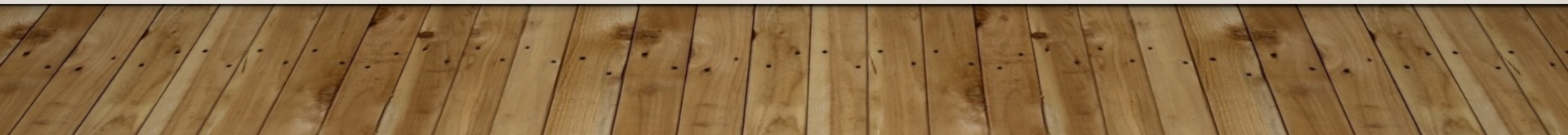
ISIT307 - WEB SERVER PROGRAMMING

LECTURE 5.1 – 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

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 opened with the `mysqli_connect()` function
- `mysqli_connect(..)` returns a resource representing the connection to the MySQL server if connection is successful or FALSE for failure

OPENING AND CLOSING A MYSQL CONNECTION

- The syntax for the `mysqli_connect()` function is:

```
$connection = mysqli_connect(host,username,password
[,dbname,port,socket]);
```

- 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)

OPENING AND CLOSING A MYSQL CONNECTION

- The database connection is assigned to the `$conn` variable

```
$conn = mysqli_connect("localhost","my_user",
                       "my_password" [,"my_db"]);
```

- The database connection can be closed using the `mysqli_close()` function

```
mysqli_close(connection);
```

  - The function returns TRUE or FALSE

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
- The `mysqli_connect_errno()` and `mysqli_connect_error()` functions return the error code and description from the attempt to connect to the database;
- `die(`*`error functions`*`)` is syntax used as a short way of writing the code that will display the error and exit the script immediately

REPORTING MYSQL ERRORS

- The `mysqli_errno()` and `mysqli_error()` functions return the results of the previous `mysqli*()` function
- For this functions the variable representing the database connection should be sent as input argument

```
mysqli_errno(connection);
mysqli_error(connection);
```

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{
        $conn = mysqli_connect($servername, $username, $password);
        echo "<p>Connection successful</p>\n";
    }
    catch (mysqli_sql_exception $e)
    {
        die("Connection failed: ". mysqli_connect_errno(). " - " .
                                            mysqli_connect_error());
    }
    //pre PHP8.1 - if (!$conn) or (mysqli_connect_errno()!=0)
    mysqli_close($conn);
?>
```

EXECUTING SQL STATEMENTS

- `mysqli_query()` function is used for sending SQL statements to MySQL

- The syntax for the `mysqli_query()` function is:

```
mysqli_query(connection, query);
```

EXECUTING SQL STATEMENTS (CONTINUED)

- The `mysqli_query()` function returns one of three values:
  - (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 `mysqli_query()` function returns a resultset (with result pointer) that represents the query results
  - A **result pointer** is a special type of variable that refers to the currently selected row in a resultset

EXECUTING SQL STATEMENTS (CONTINUED)

- (3) The `mysqli_query()` function returns a value of `FALSE`/throw an exception for any SQL statements that fail, regardless of whether they return results or not

CREATING A DATABASE

```
<?php
$servername = "localhost"; $username = "root";  $password = "";
// Create connection
try{
      $conn = mysqli_connect($servername, $username, $password); }
catch (mysqli_sql_exception $e) {
      die("Connection failed:" . mysqli_connect_errno() . "=" .
                                                mysqli_connect_error()); }
//*die() – exit the script and everything after is not executed
// Create database
$sql = "CREATE DATABASE myDB";
try {
    mysqli_query($conn, $sql);  //pre PHP8.1 - if (mysqli_query($conn, $sql))
    echo "Database created successfully"; }
catch(mysqli_sql_exception $e) {
    die("Error creating database: " . mysqli_error($conn)); }
mysqli_close($conn);  ?>
```

DELETING DATABASES

- To delete a database, use the `DROP DATABASE` statement with the `mysqli_query()` function

```
//Drop database
$sql = "DROP DATABASE myDB";
try {
    mysqli_query($conn, $sql);
    echo"Database deleted successfully";  }
catch(mysqli_sql_exception $e){
    die( "Error deleting database: " .
          mysqli_errno($conn). " - " . mysqli_error($conn));
}
```

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 `mysqli_select_db()` function is:

```
mysqli_select_db(connection, database);
```

- The function returns a value of `TRUE` if it successfully selects a database
- For simplicity and security purposes, you may choose to use an include file to connect to the MySQL server and select a database

CREATING AND DELETING TABLES

- `CREATE TABLE` statement with the `mysqli_query()` function can be used to create a new table
- if the connection function haven't included the database as argument `mysqli_select_db()` function should be used before executing the `CREATE TABLE` statement to verify that you are in the right database

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 - EXAMPLE

```
try {
       $conn = mysqli_connect($servername, $username, $password, $dbname); }
catch ( mysqli_sql_exception $e) {
       die("Connection failed: " . mysqli_connect_error()); }

$sql = "CREATE TABLE MyGuests (
              id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
              firstname VARCHAR(30) NOT NULL,
              lastname VARCHAR(30) NOT NULL,
              email VARCHAR(50))";
try {
       mysqli_query($conn, $sql);
      echo "Table MyGuests created successfully";  }
catch (mysqli_sql_exception $e){
        die("Error creating table: " . mysqli_error($conn)); }
mysqli_close($conn);
```

CREATING AND DELETING TABLES

- To delete a table, use the `DROP TABLE` statement with the `mysqli_query()` function

```
$sql = "DROP TABLE MyGuests";
try {
  mysqli_query($conn, $sql);
  echo "Table MyGuests1 deleted successfully";
}
catch (mysqli_sql_exception $e) {
   die("Error dropping table:” . mysqli_error($conn));
}
```

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 RECORDS – MYSQLI_INSERT_ID()

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

- The mysqli_insert_id() function returns the id (generated with AUTO_INCREMENT) used in the last query

```
mysqli_insert_id(connection);
```

- If the number is > max integer value, it will return a string
- The function will returns zero if there were no update or no AUTO_INCREMENT field

ADDING RECORDS – EXAMPLE

```
$sql = "INSERT INTO
      MyGuests(firstname,lastname, email)
      VALUES('Elena', 'Vlahu', 'evg@gmail.com')";
try {
     mysqli_query($conn, $sql);
     $GuestID = mysqli_insert_id($conn);
     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=" . $id ;
try {
    mysqli_query($conn, $sql);
    echo "Record updated successfully <br />"; }
catch (mysqli_sql_exception $e) {
    die("Error in updating: " . mysqli_error($conn) );  }
```

ADDING, DELETING, AND UPDATING RECORDS

- To delete records in a table, use the `DELETE` statement with the `mysqli_query()` function
- 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 `mysqli_affected_rows(`*`connection`*`)` function can be used to determine the number of affected rows

RETURNING INFORMATION ON AFFECTED ROWS - EXAMPLE

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

try {
   mysqli_query($conn, $sql);
   echo mysqli_affected_rows($conn) . "
            row(s) were deleted.<br />"; }
catch (mysqli_sql_exception $e) {
   echo "error" . mysqli_error($conn);  }
```

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 '~'";
```

USING THE `MYSQLI_INFO()` FUNCTION

- For queries that add or update records, or alter a table's structure, use the `mysqli_info(`*`connection`*`)` function 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 `mysqli_info()` function returns an empty string
- The `mysqli_info()` function returns the number of operations for various types of actions, depending on the type of query

USING THE MYSQLI_INFO() FUNCTION - 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 {
   mysqli_query($conn, $sql);
   echo "Successfully added the records.<br />";
   echo mysqli_info($conn);
}
catch (mysqli_sql_exception $e) {
   die("Unable to execute the query" .
         mysqli_errno($conn) . mysqli_error($conn));
}
```

WORKING WITH QUERY RESULTS

Function	Description
`mysqli_data_seek($Result, position)`	Moves the result pointer to a specified row in the resultset
`mysqli_fetch_array($Result, MYSQL_ASSOC | MYSQL_NUM | MYSQL_BOTH)`	Returns the fields in the current row of a resultset into an indexed array, associative array, or both, and moves the result pointer to the next row
`mysqli_fetch_assoc($Result)`	Returns the fields in the current row of a resultset into an associative array and moves the result pointer to the next row
`mysqli_fetch_row($Result)`	Returns the fields in the current row of a resultset into an indexed array and moves the result pointer to the next row

RETRIEVING RECORDS INTO AN INDEXED ARRAY

- The primary difference between the `mysqli_fetch_assoc()` function and the `mysqli_fetch_row()` function is that the `mysqli_fetch_assoc()` function 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 (($Row = mysqli_fetch_assoc($qRes)) != FALSE)
{…};
```

CLOSING QUERY RESULTS

- When finished working with query results retrieved with the `mysqli_query()` function, the `mysqli_free_result(`*`queryResults`*`)` function can be used to close the resultset
- The `mysqli_free_result()` function has one parameter - the variable containing the resultset returned by the `mysqli_query()` function

ACCESSING QUERY RESULT INFORMATION

- The `mysqli_num_rows(`*`queryResults`*`)` function returns the number of rows in a query result
- The `mysqli_num_fields(`*`queryResults`*`)` function returns the number of fields in a query result

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 "?")

```
mysqli_prepare(conn, sqlstat)
mysqli_stmt_bind_param(preparedS, argType,[arguments])
mysqli_stmt_bind_results(preparedS, 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

```
mysqli_stmt_execute (preparedS)
mysqli_stmt_fetch (preparedS)
mysqli_stmt_get_result(preparedS)
```

PREPARED STATEMENTS AND BOUND PARAMETERS

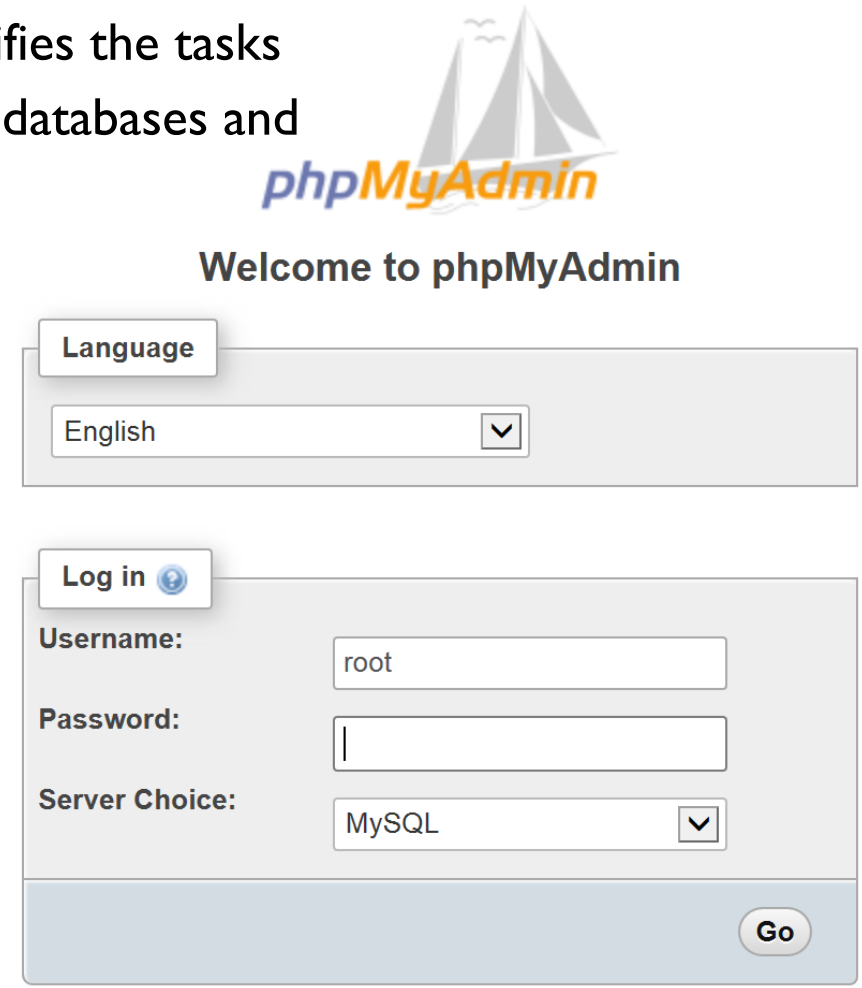
- Compared to executing SQL statements directly, prepared statements have three main advantages:
  - Prepared statements reduce parsing time as the preparation on the query is done only once
  - Bound parameters minimize bandwidth to the server as only the parameters are send each time (not the whole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

PREPARED STATEMENTS AND BOUND PARAMETERS - EXAMPLE

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

WORKING WITH PHPMYADMIN

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



WORKING WITH PHPMYADMIN