

Initiation serveur **PHP & SQL**

Base de données

C'EST PARTI !



Samih HABBANI

OBJECTIFS



PHP & MySQL



Introduction à MySQL et SQL

Découvrir les bases de **SQL** et **MySQL** pour créer, gérer et interroger des bases de données, en apprenant les commandes fondamentales pour manipuler efficacement les données.



Connexion PHP/MySQL avec PDO

Apprendre à utiliser **PDO (PHP Data Objects)** pour établir une connexion sécurisée entre un script PHP et une base de données **MySQL**, tout en facilitant la gestion des requêtes.



Le CRUD (Créer, Lire, Mettre à jour, Supprimer)

Mettre en place les opérations **CRUD** pour manipuler les données stockées dans une base de données, en permettant leur **ajout, lecture, modification et suppression**.



PHP & MYSQL



Système de connexion utilisateur (Login)

Créer un **système d'authentification** permettant aux utilisateurs de se connecter en toute sécurité à une application à l'aide d'un identifiant et d'un mot de passe.



Sécurisation avec Hash-Crypt

Renforcer la **sécurité des mots de passe** en utilisant des techniques de **hachage** comme **password_hash** et **password_verify**, garantissant ainsi la protection des données sensibles.



Création d'un serveur adapté au projet de fin d'année

Configurer un **environnement serveur** personnalisé pour déployer et tester l'application développée dans le cadre du **projet de fin d'année**, en assurant sa performance et sa fiabilité.

INITIATION SERVEUR PHP & MYSQL

PARTIE 1

Initiation serveur

- Qu'est-ce qu'un serveur PHP/MySQL

PARTIE 2

Intro MySQL & SQL

- Présentation de MySQL
- Installation d'un serveur web local
- MCD avec workbench
- Les bases de SQL
- Atelier pratique

PARTIE 3

Connexion PHP/MySQL

- Présentation de PDO
- Les requêtes avec PDO
- Atelier pratique

INITIATION SERVEUR PHP & MYSQL

PARTIE 4

Réaliser un CRUD

- Introduction au CRUD
- Mise en œuvre avec PDO
- Atelier pratique

PARTIE 5

Système de connexion

- Formulaire de connexion
- Vérifications des informations utilisateur
- Gestion de sessions
- Atelier pratique

PARTIE 6

Hash-Crypt

- Introduction au hashage
- Mise en pratique
- Atelier pratique

PARTIE 7

Création d'un serveur

- Conceptualisation du projet
- Déploiement du serveur local
- Atelier pratique

INITIATION SERVEUR PHP & MYSQL

⚠️ Rappel sur l'utilisation de l'IA ⚠️

L'utilisation de l'IA pour générer du contenu (texte, code, image, etc) que vous présentez comme vôtre dans un livrable est à la fois un délit de « **Plagiat** » et un délit de « **Fraude à un examen** » (contrôle continu).

Interdit

Sauf autorisation exceptionnelle et écrite

- Générer du contenu par IA et l'intégrer dans votre rendu pour évaluation

Autoriser

- Utiliser l'IA pour apprendre ou pour vous accompagner dans vos réflexions

Attention

- Ne pas prendre les informations fournies par une IA comme forcément vrai, revérifiez !

PARTIE 1

INITIATION SERVEUR PHP & MYSQL

Créer des applications web dynamiques repose sur l'interaction entre un serveur et une base de données. Dans cette partie vous allez apprendre à utiliser **PHP** et **MySQL** pour développer des sites capables de **traiter des informations**, de les **stocker** et de les **afficher** de manière efficace.

Vous découvrirez alors les bases nécessaires pour comprendre comment ces technologies fonctionnent ensemble pour offrir des solutions interactives et performantes.



QU'EST-CE QU'UN
SERVEUR PHP/MYSQL?



LES DIFFÉRENTS
SERVEURS DISPONIBLES



OBJECTIFS DE LA
FORMATION

PARTIE 1

Qu'est-ce qu'un serveur
PHP/MySQL?

QU'EST-CE QU'UN SERVEUR PHP/MYSQL?

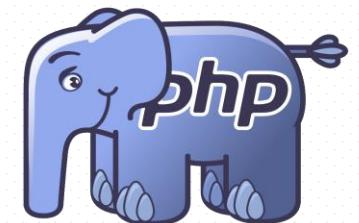
Qu'est-ce qu'un serveur PHP/MySQL ?

Un serveur PHP/MySQL est un environnement utilisé pour héberger et exécuter des applications web dynamiques.

- **PHP** : Langage de programmation côté serveur, interprète le code et génère des pages web dynamiques.
- **MySQL** : Système de gestion de bases de données relationnelles, stocke et organise les informations utilisées par les applications.

Fonctionnement

- **PHP** traite les requêtes et génère du contenu dynamique.
- **MySQL** stocke et récupère les données nécessaires au traitement des informations.
- Les deux fonctionnent ensemble pour offrir des applications interactives et connectées.



QU'EST-CE QU'UN SERVEUR PHP/MYSQL?

Serveur local vs Serveur en ligne?

Un **serveur local** est installé directement sur votre ordinateur pour **tester et développer** des applications sans connexion internet. Il est principalement utilisé pour le **développement et l'apprentissage** avant la mise en ligne d'un projet.

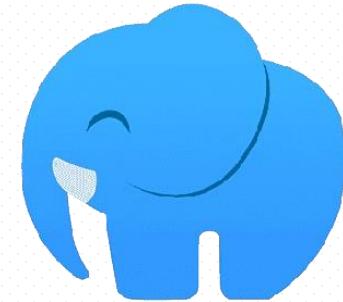
En revanche, un **serveur en ligne** est hébergé sur internet et permet de **déployer des projets accessibles au public**. Il est utilisé pour des sites en **production** et assure la **disponibilité 24/7** des applications.

Serveurs Locaux :

- XAMPP
- WAMP
- MAMP
- Laragon

Serveurs en Ligne :

- OVH
- Hostinger
- IONOS
- Bluehost



QU'EST-CE QU'UN SERVEUR PHP/MYSQL?

Le rôle de PHP pour le développement web

PHP est largement utilisé pour le développement web pour créer des sites dynamiques, gérer des formulaires, des sessions et des bases de données tout en offrant une **grande communauté de développeurs** et de nombreux **frameworks** comme **Laravel** et **Symfony** pour accélérer le développement.

Pourquoi est-il si populaire ?

- **Facile à apprendre** : Syntaxe simple et intuitive.
- **Flexible et polyvalent** : Compatible avec de nombreuses bases de données et formats.
- **Exécution côté serveur** : Génère des pages dynamiques en fonction des requêtes utilisateur.
- **Communauté active** : De nombreux frameworks (Laravel, Symfony) et ressources disponibles.
- **Interopérabilité** : Intégration facile avec HTML, JavaScript et CSS.

Exemple d'utilisation :

- Création de sites web dynamiques.
- Gestion des sessions et des utilisateurs.
- Intégration d'API et de systèmes de paiement.

QU'EST-CE QU'UN SERVEUR PHP/MYSQL?

Présentation de PDO pour la gestion des bases de données

PDO (pour PHP Data Objects) est une extension PHP qui permet de se connecter à une base de données et d'exécuter des requêtes SQL.

Les avantages de PDO :

- **Sécurité** : Protège contre les injections SQL avec des requêtes préparées.
- **Portabilité** : Compatible avec plusieurs types de bases de données (MySQL, PostgreSQL, SQLite).
- **Flexibilité** : Simplifie la gestion des erreurs et des connexions.

Exemple de connexion :

```
try {
    $pdo = new PDO('mysql:host=localhost;dbname=testdb', 'root', 'password');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connexion réussie !";
} catch (PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
```

PARTIE 2

INTRODUCTION À MYSQL & SQL

Dans cette partie vous découvrirez **MySQL** et **SQL** qui sont deux outils incontournables pour **créer, gérer et interroger** des bases de données.

Ils sont aujourd'hui très utilisés pour stocker et organiser les informations dans les applications web modernes sur internet.



PRÉSENTATION DE
MYSQL



LES BASES DE SQL



EXERCICE PRATIQUE

INTRODUCTION À MySQL & SQL

PARTIE 1

Présentation de MySQL

PRÉSENTATION DE MYSQL

La différence entre MySQL et SQL

SQL est un **langage** utilisé pour manipuler et interroger des bases de données, tandis que **MySQL** est un **système de gestion de bases de données** qui utilise **SQL** pour fonctionner. **SQL** sert à écrire des commandes pour créer, modifier ou interroger des données, alors que **MySQL** est l'outil qui exécute ces commandes et stocke les informations.

En résumé, **SQL** est le langage et **MySQL** est le logiciel qui l'interprète et le met en application.

SQL	MySQL
Langage de requêtes	Logiciel de gestion de base de données
Standardisé et universel	Sépcifique, utilise SQL pour fonctionner
Utiliser pour communiquer avec plusieurs types de SGBD	SGBD populaire pour gérer les bases de données relationnelles

PRÉSENTATION DE MYSQL

Qu'est-ce qu'une base de données relationnelle?

Une **base de données relationnelle** est un système qui organise et stocke des informations sous forme de **tables**. Chaque table est composée de **lignes** (enregistrements) et de **colonnes** (champs), ce qui permet de structurer les données de manière claire et efficace.

Les tables peuvent être **liées entre elles** à l'aide de **clés primaires** et de **clés étrangères**, ce qui facilite la gestion des relations entre les informations. Ce modèle est largement utilisé pour garantir la **cohérence, la sécurité et la fiabilité** des données.



PRÉSENTATION DE MYSQL

Propriétés et types

Les **propriétés** définissent les **caractéristiques internes** d'une colonne, telles que son **type de données**, sa **taille** ou son **comportement par défaut**. Elles déterminent **comment** les données sont stockées et gérées.

Type de données : nom **VARCHAR(50)** -- Chaîne de caractères de 50 caractères max.

Valeur par défaut : statut **VARCHAR(20) DEFAULT 'actif'**

Auto-incrémentation : id **INT AUTO_INCREMENT**

PRÉSENTATION DE MYSQL

Les différents types de données les plus courants en SQL

Les types numériques :

- **INT** – Entier pour des nombres sans décimales. (*Ex : âge, identifiants*)
- **FLOAT** – Nombre à virgule flottante pour des valeurs décimales. (*Ex : poids, température*)
- **DECIMAL(p, d)** – Nombre décimal précis avec un nombre défini de chiffres avant et après la virgule. (*Ex : prix, salaires*)

Les types chaînes de caractères :

- **VARCHAR(n)** – Chaîne de longueur variable (jusqu'à n caractères). (*Ex : noms, emails*)
- **TEXT** – Texte de grande taille pour des descriptions ou commentaires. (*Ex : bio, articles*)

PRÉSENTATION DE MYSQL

Les différents types de données les plus courants en SQL

Les types Date et Heure :

- **DATE** – Stocke uniquement la date (format : YYYY-MM-DD). (*Ex : date de naissance*)
- **DATETIME** – Stocke la date et l'heure (format : YYYY-MM-DD HH:MM:SS). (*Ex : date de création*)
- **TIMESTAMP** – Date et heure avec mise à jour automatique. (*Ex : dernière modification*)

Les types Booléens :

- **BOOLEAN** ou **BOOL** – Valeurs logiques **true (1)** ou **false (0)**. (*Ex : actif/inactif*)

Pour plus d'informations sur les types :

<https://dev.mysql.com/doc/refman/8.4/en/data-types.html>

PRÉSENTATION DE MYSQL

Contraintes

Les **contraintes** en SQL sont des règles appliquées aux colonnes d'une table pour **garantir l'intégrité et la validité des données**. Elles empêchent l'insertion ou la modification de valeurs qui ne respectent pas certaines conditions.

Par exemple, une **clé primaire (PRIMARY KEY)** garantit qu'une colonne contient des valeurs **uniques et non nulles**, tandis qu'une **clé étrangère (FOREIGN KEY)** assure la **relation** entre deux tables.

D'autres contraintes, comme **UNIQUE**, **NOT NULL** et **CHECK**, permettent de contrôler l'unicité, l'obligation de renseigner une valeur ou encore la validation des valeurs saisies.

Ces règles sont essentielles pour maintenir une base de données fiable et cohérente.

Pour plus d'informations sur les contraintes :

<https://learn.microsoft.com/fr-fr/sql relational-databases/tables/unique-constraints-and-check-constraints?view=sql-server-ver16>



PRÉSENTATION DE MYSQL

Les contraintes les plus courantes

Clé primaire :

```
id INT PRIMARY KEY
```

Valeurs uniques :

```
email VARCHAR(100) UNIQUE
```

Valeurs non nulles :

```
nom VARCHAR(50) NOT NULL
```

Clé étrangère avec contraintes de relation :

```
client_id INT,  
FOREIGN KEY (client_id) REFERENCES clients(id) ON DELETE CASCADE
```

PRÉSENTATION DE MYSQL

En résumé

Propriétés	Contraintes
Langage de requêtes	Logiciel de gestion de base de données
Standardisé et universel	Sépcifique, utilise SQL pour fonctionner
Utiliser pour communiquer avec plusieurs types de SGBD	SGBD populaire pour gérer les bases de données relationnelles

PRÉSENTATION DE MYSQL

Clef primaire (PK) et clef étrangère (FK)

Une **clé primaire (PK)** est un identifiant **unique** pour chaque ligne d'une table. Elle ne peut pas être **nulle** et garantit que chaque enregistrement est distinct.

Une **clé étrangère (FK)** est une colonne qui crée un **lien** entre deux tables en faisant référence à la **clé primaire** d'une autre table. Elle permet de **relier les données** et d'assurer leur **cohérence**.

- Table **utilisateur**

id (PK)	nom
1	Jean
2	Marie

- Table **commandes**

id (PK)	utilisateur_id	produit
1	1	Livre
2	2	Stylo

Ici, la colonne **utilisateur_id** est une **clé étrangère** qui référence à **id** clef primaire dans la table **utilisateurs**, créant ainsi un lien entre les **commandes** et les **utilisateurs**.

INTRODUCTION À MYSQL & SQL

PARTIE 2

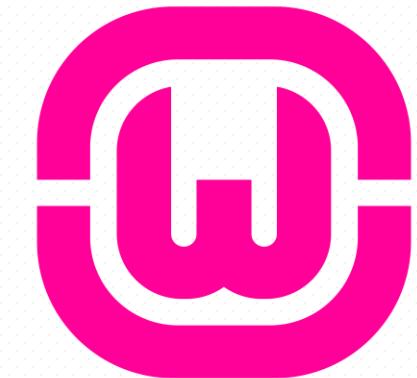
Installation d'un serveur web local

PRÉSENTATION DE MYSQL

Installation de phpMyAdmin avec WAMP

Étape 1 : Télécharger et Installer WAMP

- Rendez-vous sur le site officiel de **WAMP** : www.wampserver.com.
- Téléchargez la version compatible avec votre système (32 ou 64 bits).
- Installez WAMP en suivant les instructions.



Étape 2 : Lancer WAMP

- Ouvrez **WampServer** après l'installation.
- Vérifiez que l'icône dans la barre des tâches devient **verte** (signifie que les services sont actifs).

Étape 3 : Accéder à phpMyAdmin

- Ouvrez votre navigateur web.
- Tapez dans la barre d'adresse : `http://localhost/phpmyadmin`

PRÉSENTATION DE MYSQL

Installation de phpMyAdmin avec WAMP

Connectez-vous avec :

- **Nom d'utilisateur :** root
- **Mot de passe :** (laisser vide par défaut).

Étape 4 : Créer une base de données

- Cliquez sur **Nouvelle base de données**.
- Entrez un nom et cliquez sur **Créer**.



PRÉSENTATION DE MYSQL

Introduction à phpMyAdmin

PhpMyAdmin est un outil **gratuit et open source** qui permet de gérer facilement des bases de données **MySQL** et **MariaDB** via une **interface web**. Il est largement utilisé pour :

- **Créer, modifier et supprimer** des bases de données et des tables.
- **Exécuter des requêtes SQL** pour manipuler les données.
- **Gérer les utilisateurs** et leurs privilèges.
- **Importer et exporter** des bases de données.
- **Sauvegarder et restaurer** des données rapidement.

Il simplifie la gestion des bases de données sans nécessiter de compétences avancées en ligne de commande.

PRÉSENTATION DE MYSQL

Illustration d'une table dans phpMyAdmin

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id_logement 	int(11)			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	2 titre	varchar(225)	utf8_general_ci		Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	3 adresse	varchar(225)	utf8_general_ci		Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	4 ville	varchar(75)	utf8_general_ci		Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	5 cp	varchar(5)	utf8_general_ci		Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	6 surface	int(4)			Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	7 prix	int(11)			Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	8 photo	varchar(225)	utf8_general_ci		Oui	NULL			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	9 type	enum('location', 'vente')	utf8_general_ci		Non	Aucun(e)			 Modifier  Supprimer ▾ Plus
<input type="checkbox"/>	10 description	text	utf8_general_ci		Oui	NULL			 Modifier  Supprimer ▾ Plus

INTRODUCTION À MYSQL & SQL

PARTIE 3

Concéptualiser une base de
données avec Workbench

WORKBENCH

Introduction à MySQL Workbench

MySQL Workbench est un **outil graphique** développé par **Oracle** pour gérer et administrer des bases de données **MySQL**. Il permet de :

- **Créer, modifier et gérer** des bases de données.
- **Exécuter des requêtes SQL** facilement.
- **Concevoir des modèles de bases de données (MCD).**
- **Visualiser les relations entre les tables** avec des diagrammes.



Qu'est-ce qu'un MCD ? (Modèle Conceptuel de Données)

Un **MCD** est un **schéma visuel** qui représente les **données** d'un système et leurs **relations**.

- Il sert à **organiser** les informations sous forme de **tables** et de **liens** (relations).
- Il aide à **comprendre la structure** d'une base de données avant sa création.
- Il permet de **générer automatiquement le code SQL** d'une base de données sur Workbench

WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

1. Installer MySQL Workbench :

Téléchargez et installez-le depuis le site officiel : dev.mysql.com.

2. Ouvrir MySQL Workbench :

Lancez l'application et cliquez sur **Database > Connect to Database** pour établir la connexion.

3. Créer un MCD :

- Cliquez sur **File > New Model** pour créer un nouveau modèle.
- Sélectionnez **Add Diagram** pour ouvrir l'éditeur visuel.
- Utilisez l'outil **Table** pour ajouter des tables.
- Reliez les tables avec des **liens (Foreign Keys)** en utilisant l'outil **Relationship**.

4. Configurer les colonnes et les clés :

- Définissez les **colonnes**, les **types de données** et les **contraintes** (PRIMARY KEY, FOREIGN KEY).
- Ajoutez des relations en dessinant des lignes entre les tables.

WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

5. Exporter le MCD :

Cliquez sur **File > Export** pour enregistrer ou générer le schéma SQL.

6. Exporter le code SQL :

Cliquez sur **File > Export > Forward Engeneer SQL CREATE Script**

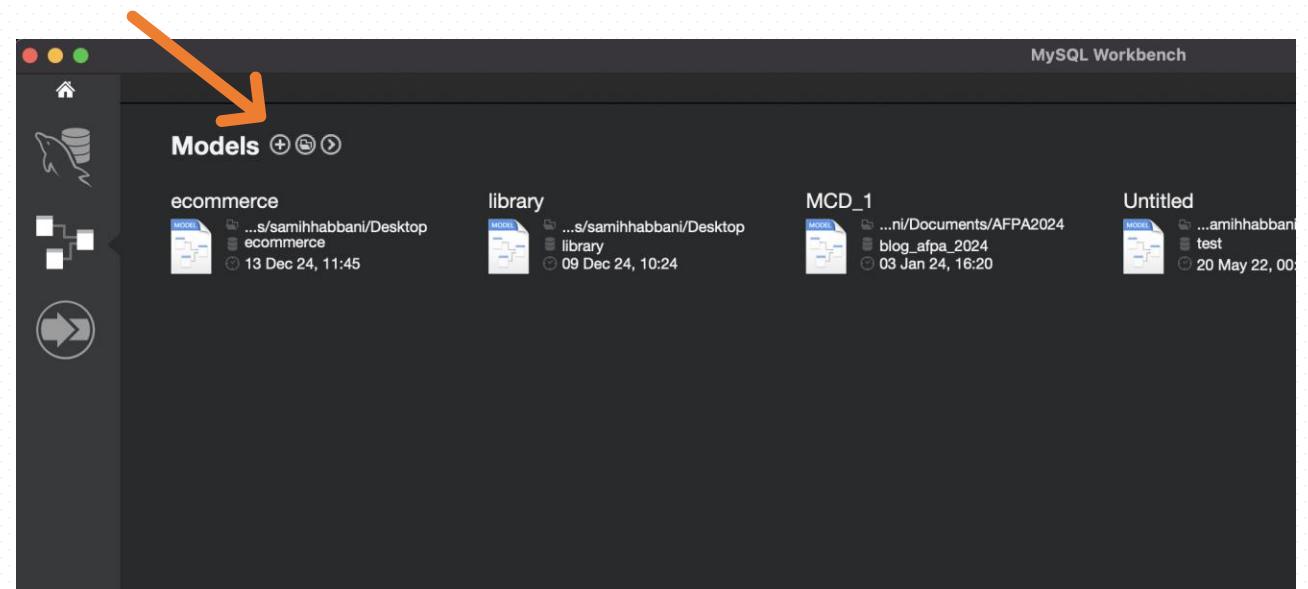
Pourquoi utiliser Workbench?

MySQL Workbench est un outil essentiel pour **visualiser, construire et documenter** nos bases de données. La création d'un **MCD** facilite la planification et la conception d'une base, garantissant une structure claire et optimisée. De plus, une fois le MCD généré, Workbench nous permet de récupérer le code SQL automatiquement généré basé sur les diagrammes représentant nos tables et leurs relations.

WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

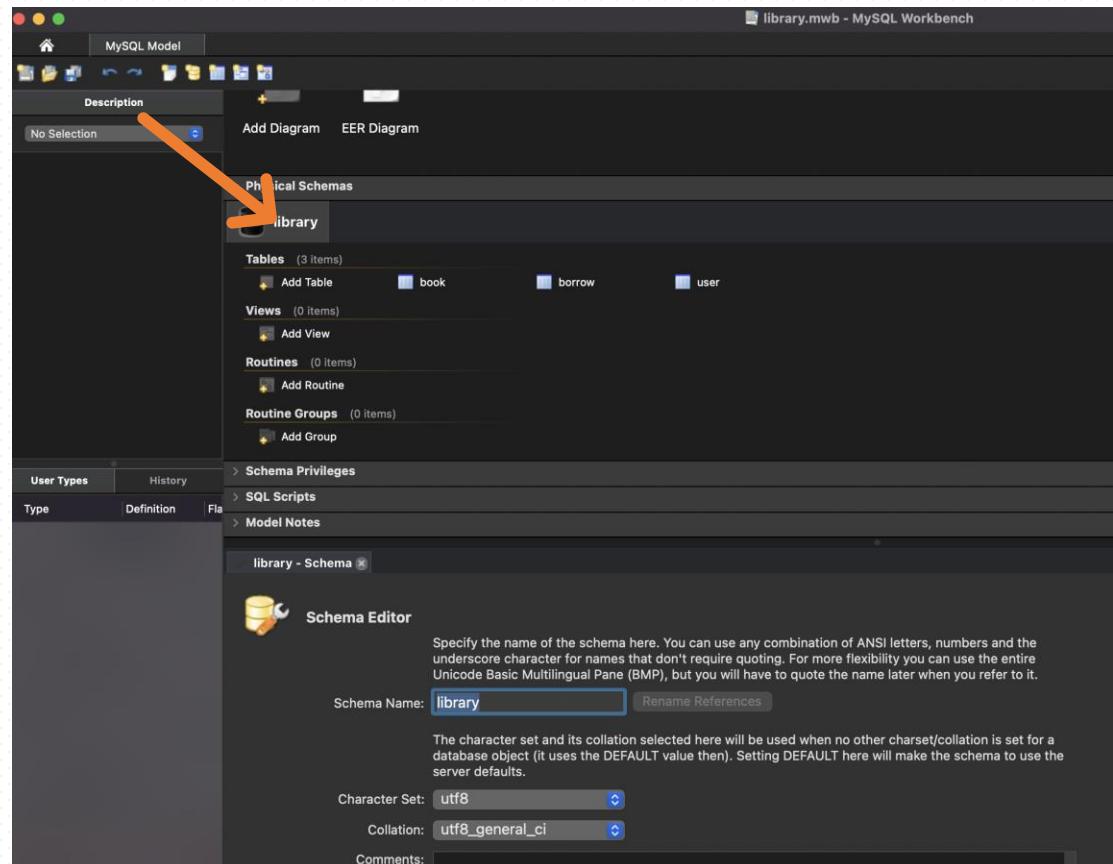
- Créer ton MCD



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

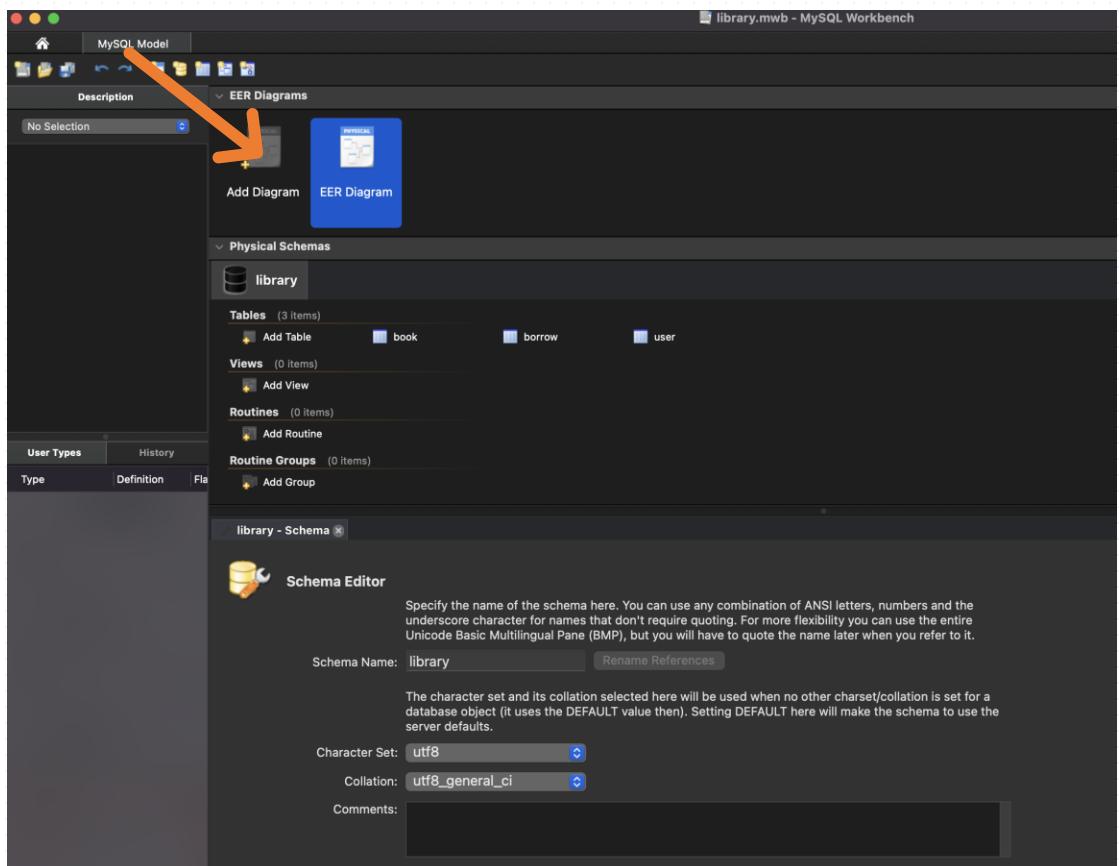
- Définir le nom de ta base de données



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

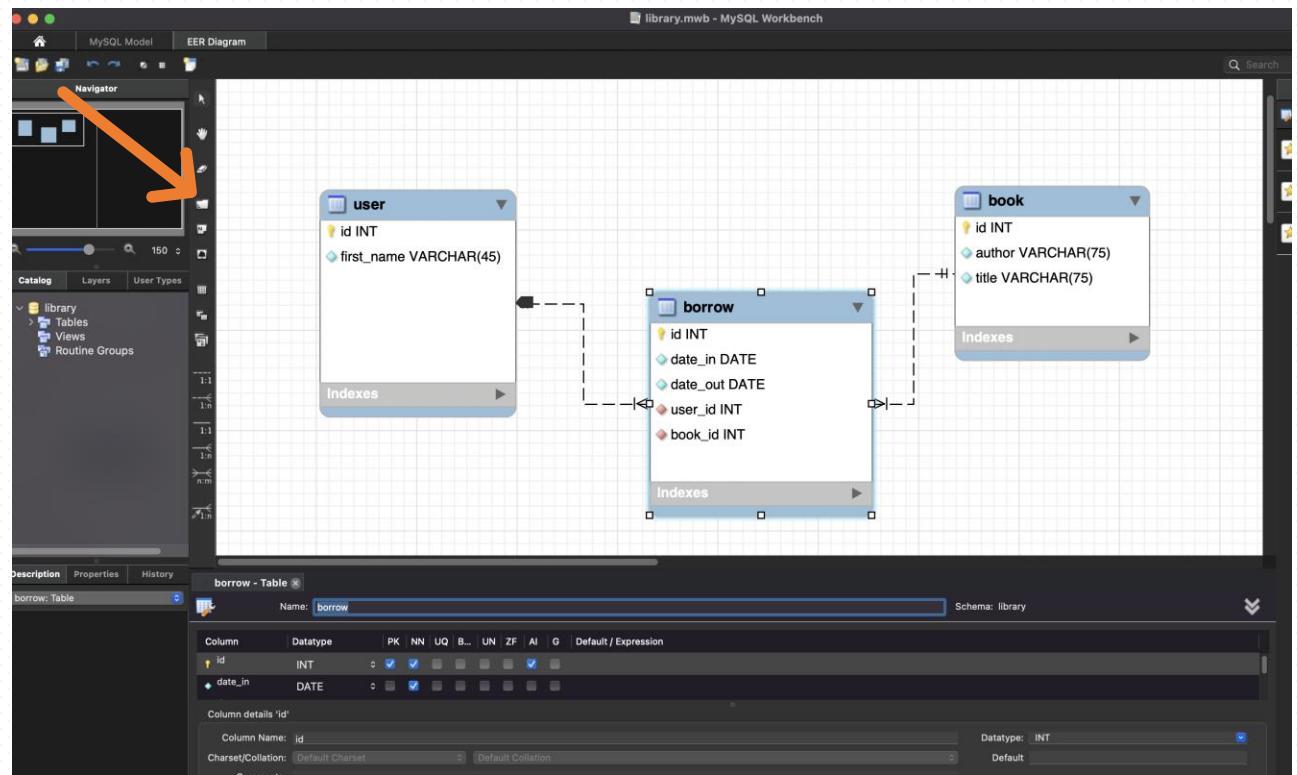
- Créer un diagramme qui représentera tes tables et leurs relations



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

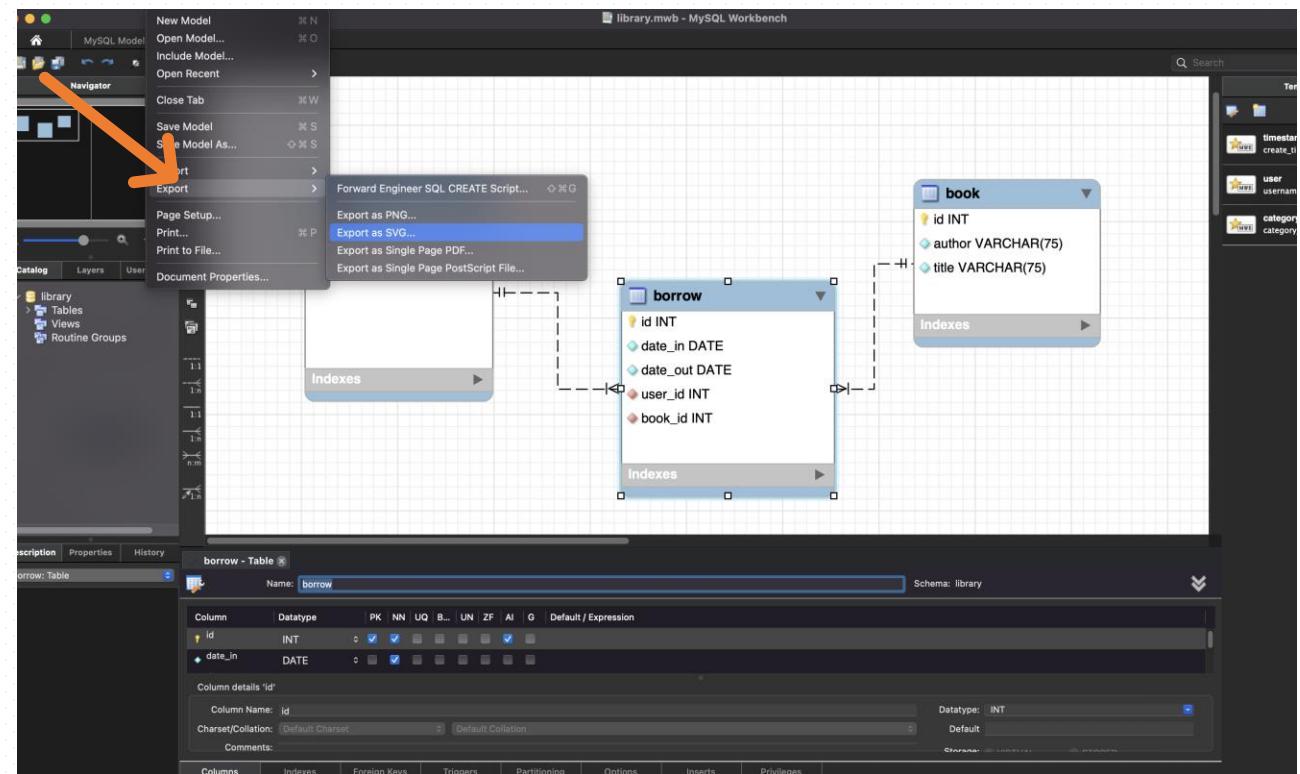
- Créer tes tables et leurs colonnes, avec leurs propriétés et contraintes



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

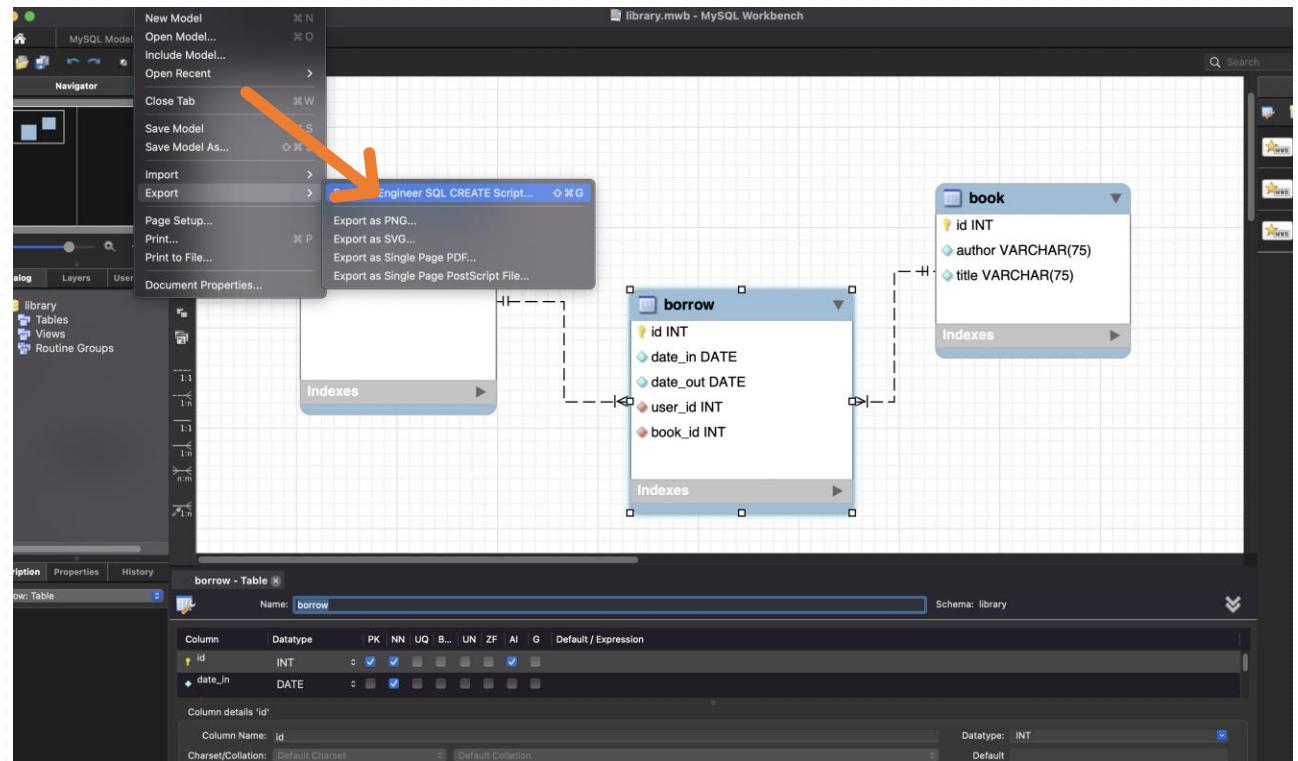
- Exporter le plan de ta base de données ou MCD



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

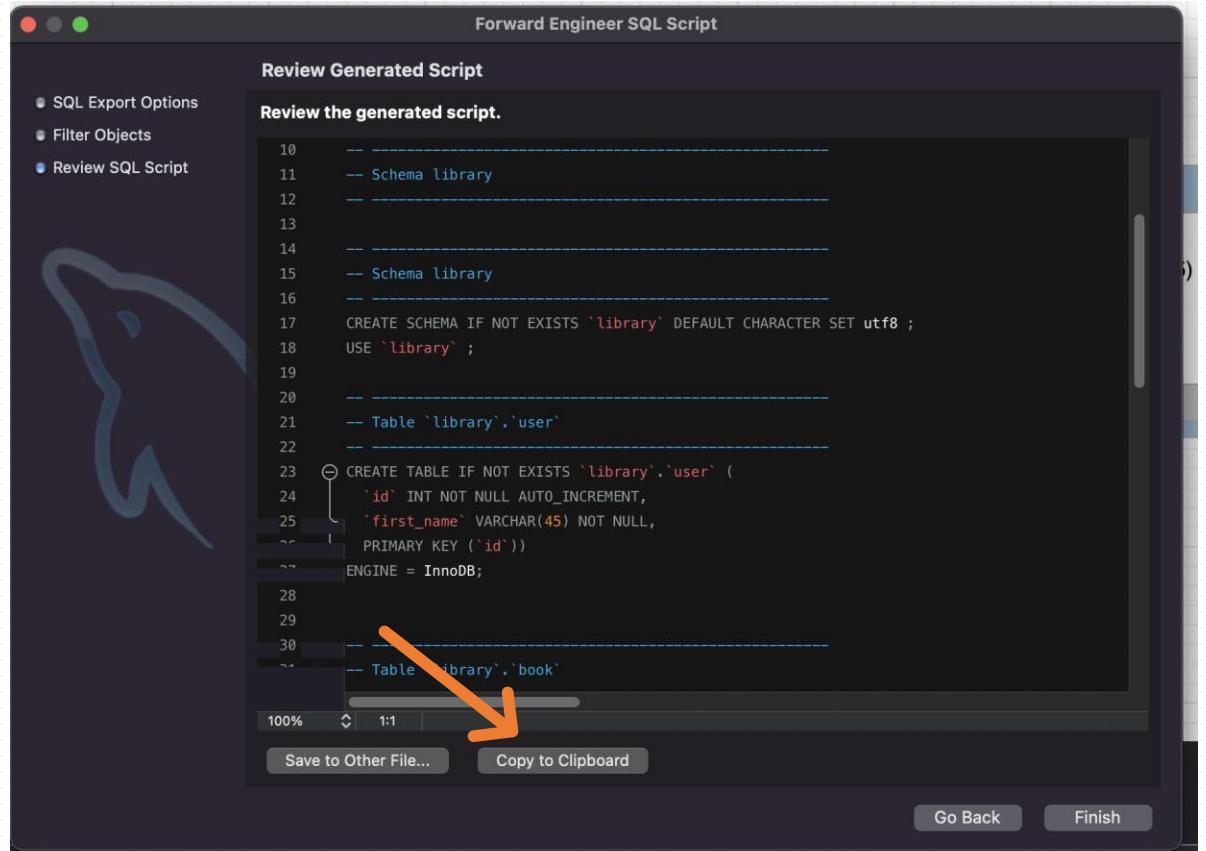
- Exporter le code SQL généré par la création de ton MCD



WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

- Récupérer le code SQL pour l'exécuter sur phpMyAdmin



The screenshot shows the "Forward Engineer SQL Script" dialog in MySQL Workbench. The title bar says "Forward Engineer SQL Script". The main area is titled "Review Generated Script" with the sub-instruction "Review the generated script.". The SQL code generated for a schema named "library" is displayed:

```
10  -- Schema library
11  -- Schema library
12
13
14
15  -- Schema library
16
17  CREATE SCHEMA IF NOT EXISTS `library` DEFAULT CHARACTER SET utf8 ;
18  USE `library` ;
19
20  -- Table `library`.`user`
21
22
23  CREATE TABLE IF NOT EXISTS `library`.`user` (
24      `id` INT NOT NULL AUTO_INCREMENT,
25      `first_name` VARCHAR(45) NOT NULL,
26      PRIMARY KEY (`id`)
27      ENGINE = InnoDB;
28
29
30  -- Table `library`.`book`
```

An orange arrow points from the bottom right towards the "Copy to Clipboard" button. At the bottom of the dialog, there are buttons for "Save to Other File...", "Copy to Clipboard", "Go Back", and "Finish".

WORKBENCH

Créer un MCD avec MySQL Workbench et récupérer le code SQL généré

- Exécuter le script SQL pour créer vos tables dans phpMyAdmin

The screenshot shows the phpMyAdmin interface. On the left, the database structure is displayed under the 'library' database, including tables like 'borrow', 'book', 'user', etc. In the center, a SQL query editor window is open, showing a CREATE TABLE statement for the 'borrow' table. The statement includes columns for 'id', 'date_in', 'date_out', 'user_id', and 'book_id', with various constraints and indexes. An orange arrow points to the 'Exécuter' (Execute) button at the bottom right of the query editor.

```
41 -->
42 -- Table `library`.`borrow`
43 --
44 CREATE TABLE IF NOT EXISTS `library`.`borrow` (
45   `id` INT NOT NULL AUTO_INCREMENT,
46   `date_in` DATE NOT NULL,
47   `date_out` DATE NOT NULL,
48   `user_id` INT NOT NULL,
49   `book_id` INT NOT NULL,
50   PRIMARY KEY (`id`),
51   INDEX `fk_borrow_user_idx` (`user_id` ASC),
52   INDEX `fk_borrow_book1_idx` (`book_id` ASC),
53   CONSTRAINT `fk_borrow_user`
54     FOREIGN KEY (`user_id`)
55       REFERENCES `library`.`user` (`id`)
56     ON DELETE NO ACTION
```

INTRODUCTION À MYSQL & SQL

PARTIE 4

Les bases de SQL

LES BASES DE SQL

La syntaxe et les bonnes pratiques

SQL suit une structure claire et précise pour écrire des requêtes. Une commande SQL commence par un **mot-clé** et se termine par un **point-virgule (;)**.

```
SELECT * FROM utilisateurs; -- Affiche toutes les colonnes et lignes de la table utilisateurs.
```

Noms de tables et de colonnes :

- Doivent commencer par une **lettre** et peuvent contenir des **chiffres**, des **soulignements (_)**, mais **pas d'espaces**.
- Les **mots réservés** (ex : SELECT, WHERE) ne peuvent pas être utilisés comme noms.
- Exemple valide : nom_utilisateur
- Exemple non valide : nom utilisateur (avec espace).

Sensibilité à la casse :

- Les mots-clés SQL (SELECT, WHERE, etc.) ne sont **pas sensibles à la casse**.
- Mais les **noms de tables et colonnes** peuvent être sensibles selon la configuration du serveur.

LES BASES DE SQL

Créer une table en SQL

Pour créer une table, on utilise la commande **CREATE TABLE** en spécifiant :

- Le **nom de la table**.
- Les **colonnes** avec leurs **types de données** et leurs **contraintes**.

```
CREATE TABLE utilisateurs (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Identifiant unique
    nom VARCHAR(50) NOT NULL,          -- Nom obligatoire
    email VARCHAR(100) UNIQUE,         -- Email unique
    age INT,                          -- Âge facultatif
    date_inscription DATE DEFAULT CURRENT_DATE -- Date par défaut
);
```

LES BASES DE SQL

Les commandes principales – SELECT, FROM, WHERE

La commande **SELECT** permet de **lire** ou **récupérer des informations** stockées dans une table. Elle est souvent utilisée avec des **filtres** et des **conditions** pour afficher des résultats spécifiques.

Exemple :

```
SELECT * FROM utilisateurs; -- Affiche toutes les colonnes et lignes de la table utilisateurs.
```

La clause **FROM** indique **de quelle table** les données doivent être récupérées.

Elle est obligatoire dans une requête **SELECT**.

La clause **WHERE** permet de **filtrer les résultats** en appliquant des **conditions** sur les données sélectionnées.

```
SELECT nom, email FROM utilisateurs WHERE age > 18;
```

LES BASES DE SQL

Les commandes principales – INSERT

La commande **INSERT** permet d'**ajouter** de nouvelles lignes dans une table en spécifiant les colonnes et leurs valeurs correspondantes.

```
INSERT INTO utilisateurs (nom, email, age)
VALUES ('Jean Dupont', 'jean.dupont@email.com', 25);
```

Ici, un nouvel utilisateur est ajouté avec un **nom**, un **email** et un **âge**.

L'id de l'utilisateur n'a pas été précisé car il s'auto-incrémente en base de données.

LES BASES DE SQL

Les commandes principales – UPDATE

La commande **UPDATE** est utilisée pour **mettre à jour** ou **modifier** des valeurs dans une table existante.

```
UPDATE utilisateurs  
SET email = 'nouveau.email@email.com'  
WHERE id = 1;
```

Ce code met à jour l'**email** de l'utilisateur avec l'identifiant (**id**) **1**.

⚠ **Attention**, sans clause **WHERE**, toutes les lignes peuvent être mises à jour par erreur.

LES BASES DE SQL

Les commandes principales – DELETE

La commande **DELETE** permet de **supprimer** des lignes dans une table, en fonction d'une condition.

```
DELETE FROM utilisateurs WHERE id = 1;
```

Ce code supprime l'utilisateur dont l'identifiant est **1**.

⚠ **Attention**, si la condition **WHERE** est omise, **toutes les lignes** de la table seront supprimées !

LES BASES DE SQL

Les clauses principales - WHERE

Comme nous l'avons vu précédemment, La clause **WHERE** permet de **filtrer les lignes** d'une table en fonction d'une ou plusieurs **conditions**. Elle est utilisée avec les commandes **SELECT**, **UPDATE** et **DELETE** pour afficher ou modifier uniquement les données qui respectent les critères spécifiés.

```
-- Filtrer par condition simple
SELECT * FROM utilisateurs WHERE age > 18;

-- Combiner plusieurs conditions
SELECT * FROM utilisateurs WHERE age > 18 AND statut = 'actif';
```

LES BASES DE SQL

Les clauses principales – ORDER BY

La clause **ORDER BY** permet de **trier les résultats** d'une requête SQL par une ou plusieurs colonnes, en ordre **croissant (ASC)** ou **décroissant (DESC)**.

```
-- Trier par ordre alphabétique
```

```
SELECT * FROM utilisateurs ORDER BY nom ASC;
```

```
-- Trier par âge décroissant
```

```
SELECT * FROM utilisateurs ORDER BY age DESC;
```

```
-- Trier par plusieurs colonnes
```

```
SELECT * FROM utilisateurs ORDER BY statut ASC, age DESC;
```

LES BASES DE SQL

Les clauses principales – GROUP BY

La clause **GROUP BY** est utilisée pour **regrouper des lignes** ayant des valeurs identiques dans une ou plusieurs colonnes et **appliquer des fonctions d'agrégation** comme **COUNT, SUM, AVG, MAX, ou MIN.**

```
-- Compter le nombre d'utilisateurs par statut
SELECT statut, COUNT(*) AS total
FROM utilisateurs
GROUP BY statut;
```

LES BASES DE SQL

Les clauses principales – GROUP BY

La clause **HAVING** est utilisée pour **filtrer** les résultats des **groupes** créés avec **GROUP BY**. Contrairement à **WHERE**, qui filtre les lignes avant le regroupement, **HAVING** filtre les résultats **après l'agrégation**.

```
-- Afficher uniquement les groupes avec plus de 5 utilisateurs
SELECT statut, COUNT(*) AS total
FROM utilisateurs
GROUP BY statut
HAVING total > 5;
```

INTRODUCTION À MYSQL & SQL

PARTIE 5

Atelier pratique

ATELIER PRATIQUE

Exercice – Créer une base de données "library"

Créer une base de données :

- Nommez la base de données et préparez-la pour gérer un catalogue de **livres**.

Créer une table :

- Ajoutez une table pour gérer les **livres** avec les colonnes suivantes :
 - **id** (clé primaire et auto-incrémentée).
 - **titre** (obligatoire).
 - **auteur** (obligatoire).
 - **annee_publication** (année).
 - **disponible** (booléen pour indiquer si le livre est disponible).

Insérer des données :

- Ajoutez au moins **quatre livres** avec des informations variées.

Lire les données :

- Affichez tous les livres dans la table.

Filtrer et trier :

- Sélectionnez uniquement les livres publiés après **2000** et triez-les par **titre** en ordre alphabétique.



PARTIE 3

CONNEXION PHP/MYSQL AVEC PDO

Cette section aborde la manière d'établir une connexion entre **PHP** et une base de données **MySQL** en utilisant **PDO**. Cet objet vous permettra d'**exécuter des requêtes SQL** et de gérer les données de manière sécurisée et flexible.



PRÉSENTATION DE PDO



LES REQUÊTES AVEC PDO



EXERCICE PRATIQUE

CONNEXION PHP/MYSQL AVEC PDO

PARTIE 1

Présentation de PDO

PRÉSENTATION DE PDO

Pourquoi utiliser PDO au lieu de mysqli?

PDO (PHP Data Objects) est une interface permettant de se connecter à des bases de données et d'exécuter des requêtes SQL en PHP.

Contrairement à **MySQLi**, qui est spécifique à MySQL, **PDO** est **compatible avec plusieurs types de bases de données** (MySQL, PostgreSQL, SQLite, etc.).

Il offre également des fonctionnalités avancées comme les **requêtes préparées** pour renforcer la sécurité contre les **injections SQL**. Son approche **orientée objet** et sa **portabilité** en font un choix plus flexible et évolutif pour les projets nécessitant une gestion robuste des bases de données.

Pour plus d'informations sur les types :

<https://www.php.net/manual/fr/book pdo.php>

PRÉSENTATION DE PDO

Connexion à une base de données avec PDO

Pour se connecter à une base de données **MySQL** avec **PDO**, on utilise un objet **PDO** avec trois paramètres :

- **DSN (Data Source Name)** : Informations sur le type de base de données, l'hôte et le nom de la base.
- **Nom d'utilisateur** : Identifiant pour accéder à la base.
- **Mot de passe** : Mot de passe associé à l'utilisateur.

```
try {  
    $pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Gestion des erreurs  
    echo "Connexion réussie !";  
} catch (PDOException $e) {  
    echo "Erreur : " . $e->getMessage(); // Affiche un message en cas d'erreur  
}
```

PRÉSENTATION DE PDO

Explications

- **mysql:host=localhost;dbname=ma_base** : Définit l'hôte et le nom de la base de données.
- **root** et '' : Identifiants de connexion (nom d'utilisateur et mot de passe, 'root' sur mac).
- **setAttribute** : Active la gestion des erreurs pour afficher des messages clairs.
- **try...catch** : Gère les exceptions pour éviter que le script ne plante en cas d'erreur.

```
try {  
    $pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Gestion des erreurs  
    echo "Connexion réussie !";  
} catch (PDOException $e) {  
    echo "Erreur : " . $e->getMessage(); // Affiche un message en cas d'erreur  
}
```

CONNEXION PHP/MYSQL AVEC PDO

PARTIE 2

Les requêtes avec PDO

LES REQUÊTES AVEC PDO

Exécution de requêtes : query() + fetch()

La méthode **query()** exécute une **requête SQL simple** et renvoie un **objet PDOStatement** contenant les résultats. Elle est utilisée pour des requêtes sans paramètres comme **SELECT**.

```
$pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');
$result = $pdo->query("SELECT * FROM utilisateurs"); // OBJ PDO STATEMENT
```

Un **objet PDOStatement** représente le **résultat d'une requête SQL**. Il contient des méthodes pour **parcourir** et **récupérer** les données retournées par la requête.

Dans cet exemple, **\$result** est un **objet PDOStatement** qui permet d'exploiter les résultats avec des méthodes comme **fetch()** et **fetchAll()**.

LES REQUÊTES AVEC PDO

Exécution de requêtes : fetch() vs fetchAll()

La méthode **fetch()** retourne **une seule ligne** à la fois sous forme de tableau ou d'objet.

```
$row = $stmt->fetch(PDO::FETCH_ASSOC);
```

PDO::FETCH_ASSOC : Retourne les données sous forme de tableau associatif, avec des clés correspondant aux noms des colonnes.

La méthode **fetchAll()** retourne **toutes les lignes** du résultat sous forme d'un tableau.

```
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

```
Array (
...
    "id" => 1,
    "nom" => "Jean",
    "email" => "jean@email.com"
)
```

```
Array (
...
    [0] => Array (
        "id" => 1,
        "nom" => "Jean",
        "email" => "jean@email.com"
    ),
    [1] => Array (
        "id" => 2,
        "nom" => "Marie",
        "email" => "marie@email.com"
    )
)
```

LES REQUÊTES AVEC PDO

Les modes de récupération des Données :

Mode	Description
PDO::FETCH_ASSOC	Retourne un tableau associatif avec des clés sur les colonnes
PDO::FETCH_NUM	Retourne un tableau indexé numériquement
PDO::FETCH_BOTH	Retourne à la fois un tableau associatif et numéroté (par défaut)
PDO::FETCH_OBJ	Retourne un objet avec des propriétés correspondant aux colonnes

En conclusion :

La méthode **query()** associée aux objets **PDOStatement** et aux méthodes **fetch()** et **fetchAll()** permet de lire facilement les données issues d'une base MySQL. Avec les modes de récupération comme **PDO::FETCH_ASSOC**, elle offre une grande flexibilité pour manipuler les résultats sous différents formats par exemple en associant nom des colonnes et valeurs dans des tableaux PHP.

LES REQUÊTES AVEC PDO

Exécution de requêtes : exec()

La méthode **exec()** en PDO est utilisée pour **exécuter des requêtes SQL qui ne retournent pas de résultats** (comme **INSERT**, **UPDATE** ou **DELETE**).

```
$pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');

// Ajouter un nouvel utilisateur
$sql = "INSERT INTO utilisateurs (nom, email, age) VALUES ('Alice', 'alice@email.com', 25)";
$pdo->exec($sql);
```

Ce que fait **exec()** :

- Exécute la requête.
- Retourne le **nombre de lignes affectées**.
- Ne récupère pas de données.

LES REQUÊTES AVEC PDO

Exécution de requêtes : exec()

La méthode **exec()** retourne un **entier** indiquant combien de lignes ont été affectées par la requête.

```
$rows = $pdo->exec("UPDATE utilisateurs SET age = 30 WHERE id = 1");
echo "Nombre de lignes mises à jour : $rows";
```

Lorsque vous insérez une nouvelle ligne avec une **clé primaire auto-incrémentée**, vous pouvez récupérer son **ID généré** avec **lastInsertId()**.

```
$sql = "INSERT INTO utilisateurs (nom, email, age) VALUES ('Bob', 'bob@email.com', 28)";
$pdo->exec($sql);
$id = $pdo->lastInsertId(); // Récupère l'ID généré
echo "Nouvel ID inséré : $id";
```

LES REQUÊTES AVEC PDO

Exécution de requêtes : `prepare()` et `execute()`

Ces deux méthodes permettent d'exécuter des **requêtes SQL sécurisées** et d'éviter les **injections SQL** en utilisant des **requêtes préparées**.

- **prepare()** : Prépare la requête SQL avec des **paramètres dynamiques**.
- **execute()** : Exécute la requête préparée avec des **valeurs sécurisées** fournies au moment de l'exécution.

Pourquoi Utiliser `prepare()` et `execute()` ?

- **Sécurité** : Protège contre les **injections SQL** en séparant la requête et les données.
- **Réutilisation** : Permet d'exécuter plusieurs fois la même requête avec des valeurs différentes.
- **Performance** : Optimise l'exécution des requêtes, surtout dans le cas de requêtes répétées.

LES REQUÊTES AVEC PDO

Exécution de requêtes : `prepare()` et `execute()`

- Insertion de Données

```
// Préparer la requête avec des paramètres
$stmt = $pdo->prepare("INSERT INTO utilisateurs (nom, email, age) VALUES (:nom, :email, :age)");

// Exécuter la requête avec des valeurs sécurisées
$stmt->execute([
    'nom' => 'Alice',
    'email' => 'alice@email.com',
    'age' => 25
]);

echo "Utilisateur ajouté avec succès !";
```

- Lecture avec Filtres

```
$stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE age > :age");
$stmt->execute(['age' => 18]);

$result = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($result as $row) {
    echo $row['nom'] . ' - ' . $row['email'] . '<br>';
}
```

LES REQUÊTES AVEC PDO

Exécution de requêtes : `prepare()` et `execute()`

Les méthodes **prepare()** et **execute()** sont essentielles pour écrire des requêtes SQL **sécurisées** et **performantes**. Elles permettent d'intégrer des valeurs dynamiques tout en garantissant la **protection des données** et en simplifiant la gestion des paramètres.

- **Paramètres Dynamiques** : On peut réutiliser la même requête avec différentes valeurs.
- **Sécurité Renforcée** : Empêche l'exécution de commandes malveillantes (injections SQL).
- **Flexibilité** : Compatible avec différents types de requêtes (INSERT, UPDATE, DELETE, SELECT).
- **Simplicité** : Réduit les erreurs en séparant la logique SQL des données utilisateur.

LES REQUÊTES AVEC PDO

Gestion des erreurs avec try...catch

En **PDO**, les erreurs peuvent survenir lors de la **connexion** à la base de données ou lors de l'**exécution de requêtes SQL**.

Le bloc **try ... catch** permet de **capturer ces erreurs et d'éviter que le script ne s'arrête brutalement**.

- **try** : Exécute un code sensible aux erreurs.
- **catch** : Intercepte l'erreur si elle survient et permet d'afficher un **message personnalisé**.
- Utilise l'objet **PDOException** pour manipuler les erreurs spécifiques à PDO.



Avantages :

- **Empêche l'arrêt du script** en cas d'erreur.
- **Affiche des messages clairs** pour faciliter le débogage.
- **Sécurise l'application** en évitant d'afficher des informations sensibles sur le serveur.
- **Personnalise les erreurs** pour guider l'utilisateur ou le développeur.

LES REQUÊTES AVEC PDO

Gestion des erreurs avec try...catch

- Connexion Sécurisée avec Gestion d'Erreurs

```
try {  
    // Connexion à la base de données  
    $pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Active la gestion des erreurs  
    echo "Connexion réussie !";  
} catch (PDOException $e) {  
    // Gestion de l'erreur  
    echo "Erreur de connexion : " . $e->getMessage(); // Affiche l'erreur  
}
```

LES REQUÊTES AVEC PDO

Gestion des erreurs avec try...catch

- Gestion des Erreurs pour les Requêtes SQL

```
try {  
    $stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE email = :email");  
    $stmt->execute(['email' => 'test@email.com']);  
    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);  
} catch (PDOException $e) {  
    echo "Erreur dans la requête : " . $e->getMessage(); // Affiche l'erreur SQL  
}
```

INTRODUCTION AU LANGAGE PHP

PARTIE 3

Atelier pratique

ATELIER PRATIQUE

Mise en pratique avec PDO

1. Connexion à la base de données :

- Établissez une connexion à la base **library** créée précédemment en utilisant **PDO**.
- Gérez les erreurs avec un bloc **try...catch** pour assurer la sécurité et la stabilité du script.

2. Récupérer et afficher tous les livres :

- Utilisez la méthode **query()** pour récupérer tous les livres de la table **livres**.
- Affichez les résultats sous forme de **liste**, en affichant le titre, l'auteur et l'année de publication.

3. Filtrer les livres publiés après 2000 :

- Préparez une requête avec **prepare()** et exécutez-la avec **execute()**.
- Affichez uniquement les livres publiés après **2000**, triés par **titre** en ordre **alphabétique**.

4. Ajouter un livre à la base :

- Utilisez une requête **préparée** avec **prepare()** et **execute()** pour insérer un nouveau livre.
- Affichez un message de confirmation avec l'**ID** du livre ajouté en utilisant **lastInsertId()**.

5. Bonus : Modifier la disponibilité d'un livre :

- Mettez à jour la disponibilité d'un livre spécifique en utilisant la méthode **exec()**.
- Affichez un message indiquant combien de lignes ont été mises à jour.

PARTIE 4

LE CRUD (CRÉER, LIRE, MAJ, SUPPRIMER)

Le **CRUD** regroupe les opérations essentielles pour manipuler des données dans une base, telles que la **création**, la **lecture**, la **mise à jour** et la **suppression**. Ces actions permettent de gérer efficacement les informations d'une application.



INTRODUCTION AU CRUD



MISE EN OEUVRE AVEC
PDO



EXERCICE PRATIQUE

LE CRUD (CRÉER, LIRE, METTRE À JOUR, SUPPRIMER)

PARTIE 1

Introduction au CRUD

INTRODUCTION AU CRUD

Définition et importance d'une application web

Le **CRUD** représente les quatre opérations de base utilisées pour manipuler des données dans une base de données :

- **Créer (Create)** : Ajouter de nouvelles informations.
- **Récupérer (Read)** : Lire ou afficher les données existantes.
- **Updater (Update)** : Modifier les informations existantes.
- **Delete (Delete)** : Supprimer des informations inutiles ou obsolètes.

Le **CRUD** joue un rôle central dans les applications web modernes en offrant une **gestion structurée des données**. Il permet d'**ajouter**, de **lire**, de **modifier** et de **supprimer** des informations de manière simple et organisée. Ce modèle permet de créer des **applications interactives** comme des **blogs**, des **boutiques en ligne** ou des **systèmes de gestion**.

Grâce à sa **flexibilité**, il facilite l'intégration avec des bases de données et/ou **API**, tout en restant **standardisé** et **facile à implémenter**, ce qui en fait un pilier du développement web.

LE CRUD (CRÉER, LIRE, METTRE À JOUR, SUPPRIMER)

PARTIE 2

Mise en Œuvre avec PDO

MISE EN ŒUVRE AVEC PDO

Lecture d'enregistrements (SELECT)

Pour insérer des informations dans une table, on utilise une **requête préparée** avec **prepare()** et **execute()** pour garantir la **sécurité** des données.

```
$stmt = $pdo->query("SELECT * FROM livres");
$result = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($result as $row) {
    echo $row['titre'] . " - " . $row['auteur'] . " (" . $row['annee_publication'] . ")<br>";
}
```

- **query()** exécute une requête simple.
- **fetchAll(PDO::FETCH_ASSOC)** récupère toutes les lignes sous forme de **tableau associatif**.
- Les données sont ensuite affichées en **boucle**.

MISE EN ŒUVRE AVEC PDO

Création d'un enregistrement (INSERT)

Pour insérer les données, on utilise **query()** ou **prepare()** avec **execute()**.

```
$stmt = $pdo->prepare("INSERT INTO livres (titre, auteur, annee_publication, disponible)
    VALUES (:titre, :auteur, :annee, :disponible)");

$stmt->execute([
    'titre' => 'Le Petit Prince',
    'auteur' => 'Antoine de Saint-Exupéry',
    'annee' => 1943,
    'disponible' => 1
]);
```

- **prepare()** prépare la requête avec des **paramètres dynamiques**.
- **execute()** injecte les valeurs de manière **sécurisée** pour éviter les **injections SQL**.
- Utilise **placeholders nommés** (`:titre`, `:auteur`).

MISE EN ŒUVRE AVEC PDO

Mise à jour d'un enregistrement (UPDATE)

On utilise **prepare()** et **execute()** pour mettre à jour des informations existantes.

```
$stmt = $pdo->prepare("UPDATE livres SET disponible = :disponible WHERE id = :id");

$stmt->execute([
    'disponible' => 0,
    'id' => 1
]);
```

- Modifie la colonne **disponible** en utilisant des **paramètres sécurisés**.
- La condition **WHERE id = :id** garantit que seule la ligne spécifiée est mise à jour.

MISE EN ŒUVRE AVEC PDO

Suppression d'un enregistrement (DELETE)

Pour supprimer une ligne, on utilise également `prepare()` et `execute()`.

```
$stmt = $pdo->prepare("DELETE FROM livres WHERE id = :id");
$stmt->execute(['id' => 2]);
```

- Supprime uniquement la ligne avec l'**id 2**.
- L'utilisation de **paramètres dynamiques** évite les erreurs et les **failles de sécurité**.

LE CRUD (CRÉER, LIRE, METTRE À JOUR, SUPPRIMER)

PARTIE 3

Atelier pratique

ATELIER PRATIQUE

Exercice pratique - implémenter un CRUD sur une table "produits"**1 - Créer la base de données et la table :**

- Créez une base de données nommée **catalogue**.
- Ajoutez une table nommée **produits** avec les colonnes suivantes :
 - **id** (clé primaire auto-incrémentée).
 - **nom** (obligatoire).
 - **description** (texte).
 - **prix** (décimal).
 - **stock** (entier).

2 - Implémenter l'opération CREATE (INSERT) :

- Créez un formulaire ou un script permettant d'ajouter un nouveau produit dans la table.
- Utilisez **PDO** et des **requêtes préparées** pour sécuriser l'insertion.



ATELIER PRATIQUE

Exercice pratique - implémenter un CRUD sur une table "produits"

Implémenter l'opération READ (SELECT) :

- Affichez la liste complète des produits avec leur nom, description, prix et stock.
- Ajoutez la possibilité de trier les résultats par **prix** ou par **nom**.

Implémenter l'opération UPDATE :

- Créez un formulaire ou un script pour modifier les informations d'un produit existant.
- Ajoutez une validation pour s'assurer que les champs obligatoires sont remplis.



ATELIER PRATIQUE

Exercice pratique - implémenter un CRUD sur une table "produits"**Implémenter l'opération DELETE :**

- Ajoutez un bouton pour **supprimer un produit** de la table.
- Demandez une **confirmation** avant la suppression.

Bonus :

- Ajoutez un filtre pour afficher uniquement les produits avec un **stock disponible**.
- Implémentez un champ de **recherche** par nom ou description.



PARTIE 5

SYSTÈME DE CONNEXION UTILISATEUR (LOGIN)

Un **système de connexion utilisateur** est essentiel pour sécuriser l'accès aux applications web. Il permet d'identifier les utilisateurs, de gérer leurs sessions et d'assurer la protection des données sensibles tout en offrant une expérience personnalisée.



FORMULAIRE DE CONNEXION



VERIFICATIONS DES INFORMATIONS UTILISATEUR



GESTION DES SESSIONS



EXERCICE PRATIQUE

SYSTÈME DE CONNEXION UTILISATEUR (LOGIN)

PARTIE 1

Formulaire de connexion

FORMULAIRE DE CONNEXION

Création d'un formulaire HTML basique

Un **formulaire de connexion** est la première étape pour sécuriser l'accès à une application web. Il permet aux utilisateurs de saisir leurs identifiants pour s'authentifier et accéder à des fonctionnalités protégées.

```
<form action="login.php" method="POST">
    <label for="email">Email :</label>
    <input type="email" id="email" name="email" required>

    <label for="password">Mot de passe :</label>
    <input type="password" id="password" name="password" required>

    <button type="submit">Se connecter</button>
</form>
```

FORMULAIRE DE CONNEXION

Création d'un formulaire HTML basique

La **méthode POST** est utilisée pour envoyer des données de manière **sécurisée**, car elles ne sont pas visibles dans l'URL. Les **champs obligatoires** sont définis avec l'attribut **required**, ce qui garantit que l'utilisateur remplit toutes les informations nécessaires avant de soumettre le formulaire.

Les **types de champs adaptés**, comme **email** pour vérifier automatiquement le format d'une adresse et **password** pour masquer les caractères saisis, assurent une meilleure expérience utilisateur.

Enfin, un **bouton d'envoi**, mis en place avec un **<input type="submit">**, valide et transmet les informations au serveur.

SYSTÈME DE CONNEXION UTILISATEUR (LOGIN)

PARTIE 2

Vérifier les infos utilisateur

VALIDER LES INFORMATIONS UTILISATEUR

Validation des données en PHP

Avant de traiter les informations reçues, il est essentiel de vérifier leur **validité** et leur **sécurité**.

Étapes :

- **Nettoyer les données** : Supprimer les espaces inutiles et caractères spéciaux.
- **Valider les formats** : Vérifier que l'email est bien structuré et que le mot de passe respecte les règles de sécurité.
- **Protéger contre les injections SQL** • Utiliser des requêtes préparées avec PDO

```
$email = trim($_POST['email']); // Nettoyer  
$password = trim($_POST['password']);  
  
// Valider l'email  
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {  
    echo "Email invalide.";  
    exit;  
}
```

- login.php

VALIDER LES INFORMATIONS UTILISATEUR

Requête pour vérifier si l'utilisateur existe en base

Après la validation des données, une **requête préparée avec PDO** est utilisée pour vérifier si l'utilisateur existe dans la base et récupérer son **mot de passe haché**. Cette approche protège contre les **injections SQL** et garantit la **sécurité des données**.

La fonction **password_verify()** est ensuite employée pour comparer le mot de passe saisi avec celui stocké en base. Si les informations sont incorrectes, une **condition** renvoie un message d'erreur afin d'informer l'utilisateur.

```
$stmt = $pdo->prepare("SELECT id, password FROM utilisateurs WHERE email = :email");
$stmt->execute(['email' => $email]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);

if ($user && password_verify($password, $user['password'])) {
    echo "Connexion réussie !";
} else {
    echo "Email ou mot de passe incorrect.";
}
```

SYSTÈME DE CONNEXION UTILISATEUR (LOGIN)

PARTIE 3

Gestion des sessions

GESTION DES SESSIONS

Utilisation de `$_SESSION` pour maintenir la connexion

Une **session** permet de stocker temporairement des informations sur un utilisateur pendant qu'il navigue sur un site. Ces informations sont disponibles sur toutes les pages tant que la session est active.

Pourquoi utiliser `$_SESSION`?

- **Maintenir la connexion utilisateur** : Une fois connecté, les informations comme l'**ID** ou le **nom** de l'utilisateur sont enregistrées pour éviter de demander ses identifiants sur chaque page.
- **Stocker des données sensibles temporairement** : Les sessions sont stockées côté serveur, ce qui est plus **sécurisé** que les cookies.
- **Personnaliser l'expérience utilisateur** : Afficher des informations spécifiques à l'utilisateur (nom, préférences).

GESTION DES SESSIONS

Les fonctions principales en PHP

- Démarrer la session et traiter le formulaire :

```
session_start(); // Démarrer la session

// Connexion à la base de données avec PDO
$pdo = new PDO('mysql:host=localhost;dbname=ma_base', 'root', '');

// Récupérer les données du formulaire
$email = trim($_POST['email']);
$password = trim($_POST['password']);
```

- Vérifier les champs et préparer la requête pour vérifier le mot de passe en base de données :

```
if (!empty($email) && !empty($password)) {
    // Préparer la requête
    $stmt = $pdo->prepare("SELECT id, nom, email, password FROM utilisateurs WHERE email = :email");
    $stmt->execute(['email' => $email]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
}
```

GESTION DES SESSIONS

Les fonctions principales en PHP

- Vérifier le mot de passe et créer la session :

```
if ($user && password_verify($password, $user['password'])) {  
    // Enregistrer les informations dans la session  
    $_SESSION['utilisateur_id'] = $user['id'];  
    $_SESSION['nom'] = $user['nom'];  
    $_SESSION['email'] = $user['email'];  
  
    // Redirection vers la page d'accueil  
    header('Location: accueil.php');  
    exit;  
} else {  
    $erreur = "Email ou mot de passe incorrect.";  
}
```

GESTION DES SESSIONS

Les fonctions principales en PHP

- Se déconnecter supprimer la session :

logout.php

```
session_start(); // Démarrer la session
session_unset(); // Supprimer toutes les variables de session
session_destroy(); // Détruire la session

header('Location: login.php'); // Redirection vers la connexion
exit;
```

```
<!-- Bouton de déconnexion -->
<form action="logout.php" method="POST">
    <button type="submit">Se déconnecter</button>
</form>
```

SYSTÈME DE CONNEXION UTILISATEUR (LOGIN)

PARTIE 4

Atelier pratique

ATELIER PRATIQUE

Exercice - implémenter une connexion avec redirection**1 - Créer une base de données et une table :**

- Créez une base de données nommée **exercice_login**.
- Ajoutez une table **utilisateurs** avec les colonnes suivantes :
 - **id** (clé primaire auto-incrémentée).
 - **email** (unique et obligatoire).
 - **mot_de_passe** (haché).

2 Créer un formulaire de connexion :

- Ajoutez un formulaire HTML avec deux champs : **email** et **mot de passe**, ainsi qu'un bouton **Se connecter**.
- Utilisez la méthode **POST** pour envoyer les données.



ATELIER PRATIQUE

Exercice - implémenter une connexion avec redirection

3 - Créer un script PHP pour vérifier les informations :

- Établissez une **connexion PDO** à la base de données.
- Récupérez et **validez les informations** saisies dans le formulaire.
- Vérifiez si l'utilisateur existe et si son **mot de passe haché** est correct.

4 - Mettre en place une session :

- En cas de réussite, enregistrez les informations utilisateur dans une **session PHP**.
- Redirigez l'utilisateur vers une page protégée (par exemple **accueil.php**).



ATELIER PRATIQUE

Exercice - implémenter une connexion avec redirection**5 - Ajouter une déconnexion :**

- Créez un bouton ou un lien pour **se déconnecter** et détruire la session.
- Redirigez l'utilisateur vers la page de connexion après déconnexion.



PARTIE 6

SÉCURISATION AVEC HASH-CRYPT

La sécurisation des mots de passe est essentielle pour protéger les données des utilisateurs. L'utilisation de techniques de hachage garantit que les mots de passe soient stockés de manière **sécurisée et inexploitables** en cas de fuite. Cette approche renforce la protection des informations sensibles dans les applications web modernes.



INTRODUCTION AU
HASHAGE



MISE EN PRATIQUE



EXERCICE PRATIQUE

SÉCURISATION AVEC HASH-CRYPT

PARTIE 1

Introduction au hashage

INTRODUCTION AU HASHAGE

Pourquoi et comment sécuriser un mot de passe?

Le **hashage** est une technique utilisée pour **protéger les mots de passe** en les transformant en une suite de caractères illisibles. Contrairement au cryptage, le hashage est **irréversible**, ce qui signifie qu'il est impossible de retrouver le mot de passe d'origine à partir de sa version hachée. Cette méthode garantit une **sécurité accrue** pour le stockage des informations sensibles dans les bases de données comme les mots de passe.

Pourquoi sécuriser?

- Protéger les données utilisateur contre les **piratages et fuites de données**.
- Empêcher l'exploitation des mots de passe en cas d'accès non autorisé à la base.

Comment sécuriser?

- **Hashage** : Transformer les mots de passe en une version chiffrée irréversible.
- **Salage** : Ajouter une chaîne aléatoire pour rendre chaque mot de passe unique.
- **Vérification sécurisée** : Comparer les mots de passe hachés sans exposer les informations réelles.

INTRODUCTION AU HASHAGE

Présentation de password_hash()

password_hash() est une fonction PHP utilisée pour **créer un hash sécurisé** à partir d'un mot de passe. Elle utilise par défaut l'algorithme **BCRYPT**, reconnu pour sa **fiabilité et sa résistance aux attaques**. Cette fonction génère également un **sel unique** automatiquement, ce qui renforce la sécurité du hash en rendant chaque mot de passe unique, même s'ils sont identiques. Cela protège contre les attaques par **tables arc-en-ciel** (aussi appelées rainbow tables).

```
$hash = password_hash('monMotDePasse', PASSWORD_BCRYPT);
```

INTRODUCTION AU HASHAGE

Présentation de password_verify()

password_verify() permet de vérifier si un mot de passe saisi par l'utilisateur correspond au hash stocké dans la base de données. Plutôt que de comparer directement les chaînes, cette fonction utilise l'**algorithme et le sel** inclus dans le hash pour effectuer la comparaison. Cela garantit une vérification sécurisée sans exposer le mot de passe d'origine.

```
if (password_verify('monMotDePasse', $hash)) {  
    echo "Mot de passe valide";  
} else {  
    echo "Mot de passe invalide";  
}
```

Pour info :

Un **sel** est une **chaîne aléatoire** ajoutée à un mot de passe avant son **hachage** pour le rendre **unique**, même si deux utilisateurs ont le même mot de passe. Cela renforce la sécurité en empêchant les attaques par **tables arc-en-ciel** (rainbow tables).

SÉCURISATION AVEC HASH-CRYPT

PARTIE 2

Mise en pratique

MISE EN PRATIQUE

Hashage des mots de passe à l'enregistrement

Sécuriser les mots de passe des utilisateurs lors de leur enregistrement en utilisant la fonction **password_hash()** pour générer un hash sécurisé.

- Création du formulaire d'inscription

```
<form action="register.php" method="POST">
    <label for="email">Email :</label>
    <input type="email" id="email" name="email" required>

    <label for="password">Mot de passe :</label>
    <input type="password" id="password" name="password" required>

    <button type="submit">S'inscrire</button>
</form>
```

MISE EN PRATIQUE

Hashage des mots de passe à l'enregistrement

- Hashage du mot de passe et insertion en base de données dans register.php

```
// Vérifier si les données ont été envoyées
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Récupérer et nettoyer les données du formulaire
    $email = trim($_POST['email']);
    $password = trim($_POST['password']);

    // Vérifier si les champs sont remplis
    if (!empty($email) && !empty($password)) {
        // Hacher le mot de passe
        $hashedPassword = password_hash($password, PASSWORD_BCRYPT);

        // Insérer les données dans la base de données
        $stmt = $pdo->prepare("INSERT INTO utilisateurs (email, mot_de_passe) VALUES (:email, :mot_de_passe)");
        $stmt->execute([
            'email' => $email,
            'mot_de_passe' => $hashedPassword
        ]);

        echo "Enregistrement réussi!";
    } else {
        echo "Veuillez remplir tous les champs.";
    }
}
```

SÉCURISATION AVEC HASH-CRYPT

PARTIE 3

Atelier pratique

ATELIER PRATIQUE

Mise en pratique – Ajouter le hashage de mot de passe à l'application CRUD

Modifier la base de données :

- Ajoutez une colonne **mot_de_passe** dans la table existante (par exemple **utilisateurs**) pour stocker les mots de passe hachés.

Créer un formulaire d'enregistrement :

- Ajoutez un champ **mot de passe** au formulaire d'ajout d'un utilisateur.
- Utilisez la méthode **POST** pour sécuriser l'envoi des données.

Hashage des mots de passe :

- Lors de l'ajout d'un utilisateur, récupérez et **hachez le mot de passe** avant de l'enregistrer dans la base de données.
- Assurez-vous que chaque mot de passe est stocké de manière sécurisée avec un **sel automatique**.

Modifier la lecture des données :

- Vérifiez que les mots de passe ne sont pas affichés directement lors de la lecture des utilisateurs.

Authentification avec vérification :

- Ajoutez une fonctionnalité permettant de **vérifier le mot de passe haché** lors de la connexion d'un utilisateur avec **password_verify()**.

PARTIE 7

CRÉATION D'UN SERVEUR

Dans le cadre de votre projet d'axe, vous devrez concevoir une **plateforme de collection et d'échange de cartes** en utilisant des technologies web modernes. Ce projet permettra d'appliquer les notions vues en cours, notamment **PHP**, **MySQL** et **PDO**, pour gérer les données et sécuriser les interactions.



CONCÉPTUALISATION DU PROJET



DÉPLOIEMENT DU SERVEUR LOCAL



EXERCICE PRATIQUE

CRÉATION D'UN SERVEUR

PARTIE 1

Conceptualisation du projet

CONCEPTUALISATION DU PROJET

Analyse des besoins du projet d'axe

- **Analyse des besoins** : Comprendre les fonctionnalités attendues et structurer les données nécessaires au projet.
- **Base de données relationnelle** : Créer et organiser les tables pour stocker les utilisateurs, les cartes et leurs interactions.
- **Déploiement d'un serveur local** : Installer un environnement de développement avec **XAMPP** ou **WAMP** pour tester et exécuter l'application.
- **Sécurisation des données** : Implémenter des sessions et des mots de passe hachés pour garantir la confidentialité des utilisateurs.

CONCEPTUALISATION DU PROJET

Création de la structure de la base de données

Avant de créer la base de données, il est essentiel d'**analyser les besoins** du projet pour identifier :

- Les **entités principales** (ex. : utilisateurs, cartes, échanges).
- Les **relations** entre ces entités.
- Les **informations** à stocker pour chaque entité (colonnes et types de données).

MCD :

- Identifier les **tables** nécessaires.
- Définir les **clés primaires (PK)** et **clés étrangères (FK)** pour les relations.
- Préparer les **contraintes (UNIQUE, NOT NULL)** pour sécuriser les données.

CRÉATION D'UN SERVEUR

PARTIE 2

Déploiement d'un serveur local

DÉPLOIEMENT D'UN SERVEUR LOCAL

Utilisation d'un outil comme XAMPP ou WAMP

- Téléchargez WAMP sur www.wampserver.com.
- Installez et lancez WAMP.
- Vérifiez que l'icône devient **verte** dans la barre des tâches pour confirmer que les services sont actifs.

Accéder au serveur web local :

- Accédez à votre serveur en ouvrant un navigateur et en tapant : <http://localhost>
- Pour gérer la base de données, ouvrez phpMyAdmin : <http://localhost/phpmyadmin>

Configuration du projet :

- Placez les fichiers du projet dans le dossier **www** (pour WAMP) dans le disque local **c://**.
- Vérifiez que votre fichier d'accueil est nommé **index.php** pour qu'il soit reconnu automatiquement.
- Testez l'accès au projet en entrant dans votre navigateur

DÉPLOIEMENT D'UN SERVEUR LOCAL

Utilisation d'un outil comme XAMPP ou WAMP

Création et configuration de la base de données :

- Accédez à **phpMyAdmin**.
- Créez une nouvelle base de données pour votre projet.
- Importez ou créez les tables nécessaires via des requêtes SQL.

Test et vérifications :

- Vérifiez que votre serveur fonctionne en testant une **page PHP simple**
- Testez également la **connexion à la base de données** avec PDO.

SOURCES

Liste des sources



MySQL

<https://www.mysql.com/fr/>



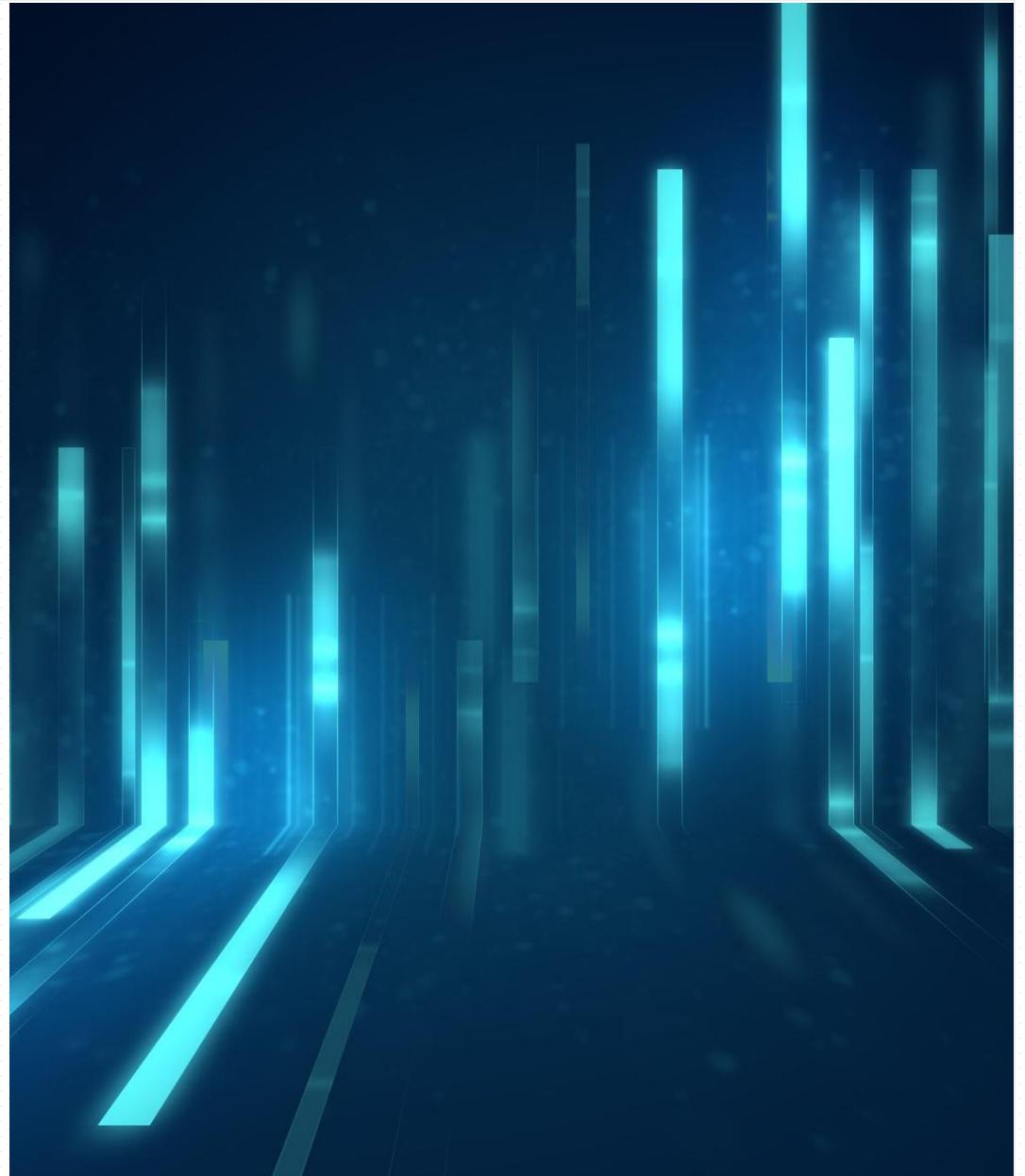
PDO

<https://www.php.net/manual/fr/book pdo.php>



WAMP

<https://www.wampserver.com/>



INITIATION SERVEUR PHP & MYSQL

Rappel des Rendus

Création d'une réplique de **TWITTER** avec l'utilisation de **PHP** et **MYSQL** :

- **Il y aura des utilisateurs qui pourront :**
 - S'inscrire
 - Rechercher des tweets
- **Il y aura des tweets que l'on pourra :**
 - Voir du plus récent au plus ancien
 - Ajouter
 - Supprimer
- **Lier votre utilisateur à ses tweets vous permet de savoir qui a tweeté quoi, et même de filtrer les tweets par utilisateur !**
 - Ajouter une colonne “user_id” dans votre table tweet
 - Celle-ci contiendra donc l'id de l'utilisateur qui a écrit ce tweet
 - Cela suffit pour lier les deux tables ensemble !



INITIATION SERVEUR PHP & MYSQL

Rappel des Rendus

- Pour récupérer tous les tweets avec les données de l'user associé :

```
'SELECT * FROM tweet INNER JOIN users ON tweet.user_id = users.id'
```

Bonus :

Cette partie est en bonus car ça se complexifie un peu. Nous avons trouvé comment lier les deux tables ensemble et récupérer les infos, mais comment stocker l'id de l'utilisateur à la création d'un tweet ? Il y a plusieurs façons de faire :

- **Ajouter un champ de formulaire pour le pseudo de l'user lors de l'ajout du tweet**
 - Avec ce pseudo, on pourra récupérer le user correspondant et donc, fournir son id lors de la création du tweet
 - Difficulté : Medium / UserFriendly : null / Prof: **kind of impressed**
- **Faire un système d'authentification avec \$_SESSION (Aide)**
 - Ça sera donc l'user qui est connecté qui sera lié au tweet qu'il a écrit
 - Au moment du logging vous récupérez l'id de l'utilisateur dans la base de données, le stockez dans \$_SESSION et vous l'aurez lors de la création d'un tweet
 - Difficulté : Hard / UserFriendly : Perfect / Prof : **very very impressed**



INITIATION SERVEUR PHP & MYSQL

Rappel des Rendus

Attention :

- Bien commenter son code !
- Interdiction d'utiliser mysqli !
- Toutes fonctionnalités complémentaires comptent comme un BONUS !
- Bien ranger ses fichiers et dossier !
- Bien vérifier le contenu de son rendu !
- Bien vérifier que les liens sont bien **accessible par autrui** !
- Bien mettre en avant les fonctionnalités qui sont évalués !
- Bien vérifier sa démonstration vidéo !

Deadline : Dimanche à 23h59

Deadline tiers temps : Mercredi à 23h59 (merci de me prévenir avant)

Vous mettrez votre code sur **GitHub**

Et un **export de votre base de données** :

- dans phpMyAdmin => clic sur votre base, onglet “Export” en haut
- bien choisir “SQL”
- vérifiez qu'il y a bien toutes vos tables dans le fichier

Le rendu se fera sur **MOODLE** et par email web.start.contact@gmail.com



DES QUESTIONS?

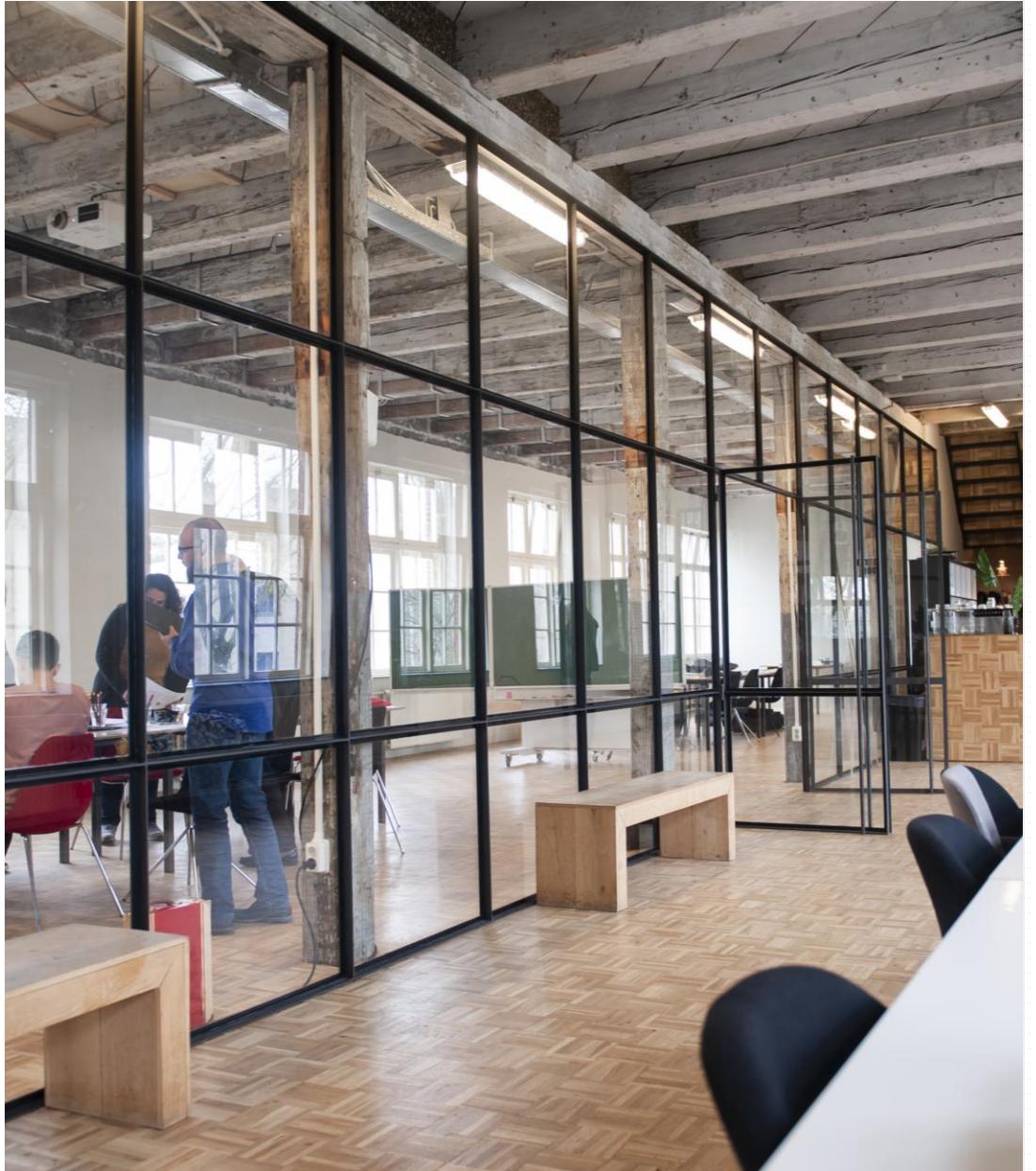
Me contacter



samih@academiews.fr



07 82 14 81 41



MERCI POUR VOTRE ATTENTION

Bonne révision !

