CPET 499/ITC 250 Web Systems

Chapter 14
Working with Databases
Part 2 of 3

Text Book:
*** Fundamentals of Web Development, 2nd edition, by Randy Connolly and Ricardo Hoar, published by Pearson**

Paul I-Hai Lin, Professor of Electrical and Computer Engineering Technology
http://www.etcs.pfw.edu/~lin

Topics

- Accessing MySQL in PHP
- More PHP Data Object (PDO) and mysqli Procedural Style APIS
- Integrating User Input Data Into Query
- PHP and MySQL Tasks
  - Making MySQL connection and closing connection
  - Display a List of Links
  - Search and Results Page
  - Editing a Record
  - Saving and Displaying Raw Files in the Database
  - Displaying BLOBs from the Database
  - Using Transactions
- **Database Schemas:**
  - Art Database, Book CRM Database, Travel Photo Database

Accessing MySQL in PHP

Basic Connection Algorithm

1. Connect to the database
2. Handle connection errors
3. Execute the SQL query
4. Process the results
5. Free resources and close connection

Connecting to MySQL Database Using PHP Data Object (PDO)

Figure 14.22 Basic Database Connection using PHP PDO

```
<?php

try {
   $connString = "mysql:host=localhost;dbname=bookcrm";
   $user = "testuser";
   $pass = "mypassword";

   $pdo = new PDO($connString,$user,$pass);
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   $sql = "select * from Categories order by CategoryName";
   $result = $pdo->query($sql);

   while ($row = $result->fetch()) {
      echo $row['ID'] . " - " . $row['CategoryName'] . "<br/>";
   }
   $pdo = null;
}
catch (PDOException $e) {
   die( $e->getMessage() );
}

?>
```

1
2
3
4
5

Listings 14-3 Connecting to a database with mysqli (procedural)

```
<?php
// modify these variables for your installation
$host = "localhost";
$database = "bookcrm";
$user = "testuser";
$pass = "mypassword";
$connection = mysqli_connect($host, $user, $pass,
                             $database);
?>
```

Listings 14-4 Connecting to a database with PDO (object-oriented)

```
<?php
// modify these variables for your installation
$connectionString = "mysql:host=localhost;
                     dbname=bookcrm";
$user = "testuser";
$pass = "mypassword";
$pdo = new PDO($connectionString, $user, $pass);
?>
```


Listings 14-5 Defining connection details via constants in a separate file (config.php)

```
<? php
define('DMHOST', 'localhost');
define('DBNAME', 'bookcrm');
define('DBUSER', 'testuser');
define('DBPASS', 'mypassword');
?>
```

Listings 14-6 Using the connection constants

```
<?php
require_once(protected/config.php);
$connection = mysqli_connect(DBHOST, DBUSER,
DBPASS, DBNAME);
?>
```


Handling Connection Errors

Listings 14-8 Handling connection errors with mysqli (version 2)

```
<?php
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);
// mysqli_connect_errno returns the last error code
if ( mysqli_connect_errno() ) {
        die( mysqli_connect_error() );
        // die() is equivalent to exit()
}
?>
```


Listings 14-9 Handling connection errors with PDO

```
<?php
try {
        $connString = "mysql:host=localhost;dbname=bookcrm";
        $user = DBUSER;
        $pass = DBPASS;
        $pdo = new PDO($connString,$user,$pass);
        . . .
}
catch (PDOException $e) {
        die( $e->getMessage() );
}
?>
```

PDO Exception Modes

Three different error-handling modes/approaches

- PDO::ERRORMODE_SELENT
  - For normal production use
- PDO::ERRORMODE_WARING
  - For use during debugging/testing phase
- PDO::ERRORMODE_EXCEPTION
  - For use during debugging phase
  - Stop the script at the point of error

Executing Query

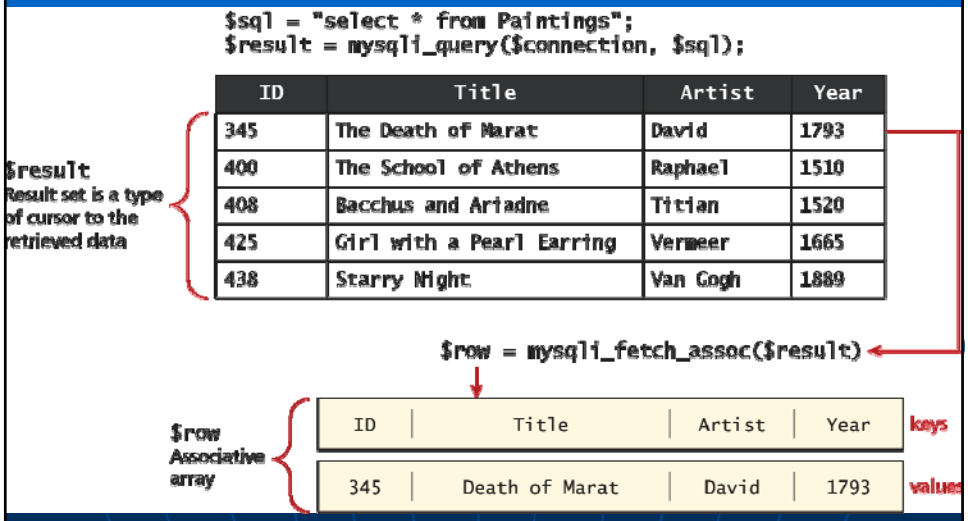
Listings 14.11 and 12 Executing a SELECT query (mysqli and PDO)

```
<?php
//Listing 14.11 Executing a SELECT query (mysqli)
$sql = "SELECT * FROM Categories ORDER BY CategoryName";
// returns a mysqli_result object
$result = mysqli_query($connection, $sql);
?>

<?php
//Listing 14.12 Executing a SELECT query (pdo)
$sql = "SELECT * FROM Categories ORDER BY CategoryName";
// returns a PDOStatement object
$result = $pdo->query($sql);
?>
```


Processing the Query Results

Figure 14.23 Fetching From a Result Set



Fetches and Displays Result Rest
Listing 14.13 Looping through the result set (PDO)

```
<?php
//Listing 14.13 Looping through the result set (PDO)
$sql = “SELECT * FROM Categories ORDER by CategoryName";
// run the query
$result = $pdo->query($sql);
while ( $row = $result->fetch() ) {
   echo $row['ID'] . " - " . $row['CategoryName'] ;
   echo  "<br/>";
  }
?>
```

PHP MySQL Fetching Functions

- **mysqli_fetch_all()**: Fetches all result rows as an associate array, a numeric array, or both
  - http://php.net/manual/en/mysqli-result.fetch-all.php
- **mysqli_fetch_array()**: Fetches a result row as an associate array, a numeric array, or both
  - http://php.net/manual/en/mysqli-result.fetch-array.php
- **mysqli_fetch_assoc():** Fetches a result row as an associate array
  - http://php.net/manual/en/mysqli-result.fetch-assoc.php
- **mysqli_fetch_field():** Returns the definition of one column of a result set as an object. Call this function repeatedly to retrieve information about all columns in the result set.
  - http://php.net/manual/en/mysqli-result.fetch-field.php

PHP MySQL: Procedural Style Fetching Functions

- **mysqli_fetch_fields():** Returns an array of objects which contains field definition information or FALSE if no filed information is available
  - http://php.net/manual/en/mysqli-result.fetch-fields.php
- mysqli_fetch_object(): Returns the current row of a result as an object
  - http://php.net/manual/en/mysqli-result.fetch-object.php
- mysqli_fetch_row(): Fetch one row of data from the result set as an numeric array
  - http://php.net/manual/en/mysqli-result.fetch-row.php

Fetching Into An Object

Book Class, page 659

```
class Book {
        public $ID;
        public $Title;
        public $CopyrightYear;
        public $Description;
}
```

Fetching Into an Object, page 658

```
<?php
//Listing 14.14 Populating an object from a result set (PDO)
$id = $_GET['id'];
$sql = "SELECT * FROM Books";
$results = $pdo->query($sql);
while ($b = $result -> fetchObject('Book') )
{
  echo 'ID: ' . $b->ID . '<br/>';
  echo 'Title: ' . $b->Title . '<br/>';
  echo 'Year: ' . $b->CopyrightYear . '<br/>';
  echo 'Description: ' . $b->Description . '<br/>';
  echo “<hr>”;
?>
```


Fetching Into an Object, pages 659-660

```
<?php
//Listing 14.15 Letting an object populate itself from a result set
class Book {
    public $id;
    public $title;
    public $year;
    public $description;
   function __construct($record)  {
   // the references to the field names in associative array must
   // match the case in the table
    $this->id = $record['ID'];
    $this->title = $record['Title'];
    $this->year = $record['CopyrightYear'];
    $this->description = $record['Description'];
} }
```

Fetching Into an Object, pages 659-660

```
//Listing 14.15 Letting an object populate itself from a result set
//...
// in some other page or class
$sql =“SELECT * FROM Books”;
$results = $pdo -> query($sql);
// fetch a record normally
while ($row = $result -> fetch() ){
  $b = new Book($row);
  echo 'ID: ' . $b->id . '<br/>';
  echo 'Title: ' . $b->title . '<br/>';
  echo 'Copyright Year: ' . $b->year . '<br/>';
  echo ‘Description: ‘ . $b->description . </br>’;
  echo “<hr>”;
?>
```


Freeing Resources and Closing Connection

Freeing Resources and Closing Connection, page 660

```
<?php  //Listing 14.16 Closing the connection
// mysqli approach
$connection = mysqli_connect($host, $user, $pass, $database);
$result= mysqli_query($connection, "SELECT … FROM …");
//...
// release the memory used by the result set. This is necessary if
// you are going to run another query on this connection
mysqli_free_result($result);
//...
// close the database connection
mysqli_close($connection);
```


Freeing Resources and Closing Connection

```
<?php  //Listing 14.16 Closing the connection
// PDO approach
$pdo = new PDO($connString,$user,$pass);
//...
// closes connection and frees the resources used by the PDO
object
$pdo = null;
?>
```

Working with Parameters

Working with Parameters, page 661

- **SQL Statements that perform action on the data**
  - INSERT
  - UPDATE
    DELETE
- Integrating User Data
- Sanitizing User Data
- Prepare Statements

Listing 14.18 Executing a query that doesn't return data (mysqli) - UPDATE

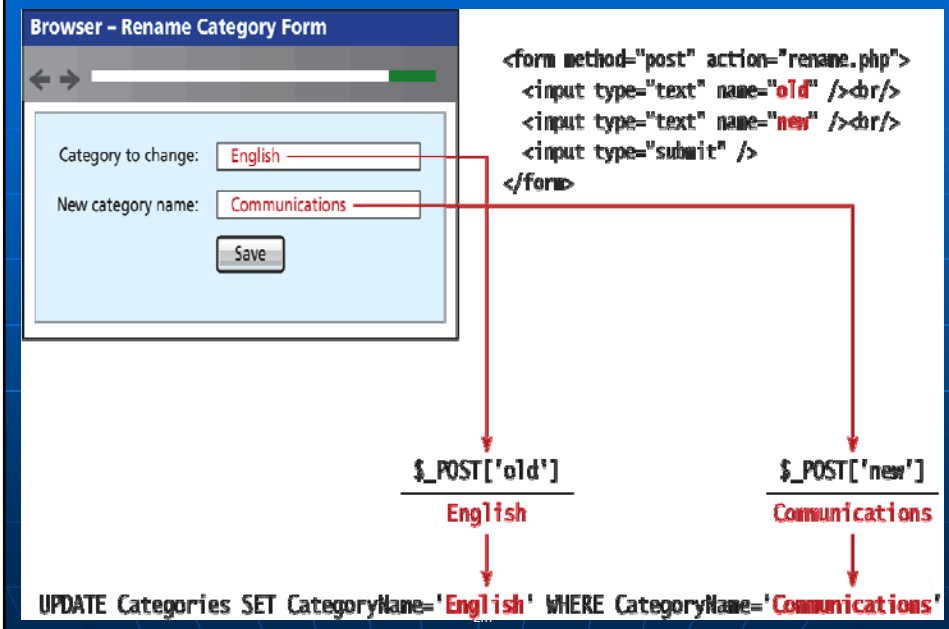
```
<?php
//Listing 14.18 Executing a query that doesn't return data (mysqli)
$sql = "UPDATE Categories SET CategoryName='Web' WHERE
CategoryName='Business'";

if ( mysqli_query($connection, $sql) ) {
   $count = mysqli_affected_rows($connection);
   echo "<p>Updated " . $count . " rows</p>";}
?>
```


Listing 14.17 Executing a query that doesn't return data (PDO) - UPDATE

```
<?php
//Listing 14.17 Executing a query that doesn't return data (PDO)
$sql = "UPDATE Categories SET CategoryName='Web' WHERE
CategoryName='Business'";
$count = $pdo->exec($sql);
echo "<p>Updated " . $count . " rows</p>";
?>
```

Figure 14.24 Integrating user input data into a query



- Integrating User Data into An Query
- Sanitizing User Data
- Prepare Statements

Listing 14.19 Integrating user input into a query (first attempt)

```
<?php
//Listing 14.19 Integrating user input into a query (first attempt)
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE
CategoryName='$from'";

$count = $pdo->exec($sql);
?>
```


Sanitizing User Input Data

- Remove any special characters from a desired piece of text
  - mysqli_real_escape_string()
  - quote() - PDO

```
<?php// Sanitizing user input before use in an SQL query
$from = $pdo->quote($from);
$to = $pdo->quote($to);
$sql = "UPDATE Categories SET CategoryName=$to WHERE
CategoryName=$from";
$count = $pdo->exec($sql);?>
```

Prepared Statements

- Prepared Statements
  - A way to improve performance for queries that need to be executed multiple times
  - It also integrates sanitization into each user input automatically, so it can protect SQL Injection
- To fully protect against attack called “SQL injection”
  - Go beyond “user input sanitization”
  - Use prepared statement technique (best)

Listing 14.20 Using a prepare statement (PDO)

```
<?php
//Listing 14.20 Using a prepared statement (PDO)
// retrieve parameter value from query string
$id = $_GET['id'];
/* method 1 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $id);
$statement->execute();
/* method 2 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = :id";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id', $id);
$statement->execute();?>
```

Listing 14.21 Using named parameters (PDO)

```
<?php//Listing 14.21 Using named parameters (PDO)
/* technique 1 - question mark placeholders */
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,ProductionStatusId, TrimSize, Description)
VALUES(?,?,?,?,?,?,?)";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $_POST['isbn']);
$statement->bindValue(2, $_POST['title']);
$statement->bindValue(3, $_POST['year']);
$statement->bindValue(4, $_POST['imprint']);
$statement->bindValue(4, $_POST['status']);
$statement->bindValue(6, $_POST['size']);
$statement->bindValue(7, $_POST['desc']);
$statement->execute();
```


Listing 14.21 Using named parameters (PDO)

```
/* technique 2 - named parameters */
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear,
ImprintId,ProductionStatusId, TrimSize, Description) VALUES
(:isbn,:title, :year, :imprint, :status, :size, :desc) ";
$statement = $pdo->prepare($sql);
$statement->bindValue(':isbn', $_POST['isbn']);
$statement->bindValue(':title', $_POST['title']);
$statement->bindValue(':year', $_POST['year']);
$statement->bindValue(':imprint', $_POST['imprint']);
$statement->bindValue(':status', $_POST['status']);
$statement->bindValue(':size', $_POST['size']);
$statement->bindValue(':desc', $_POST['desc']);
$statement->execute();
?>
```

Listing 14.22 Using a prepare statement (mysqli)

```
<?php
//Listing 14.22 Using a prepared statement (mysqli)
// retrieve parameter value from query string
$id = $_GET['id'];
// construct parameterized query – notice the ? Parameter
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";
// create a prepared statement
if ($statement = mysqli_prepare($connection, $sql)) {
// Bind parameters s - string, b - blob, i - int, etc
mysqli_stmt_bindm($statement, 'i', $id);
// execute query
mysqli_stmt_execute($statement);
// learn in next section how to access the returned data  //...}
?>
```


Using Transactions

Fetches and Displays Result Rest
Looping through the result set (mysqli)

```
<?php
//Looping through the result set
// (mysqli—not prepared statements)
$sql = "select * from Categories order by CategoryName";
 // run the query
if ($result = mysqli_query($connection, $sql)) {
  // fetch a record from result set into an associative array
while($row = mysqli_fetch_assoc($result))    {
// the keys match the field names from the table
 echo $row['ID'] . " - " . $row['CategoryName'] ;
 echo "<br/>";
 }
 }
?>
```


Using Transactions, page 666

- **Transactions**
  - Unnecessary when retrieving database data
  - Should be used for most scenarios involving any database “writes”

Listing 14.23 Using Transactions (mysqli)

```
<?php
//Listing 14.23 Using transactions (mysqi extension)
$connection = mysqli_connect($host, $user, $pass, $database);
//...
/* set autocommit to off. If autocommit is on, then mysql will
commit (i.e., make the data change permanent) each command
afterit is executed */
mysqli_autocommit($connection, FALSE);
/* insert some values */
$result1 = mysqli_query($connection,"INSERT INTO
Categories (CategoryName) VALUES ('Philosophy')");

$result2 = mysqli_query($connection,"INSERT INTO
Categories (CategoryName) VALUES ('Art')");
```


Listing 14.23 Using Transactions (mysqli)

```
<?php
//Listing 14.23 Using transactions (mysqi extension)
if ($result1 && $result2) {
 /* commit transaction */
  mysqli_commit($connection);
 }
else
 {
   /* rollback transaction */
  mysqli_rollback($connection);}
?>
```

Listing 14.24 Using Transactions (PDO)

```
<?php
//Listing 14.24 Using transactions (PDO)
$pdo = new PDO($connString,$user,$pass);
// turn on exceptions so that exception is thrown if error occurs
$pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
//...
try {
 // begin a transaction
$pdo->beginTransaction();
 // a set of queries: if one fails, an exception will be thrown
$pdo->query("INSERT INTO Categories (CategoryName) VALUES
('Philosophy')");
$pdo->query("INSERT INTO Categories (CategoryName) VALUES
('Art')");
```


Listing 14.24 Using Transactions (PDO)

```
//Listing 14.24 Using transactions (PDO)

// if we arrive here, it means that no exception was thrown
// which means no query has failed, so we can commit the
// transaction
$pdo->commit();
}
catch (Exception $e)
 {
// we must rollback the transaction since an error occurred
// with insert
$pdo->rollback();
}
?>
```

Q & A