PHP Data Objects

CSC 242, Web Programming

PHP Data Objects (PDO)

- PHP Data Objects (PDO) is a PHP extension that defines an interface for accessing databases.
- Benefits of PDO:
  - Security (prepared statements)
  - Reusability (access different DBMS)

Connecting to a DBMS

- A connection to a database is created by constructing an instance of a PDO object.
- The PDO object has functions to access the database.
- Syntax to create a connection:
  `$db = new PDO($dsn, $user, $password)`
  - `$dsn` the data source name
  - `$user` optional user name
  - `$password` optional password

Example: Connect to a SQLite Database

```
// connect to the SQLite database
// named example.db
$dsn = "sqlite:example.db";
$db = new PDO($dsn);

// do some stuff with the database

// close the database connection
$db = null;
```

The `PDO::query` Function

- The `PDO::query` function can be used to query the database.
- The `PDO::query` function is used for SQL SELECT statements.
- The `PDO::query` function returns a `PDOStatement` object.
- The `PDOStatement` object has functions to access the result of executing the query.

PDO::query Example

```
// connect to example.db
$db = new PDO("sqlite:example.db");

// make a query
$sql = "SELECT * FROM table_name";

// execute the query with PDO::query
$stmt = $db->query($sql);
```

Retrieve Data from a `PDOStatement`

- Using a foreach:

```
foreach($stmt as $row) {
  // do something with the row
}
```

- Using `PDOStatement::fetch`

```
while ($row = $stmt->fetch()) {
  // do something with the row
}
```

- Using `PDOStatement::fetchAll`

```
$all_rows = $stmt->fetchall();
```

Options for `PDOStatement::fetch`

- The form of the return value from `PDOStatement::fetch` and `PDOStatement::fetchall` can be changed by a parameter:
  - `PDO::FETCH_NUM` return a numeric array
  - `PDO::FETCH_ASSOC` return an associative array with the column names as keys.
  - `PDO::FETCH_BOTH` returns both of the above
- Example

```
// return all records as associative arrays
$records = $stmt->fetchall(PDO::FETCH_ASSOC);
```

The `PDO::exec` Function

- The `PDO::exec` function exectues an SQL statement.
- The `PDO::exec` function returns the number of rows affected by the SQL statment
- The `PDO::exec` function should be used for the SQL statements: INSERT, UPDATE, and DELETE.

PDO Prepared Statements

- Prepared statments provide protection against SQL injections.
- A prepared statement is the only proper way to run a query.
- Prepared statements use placeholders for variables.
- PDO prepared statments make use of the functions `PDO::prepare` and `PDOStatement::execute`

The `PDO::prepare` Function

- The `PDO::prepare` function takes a SQL statement string with *placeholders* where the real values will be substituted when the statment is executed.
- There are two kinds of placeholders:
  - Positional: use the question mark (?) for placeholders
  - Named: use named (:name) placeholders
- The `PDO::prepare` function returns a `PDOStatement` object.

The `PDOStatement::execute` Function

- The `PDOStatement::execute` function takes the result of the `PDO::prepare` function substitutes the placeholders with real values and executes the statement.
- The argument depends on the kind of placeholders:
  - Positional: the argument is a numeric array with the elements in positional order.
  - Named: the argument is an associative array with the placeholder names as keys.

PDO Prepared Statement Examples

- Positional placeholders

```
$sql =
"SELECT * FROM users
 WHERE name = ? and email = ?";
$stmt = $db->prepare($sql);
$stmt->execute(['Bob', 'bob@axample.com']);
```

- Named placeholders

```
$sql =
"SELECT * FROM users
 WHERE name = :name and email = :email";
$stmt = $db->prepare($sql);
$stmt->execute(
  array('name' => 'Bob',
        'email' => bob@axample.com')
  );
```