



MYSQL + PHP

Update, Delete, Join, and PHP Integration

COS216

AVINASH SINGH

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF PRETORIA

MYSQL – RECAP

- Remember from the previous lecture this is how we created tables using SQL.
- Also remember that you must choose the correct data types because structured databases cannot easily be restructured and changed.
- It is also important that your data type matches the data you store, thus maximizing performance and storage.

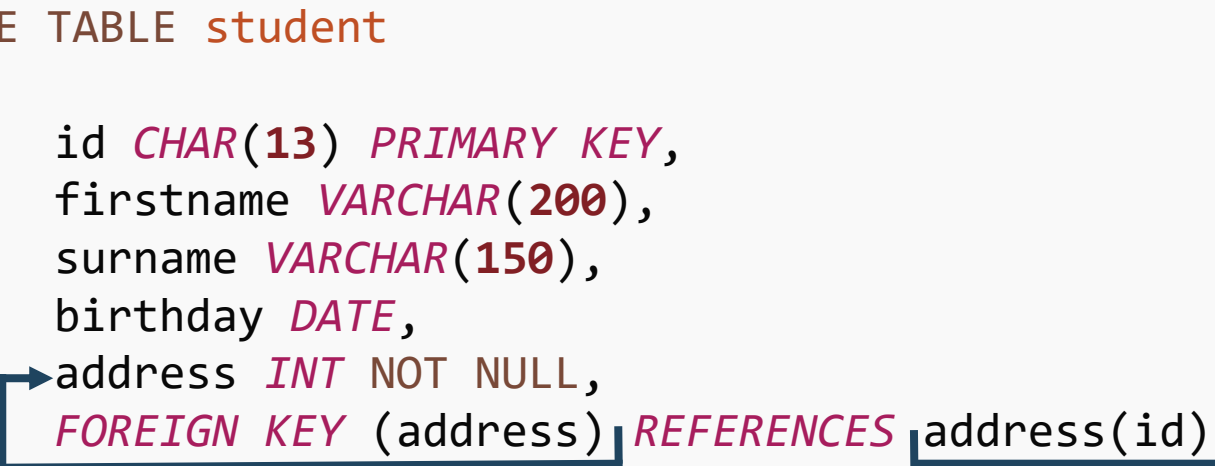
```
CREATE TABLE address
(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    street VARCHAR(200) NOT NULL,  
    city VARCHAR(100) NOT NULL,  
    zipcode CHAR(4)  
);
```

```
CREATE TABLE student
(  
    id CHAR(13) PRIMARY KEY,  
    firstname VARCHAR(200),  
    surname VARCHAR(150),  
    birthday DATE,  
    address INT NOT NULL,  
    FOREIGN KEY (address) REFERENCES address(id)  
);
```

MYSQL – RECAP

```
CREATE TABLE address
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    street VARCHAR(200) NOT NULL,
    city VARCHAR(100) NOT NULL,
    zipcode CHAR(4)
);
```

```
CREATE TABLE student
(
    id CHAR(13) PRIMARY KEY,
    firstname VARCHAR(200),
    surname VARCHAR(150),
    birthday DATE,
    address INT NOT NULL,
    FOREIGN KEY (address) REFERENCES address(id)
);
```



MYSQL – UPDATE

- Update and **existing entry**

```
UPDATE student  
SET firstname="Notsatoshi", surname="Notnakamoto"  
WHERE id = "9401234052087";
```

- Update the values of specific columns
- Update the values of a row with specific attributes (WHERE), this allows for more precision in the data you want to update, for the example above we do not want the update to happen for all data in the table therefore using the WHERE clause we are able to limit or control what updates should happen, in this case, it's based on the id.

MYSQL – DELETE

- Delete an **existing entry**

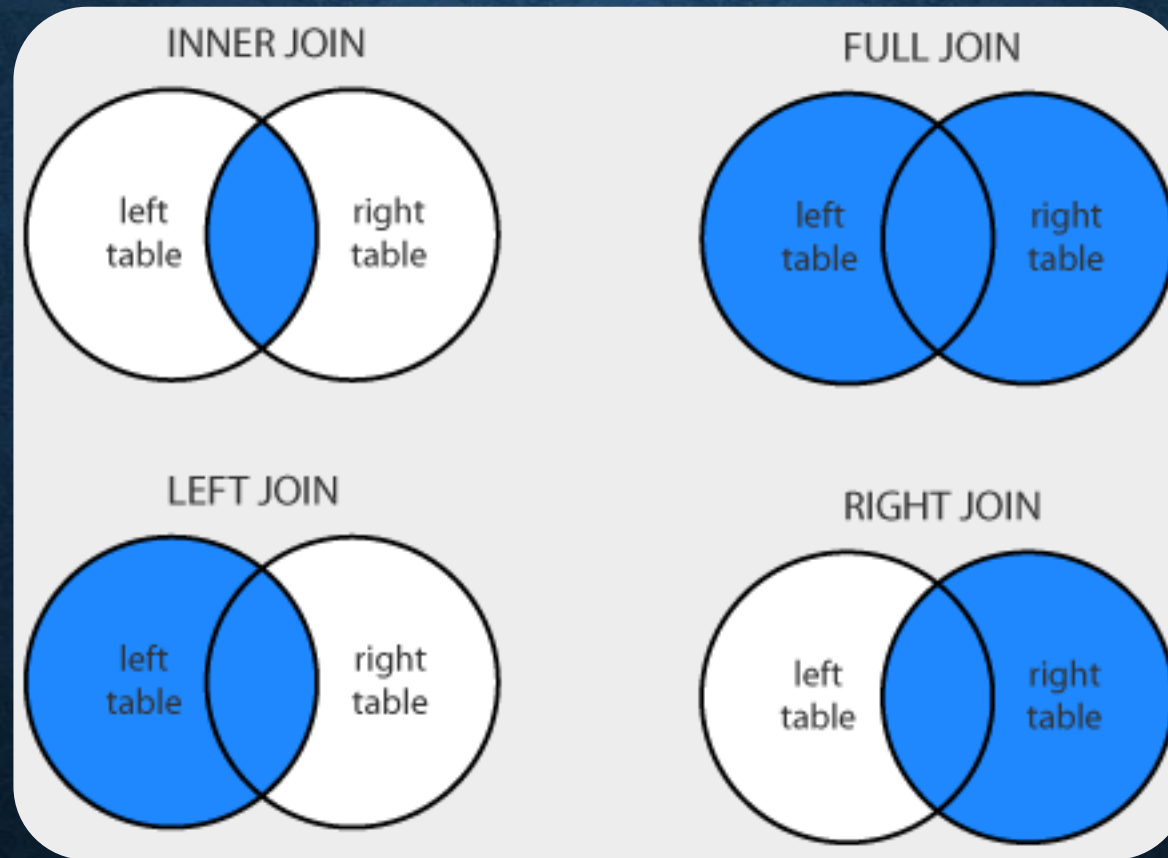
```
DELETE FROM student  
WHERE id = "9401234052087";
```

- Delete an entire row according to specific attributes (WHERE).
- **NOTE:** the **WHERE** clause is important in both **UPDATE** and **DELETE**

MYSQL – JOIN

- Joins are used to combine different tables
- These tables are typically related with regard to foreign keys
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in the either left or right table
- More info at https://www.w3schools.com/sql/sql_join.asp

MYSQL - JOIN



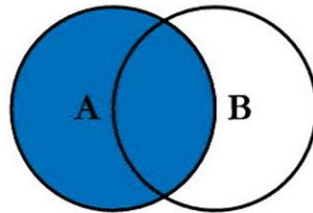
MYSQL – JOIN

```
SELECT student.firstname, address.city  
FROM student  
INNER JOIN address ON student.address = address.id;
```

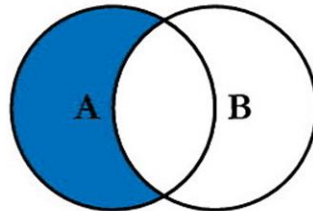
- Retrieve students first names and the city they live in
- Join the two tables on the address ID (student and address)
- The address ID is the primary key in the address table and a foreign key in the student table
- Since INNER JOIN is used the data Selected would be the intersection of the 2 tables.

MYSQL - JOIN

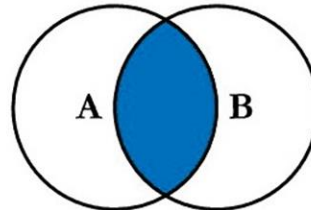
SQL JOINS



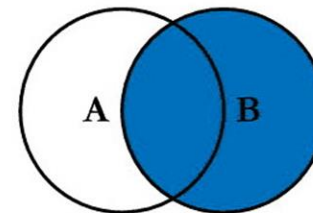
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



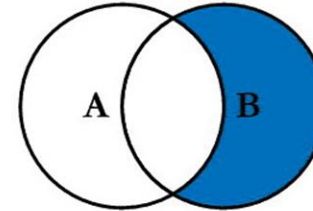
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



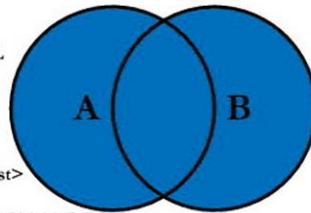
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



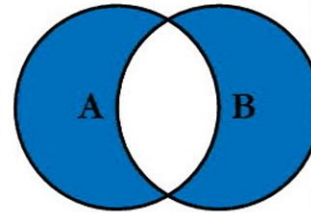
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

MYSQL – PHP INTEGRATION

- PHP has various extensions to access databases through libraries
- Amongst others, two main extensions can be used for MySQL
 - MySQLi (easier to use) and PDO (more features)
- The MySQL PHP extension was deprecated and replaced with MySQLi (improved MySQL) – This method can follow a procedural approach or object orient approach
- The PDO (PHP Data Objects) supports multiple databases and provides better object manipulation – This method follows an object oriented approach

MYSQL – PHP INTEGRATION

Feature	MySQLi	PDO
MySQL Support	Yes	Yes
Other Non-MySQL Database Support	No	Yes (19 others)
Procedural API	Yes	No
Object-Oriented API	Yes	Yes
Prepared Statements	Yes	Yes

MYSQL – SINGLETON

- Write a Singleton wrapper class for the database (eg: DataManager.php)
- Automatically restrict to a single database connection per request
- Design in the following way
 - A user sends a GET/POST request to the server
 - If the request is received, create a singleton instance which automatically establish a connection to the database (remember a singleton only allows for one instance)
 - Execute various queries using the singleton, depending on the GET/POST request
 - If the GET/POST request is completed, let the singleton run out of scope (or delete it manually). On destruction, disconnect from the database

MYSQL – SINGLETON

```
<?php
class Database
{
    public static function instance()
    {
        static $instance = null; // remember that this only ever gets called once
        if($instance === null) $instance = new Database();
        return $instance;
    }
    private function __construct() { /* Connect to the database */ }
    public function __destruct() { /* Disconnect from the database */ }
    public function addUser($username){ /* Add to the database */ }
    public function retrieveUser($username){ /* Retrieve from the database */ }
}

$instance = Database::instance();
$instance->addUser("satoshi");
$user = $instance->retrieveUser("satoshi");
?>
```

PHP - LOCALHOST

- If the MySQL server is on the same machine as the Apache server (like Wheatley)
 - You can use **localhost** for the connection instead of the **FQDN**
 - Most webserver (80%+) run on Linux
 - Linux often does not have **localhost** as a DNS entry, rather use **127.0.0.1** to be safe
- If the MySQL server is not on the same server as Apache, use the **FQDN** or **IP address** of the MySQL server to establish a connection
- To connect on Wheatley: “**127.0.0.1**” or “**wheatley.cs.up.ac.za**”

PHP - MYSQLI

```
<?php    // OBJECT ORIENTED APPROACH (Preferred*)
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username,
$password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn-
>connect_error);
}
echo "Connected successfully";
?>
```

```
<?php // PROCEDURAL APPROACH
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername,
$username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}
echo "Connected successfully";
```

PHP - MYSQLI

```
<?php
```

```
    $host = "127.0.0.1";  
    $username = "username";  
    $password = "password";
```

```
    // Create a new connection and select database using object oriented  
    $connection = new mysqli($host, $username, $password);
```

```
    // Check if connected
```

```
    // Might fail if password is incorrect, server is unreachable, etc
```

```
    if($connection->connect_error)  
        die("Connection failure: " . $connection->connect_error);
```

```
    else
```

```
    {
```

```
        $connection->select_db("216_database");
```

```
        echo "Connection success";
```

```
    }
```

```
    // Do some queries and once done close the connection
```

```
    $connection->close();
```

```
?>
```


PHP - MYSQLI

```
<?php
    // Connect to the database first
    // Insert a new row into the database
    $query = "INSERT INTO address (street, city, zipcode)
              VALUES ('23 Boom Str', 'Pretoria', '0001')";

    if($connection->query($query) === true)
        echo "New record created successfully";
    else
        echo "Error: " . $query . "<br>" . $connection->error;
    // Close the database connection
?>
```

- Remember that SQL queries have to be enclosed within Quotes (") because PHP does not natively know SQL. Hence why an extension is used. Also, look carefully at how the query is structured paying special attention to the (') single quote

PHP - MYSQLI

```
<?php
    // Connect to the database first
    $query = "SELECT street, city, zipcode FROM address";
    $result = $connection->query($query);

    if($result->num_rows > 0) // check if there is data returned
    {
        // Iterate over all returned rows and print the values
        while($row = $result->fetch_assoc()) //fetch associative arrays
        {
            echo $row["street"] . ", " . $row["city"]
                . ", " . $row["zipcode"] . "<br>";
        }
    }
    else echo "No results";
    // Close the database connection
?>
```


PHP - PDO

Remember since this is an OO approach you have to use try-catch statements as it will throw exceptions when there are errors.

```
<?php
    $host = "127.0.0.1";
    $username = "username";
    $password = "password";
    try
    {
        $connection = new PDO("mysql:host=$host;dbname=216_database",
                               $username, $password);

        // Set PDO error mode to exception
        $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        echo "Connection success";
        // Close the connection by letting garbage collection delete it
        $connection = null;
    }
    catch(PDOException $error)
    {
        echo "Connection failure: " . $error->getMessage();
    }
?>
```

PHP - PDO

```
<?php
    try
    {
        // Connect to the database first

        $query = "SELECT street, city, zipcode FROM address";
        $connection->exec($query);
        echo "Executed Query";

        // Close the database connection
    }
    catch(PDOException $error)
    {
        echo "Error: " . $error->getMessage();
    }
?>
```


PHP - PDO

```
<?php
    try
    {
        // Connect to the database first
        $query = "SELECT street, city, zipcode FROM address";
        $statement = $connection->prepare($query);
        $statement->execute();
        // Convert results to associative array
        $result = $statement->setFetchMode(PDO::FETCH_ASSOC);
        foreach(new TableRows(new RecursiveArrayIterator(
            $statement->fetchAll())) as $k => $v)
        {
            echo $v . ", ";
        }
        // Close the database connection
    }
    catch(PDOException $error)
        echo "Error: " . $error->getMessage();
?>
```

MORE RESOURCES

- https://www.w3schools.com/php/php_mysql_intro.asp

SINGLETON CLASS EXAMPLE

```
<?php
class Database {
    public static function instance() {
        static $instance = null; // remember this only ever gets called once, why?
        if($instance === null)
            $instance = new Database();
        return $instance; }
    private function __construct() { /* Connect to the database */ }
    public function __destruct() { /* Disconnect from the database */ }
    public function addUser($username){ /* Add to the database */ }
    public function retrieveUser($username){ /* Retrieve from the database */ }
}

$user = $_POST['username']; // If the request is FORM URL encoded
// if the request is JSON data e.g. {'username': 'Satoshi'}
$json = file_get_contents('php://input');
$data = json_decode($json); // Converts it into a PHP object
$user = $data['username'];
$instance = Database::instance();
$instance->addUser($user);
?>
```

SINGLETON ISSUES?

- Remember each time a script completes execution the data gets destroyed, hence (Stateless HTTP)
- This means that even concurrent requests don't share data. So an instance would be created every time the script is executed.
- So why use Singletons?
 - - Added benefits since it is OOP
 - - Customized functionality is relatively easy
 - - Don't need to worry about closing connections, since it will be called on `__destruct()`
 - - Reusability!!!! Provides a constant interface, which is what an API requires.

