Manipulate data using PDO

Objectives

- Selecting data using PDO
- Transaction
- Call StoredProcedure
- Demo code using PDO

Selecting Data

- Data is obtained via the ->fetch(), a method of your statement handle. Before calling fetch, it's best to tell PDO how you'd like the data to be fetched.

- In reality, there are three which will cover most situations: FETCH_ASSOC, FETCH_CLASS, and FETCH_OBJ. In order to set the fetch method, the following syntax is used:

```
$STH->setFetchMode(PDO::FETCH_ASSOC);
```

Selecting Data

- You have the following options:
  - **PDO::FETCH_ASSOC:** returns an array indexed by column name
  - **PDO::FETCH_BOTH (default):** returns an array indexed by both column name and number
  - **PDO::FETCH_BOUND:** Assigns the values of your columns to the variables set with the ->bindColumn() method
  - **PDO::FETCH_CLASS:** Assigns the values of your columns to properties of the named class. It will create the properties if matching properties do not exist
  - **PDO::FETCH_INTO:** Updates an existing instance of the named class
  - **PDO::FETCH_LAZY**: Combines PDO::FETCH_BOTH/PDO::FETCH_OBJ, creating the object variable names as they are used
  - **PDO::FETCH_NUM:** returns an array indexed by column number
  - **PDO::FETCH_OBJ:** returns an anonymous object with property names that correspond to the column names

FETCH_ASSOC

- This fetch type creates an associative array, indexed by column name. This should be quite familiar to anyone who has used the mysql/mysqli extensions.

```
# using the shortcut ->query() method here since there are no variable
# values in the select statement.
$STH = $DBH->query('SELECT name, addr, city from folks');

# setting the fetch mode
$STH->setFetchMode(PDO::FETCH_ASSOC);

while($row = $STH->fetch()) {
    echo $row['name'] . "\n";
    echo $row['addr'] . "\n";
    echo $row['city'] . "\n";
}
```

- The while loop will continue to go through the result set one row at a time until complete.

FETCH_OBJ

- This fetch type creates an object of class for each row of fetched data.

```
# creating the statement
$STH = $DBH->query('SELECT name, addr, city from folks');

# setting the fetch mode
$STH->setFetchMode(PDO::FETCH_OBJ);

# showing the results
while($row = $STH->fetch()) {
    echo $row->name . "\n";
    echo $row->addr . "\n";
    echo $row->city . "\n";
}
```

FETCH NUM

- PDO::FETCH_NUM produces a numerical index of the result set rather than the field names.

```php
/*** The SQL SELECT statement ***/
$sql = "SELECT * FROM animals";

/*** fetch into an PDOStatement object ***/
$stmt = $dbh->query($sql);

/*** echo number of columns ***/
$result = $stmt->fetch(PDO::FETCH_NUM);

/*** loop over the object directly ***/
foreach($result as $key=>$val)
{
echo $key.' - '.$val.'<br />';
}

/*** close the database connection ***/
$dbh = null;
```

- Result

```
0 - 1
1 - emu
2 - bruce
```

FETCH BOTH

- There may be times you need to fetch both numerical and associative indexes. PDO::FETCH_BOTH produces a numerical and associative index of the result set so you can use either, or both.

```php
$dbh = new PDO("mysql:host=$hostname;dbname=animals", $username, $password);
/*** echo a message saying we have connected ***/
echo 'Connected to database<br />';

/*** The SQL SELECT statement ***/
$sql = "SELECT * FROM animals";

/*** fetch into an PDOStatement object ***/
$stmt = $dbh->query($sql);

/*** echo number of columns ***/
$result = $stmt->fetch(PDO::FETCH_BOTH);

/*** loop over the object directly ***/
foreach($result as $key=>$val)
{
echo $key.' - '.$val.'<br />';
}
```

- Result

```
Connected to database
animal_id - 1
0 - 1
animal_type - emu
1 - emu
animal_name - bruce
2 - bruce
```

FETCH CLASS

- PDO::FETCH_CLASS instantiates a new instance of the specified class. The field names are mapped to properties (variables) within the class called. This saves quite a bit of code and speed is enhanced as the mappings are dealt with internally.
- Example

```php
$dbh = new PDO("mysql:host=$hostname;dbname=animals", $username, $password);
/*** echo a message saying we have connected ***/
echo 'Connected to database<br />';

/*** The SQL SELECT statement ***/
$sql = "SELECT * FROM animals";

/*** fetch into an PDOStatement object ***/
$stmt = $dbh->query($sql);

/*** fetch into the animals class ***/
$obj = $stmt->fetchALL(PDO::FETCH_CLASS, 'animals');
```

FETCH LAZY

- PDO::FETCH_LAZY is odd as it combines PDO::FETCH_BOTH and PDO::FETCH_OBJ.
- Example

```php
$dbh = new PDO("mysql:host=$hostname;dbname=animals", $username, $password);
/*** echo a message saying we have connected ***/
echo 'Connected to database<br />';

/*** The SQL SELECT statement ***/
$sql = "SELECT * FROM animals";

/*** fetch into an PDOStatement object ***/
$stmt = $dbh->query($sql);

/*** echo number of columns ***/
$result = $stmt->fetch(PDO::FETCH_BOTH);

/*** loop over the object directly ***/
foreach($result as $key=>$val)
{
echo $key.' - '.$val.'<br />';
}
```

Transactions

- A PDO transaction begins with the with PDO::beginTransaction() method. This method turns off auto-commit and any database statements or queries are not committed to the database until the transaction is committed with PDO::commit.
- When PDO::commit is called, all statements/queries are enacted and the database connection is returned to auto-commit mode.

Transactions

- Example

```php
<?php
/*** mysql hostname ***/
$hostname = 'localhost';

/*** mysql username ***/
$username = 'username';

/*** mysql password ***/
$password = 'password';

/*** database name ***/
$dbname = 'animals';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username, $password);
    /*** echo a message saying we have connected ***/
    echo 'Connected to database<br />';

    /*** set the PDO error mode to exception ***/
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    /*** begin the transaction ***/
    $dbh->beginTransaction();
```

Transactions

```
/*** CREATE table statements ***/
$table = "CREATE TABLE animals ( animal_id MEDIUMINT(8) NOT NULL AUTO_INCREMENT PRIMARY KEY
animal_type VARCHAR(25) NOT NULL,
animal_name VARCHAR(25) NOT NULL
)";
$dbh->exec($table);
/***  INSERT statements ***/
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('emu', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('funnel web', 'bruce')")
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('lizard', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('dingo', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('kangaroo', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('wallaby', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('wombat', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('koala', 'bruce')");
$dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES ('kiwi', 'bruce')");

/*** commit the transaction ***/
$dbh->commit();
```

```
    /*** echo a message to say the database was created ***/
    echo 'Data entered successfully<br />';
}
catch(PDOException $e)
    {
    /*** roll back the transaction if we fail ***/
    $dbh->rollback();

    /*** echo the sql statement and error message ***/
    echo $sql . '<br />' . $e->getMessage();
    }
?>
```

Call MySQL Stored Procedures

- The steps of calling a stored procedure that returns a result set using PHP PDO are similar to querying data from MySQL database table using the SELECT statement.
- Instead of sending a SELECTstatement to MySQL database, you send a stored procedure call statement.
- Example

```
CREATE PROCEDURE GetCustomers()
BEGIN
 SELECT customerName, creditlimit
 FROM customers;
    END$$
```

Call MySQL Stored Procedures

```
<html>
    <head>
        <title>PHP MySQL Stored Procedure Demo 1</title>
        <link rel="stylesheet" href="css/table.css" type="text/css" />
    </head>
    <body>
        <?php
        require_once 'dbconfig.php';
        try {
            $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password
);
            // execute the stored procedure
            $sql = 'CALL GetCustomers()';
            // call the stored procedure
            $q = $pdo->query($sql);
            $q->setFetchMode(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            die("Error occurred:" . $e->getMessage());
        }
        ?>
         <table>
             <tr>
                 <th>Customer Name</th>
                 <th>Credit Limit</th>
             </tr>
             <?php while ($r = $q->fetch()): ?>
                 <tr>
                     <td><?php echo $r['customerName'] ?></td>
                     <td><?php echo '$' . number_format($r['creditlimit'], 2) ?>
                     </td>
                 </tr>
             <?php endwhile; ?>
         </table>
    </body>
</html>
```

Call MySQL Stored Procedures

- Output:

Customer Name	Credit Limit
Atelier graphique	$21,000.00
Signal Gift Stores	$71,800.00
Australian Collectors, Co.	$117,300.00
La Rochelle Gifts	$118,200.00
Baane Mini Imports	$81,700.00
Mini Gifts Distributors Ltd.	$210,500.00
Havel & Zbyszek Co	$0.00

Demo

- Teacher demo code about database CRUD operations using PDO

Summary

- Selecting data using PDO
- Transaction
- Call StoredProcedure
- Demo code using PDO