

# Storage

1PHPD

# 1PHPD – Storage

## Course Objectives

By the end of the course, students should:

- Know the different storage types
- Be able to use the filesystem
- Work with a SQL database

Time:

- course: 3h
- exercises: 4h

# *Summary*

1. Files
2. Databases
3. Exceptions



# 1. Files

# 1. Files

PHP provides a robust set of functions for file handling, allowing you to create, read, write, delete, and manipulate files on the server.

This capability is essential for many web applications, from simple logging mechanisms to complex data storage and processing systems.

# 1. Files

Before you can read from or write to a file, you need to open it using the `fopen()` function.

This function requires at least two arguments: the path to the file and the mode in which to open the file.

# 1. Files

The mode can be, for example

- 'r' for read-only,
- 'w' for write (which will erase the file and start fresh),
- 'a' for append (which will keep the file's contents and add to the end)

# 1. Files

## Example

```
$file = fopen("example.txt", "r");
```



# 1. Files

After opening a file, you can read its content.

In PHP, there are several ways to read the content of a file.

The best method depends on your specific needs, such as whether you want to read the file line by line, all at once, or in chunks

`feof()` checks if the "end-of-file" (EOF) has been reached, and `fclose()` closes an open file pointer.

# 1. Files

For more control, especially with larger files, you can use `fopen()` in combination with `fgets()` (to read line by line) or `fread()` (to read a specific amount of bytes at a time).

# 1. Files

## Example

```
$file = fopen('example.txt', 'r');  
if ($file) {  
    while (($line = fgets($file)) !== false) {  
        echo $line;  
    }  
    fclose($file);  
} else {  
    // error opening the file  
}
```

# 1. Files

## Example

```
$file = fopen('example.txt', 'r');  
if ($file) {  
    while (!feof($file)) {  
        $content = fread($file, 1024); // Reads up to 1024 bytes  
        echo $content;  
    }  
    fclose($file);  
} else {  
    // error opening the file  
}
```

# 1. Files

The function “`file_get_contents()`” reads the entire content of a file into a string.

It's the simplest method to read a whole file at once.

This method is best for reading smaller files or when you need the entire file content at once.

# 1. Files

## Example

```
$content = file_get_contents('example.txt');  
echo $content;
```

# 1. Files

The `file()` function reads the entire file into an array, with each line of the file as an element of the array.

This is useful for processing a file line by line while having access to all lines at once.

# 1. Files

## Example

```
$lines = file('example.txt');  
foreach ($lines as $line) {  
    echo $line;  
}
```



# 1. Files

The `readfile()` function reads a file and writes it directly to the output buffer.

It's a convenient way to output the content of a file to the browser.

This function is useful when you need to simply display a file's contents as part of a response without manipulating or processing the data.

# 1. Files

## Example

```
readfile('example.txt');
```

# 1. Files

To write content to a file, you can use `fwrite()` after opening the file in a write (w) or append (a) mode.

The content will be written “as is” without modification, and everything will be considered as a string.

You cannot keep a datatype (boolean or float) when you write or read the data from a file, you will have to cast the value.

# 1. Files

## Example

```
$file = fopen("example.txt", "w");  
fwrite($file, "Hello, World!");  
fclose($file);
```

# 1. Files

It's important to close a file after you're done with it to free up system resources.

Use `fclose()` for this purpose.

# 1. Files

## Example

```
fclose($file);
```

# 1. Files

PHP allows you to delete a file from the server using the `unlink()` function.

# 1. Files

## Example

```
unlink("example.txt");
```



# 1. Files

Before working with a file, it's often necessary to check if it exists to avoid errors.

Use `file_exists()` for this check.

# 1. Files

## Example

```
if(file_exists("example.txt")) {  
    // File exists  
}
```

# 1. Files

When dealing with files, be mindful of file permissions.

PHP runs as a server module, and its permissions to access and modify files are determined by the server configuration and the permissions set on the files/directories themselves.

# 1. Files

Working with files can be a challenge, especially for security, so you have to take care of it.

Validate all input to avoid path traversal vulnerabilities.

Be cautious when allowing file uploads or any operation that involves user-supplied content to avoid security risks, such as code execution or overwriting important files.

Consider using higher-level functions or libraries for complex operations, such as file uploading, to benefit from built-in security features.

# 1. Files

When working with files, always consider the file's size and the memory limitations of your PHP environment.

Reading a very large file into memory with `file_get_contents()` or `file()` might lead to memory exhaustion errors.

Ensure you have the appropriate permissions to read the file, and consider the security implications of exposing file paths and contents.

Always check if the file exists using `file_exists()` before attempting to read it to avoid errors.



Exercises

## 2. Database

## 2. Database

Files can be a great way to store information in different formats, like text, csv or binary.

But as you can see it is complex to search a specific information in a file, query-specific data or just find an item without reading the full file.

That is where databases shine - and they use files behind the scenes to store data.



## 2. Database

You already know about SQL and database so we will focus on learning how to use and query the database from PHP.

For the graphical interface we will use PHPMyAdmin. If you used Mamp to install the full ecosystem it should already be running and you can access it on your localhost

On Linux you will need to install the package (it should be phpmyadmin ) and use the right port.

## 2. Database

PHP is widely used for developing dynamic web applications, with database interaction being a core aspect of many such applications.

PHP supports various databases, including MySQL, PostgreSQL, SQLite, Oracle, and Microsoft SQL Server, among others. On our side we will only work with a MySQL database

The language offers several ways to connect to databases and manipulate data, ensuring flexibility and scalability for developers.

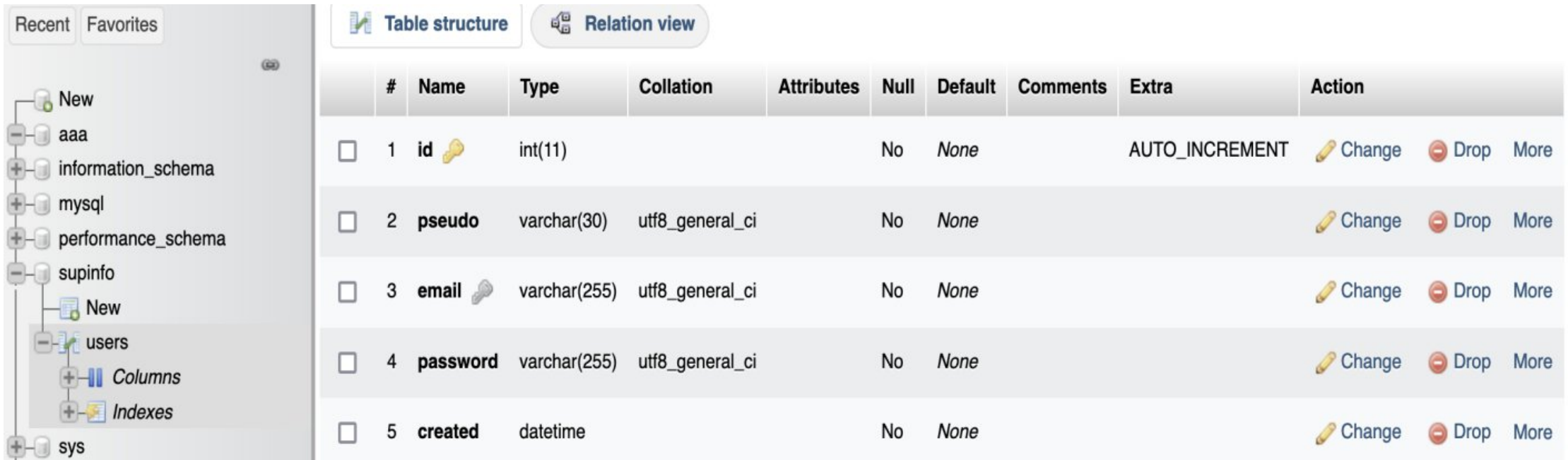
## 2. Database

The first step is to create a base and a table. All of that can be done manually from the web interface. To create a base:

- On the left, click on "new"
- Choose the name you want and keep utf8\_general\_ci > Create
- Create a table with the number of columns you want
- If you are running PHP@8.2 you may have some warning. Ignore all
- You should see your table with empty columns. Name them then
- save

## 2. Database

You should have a table fully created ! Here is an example of the said table (but you can create another one different if you want).



The screenshot shows a database management interface. On the left is a tree view of the database structure, including a 'supinfo' database with a 'users' table. The main area displays the 'Table structure' view for the 'users' table. It features a table with 5 columns: 'id' (int(11), primary key, AUTO\_INCREMENT), 'pseudo' (varchar(30), utf8\_general\_ci), 'email' (varchar(255), utf8\_general\_ci), 'password' (varchar(255), utf8\_general\_ci), and 'created' (datetime). Each row has checkboxes for selection and action buttons (Change, Drop, More).

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	<b>id</b>	int(11)			No	None		AUTO_INCREMENT	Change  Drop  More
<input type="checkbox"/>	2	<b>pseudo</b>	varchar(30)	utf8_general_ci		No	None			Change  Drop  More
<input type="checkbox"/>	3	<b>email</b>	varchar(255)	utf8_general_ci		No	None			Change  Drop  More
<input type="checkbox"/>	4	<b>password</b>	varchar(255)	utf8_general_ci		No	None			Change  Drop  More
<input type="checkbox"/>	5	<b>created</b>	datetime			No	None			Change  Drop  More

## 2. Database

To access the database and be able to execute queries from our PHP code we will need a "driver" to connect to the base.

There are multiple drivers depending on which database you want to use. But there is also something called PDO (PHP Data Objects) that is a wrapper around them and allow you to write the same code independently of the database.

PDO is the main solution since PHP@5 to use a database - except if the driver specifically doesn't support PDO (don't implement PDO).

## 2. Database

Let's see a first driver without PDO.

MySQLi ("i" stands for improved) is a PHP extension designed to work with MySQL and MariaDB databases.

It provides both procedural (first example) and object-oriented (second example) interfaces.

## 2. Database

```
$connection = mysqli_connect("localhost", "username", "password", "database_name");

if (!$connection) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM users";
$result = mysqli_query($connection, $sql);

if (mysqli_num_rows($result) > 0) {
    while($row = mysqli_fetch_assoc($result)) {
        echo "ID: " . $row["id"]. " - Name: " . $row["firstname"]. " "
    }
} else {
    echo "0 results";
}

mysqli_close($connection);
```

## 2. Database

```
$mysqli = new mysqli("localhost", "username", "password", "database_name");

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM users";
$result = $mysqli->query($sql);

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"];
    }
} else {
    echo "0 results";
}

$mysqli->close();
```



## 2. Database

PDO provides a data-access abstraction layer, meaning you can use the same functions to connect to various databases without needing to rewrite code for different database engines.

## 2. Database

```
try {
    $pdo = new PDO("mysql:host=localhost;dbname=database_name", "username", "password")
    // Set the PDO error mode to exception
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->prepare("SELECT id, firstname, lastname FROM users");
    $stmt->execute();

    // Set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new RecursiveArrayIterator($stmt->fetchAll()) as $k=>$v) {
        echo $v["id"]. " - Name: " . $v["firstname"]. " " . $v["lastname"]. "<br>";
    }
} catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$pdo = null;
```

## 2. Database

Using PDO (PHP Data Objects) in PHP to interact with a database involves several steps, including

- establishing a connection
- executing queries
- handling results
- closing the connection

## 2. Database

First, you need to create a new instance of the PDO class to establish a connection to the database.

You'll need to provide a DSN (Data Source Name), which includes the database type, host, and the database name, along with a username and password.

## 2. Database

To connect to a database, PDO only needs 5 information

- Your database type: “mysql”
- The address of the database “localhost”
- The name of the database “supinfo”
- A valid login “root”
- A valid password “toor”

## 2. Database

### Example

```
$pdo = new PDO('mysql:host=localhost;dbname=supinfo', root, toor);  
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

The `setAttribute()` method is used to set the error reporting mode. `PDO::ERRMODE_EXCEPTION` tells PDO to throw exceptions whenever a database error occurs (we will see exceptions in the next chapter).

## 2. Database

After establishing a connection, you can execute SQL queries.

For non-prepared statements (direct execution), use the `exec()` method for SQL statements that don't return a result set (like `INSERT`, `UPDATE`, or `DELETE`) or the `query()` method for SQL statements that retrieve data.

## 2. Database

Example: Inserting data using the “exec” function

```
$sql = "INSERT INTO users (name, age, email) VALUES ('John', 33, 'john@ex.com')";  
$pdo->exec($sql);
```



## 2. Database

Example: Query the data from the database

```
foreach ($pdo->query('SELECT * FROM users') as $row) {  
    print_r($row);  
}
```

## 2. Database

Prepared statements are a better choice for executing SQL queries, especially with user-supplied data, as they prevent SQL injection attacks.

In the next example, :email is a named placeholder that will be replaced by the actual value provided in the execute() method.

## 2. Database

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE email = :email");  
$stmt->execute(['email' => 'john.doe@example.com']);  
  
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {  
    print_r($row);  
}
```

## 2. Database

The connection is automatically closed when the PDO object is destroyed, but you can explicitly close the connection by setting the PDO object to null.

```
$pdo = null;
```

## 2. Database

Most of the time you will rely on 4 main types of requests, requests that are really close to your HTTP requests.

Operation	HTTP	SQL
Create	POST	INSERT INTO
Read	GET	SELECT
Update	PUT/PATCH	UPDATE
Delete	DELETE	DELETE

## 2. Database

There is some good practices to follow when working with a database.

- **Use Prepared Statements:** Always use prepared statements for executing SQL queries with user-supplied data.
- **Exception Handling:** Use try-catch blocks for connection and query execution to handle exceptions properly.
- **Persistent Connections:** Consider using persistent connections cautiously, as they can improve performance by reusing existing connections but may lead to issues if not managed carefully.



Exercises

# 3. Exception



### 3. Exception

In PHP, Exceptions provide a modern way to handle errors through the use of the try-catch block.

Exceptions are used for error handling in object-oriented programming (OOP) and can make your code more robust and easier to maintain.

PHP's exception model is based on classes; thus, exceptions are objects created from exception classes.

### 3. Exception

An exception can be thrown, and caught ("caught") within PHP.

Throwing an exception is like creating a special object that contains information about an error or unexpected behavior.

Catching an exception is the process of responding to that error.

## 3. Exception

### Syntax

```
try {  
    // Code that may throw an exception  
    if (something goes wrong) {  
        throw new Exception("Something went wrong.");  
    }  
} catch (Exception $e) {  
    // Code to handle the exception  
    echo 'Caught exception: ', $e->getMessage(), "\n";  
}
```

### 3. Exception

We can see three key components in our exception

- try
- catch
- throw
- finally (not present in the example)

### 3. Exception

try

The try block contains code that might potentially throw an exception.

PHP executes the code within the try block as normal until an exception is thrown.

### 3. Exception

#### throw

The throw keyword is used to trigger an exception.

You can throw an instance of the Exception class or a subclass of Exception.

### 3. Exception

#### catch

The catch block is used to catch exceptions that are thrown within the try block.

When an exception is caught, the code within the catch block is executed. You can specify which exception type you want to catch.

### 3. Exception

#### finally

PHP 5.5 introduced a finally block that gets executed after the try and catch blocks, regardless of whether an exception was thrown or not.

The finally block is useful for cleaning up resources, like closing files or database connections, that may have been opened during the execution of the try block.



## 3. Exception

### Complete example

```
try {  
    // Code that may throw an exception  
} catch (Exception $e) {  
    // Code to handle the exception  
} finally {  
    // Code that will run regardless of whether an exception was thrown or not  
}
```

### 3. Exception

Using our previous PDO setup, we can have - and should have - a simple way to handle the exception. As you cannot guarantee that the database will always be accessible, you should always handle an exception if you are working with other system, like databases, files or APIs.

```
try {  
    $pdo = new PDO('mysql:host=localhost;dbname=supinfo', root, toor);  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
} catch (PDOException $e) {  
    echo "Connection failed: " . $e->getMessage();  
}
```

### 3. Exception

PHP provides a built-in Exception class, and you can create your own exception classes by extending it.

The Exception class provides several methods that you can use to retrieve information about the exception

- message
- code
- trace

### 3. Exception

`getMessage()`: Returns the message of the exception.

`getCode()`: Returns the exception code.

`getFile()`: Returns the filename in which the exception was created.

`getLine()`: Returns the line number where the exception was thrown.

`getTrace()`: Returns the stack trace as an array.

`getTraceAsString()`: Returns the stack trace as a string.



Exercises



