ISIT307 - WEB SERVER PROGRAMMING

LECTURE 5.2 – WORKING WITH DATABASES USING PHP

LECTURE PLAN

- Connect to MySQL from PHP
- Work with MySQL databases using PHP
- Create, modify, and delete MySQL tables with PHP
- Use PHP to manipulate MySQL records and retrieve database records
- PHP prepared statements

Look for additional resources:
- https://www.w3schools.com/php/default.asp; https://www.php.net
- Nixon, Robin. Learning PHP, MySQL and JavaScript : With JQuery, CSS and HTML5, O'Reilly Media, Incorporated, 2014. ProQuest Ebook Central

DATABASES VS FILE-SYSTEMS

- use of indexing - makes calculation, retrieval and search extremely fast and efficient
  - file systems - retrieval and search are done manually
  - Databases - DBMS provides automated, organized, and effective methods
- controlled redundancy
- minimum maintenance required
- have a strong logging mechanism and can provide multiple user interfaces
- provide back-up and recovery

CONNECTING TO DATABASES WITH PHP

- PHP has the ability to access and manipulate any database that is ODBC compliant
- PHP includes functionality that allows you to work directly with different types of databases, without going through ODBC or PEAR DB
- mySQLi
- PDO

PHP DATA OBJECTS - PDO

- lightweight and consistent interface for accessing databases in PHP
- data access layer that uses a unified API (a database-specific PDO driver must be used to access a database server)
- another way to access a MySQL database from PHP

MYSQLI PACKAGE

- The `mysqli` (MySQL Improved) package became available with PHP 5 and is designed to work with MySQL version 4.1.3 and later
- Earlier versions must use the `mysql` package
- With PHP 5.5.x the `mysql` package is deprecated, so `mysqli` package should be used
- The `mysqli` package is the object-oriented equivalent of the `mysql` package
- The mysqli extension features a dual interface - it supports the procedural and object-oriented programming paradigm

OPENING AND CLOSING A MYSQL CONNECTION

- A connection to a MySQL database server can be created by instantiating a new object of mysqli

```
$conn = new mysqli(host, username, password[,dbname,
                                              port,socket])
```

- `$conn` represents the connection to the MySQL server
- The database connection can be closed using the `close()` method

```
$conn ->close()
```

OPENING AND CLOSING A MYSQL CONNECTION

- The *host* argument specifies the host name where the MySQL database server is installed
- The *user* and *password* arguments specify a MySQL account name and password
- The *dbname* argument specify the database name (default database to be used when performing queries)

REPORTING MYSQL ERRORS

- Reasons for not connecting to a database server include:
  - The database server is not running
  - Insufficient privileges to access the data source
  - Invalid username and/or password

REPORTING MYSQL ERRORS

- if the connection to the database server is unsuccessful, the error code and description of the last connection error can be retrieved from the `connect_errno` and `connect_error` properties (data members)
- The error code and description of the most recent `mysqli*` method call can be retrieved from the `errno` and `error` properties (of the connection object)

```
$conn->connect_errno, $conn->connect_error,
$conn->error, $conn->errno
```

- `die(`*`error properties`*`)` is syntax used as a short way of writing the code that will display the error and exit the script immediately

SUPPRESSING ERRORS WITH THE ERROR CONTROL OPERATOR

- Use the **error control operator (@)** to suppress error messages
  - The error control operator can be prepended to any expression although it is commonly used with built-in PHP functions that access external data sources

PHP 8 & ERROR CONTROL OPERATOR

- In **PHP 8.0**, the @ operator does not suppress certain types of errors that were silenced prior to PHP 8.0., including:
  - E_ERROR - Fatal run-time errors
  - E_CORE_ERROR - Fatal errors occurred in PHP's initial startup
  - E_COMPILE_ERROR - Fatal compile-time errors (from Zend engine)
  - E_USER_ERROR - User-triggered errors with trigger_error() function
  - E_RECOVERABLE_ERROR - Catchable fatal error
  - E_PARSE - Compile-time parse errors
- All of these errors halts the rest of the application from being run
- The @ operator in PHP 8 continue to silent warnings and notices

EXCEPTION HANDLING

- Since PHP 7, most errors are reported by throwing an exception (generating a special type of object that contains details of what caused the error and where)
- In PHP 8.1, the default error handling behaviour of the MySQLi has changed to throw an exception on errors

https://www.w3schools.com/php/php_exception.asp

EXCEPTION HANDLING

- Dealing with errors
  - **Exception handling** is used to change the normal flow of the code execution if a specified/exceptional error condition (called an exception) occurs
- This is what normally happens when an exception is triggered:
  - The current code state is saved
  - The code execution will switch to a predefined (custom) exception handler function
  - Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

https://www.w3schools.com/php/php_exception.asp

EXCEPTION HANDLING

- **try -** to facilitate the catching of potential exceptions, the code should be surrounded in a try block
- **catch -** defines how to respond to a thrown exception
- **throw -** throw an exception; halts execution of the current method and passes responsibility for handling the error to a catch statement
- **finally -** code within the finally block will always be executed after the try and catch blocks (regardless of whether an exception has been thrown or not)

https://www.w3schools.com/php/php_exception.asp

OPENING AND CLOSING A MYSQL CONNECTION

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

    try{
        $conn = new mysqli($servername, $username, $password);
        echo "<p>Connection successful</p>\n";
    }
    catch (mysqli_sql_exception $e)
    {
        die ($e->getCode(). ": " . $e->getMessage());
    }
    $conn->close();
?>
```

EXECUTING SQL STATEMENTS

- `query()` method is used for sending SQL statements to MySQL (performs a query on the database)

- The syntax is

```
$conn->query(query);
```

EXECUTING SQL STATEMENTS (CONTINUED)

- The `query()` method returns:
  - (1) For SQL statements that do not return results (`CREATE DATABASE` and `CREATE TABLE` statements) it returns a value of `TRUE` if the statement executes successfully

EXECUTING SQL STATEMENTS (CONTINUED)

- (2) For SQL statements that return results (`SELECT` and `SHOW` statements) the `query()` method returns a resultset object

- (3) The `query()` method throws an exception for any SQL statements that fail, regardless of whether they return results or not

CREATE/DROP A DATABASE

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

// Create connection
try {
    $conn = new mysqli($servername, $username, $password);
}
catch (mysqli_sql_exception $e){
    die("Connection failed: " . $e->getCode(). ": " . $e->getMessage());
}

?>
```

CREATE/DROP A DATABASE

```
<?php
include "inc_dbconnect.php";

// Create database
$sql = "CREATE DATABASE myDB2";
try {
    $conn->query($sql);
    echo "Database created successfully"; }
catch(mysqli_sql_exception $e) {
    die("Error creating database: " . $e->getCode(). ": " . $e->getMessage()); }

//Drop database
$sql = "DROP DATABASE myDB2";
try {
    $conn->query($sql);
    echo"Database deleted successfully";
 }
catch(mysqli_sql_exception $e){
    die( "Error deleting database: " . $e->getCode(). ": " . $e->getMessage());}

$conn->close();
?>
```

SELECTING A DATABASE

- If the connection function haven't included the database as argument then the database needs to be selected before use
- The syntax for the `select_db()` method is:

```
$conn->select_db(database);
```

- The method returns a value of `TRUE` if it successfully selects a database

CREATING AND DELETING TABLES - EXAMPLE

```
<?php
       include 'inc_dbconnect.php';
       $conn->select_db("mydb");
    // sql to create table
    $sql = "CREATE TABLE MyGuests1 (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        firstname VARCHAR(30) NOT NULL,
        lastname VARCHAR(30) NOT NULL,
        email VARCHAR(50),
        reg_date TIMESTAMP
    )";
    // sql to delete table:  $sql = "DROP TABLE MyGuests1";
    try {
        $conn->query($sql);
        echo "Table MyGuests1 created successfully";  }
    catch (mysqli_sql_exception $e) {
        die("Error creating table: " . $e->getCode(). ": " . $e->getMessage());}

$conn->close();
?>
```

CREATING AND DELETING TABLES

- To identify a field as a primary key in MySQL, the `PRIMARY KEY` keywords needs to be included in a field definition with the `CREATE TABLE` statement
- The `AUTO_INCREMENT` keyword is often used with a primary key to generate a unique ID for each new row in a table
- The `NOT NULL` keywords are often used with primary keys to require that a field include a value

CREATING AND DELETING TABLES (CONTINUED)

- `SHOW TABLES LIKE` command can be used to prevent code from trying to create a table that already exists

```
$sql = "SHOW TABLES LIKE 'MyGuests'";
```

ADDING, DELETING, AND UPDATING RECORDS

- To add multiple records to a database, use the `LOAD DATA` statement with the name of the local text file containing the records you want to add

```
$sql = "LOAD DATA INFILE 'myFile.txt'
        INTO TABLE MyGuests
        FIELDS TERMINATED BY '~'";
```

- To add records to a table, use the `INSERT` and `VALUES` keywords with the `query()` method

ADDING RECORDS – INSERT_ID

- To add records to a table, use the `INSERT` and `VALUES` keywords with the `query()` method

- The insert_id property returns the id (generated with AUTO_INCREMENT) used in the last query

```
$conn->insert_id;
```

- If the number is > max integer value, it will return a string
- It will return zero if there were no updates or no AUTO_INCREMENT field

ADDING RECORDS – EXAMPLE

```
$sql = "INSERT INTO
      myguests(firstname,lastname, email)
      VALUES('Elena', 'Vlahu', 'evg@gmail.com')";
try {
     $conn->query($sql);
     $GuestID = $conn->insert_id;
     echo "Your ID is $GuestID <br />";
}
catch (mysqli_sql_exception $e) {
      echo "Unable to insert the the record";
}
```

ADDING, DELETING, AND UPDATING RECORDS

- To update records in a table, use the `UPDATE` statement
- The `UPDATE` keyword specifies the name of the table to update
- The `SET` keyword specifies the value to assign to the fields in the records that match the condition in the `WHERE` clause

```
$sql = "UPDATE MyGuests SET email='" . $email
                                    . "' WHERE id=" . $id ;
try {
    $conn->query($sql);
    echo "Record updated successfully <br />"; }
catch (mysqli_sql_exception $e) {
    die("Error in updating: " . $e->getMessage() );  }
```

ADDING, DELETING, AND UPDATING RECORDS

- To delete records in a table, use the `DELETE` statement with the `query()` method
- Omit the `WHERE` clause to delete all records in a table

RETURNING INFORMATION ON AFFECTED ROWS

- With queries that modify tables (`INSERT`, `UPDATE`, and `DELETE` queries), the `affected_rows` property can be used to determine the number of affected rows

RETURNING INFORMATION ON AFFECTED ROWS - EXAMPLE

```
$sql = "DELETE FROM MyGuests where id=1";

try {
   $conn->query($sql);
   echo $conn->affected_rows .
            " row(s) were deleted.<br />";
}
catch (mysqli_sql_exception $e) {
   echo "error" . $e->getMessage();
}
```

USING THE `INFO` PROPERTY

- For queries that add or update records, or alter a table's structure, use the `info` property to return information about the last query that was executed on the database connection
  - `INSERT INTO...SELECT...`
  - `INSERT INTO...VALUES (...),(...),(...)`
  - `LOAD DATA INFILE ...`
  - `ALTER TABLE ...`
  - `UPDATE`
  - For any queries that do not match one of these formats, the `$conn->info` returns an empty string
- The `$conn->info` returns the string including number of operations for various types of actions, depending on the type of query

USING THE INFO PROPERTY - EXAMPLE

```
$sql = "INSERT INTO MyGuests " .
     " (firstname, lastname, email) " .
     " VALUES " .
     " ('Tom', 'Hon', 'tt@gmail.com'), " .
     " ('Tara', 'Davis', 'tara@gmail.com'), " .
     " ('Kate', 'Smith', 'kate@gmail.com')";

try {
   $conn->query($sql);
   echo "Successfully added the records.<br />";
   echo $conn->info;
}
catch (mysqli_sql_exception $e) {
   die("Unable to execute the query" .
         $e->getCode(). ": " . $e->getMessage());
}
```

WORKING WITH QUERY RESULTS

Method	Description
`fetch_row()`	Fetches one row of data from the result set and returns it as an enumerated array (each subsequent call to this function will return the next row within the result set)
`fetch_assoc()`	Fetches one row of data from the result set and returns it as an associative array (each subsequent call to this function will return the next row within the result set)
`data_seek(position)`	Moves the result pointer to a specified row in the result set
`fetch_all(MYSQL_ASSOC|MYSQL_NUM)`	Returns an array of associative or numeric arrays holding result rows

RETRIEVING RECORDS INTO AN INDEXED ARRAY

- The primary difference between the `fetch_assoc()` and the `fetch_row()` method is that the `fetch_assoc()` returns the fields into an associative array and uses each field name as the array key
- The both function return NULL when there are no records in the resultset

```
while ($row = $result->fetch_assoc())
{…};
```

CLOSING QUERY RESULTS

- When finished working with query results retrieved with the `query()` method, the

```
free_result(), free(), close()
```

methods can be used to close/free the memory associated with the result set

ACCESSING QUERY RESULT INFORMATION

- The `num_rows` property returns the number of rows in a query result set
- The `field_count` property returns the number of fields in a query result set

EXAMPLE – NEWSLETTER SUBSCRIBERS

PREPARED STATEMENTS AND BOUND PARAMETERS

- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency
- Prepare
  - an SQL statement template is created and sent to the database
  - parameters - certain values are left unspecified (by adding "?")

```
prepare(sqlstat)
bind_param(argType,[arguments])
bind_result(mixed &$var1 [, mixed &$... ] )
```

  - Argument type can be
    - i – integer, d – double, s – string, b – BLOB (Binary large object)

PREPARED STATEMENTS AND BOUND PARAMETERS

- The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
- Execute
  - at a later time, the application binds the values to the parameters, and the database executes the statement
  - the application may execute the statement as many times as it wants with different values

```
execute()
fetch()
get_result()
```

PREPARED STATEMENTS AND BOUND PARAMETERS

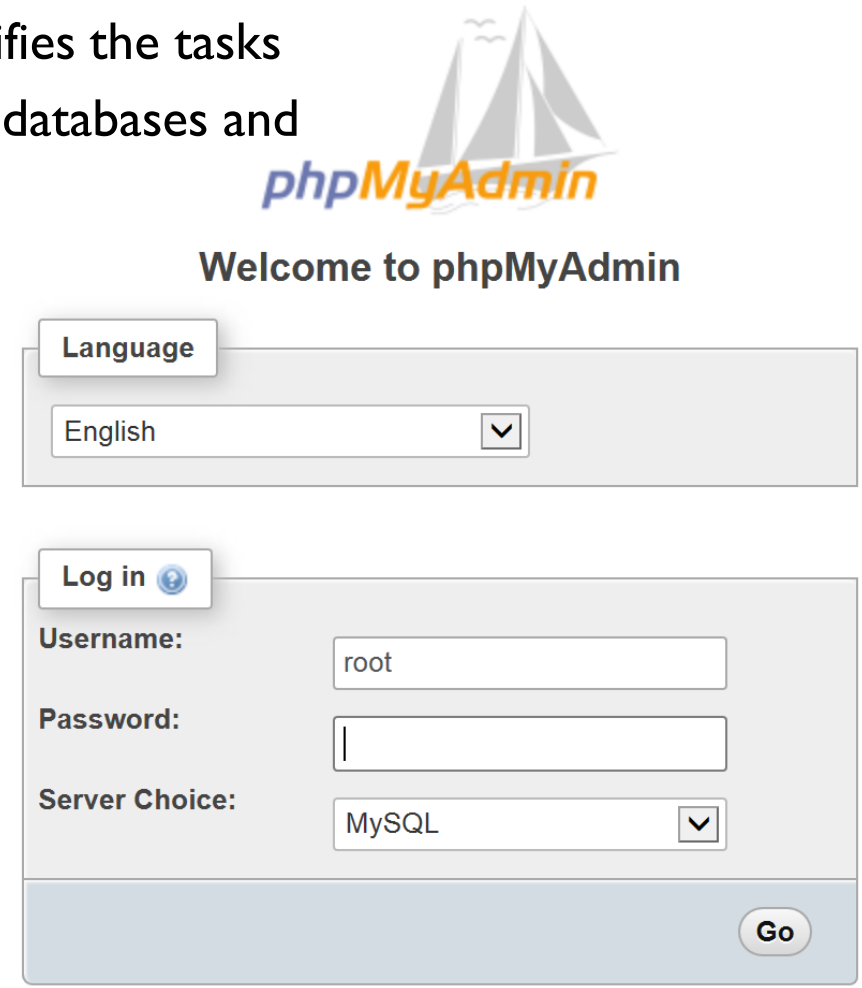
- Compared to executing SQL statements directly, prepared statements have three main advantages:
  - Prepared statements reduce parsing time as the preparation on the query is done only once
  - Bound parameters minimize bandwidth to the server as only the parameters are send each time (not the whole query)
  - Prepared statements are very useful against SQL injections

PREPARED STATEMENTS AND BOUND PARAMETERS - EXAMPLE

```
// prepare and bind
    $stmt = $conn->prepare("INSERT INTO MyGuests
                                    (firstname, lastname, email)
                                            VALUES (?, ?, ?)");
    $stmt->bind_param("sss", $fname, $lname, $email);
    // set parameters and execute
    $fname = "John";
    $lname = "Doe";
    $email = "john@example.com";
    $stmt->execute();
    $stmt->close();
    $conn->close();
```

WORKING WITH PHPMYADMIN

- The **phpMyAdmin** graphical tool simplifies the tasks associated with creating and maintaining databases and tables



WORKING WITH PHPMYADMIN