Selected Topics in CS

PHP - Advanced Concept

2020/2021(2013) AC

Contents

- OOP with PHP
  - OOP concept
  - Class and Object
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Overloading
  - Overriding
  - Abstraction
  - …
- Database Programming

PHP - OOP

OOP

- An approach to software development
- Models application around real world entity
  - Students, Courses, etc.
- A class defines
  - Properties
  - Methods
- An object is an occurrence of a class

OOP …

- The basic components of object orientation
  - Object oriented analysis
    - functionality of the system
  - Object oriented designing
    - architecture of the system
  - Object oriented programming
    - implementation of the application

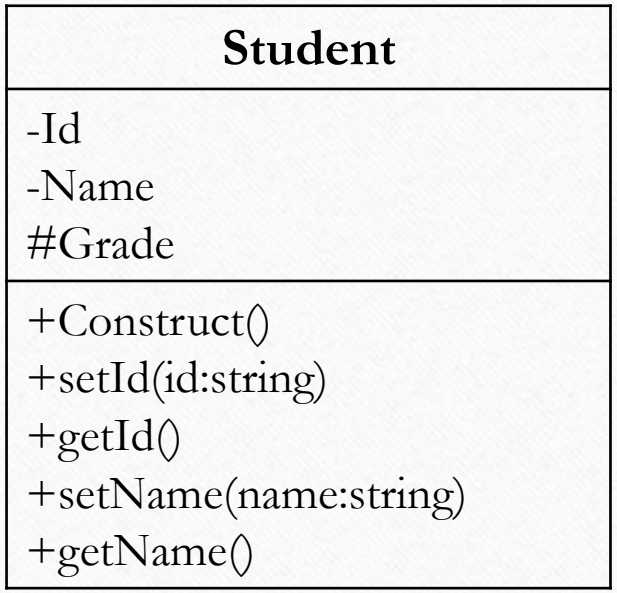
OOP principle

- **Encapsulation**
  - Hiding implementation details
  - Expose only the method
  - Reduce software complexity
  - Protect internal state of an object
  - Implementation can be changed with out code breaking uses the class
- **Inheritance**
  - Relationship between classes-> Parent –child
  - Re-usability
- **Polymorphism**
  - Single form – many different implementation
  - Simplify maintaining application – more extendable

UML

- Unified Modeling Language
- is a technique used to
  - design and
  - document object oriented systems
- UML produces a number of documents
  - focus on **class** diagram
    - very important to object oriented programming

UML …



- Class diagram component
  - >Upper box contains the class name
  - >Middle box contains the class variables
  - >Lower box contains the class methods
  - >Minus (-) sign means private scope
  - >Plus (+) sign means public scope
  - >Hash (#) sign means protected scope

OOP in PHP

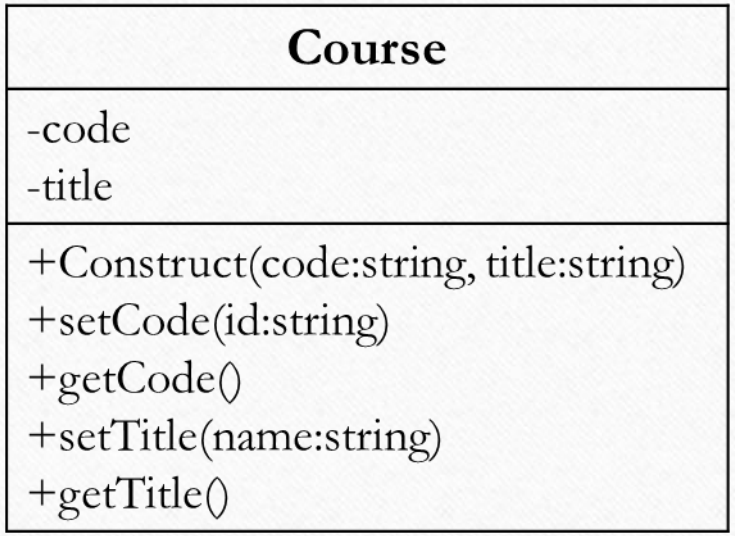
- **Encapsulation**
  - via the use of "get" and "set" methods etc.
- **Inheritance**
  - via the use of extends keyword
- **Polymorphism**
  - via the use of implements keyword

Class in PHP

- The class keyword is used to define a class in PHP
- class name should start with a letter
- class name cannot be a PHP reserved word
- class name cannot contain spaces
- Example:
  - a class for representing Course
  - Properties
    - Code, Title, Credit, …
  - Methods
    - getCode, setCode, getTitle, setTitle, …

Class in PHP …

Class Diagram



Class Implementation

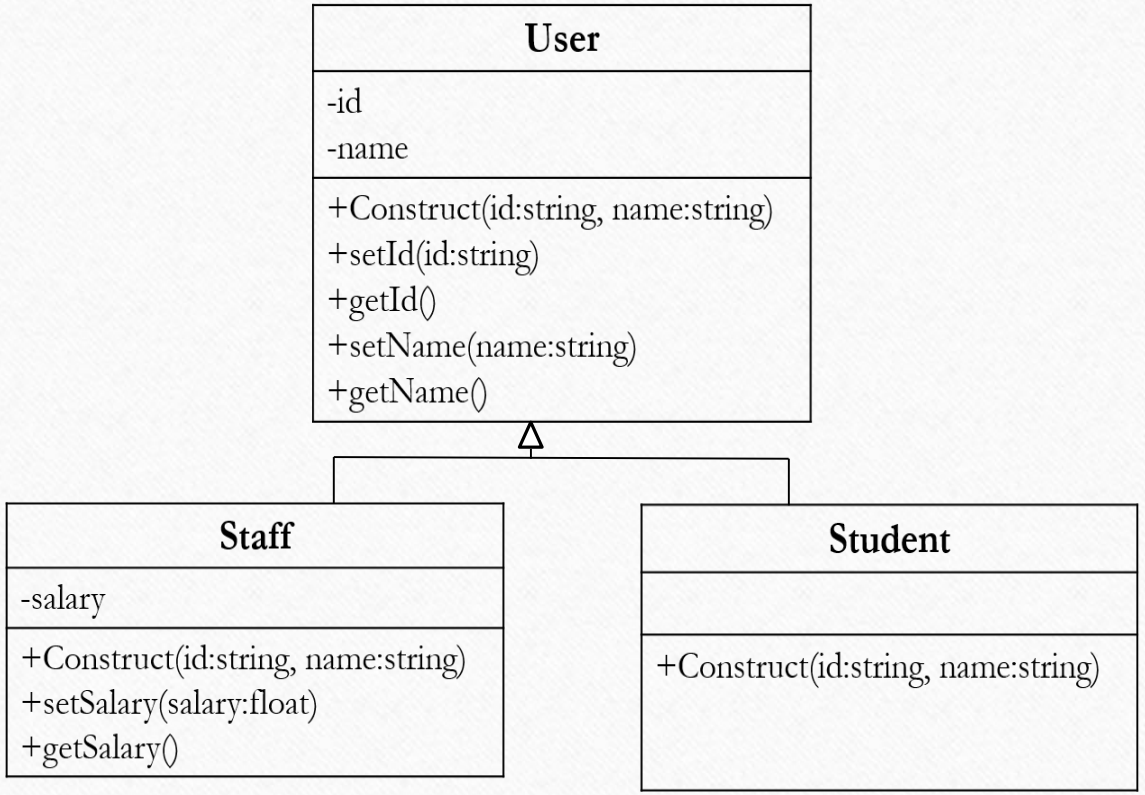
```php
<?php
class Course
{
    private $code;
    private $title;
    public function __construct($code, $title) {
        $this->code = $code;
        $this->title = $title;
    }
    public function getCode() {
        return $this->code;
    }
    public function setCode($code){
        $this->code = $code;
    }
    //.....
}
?>
```

Class in PHP …

- "private $code, $title"
  - the variables cannot be accessed directly outside the class (Encapsulation)
- "public function __construct($code…)"
  - is the php constructor method.
  - This function is called whenever an instance of the class has been created
- "public function get…()"
  - is the method used to access the code or title value (Encapsulation)
- "public function set…()"
  - is the method used to set the code or title value (Encapsulation)

Inheritance in PHP

- implemented with extends key word
- Student and Staff inherit from User class
- Staff inherits from the User class and defines its own variable and methods too



Inheritance in PHP …

User Class

```php
<?php
class User
{
    private $id;
    private $name;
    public function __construct($id, $name) {
        $this->id=$id;
        $this->name = $name;
    }
    public function setId($id) {
        $this->id = $id;
    }
    public function getId() {
        return $this->id;
    }
    public function setName($name) {
        $this->name = $name;
    }
    public function getName(){
        return $this->name;
    }
}
?>
```

Inheritance in PHP …

Staff Class

```php
<?php
class Staff extends User {
    private $salary;
    public function __construct($id, $name)    {
        parent::__construct($id, $name);
    }
    public function setSalary($salary)    {
        $this->salary = $salary;
    }
    public function getSalary()    {
        return $this->salary;
    }
}
?>
```

Student Class

```php
<?php
class Student extends User
{
    public function __construct($id, $name)
    {
        parent::__construct($id, $name);
    }
}
?>
```

Inheritance in PHP …

- "class … extends User"
  - makes the Staff and Student use methods from the User class (Inheritance).

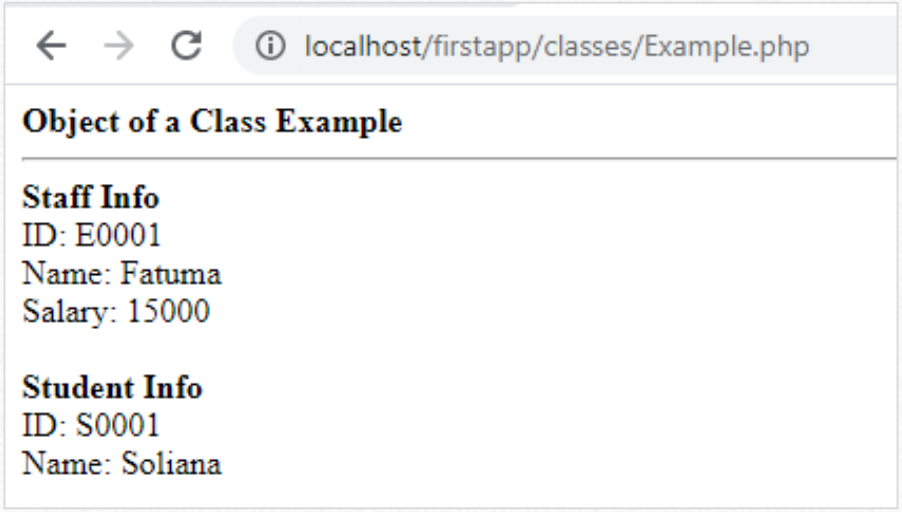
Example - Create object of the class

- Create the application that uses our classes

```
<?php
    require 'User.php';
    require 'Staff.php';
    require 'Student.php';
    echo '<b>Object of a Class Example</b> <br> <hr/>';
    $staff  = new Staff('E0001', 'Fatuma');
    $staff->setSalary(15000);
    $student = new Student('S0001', 'Soliana');
    echo '<b>Staff Info</b> <br>';
    echo 'ID: ' . $staff->getId() . '<br>Name: ' . $staff->getName() . '<br>Salary: '. $staff->getSalary(). '<br><br>';
    echo '<b>Student Info</b> <br>';
    echo 'ID: ' . $student->getId() . '<br>Name: ' . $student->getName();
?>
```

Example – Run the application

- Let's now view the application in a web browser



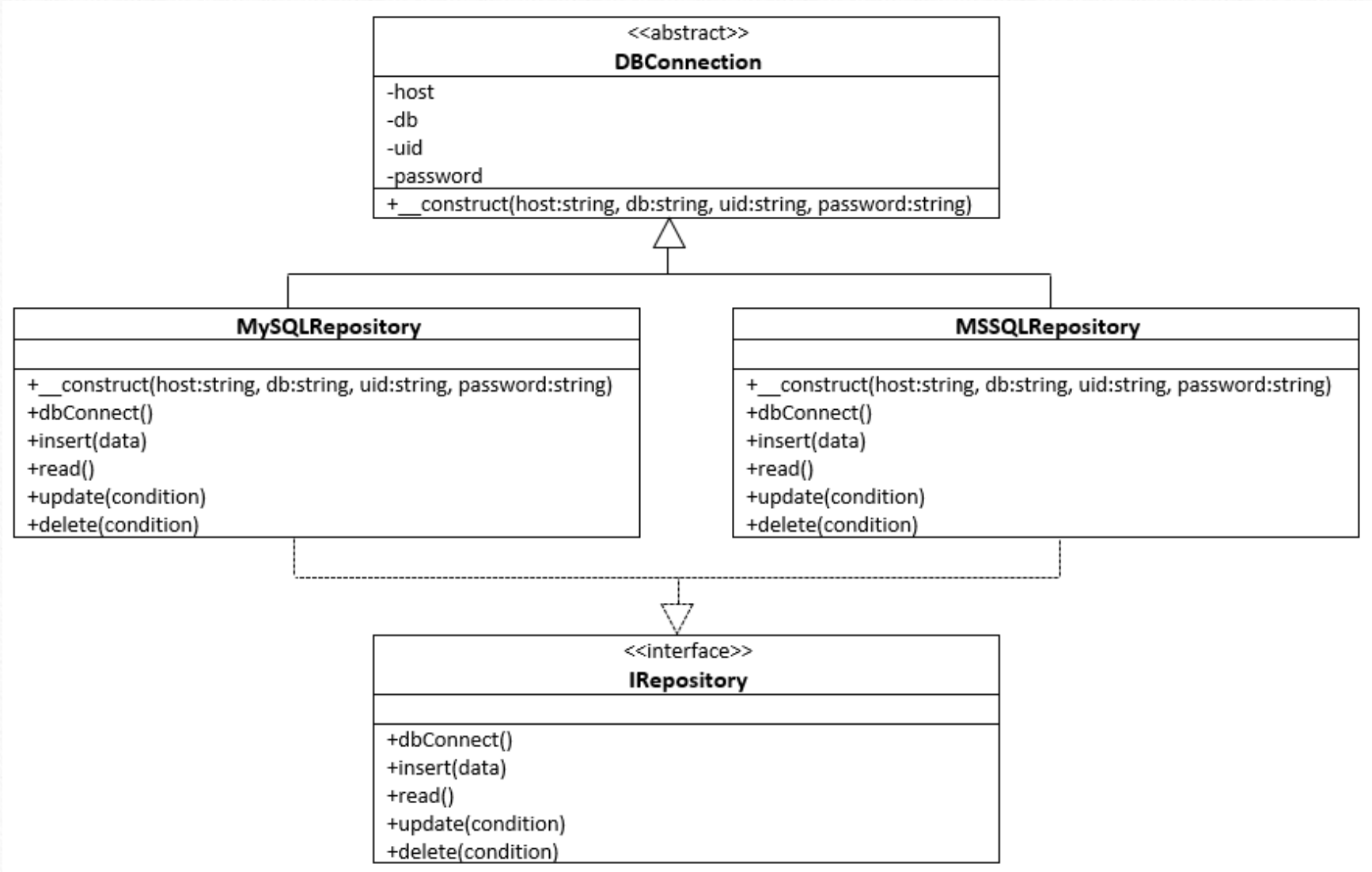
Polymorphism in PHP

- Having sing form/structure – different implementation
- Implemented by implements key word
- Example:
  - develop an application that connects to different database engines
    - Oracle, MySQL and SQL Server but use the same uniform interface
  - create
    - an interface that defines the standard methods
    - an abstract class that implements the common methods

Polymorphism in PHP …

- **Interface**
  - similar to a class
  - only defines the methods and parameters – Signature
  - Created with interface keyword
- **Abstract class**
  - a class that cannot be used to create an object directly
  - provide partial or whole implementations of common methods
  - Created with abstract keyword

Polymorphism in PHP …



Polymorphism in PHP …

```php
<?php
abstract class DBConnection
{
    private $host;
    private $db;
    private $uid;
    private $password;
    public function __construct($host, $db, $uid, $password)     {
        $this->host     = $host;
        $this->db       = $db;
        $this->uid      = $uid;
        $this->password = $password;
    }
}
?>
```

Polymorphism in PHP …

```php
<?php
interface IRepository
{
    public function dbConnect();
    public function insert($data);
    public function read($where);
    public function update($where);
    public function delete($where);
}
?>
```

Polymorphism in PHP …

```php
<?php
class MySQLRepository extends DBConnection implements IRepository
 {
   public function __construct($host, $db, $uid, $password) {
   parent::__construct($host, $db, $uid, $password);
   }
   public function dbConnect() {
      //connect code goes here
   }
   public function delete($where) {
      //delete code goes here
   }
   public function insert($data) {
      //insert code goes here
   }
   public function read($where) {
      //read code goes here
   }
   public function update($where) {
      //update code goes here
   }
}
?>
```

Polymorphism in PHP …

```php
<?php
class MSSQLRepository extends DBConnection implements IRepository
 {
   public function __construct($host, $db, $uid, $password) {
   parent::__construct($host, $db, $uid, $password);
   }
   public function dbConnect() {
      //connect code goes here
   }
   public function delete($where) {
      //delete code goes here
   }
   public function insert($data) {
      //insert code goes here
   }
   public function read($where) {
      //read code goes here
   }
   public function update($where) {
      //update code goes here
   }
}
?>
```

Polymorphism in PHP …

- "class … extends DBConnection"
  - use the methods in the DBConnection abstract class
- "… implements IRepository"
  - ensures that the class provides standard methods regardless of the database driver used
- Usage of the above code

```
<?php
        $db = new MySQLRepository($host,$db,$uid,$password);
?>
OR
<?php
        $db = new MSSQLRepository($host,$db,$uid,$password);
?>
```

- The rest of the code would be the same for both drivers such as;

```
<?php
    $db->dbConnect();
    $db->insert($data);
?>
```

Object Oriented Concepts summary

- Class
  - Real world entity
- Object
  - individual instance of a class
- Member Variable
  - variables defined inside a class.
- Member function
  - function defined inside a class and are used to access object data
- Inheritance
  - a class is defined by inheriting existing function of a parent class

OO concepts …

- Parent class/ base class /super class
  - A class that is inherited from by another class.
- Child Class/ subclass /derived class.
  - A class that inherits from another class. also called a subclass or derived class
- Polymorphism
  - same function can be used for different purposes
- Overloading
  - polymorphism in types of argument(s)
  - also be overloaded with different implementation
- Overriding
  - change the behavior of parent class method

OO concepts …

- Data Abstraction
  - the implementation details are hidden (abstracted)
- Encapsulation
  - hide all the data and member functions together to form an object
- Constructor
  - special type of function which will be called automatically whenever there is an object formation from a class
- Destructor
  - special type of function which will be called automatically whenever an object is deleted or goes out of scope

PHP Access Modifiers

- public
  - accessible from anywhere, even from outside of the class scope
  - Applicable to
    - Functions and variables
- private
  - accessed from within the class itself
  - Applicable to
    - Functions and variables
- protected
  - Can be accessed from its
  - Applicable to
    - Functions and variables
- abstract
  - used for PHP classes and its member functions.
- final
  - can not be changed or overriden by any subclass

PHP - OOP

Questions?

PHP – Database Programming

Database Programming

- Introduction to Relational Database System
- Relational Database System Management System Software
- Structured Query Language
- Connecting PHP application to Database System
- Creating and executing statements
- Working on populated data/Result set
- Using database objects in application program

Database System – Review

- A database is
  - a collection of related,
  - logically coherent data used by the application programs in an organization.
- Advantages of databases
  - Less redundancy
  - Inconsistency avoidance
  - Efficiency
  - Data integrity
  - Confidentiality

RDMS

- A database management system (DBMS)
  - defines,
  - creates and
  - maintains a database
  - allows controlled access to data in the database.
- A DBMS is a combination of five components:
  - hardware,
  - software,
  - data,
  - users and
  - procedures

Relational Database Model

- Data are organized in two-dimensional tables called relations.
- The tables are related to each other.
- A relation in an RDBMS has the following features:
  - Name of entity
  - Attributes
  - Tuples

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of constraints in the relational model:
  - **Key** constraints
    - Primary Key
    - Unique Key
    - Foreign Key
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another implicit constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)
- Check constraints
- Default value constraint

SQL

- SQL is the language standardized by
  - the American National Standards Institute (ANSI) and
  - the International Organization for Standardization (ISO) for use on relational databases.
- It is a *declarative* rather than *procedural* language,
  - which means that users declare what they want without having to write a step-by-step procedure.

SQL …

- In a relational database,
  - can define several operations to create new relations out of the existing ones.
- Basic operations:
  - Insert
  - Delete
  - Update
  - Select
  - Project
  - Join
  - Union
  - Intersection
  - Difference

SQL … violation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
    - Primary Key
    - Unique Key
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

SQL … Violation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL, SET DEFAULT
      - RESTRICT/NO ACTION option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
      - SET DEFAULT option : set the foreign keys of the referencing tuples to the default value if any
  - One of the above options must be specified during database design for each foreign key constraint

SQL … violation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

RDMS

- Networked RDBMS
  - Oracle
    - For large mission critical systems runs on Linux
  - MS SQL Server
    - Used in small to medium sized systems run in Windows server
  - DB2
    - For large mission critical systems runs on IBM OS
  - MySQL/MariaDB
    - Open source, runs on major operating system, commonly used in web application
  - Apache Derby/Java DB
    - New java based relation database, can be embedded in application
  - …
- Embedded RDBMS
  - Apache Derby/Java DB
  - MS Access
  - Compact/SQLite version of MS SQL, Oracle

SQL Skill

- Creating and/or modifying database objects
- Retrieving data from a table
- Insert, Delete, update data
- Using functions
  - String function
  - Numeric functions
  - Date/Time functions
  - Other functions
- Creating View
- Transact-SQL programming
  - Scripts
  - Creating triggers
  - Stored procedures
  - Functions
- Working with Transaction

PHP – MySQL database

- Mysql
  - mysql_connect
    - Open a connection to a MySQL Server
    - deprecated in PHP 5.5.0
    - removed in PHP 7.0.0.
    - Syntax
      - $conn = mysql_connect('localhost', user', 'password‘,’database’);
- MySQLi
  - MySQL Improved Extension
  - allows access to the functionality provided by MySQL 4.1 and above
  - Procedural and OO API
  - MySQL database
  - Syntax
    - Procedural
      - $conn = mysqli_connect("hostname", "username", "password", "database“, port);
    - OO
      - $conn = new mysqli("hostname", "username", "password", "database“, port);

PHP – MySQL …

- PDO
  - PHP Data Objects (PDO)
  - defines a lightweight, consistent interface for accessing databases in PHP
  - provides a data-access abstraction layer
  - OO API
  - Support different database
  - Syntax
    - $pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");

Example - mysql_connect()

```
<?php
$conn = mysql_connect('localhost', 'mysql_user', 'mysql
_password');
if (!$conn) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($conn);
?>
```

Example - MySQLi - Object-oriented

```php
<?php
    $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);
    }

    // Create database
    $sql = "CREATE DATABASE testdb";
    if ($conn->query($sql) === TRUE) {
      echo "Database created successfully";
    } else {
      echo "Error creating database: " . $conn->error;
    }

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

MySQLi …

- New database – three argument
  - mysqli object (servername, username and password).
- If you have to use a specific port,
  - add an empty string for the database-name argument
  - new mysqli("localhost", "username", "password", "", port)

MySQLi – Procedural

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

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE testdb";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

PDO - PHP Data Objects

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

try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE testdb";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Database created successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

- PDO
  - has exception class to handle any problems

PDO – insert operation

- INSERT INTO statement is used
- $pdo = new PDO("mysql:host=localhost;dbname=registrar", "root", "");
- $sql = "INSERT INTO college (name, phone) VALUES (‘College', ‘09123654789')";
- $pdo->execute($sql);

PDO – Prepared statement

- A prepared statement
  - parameterized statement
  - SQL query template containing placeholder instead of the actual parameter values
  - Placeholders will be replaced by the actual values at the time of execution of the statement
- MySQLi supports
  - positional placeholder (?)
- PDO supports
  - positional placeholder (?) and
  - named placeholders.
    - A named placeholder begins with a colon (:) followed by an identifier
- Example
  - Positional placeholder
    - INSERT INTO college (name, phone) VALUES (?, ?);
  - Named placeholder
    - INSERT INTO college (name, phone) VALUES (:name, :phone);

PDO – Prepared …

- prepared statement execution consists
  - Prepare
    - SQL statement template is created and
    - sent to the database server.
    - The server parses the statement template,
    - performs a syntax check and query optimization, and stores it for later use.
  - Execute
    - parameter values are sent to the server.
    - The server creates a statement from the statement template and these values to execute it.

PDO – Prepared …

- Advantage
  - Efficiency
    - the statement is parsed only once again, while it can be executed multiple times
  - Minimize bandwidth
    - upon every execution only the placeholder values need to be transmitted to the database server instead of the complete SQL statement.
  - Provide strong protection against SQL injection
    - parameter values are not embedded directly inside the SQL query string

PDO – Prepared …

- Database connection
  - $pdo = new PDO("mysql:host=localhost;dbname=registrar", "root", "");
- SQL Statement
  - $sql = "INSERT INTO college (name, phone) VALUES (:name, :phone)";
- Prepare query
  - $query = $pdo -> prepare($sql);
- Bind the placeholders to the variables:
  - $query->bindParam(':name',$name);
  - $query->bindParam(':phone',$phone);
-

PDO – Prepared …

- add a third parameter as a datatype:
  - $query->bindParam(':name',$name,PDO::PARAM_STR);
  - $query->bindParam(':phone',$phone,PDO::PARAM_INT);
- Also use bindValue to insert static value
  - bindValue(':name',‘College of Natural Science’,PDO::PARAM_STR);
- Assign values
  - $name = “College of Education";
  - $phone = “01123654789";
- Execute the query
  - $pdo->execute($sql);

PDO – Select operation

- Database connection
  - $pdo = new PDO("mysql:host=localhost;dbname=registrar", "root", "");
- Sql
  - $sql = “Select * from college”;
- Prepare query
  - $query = $pdo->prepare($sql);
- Execute
  - $query -> execute();
- Assign the data
  - $results = $query -> fetchAll(PDO::FETCH_OBJ);

PDO – Select operation

- Display result

```
if($query -> rowCount() > 0) {
    foreach($results as $result) {
        echo $result -> Id . ", ";
        echo $result -> Name . ", ";
        echo $result -> phone;
    }
}
```

PDO – Update operation

- Update statement
- Database connection
  - $pdo = new PDO("mysql:host=localhost;dbname=registrar", "root", "");
- Sql
  - $sql = “Update college set phone=:phone where id=:id ”;
- Prepare query
  - $query = $pdo->prepare($sql);
- Bind value
  - $query -> bindParam(':phone', $phone);
  - $query -> bindParam(':id', $id,);
  - $phone = ‘012365478’;
  - $id = 1;
- Execute
  - $query -> execute();

PDO – Update …

- Check update operation

```
if($query -> rowCount() > 0) {
    $count = $query -> rowCount();
    echo $count . " Records were updated.";
}
else {
    echo "No record updated.";
}
```

PDO – Delete operation

- Delete statement
- Database connection
  - > $pdo = new PDO("mysql:host=localhost;dbname=registrar", "root", "");
- Sql
  - > $sql = “Delete from college where id=:id ”;
- Prepare query
  - > $query = $pdo->prepare($sql);
- Bind value
  - > $query -> bindParam(':id', $id,);
  - > $id = 1;
- Execute
  - > $query -> execute();

PDO – Delete …

- Check update operation

```
if($query -> rowCount() > 0) {
    $count = $query -> rowCount();
    echo $count . " Records were deleted.";
}
else {
    echo "No record deleted.";
}
```

Questions?