our database credentials into our app with PHP, connect to the database using PHP, import data from the database using SQL to display on our page using PHP and send data from a form to the database using SQL.

There used to be a built-in library for managing ***MySQL*** databases using PHP which provided a series of functions such as `mysql_connect()` and `mysql_query()`. However, this library is now deprecated and has been removed in PHP 7.

Even prior to PHP 7, it had been replaced with a newer built-in library called ***MySQLi***. The "i" in MySQLi stands for "improved" because it made improvements to the now outdated MySQL library. The ***MySQLi*** library uses the same functions as the previous MySQL library except they are named with an "i" such as `mysqli_connect()` and `mysqli_query()`.

PHP Resources

1. PHP Website
2. XAMPP
3. Documentation
4. W3Schools
5. CSS Tricks
6. Scotch.io
7. Stack Overflow

You should never use the old MySQL functions, and should always use the newer MySQLi functions or PDO (See ***10. PDO (PHP Data Object)***) instead.

9.2 Connecting to an SQL Database with MySQLi

First our database credentials are saved as variables. Because we are working locally, the host is localhost, PHPMyAdmin's default username is "root" and the default password is blank. If you are using MAMP then it is required that you have a password so you will need to set one. In this case, our database name is "dolphins".

```
$db_host = 'localhost';
$db_user = 'root';
$db_password = '';
```

```
  $db_name = 'dolphins';
```

The database credentials are used as parameters in the `mysqli_connect()` function which will connect us to the database.

```
  $connection = mysqli_connect($db_host, $db_user, $db_password,
  $db_name);
```

9.3 Displaying the Data with MySQLi

First we will save a query to select all rows from our `dolphins` table in a variable. Then, we can use our `$connection` variable and our `$select_query` variable in the `mysqli_query()` function to run a query in the SQL database. Be careful to use ***backticks*** and not single quotes for the table name.

```
  $select_query = SELECT * FROM `dolphins`;
  $select_result = mysqli_query($connection, $select_query);
```

Now that we have selected all of the dolphin rows from the database, we will store all of the rows in an associative array using the `mysqli_fetch_assoc()` function.

Once they are in an array we can loop through them. We could use any of the loops but let's use the while loop. The while loop will loop while the `mysqli_fetch_assoc()` function has rows. We can then echo out individual array items. When echoing data which has been retrieved from the database we would want to escape HTML using `htmlspecialchars()` at this point.

```
  <table>
     <thead>
          <tr>
               <th>#</th>
               <th>Name</th>
               <th>Age</th>
               <th>Image</th>
          </tr>
     </thead>
     <tbody>
          <?php while ($row = mysqli_fetch_assoc($select_result)): ?>
          <tr>
               <td><?php echo htmlspecialchars($row['id']); ?></td>
               <td><?php echo htmlspecialchars($row['name']); ?></td>
               <td><?php echo htmlspecialchars($row['age']); ?></td>
               <td><img src="img/<?php echo
                    strtolower(htmlspecialchars($row['name'])); ?>"
                    alt="<?php echo htmlspecialchars($row['name']); ?>">
               </td>
          </tr>
          <?php endwhile; ?>
     </tbody>
  </table>
```

9.4 Adding New Data with MySQLi

Simply displaying data is something we could have just used HTML for. The advantage of using a database is that we can allow the user to dynamically add data. To do this we will need to send data from a form to the database.

First we need to save this file name as a variable and perform escaping procedures on it.

```
$this_file = basename(htmlspecialchars($_SERVER[‘PHP_SELF’], ENT_
QUOTES, ‘UTF-8’));
```

Then we can make a form which has an action that points to this variable. To submit data to a database we would normally want to use a form with a method of "post".

```
<form action=”<?php echo $this_file; ?>” method=”post”>

   <label for=”name”>Name:</label>
   <input type=”text” name=”name” id=”name”>

   <label for=”age”>Age:</label>
   <input type=”number” name=”age” id=”name”>

   <button type=”submit”>Submit</button>
</form>
```

Once we have made our form, we can import our form data using the `$_POST` superglobal. First, we need to check to make sure that the form data is not empty by using the `empty()` function.

Once we have confirmed that the `$_POST` array items are not empty we can save our post data as variables and filter it.

```
// Run this code if name and age are not empty
if (!empty($_POST[‘name’]) && !empty($_POST[‘age’])):
   // Retreive the user input from the $_POST array and filter
   $name = trim(strip_tags($_POST['name']));
   $age = trim(strip_tags($_POST['age']));
else:
   // Else display an error message if the form is not filled out
   ?>
   <div class="error">We need more information!</div>
   <?php
endif;
```

When saving data to an SQL database using the MySQLi library, it is also important to use the `mysqli_real_escape_string()` function to escape any special characters such as quotes and apostrophes before they are sent to the database. It is necessary to include the `$connection` variable as the first parameter when using `mysqli_real_escape_string()`.

```
$name = trim(strip_tags($_POST['name']));
$name = mysqli_real_escape_string($connection, $name);

$age = trim(strip_tags($_POST['age']));
$age = mysqli_real_escape_string($connection, $age);
```

Then we can insert them into an SQL query using interpolation. To use interpolation we MUST use ***double quotes***. Be careful to use ***backticks*** and not single quotes for the table and column names. However, column ***values*** are wrapped in regular ***single quotes***.

```
$insert_query = "INSERT INTO `dolphins` (`name`, `age`) VALUES
('${name}', '${age}')";
```

Finally, we will use the `mysqli_query()` function as a condition in an if else statement to provide a condition for if the function runs successfully and a fallback for if it fails. In this case we are just going to display success and error messages in our HTML.

```
// Run this code if the database query runs successfully
if (mysqli_query($connection, $select_query)):
   ?>
   <div class="success">You have successfully added
        <?php echo htmlspecialchars($name) ?>to the database!</div>
   <?php
else:
   // Else display an error message
   ?>
   <div class="error">Database error!</div>
   <?php
endif;
```

The complete code would look like this.

```
// Run this code if name and age are not empty
if (!empty($_POST['name']) && !empty($_POST['age'])):
   // Retreive the user input from the $_POST array and filter
   $name = trim(strip_tags($_POST['name']));
   $age = trim(strip_tags($_POST['age']));

   // Escape special characters before sending to SQL database
   $name = mysqli_real_escape_string($connection, $name);
   $age = mysqli_real_escape_string($connection, $age);

   // Interpolate variables into SQL INSERT INTO query
   $insert_query = "INSERT INTO `dolphins` (`name`, `age`) VALUES
        ('${name}', '${age}')";

   // Run this code if the database query runs successfully
   if (mysqli_query($connection, $select_query)):
        ?>
        <div class="success">
             You have successfully added <?php echo
             htmlspecialchars($name) ?>to the database!
        </div>
        <?php
   else:
        // Else display an error message if database fails
        ?>
        <div class="error">Database error!</div>
        <?php
    endif;
else:
    // Else display an error message if the form is not filled out
?>
    <div class="error">We need more information!</div>
<?php
endif;
?>
```

10. PDO (PHP Data Object)

10.1 PDO vs. MySQLi

PDO, or the PHP Data Object, is a built-in object in PHP which is used for connecting to databases and running SQL queries.

It works very similarly to MySQLi but it supports eighteen different databases while MySQLi only supports MySQL/MariaDB. PDO also uses an object-oriented interface. Although MySQLi offers an object-oriented interface as well, we used the procedural version of MySQLi to compare the difference between the two programming styles. For more information on the differences between PDO and MySQLi, see this article.

10.2 Connecting to an SQL Database with PDO

We can connect to the database by creating a new instance of the PDO class, and using our credentials as the parameters.

```
$pdo = new PDO(
    'mysql:host=' . $db_host .
    ';dbname=' . $db_name,
    $db_user,
    $db_password
);
```

If PDO is not able to connect to the database then this will display an ugly PHP error message. We can use a ***try catch*** statement to display a nicer looking and more user friendly error message in case the database connection fails.

```
try {
    $pdo = new PDO(
        'mysql:host=' . $db_host .
        ';dbname=' . $db_name,
        $db_user,
        $db_password
    );
} catch (PDOException $e) {
    ?>
    <div class="error">
         <h2>No Database Connection!</h2>
         <p><?php echo $e->getMessage() ?></p>
    </div>
    <?php
}
```

10.3 Displaying the Data with PDO

First we will save a query to select all rows from our `dolphins` table in a variable. We can now use PDO's `prepare()` method to write an SQL query to select all rows from the `dolphins` table. Be careful to use ***backticks*** and not single quotes for the table name.

```
$select_query = $pdo->prepare("SELECT * FROM `dolphins`");
```

Now that we have selected all of the dolphin rows from the database, we will store all of the rows in an associative array using PDO's `fetchAll()` method.

```
$all_rows = $select_query->fetchAll();
```

Once they are in an array we can loop through them. We will use a foreach loop. We can then echo out individual array items. When echoing data which has been retrieved from the database we would want to escape HTML using `htmlspecialchars()` at this point.

```
<table>
    <thead>
         <tr>
              <th>#</th>
              <th>Name</th>
              <th>Age</th>
              <th>Image</th>
         </tr>
    </thead>
    <tbody>
         <?php foreach ($all_rows as $row_number => $row): ?>
         <tr>
              <td><?php echo htmlspecialchars($row['id']); ?></td>
              <td><?php echo htmlspecialchars($row['name']); ?></td>
              <td><?php echo htmlspecialchars($row['age']); ?></td>
              <td><img src="img/<?php echo
                   strtolower(htmlspecialchars($row['name'])); ?>"
                   alt="<?php echo htmlspecialchars($row['name']); ?>">
              </td>
         </tr>
         <?php endforeach; ?>
    </tbody>
</table>
```

10.4 Adding New Data with PDO

When saving data to an SQL database using PDO, the `prepare()` method automatically escapes any special characters such as quotes and apostrophes before they are sent to the database so it is not necessary to use any additional functions for escaping characters.

We will interpolate our variables into an SQL query which is inserted directly as the parameter of the `prepare()` method.

To use interpolation we MUST use ***double quotes***. Be careful to use ***backticks*** and not single quotes for the table and column names. However, column ***values*** are wrapped in regular ***single quotes***.

```
$name = trim(strip_tags($_POST['name']));
$age = trim(strip_tags($_POST['age']));

$insert_query = $pdo->prepare("INSERT INTO `dolphins` (`name`, `age`)
VALUES ('${name}', '${age}')");
```

Finally, we will use PDO's `execute()` method as a condition in an if else statement to provide a condition for if the function runs successfully and a fallback for if it fails.

```
// Run this code if the database query runs successfully
if ($query->execute()):
   ?>
   <div class="success">You have successfully added
        <?php echo htmlspecialchars($name) ?>to the database!</div>
   <?php
else:
   // Else display an error message if database fails
   ?>
   <div class="error">Database error!</div>
   <?php
endif;
```

The complete code would look like this.

```
if (!empty($_POST['name']) && !empty($_POST['age'])):
   // Retreive the user input from the $_POST array and filter
   $name = trim(strip_tags($_POST['name']));
   $age = trim(strip_tags($_POST['age']));

   // Escape special characters before sending to SQL database
   // Interpolate variables into SQL INSERT INTO query
   $insert_query = $pdo->prepare("INSERT INTO `dolphins` (`name`,
        `age`) VALUES ('${name}', '${age}')");

   // Run this code if the database query runs successfully
   if ($query->execute()):
        ?>
        <div class="success">
             You have successfully added <?php echo
             htmlspecialchars($name) ?>to the database!
        </div>
        <?php
    else:
        // Else display an error message if database fails
        ?>
        <div class="error">Database error!</div>
        <?php
    endif;
else:
    ?>
    <div class="error">We need more information!</div>
    <?php
endif;
?>
```