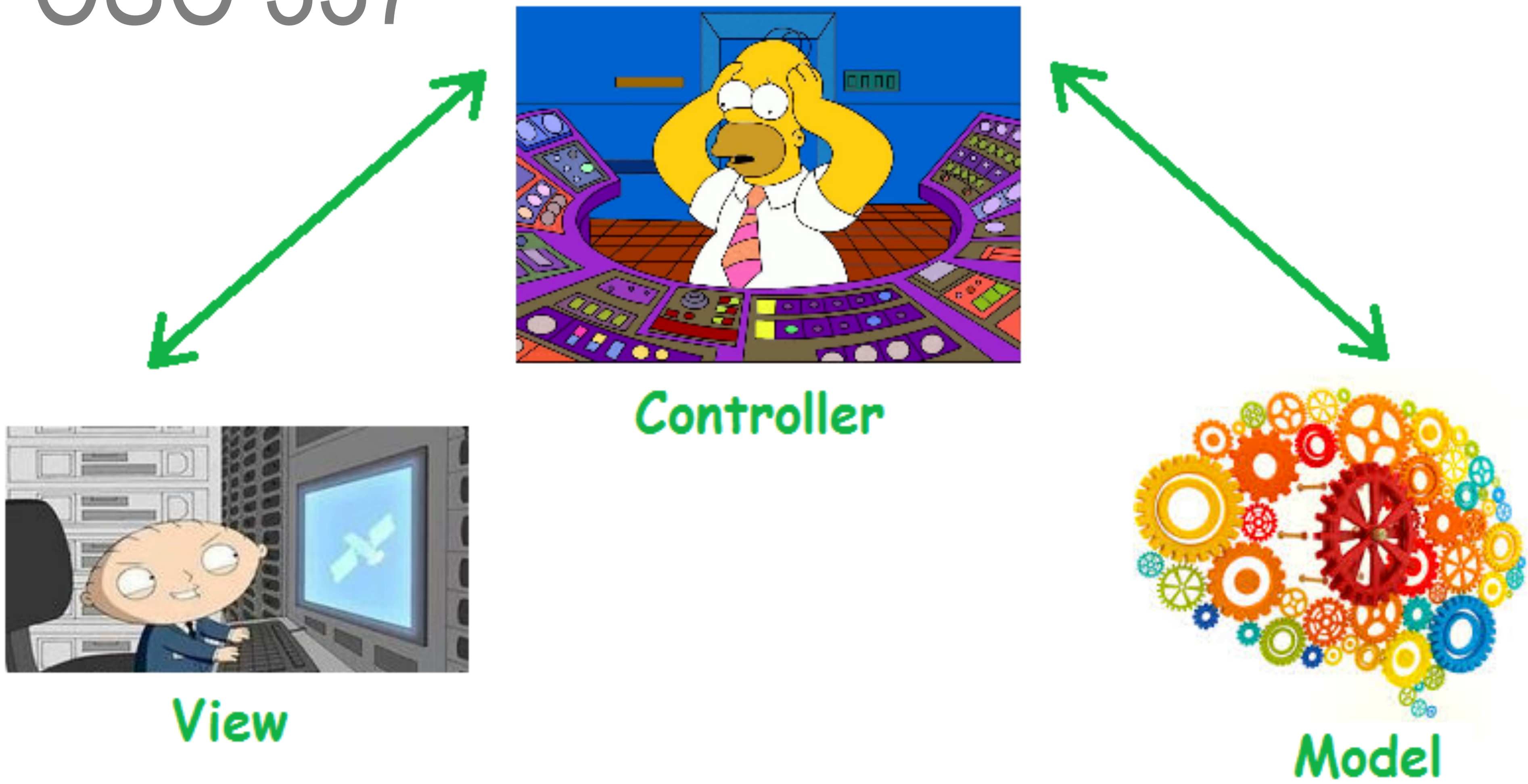
CSC 337



Model View Controller (MVC)

PHP Data Objects (PDO)

Rick Mercer

Model View Controller (MVC)

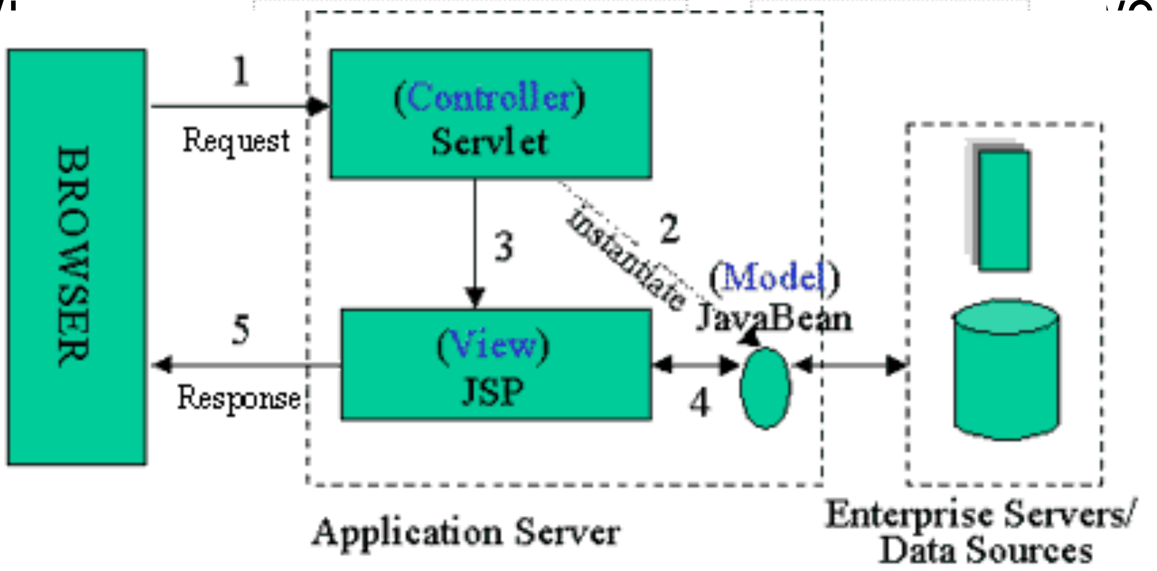
- In the MVC paradigm, the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of objects, each specialized for its task.
  - model
  - view
  - controller

Model View Controller

- The **view** manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application.
- The **controller** interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.
- The **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

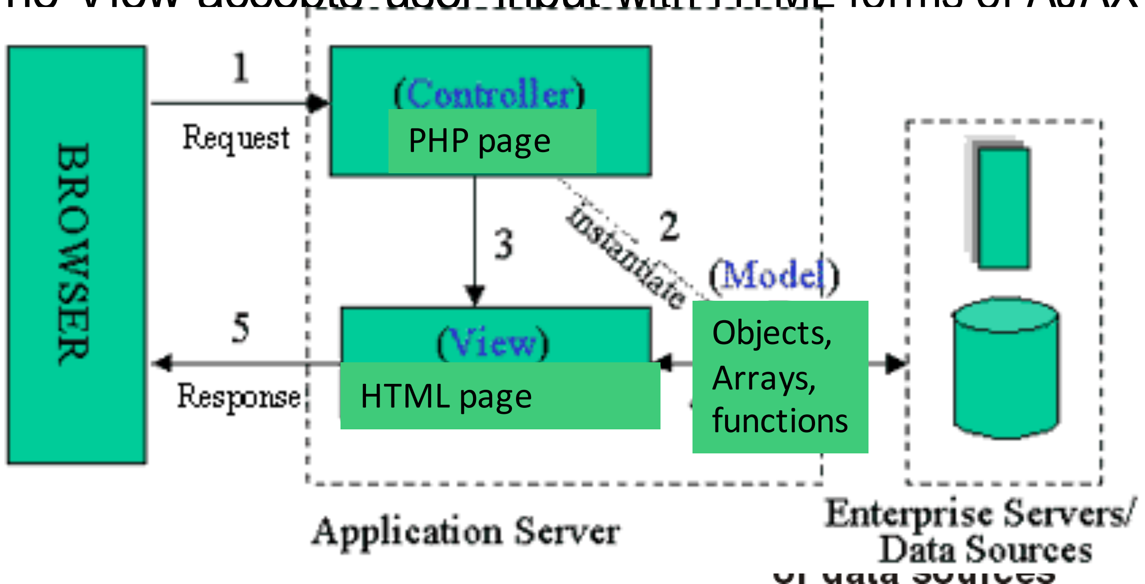
Java Server Pages

- Model 2 Architecture to serve dynamic content
  - Model: Enterprise Beans with data in the DBMS
    - JavaBean: a class that encapsulates objects and can be displayed graphically
  - Controller: Servlets create beans, decide which JSP to return, do the bulk of the processing
  - Vi… (the b…



MVC in CSC 337

- Model: Arrays, the database, PHP objects, functions
- Controller: Coordinates between the view (the browser) and database
  - This will be a separate file in a CSC 337 project
- View: the browser
  - The View accepts user input with HTML forms or AJAX opens



MVC Benefits

- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one don't affect the others
  - can develop in parallel once you have the interfaces
- Multiple views
  - Spreadsheets, Powerpoint, Eclipse…
  - Demo different views in these three apps…

MVC in 337

- Model: the data
  - file processing we've seen
  - databases we now begin to use with PDO (next slides)
- View:
  - Accepts user input (input types like buttons, text fields, hovers, …
  - Displays the state of the model to the user with the rendering (HTML page)
- Controller:
  - the go between user input and the data
  - this will be a PHP script

PHP Data Objects (PDO)

PDO functions in a class connecting to a data base

```
class DatabaseAdaptor {

  private $DB; // The instance variable used in every method

  // Connect to an existing data based named 'first'
  public function __construct() {
    $dataBase = 'mysql:dbname=first; charset=utf8; host=127.0.0.1';
    $user = 'root';
    $password = '';
    try {
      $this->DB = new PDO ( $ dataBase, $user, $password );
      $this->DB->setAttribute ( PDO::ATTR_ERRMODE,
                                PDO::ERRMODE_EXCEPTION );
      } catch ( PDOException $e ) {
     echo ('Error establishing Connection');
     exit ();
    }
 } // . . .  continued
```

```php
// Return all customer records as a PHP associative array.
public function getAllRecords () {
   $stmt = $this->DB->prepare( "SELECT * FROM customers" );
   $stmt->execute ();
   return $stmt->fetchAll( PDO::FETCH_ASSOC );
  }
} // End class DatabaseAdaptor

// Test code: Run as CLI console app
$theDBA = new DatabaseAdaptor ();
// Remove the following test lines after tested
$arr = $theDBA-> getAllRecords ();
echo $arr[0]['ID'] . PHP_EOL;
echo $arr[1]['Name'] . PHP_EOL;
echo $arr[2]['City'] . PHP_EOL;
print_r($arr);
?>
```

Output (print_r output shown to the the right)

```
1
Ana
Douglas
```

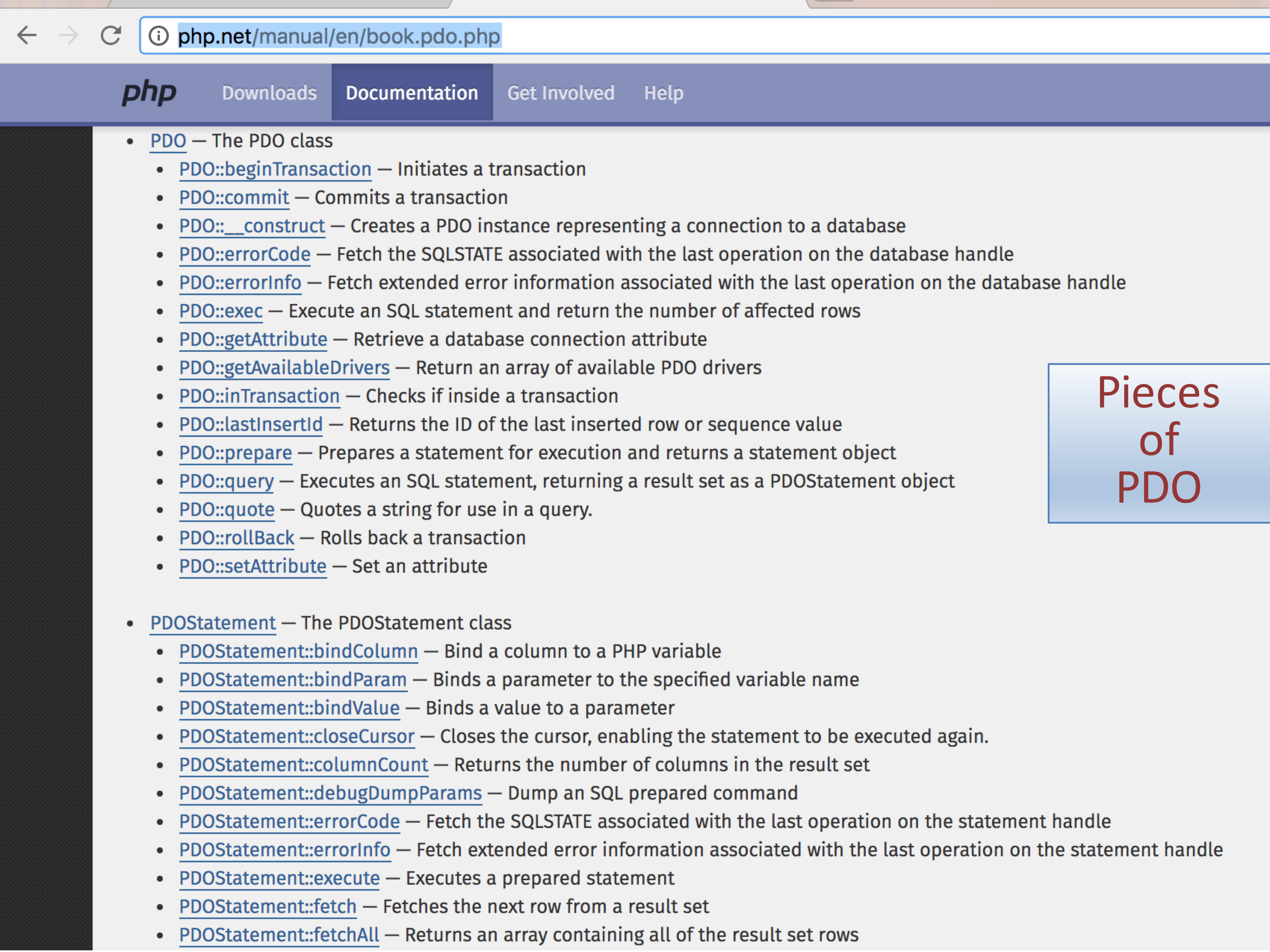
```
Array
(
    [0] => Array
        (
            [ID] => 1
            [Name] => Maria
            [City] => Tucson
        )

    [1] => Array
        (
            [ID] => 2
            [Name] => Ana
            [City] => Yuma
        )

    [2] => Array
        (
            [ID] => 3
            [Name] => Antonio
            [City] => Douglas
        )

    [3] => Array
        (
            [ID] => 4
            [Name] => Thomas
            [City] => Phoenix
        )
)
```

Pieces of PDO



- PDO — The PDO class
  - PDO::beginTransaction — Initiates a transaction
  - PDO::commit — Commits a transaction
  - PDO::__construct — Creates a PDO instance representing a connection to a database
  - PDO::errorCode — Fetch the SQLSTATE associated with the last operation on the database handle
  - PDO::errorInfo — Fetch extended error information associated with the last operation on the database handle
  - PDO::exec — Execute an SQL statement and return the number of affected rows
  - PDO::getAttribute — Retrieve a database connection attribute
  - PDO::getAvailableDrivers — Return an array of available PDO drivers
  - PDO::inTransaction — Checks if inside a transaction
  - PDO::lastInsertId — Returns the ID of the last inserted row or sequence value
  - PDO::prepare — Prepares a statement for execution and returns a statement object
  - PDO::query — Executes an SQL statement, returning a result set as a PDOStatement object
  - PDO::quote — Quotes a string for use in a query.
  - PDO::rollBack — Rolls back a transaction
  - PDO::setAttribute — Set an attribute

- PDOStatement — The PDOStatement class
  - PDOStatement::bindColumn — Bind a column to a PHP variable
  - PDOStatement::bindParam — Binds a parameter to the specified variable name
  - PDOStatement::bindValue — Binds a value to a parameter
  - PDOStatement::closeCursor — Closes the cursor, enabling the statement to be executed again.
  - PDOStatement::columnCount — Returns the number of columns in the result set
  - PDOStatement::debugDumpParams — Dump an SQL prepared command
  - PDOStatement::errorCode — Fetch the SQLSTATE associated with the last operation on the statement handle
  - PDOStatement::errorInfo — Fetch extended error information associated with the last operation on the statement handle
  - PDOStatement::execute — Executes a prepared statement
  - PDOStatement::fetch — Fetches the next row from a result set
  - PDOStatement::fetchAll — Returns an array containing all of the result set rows

Now the model is tested

- Remove all test echos from the model

```
$theDBA = new DatabaseAdaptor ();
// Test code: Run as CLI console app
// Remove the follwong code after tested
// $arr = $theDBA->getAllRecords ();
// echo $arr[0]['ID'] . PHP_EOL;
// echo $arr[1]['Name'] . PHP_EOL;
// echo $arr[2]['City'] . PHP_EOL;
// print_r($arr);
```

- Add a controller to send a PHP array to JS

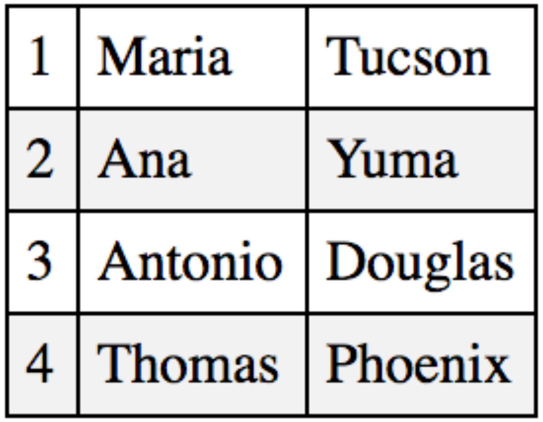
```
<?php
// File name controller.php
// Acts as the go between the view and the model.
include "model.php";
$theDBA = new DatabaseAdaptor();
echo json_encode($theDBA->getAllRecords());
?>
```

The View part of MVC

```
anObj.onreadystatechange = function () {
  if (anObj.readyState == 4 && anObj.status == 200) {
  // Read an array of arrays, the 2nd subscript is a string key
  array = JSON.parse(anObj.responseText);

  str = "<table>";
  for (var i = 0; i < array.length; i++) {
    str += "<tr>";
    str += "<td>" + array[i]['ID'] + "</td>";
    str += "<td>" + array[i]['Name'] + "</td>";
    str += "<td>" + array[i]['City'] + "</td>";
    str += "</tr>";
  }
  str += "</table>";

// Change the DOM to show the table
```



1	Maria	Tucson
2	Ana	Yuma
3	Antonio	Douglas
4	Thomas	Phoenix