# Web Programming

1st Semester, 2020/2021

Lecture #4

MM Irfan Subakti

# PHP & MySQL (MariaDB)

- PHP: Hypertext Preprocessor
- MySQL → MariaDB
- XAMPP → PHP + MySQL


# Redirection

- When the login data has been processed/validated then redirection can be used if the new webpage want to be visited
- `header("Location: URL");`

```
header("Location: http://31.31.198.216/web/main.php");
```

- Common techniques
  - Starting webpage = login webpage
  - Login webpage validates the user & set the cookies
  - Redirect to the new webpage
  - The new webpage uses the cookies' data to access the database information

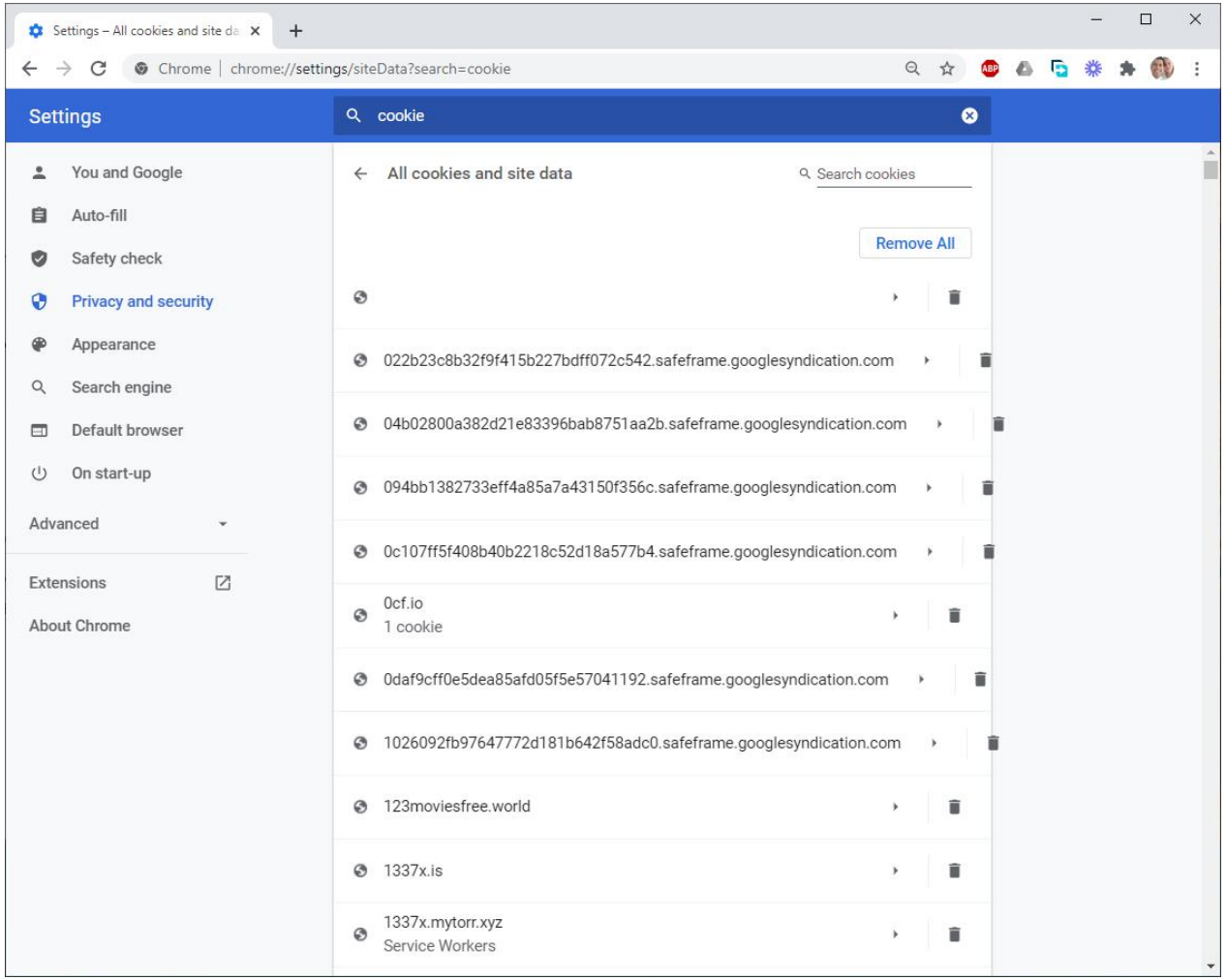
# Maintaining state

- HTTP is the stateless protocol → each client/server transaction is the different entity
- Thus, webserver doesn't have an automatic mechanism to remember the browser's information about any website
- On the other hand, a lot of web-based application needs for maintaining the state. E.g., provide the state of a user in her/his "shopping cart" stage before continue to the "checkout" stage.

# Cookies

- It’s used to overcome the “stateless” characteristic of web (HTTP protocol)
- Cookies stored in the client storage
- Actions to cookies
  - Create
  - Access
  - Delete

# Cookies in the browser



# Cookies: creating

- setcookie(name, value, expiration);
  - `setcookie("hobby", "swimming", time()+3600);`
  - Cookie'name → `"hobby"`
  - Cookie's value → `"swimming"`
  - Will be deleted 3600 seconds = 60 minutes = 1 hour from the current time
- The cookie's value will be sent as a part of HTTP header

# Cookies: accessing

- `$_COOKIE` → containing the value of the current active cookie
- E.g.,

```
<?
     foreach($_COOKIE as $name => $value) {
          echo "<br>$name => $value";
     }
?>
```

# Cookies: deleting

- A cookie will be automatically deleted once the designated expiration time's up, or
- Manually will be deleted by setting the given cookie with the time variable's negative value

```
setcookie("username", "", time()-3600);
```

# Cookies: the problems

- Cookies can be disabled → the user can set the browser not to run the cookies
- The cookies can be seen by other users
- Only can save 20 cookies → max 4 kB
- Some browsers show the correct cookies only if the options have been all set in `setcookie()`

# Session

- Luckily, PHP provides a simple mechanism for maintaining the state information → session
- Session is the sequential HTTP requests from a given browser → the problem is how to recognise the first request & the second one are from the same browser
- E.g.,
  - A user can log in a given system. She has some activities on a given webpage (i.e., the first webpage) – the browser, of course, knows that the one who is doing activity is her, she just log in.
  - The browsing activities on the next webpages still recognise that the user who is doing activity is the same user as in the first webpage

# Session: the setting

- Session in PHP can be set by a super global array `$_SESSION`
- A PHP script can create a variable in that array & this variable will be available for other scripts in the same session
- E.g.,
  - A script has successfully validated a user → this script can set a variable which save who is the user, then the other scripts can check whether that variable has already been set or not

# Session: the initialisation

- A script which will use a session has to call `session_start()`
- Then, the script can write or read the array's content of `$_SESSION`
- That script can be put in the login webpage → validate the user's detail & set the session variable

```
<?php
    session_start();
    // validate the user's detail on login
    $_SESSION['user_id'] = "admin@subakti.com";
?>
```

# Session: the next

- The next scripts will check whether variable `$_SESSION` has already been set or not

```
<?php
      session_start();
      if (isset($_SESSION['user_id'])) {
        //we knew who is she & can customise page
      } else {
        //provide free content/redirect to login page
      }
?>
```

# Session: how it works

- It works by using cookie. When the first session created, PHP will create a session id which will be sent to the browser as a cookie – the information saved in the browser
- Variable created in the array of `$_SESSION` saved in the server → in the area identified by session id
- For each next HTTP request, the browser automatically send back the cookie to the web server, then it saves the value to the array of `$_SESSION` so that it can be accessed by the new/next script

# Session: how if it’s turned off

- If the browser has been set not to accept cookie sent by the web server, PHP automatically will send session id along with the URL
- E.g., if session id is 9876544210123456789 then PHP automatically will add up this value to every links in the webpage so that the link cart.php becomes cart.php?PHPSESID=9876544210123456789
- When the user clicked the next webpage, session id will be sent back to the server along with the URL

# MySQL: introduction

- GNU (General Public License) free relational database (DB) server
- Open-source relational database management system (RDBMS)
- Multiplatform
- Server networking form → no GUI as MS Access

# phpMyAdmin

- MySQL client written in PHP
- Web-based for managing
  - Database (DB)
  - MySQL users
  - Submit query
- Recommended for a newbie

# MySQL: basic commands

- Create database, drop database
- Create table, alter table, drop table
- Lock tables, unlock tables
- Select, delete, insert into, describe, update

# Database connection

- PHP supports database connection in various ways
- One of them is direct connection to MySQL DB via functions
  - `mysql_connect(),mysql_select_db(),` etc.

```
$username = "rahayu"; $password = "dewi";
$database = "mystore";
$conn = mysql_connect("localhost", $username, $password);
if (!$conn) {
      die(“Connection failed: “ . mysql_connect_error());
}
$db_selected = mysql_select_db($database);
if (!$db_selected) {
      die("Unable to select database“ . mysql_error());
}
```

# Query: submit to DB

```
$query = "SELECT userID FROM users WHERE username =
'rahayu'";
$result = mysql_query($query);
// no result -> no user, so return false
if (!$result) {
     … no result (error!)
}
```

# Query: processing the result

- If there is no error then `$result` refers to the result's object
- This object is like a cursor wherein there's `fetchRow()` which will fetch current row (in array) and then move to the next row

```
while ($row =& $result -> fetchRow()) {
    // do something in here
}
```

- `fetchRow()` returns `NULL` if there's no more row can be fetched, so that by using a `while` loop each row can be processed

# Query: showing the result

- Showing the data from each row

```
while ($row =& $result -> fetchRow()) {
     print "<p>$row[0] has a population of $row[1]<p>";
}
```

- Each row is an array whose 2 elements by the index `$row[0]` and `$row[1]`

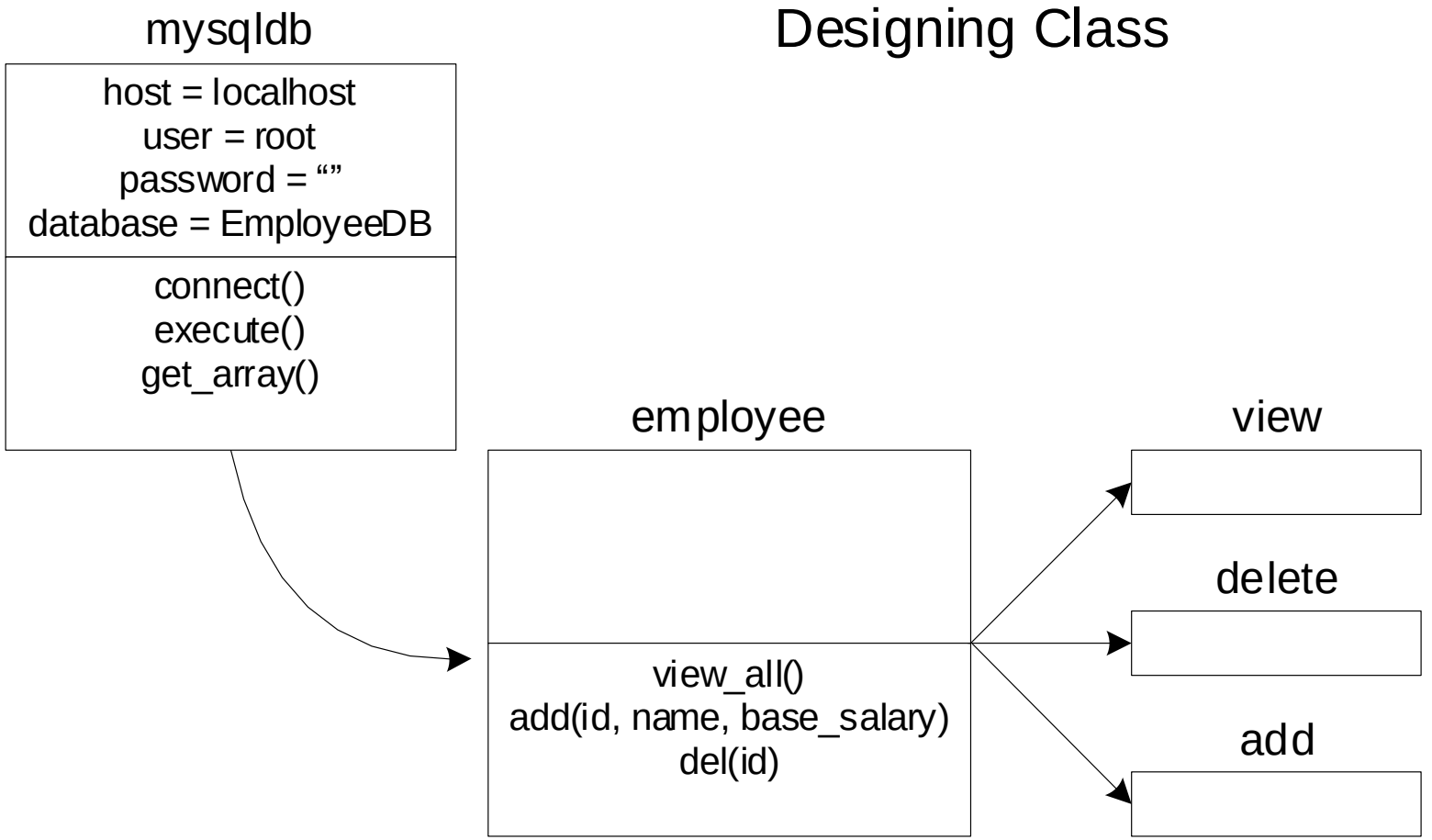
# Query: clean-up

- Finally, the script will release all of the current resources used

```
$result -> free();
mysql_close($conn);
```

- Actually, when this script ends, the resources will be freed automatically. However, if there's a long script where there are a lot of DB connections → the resources needs to be released explicitly once they're finished

# PHP + MySQL with OOP



# Class mysqldb

mysqldb.php

```php
<?
    class mysqldb {
        private $db;
        private $host;
        private $user;
        private $password;
        private $database;
        private $query;
        private $result;
        private $row;

        function mysqldb() {
            include "conf.php";
            $this -> host = $host;
            $this -> user = $user;
            $this -> password = $password;
            $this -> database = $database;
        }

        function connect() {
            $this -> db = mysql_connect(
                $this -> host,
                $this -> user,
                $this -> password);
            if (!$this -> db) {
                die('ERROR: ' . mysql_error());
            }
            mysql_select_db($this -> database, $this -> db);
        }

        function execute($query) {
            $this -> query = $query;
            $this -> result = mysql_query($this -> query, $this -> db);
            $this -> error_message($result);
        }

        function get_array() {
            if ($this -> row = mysql_fetch_array($this -> result, MYSQL_ASSOC)) {
                return $this -> row;
            } else {
                return false;
            }
        }
    }
?>
```

# Class employee

employee.php

```php
<?
    require_once ("mysqldb.php");
    class employee extends mysqldb {
        private $db;
        private $host;
        private $user;
        private $password;
        private $database;
        private $query;
        private $result;

        function view_all() {
            $this -> execute("SELECT * FROM employee");
        }

        function add($id, $name, $base_salary) {
            $this -> execute("INSERT into employee(id, name, base_salary) "
                ."VALUES ('$id', '$name', '$base_salary')");
        }

        function del($id) {
            $this -> execute("DELETE FROM employee WHERE id = '$id'";
        }
    }
?>
```

# Objects: the implementation

```php
<?
    // view
    require_once ("employee.php");
    $view = new employee;
    $view -> connect();
    $view -> view_all();

    while ($result = $view -> get_array()) {
        echo $result["name"];
        echo "<br>";
    }
?>

<?
    // del
    require_once ("employee.php");
    $del = new employee;
    $del -> connect();
    $del -> del('7');
?>

<?
    // add
    require_once ("employee.php");
    $add = new employee;
    $add -> connect();
    $add -> add('7', 'Pandanwangi', '15000000');
?>
```