phpMyAdmin : le même travail côté web

Refaire la création d'une base, de tables et de clés étrangères dans phpMyAdmin — l'interface web universelle des hébergements mutualisés. Même serveur MySQL, même SQL en dessous, autre habillage.

4TTR | niveau

Tu sais maintenant créer une base, des tables et des clés étrangères avec HeidiSQL. Très bien — sauf que **HeidiSQL n'est pas installé sur ton hébergement web**. Là-bas, ce que tu trouveras dans le panneau d'administration, c'est presque toujours **phpMyAdmin**.

phpMyAdmin n'est pas un autre SGBDR : c'est un **autre client**. Il parle au **même serveur MySQL** dans la **même langue (SQL)**. Tu vas le constater dans ce cours en refaisant tout le travail des deux séances précédentes — base, tables et clés étrangères — depuis l'interface web.

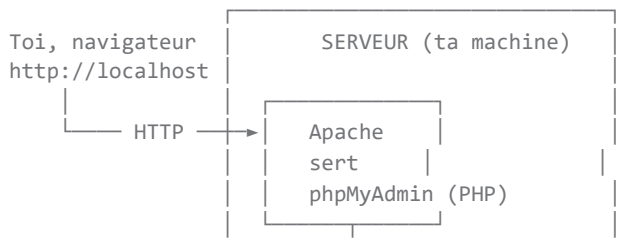
🎯 Objectifs

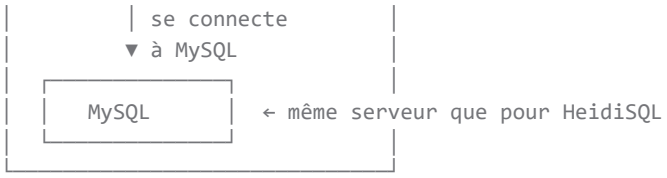
À la fin de cette séquence, tu seras capable de :

1. Ouvrir phpMyAdmin depuis Laragon
2. Naviguer dans son interface (panneau gauche, onglets)
3. Créer une base de données et ses tables
4. Définir des clés primaires et étrangères
5. Importer un dump SQL produit ailleurs
6. Identifier les différences pratiques avec HeidiSQL

Partie 1 — phpMyAdmin, qu'est-ce que c'est exactement ?

phpMyAdmin est une **application web** écrite en PHP. Elle est servie par **Apache** (le serveur web démarré par Laragon en même temps que MySQL).





Tu ouvres phpMyAdmin dans un navigateur. Apache exécute le code PHP. Ce code se connecte à MySQL et te renvoie une page HTML avec les résultats.

> 💡 Sur un hébergement web réel, le scénario est identique. Le navigateur de l'admin (toi) parle à Apache distant, Apache parle au MySQL distant. Tu n'as rien à installer sur ta machine — c'est le grand intérêt.

Partie 2 — Lancer phpMyAdmin

Depuis Laragon

1. Vérifie que **Apache** *et* **MySQL** tournent (pastilles vertes dans Laragon).
2. Dans le menu Laragon, clique sur **MySQL** → **phpMyAdmin**.

Ton navigateur s'ouvre sur une URL du type `http://localhost/phpmyadmin/`.

L'écran de connexion

Selon la configuration de Laragon, deux cas :

- **Connexion automatique** : tu arrives directement sur l'interface
- **Formulaire de connexion** : remplis `Utilisateur : root`, mot de passe vide

> ⚠️ Comme pour HeidiSQL : `root` sans mot de passe, **uniquement en local**. Sur un hébergement, on a un utilisateur dédié à la base.

Que vois-tu ?

- **Panneau gauche** : la liste des bases existantes — exactement les mêmes que dans HeidiSQL, puisque c'est le même serveur. Tu y trouves notamment la base `billetterie` que tu as créée au cours 08.
- **Panneau central** : tableau de bord, onglets *Bases de données*, *SQL*, *État*, *Comptes utilisateurs*, *Importer*, *Exporter*…

> 🧠
> **VÉRIFICATION IMPORTANTE**

: clique sur `billetterie` dans le panneau gauche. Tu vois `client`, `evenement`, `ticket` — avec leurs lignes ! C'est la **preuve** que phpMyAdmin et HeidiSQL parlent au même serveur. Tu peux gérer les mêmes données avec l'un ou l'autre indifféremment.

Partie 3 — Refaire le travail dans une nouvelle base

Pour bien voir l'équivalence, on va créer une **nouvelle base** `billetterie_phpma` et refaire dedans le travail complet : tables `client`, `evenement`, `ticket` avec ses FK.

Créer la base

Onglet **Bases de données** (en haut). Sous *Créer une base de données* :

Champ	Valeur
Nom	`billetterie_phpma`
Interclassement	`utf8mb4_unicode_ci`

Clique **Créer**. La base apparaît dans le panneau gauche.

Partie 4 — Créer la table `client`

Clique sur `billetterie_phpma` dans le panneau gauche. Tu vois *Aucune table dans la base de données*. Formulaire **Créer une table** :

Champ	Valeur
Nom	`client`
Nombre de colonnes	`5`

Clique **Créer**. Tu arrives sur le tableau de définition des colonnes (une grosse grille).

Remplir les colonnes

#	Nom	Type	Taille	NULL	Index	A_I
1	`id`	`INT`	—	non	**PRIMARY**	☑
2	`nom`	`VARCHAR`	`100`	non	—	—
3	`prenom`	`VARCHAR`	`100`	non	—	—
4	`email`	`VARCHAR`	`150`	oui	—	—
5	`telephone`	`VARCHAR`	`20`	oui	—	—

> 💡 La colonne **A_I** (*AUTO_INCREMENT*) est une simple case à cocher dans phpMyAdmin. Pour la PK, sélectionne *PRIMARY* dans la colonne **Index**.

Tout en bas, clique **Enregistrer**.

Voir le SQL généré

Onglet **SQL** (en haut, ou bouton *Aperçu SQL* avant d'enregistrer) : phpMyAdmin te montre exactement la requête envoyée.

```sql
CREATE TABLE `billetterie_phpma`.`client` (
    `id`        INT          NOT NULL AUTO_INCREMENT,
    `nom`       VARCHAR(100) NOT NULL,
    `prenom`    VARCHAR(100) NOT NULL,
    `email`     VARCHAR(150) NULL,
    `telephone` VARCHAR(20)  NULL,
    PRIMARY KEY (`id`)
) ENGINE = InnoDB;
```

> 🧠 **COMPARE AVEC LE SQL GÉNÉRÉ PAR HEIDISQL AU COURS 08.**
>
> Tu verras qu'ils sont **identiques** (à quelques détails de mise en forme près). C'est la même langue, donc le même résultat — peu importe le client.

Partie 5 — Créer `evenement` et `ticket`

Refais la procédure pour les deux autres tables.

evenement

#	Nom	Type	Taille	NULL	Index	A_I
1	`id`	`INT`	—	non	PRIMARY	☑
2	`titre`	`VARCHAR`	`200`	non	—	—
3	`date`	`DATETIME`	—	non	—	—
4	`prix_base`	`DECIMAL`	`6,2`	non	—	—

ticket (sans FK pour l'instant)

#	Nom	Type	Taille	NULL	Index	A_I
1	`id`	`INT`	—	non	PRIMARY	☑
2	`prix`	`DECIMAL`	`6,2`	non	—	—
3	`statut`	`VARCHAR`	`20`	non	—	—
4	`date_achat`	`DATETIME`	—	non	—	—
5	`client_id`	`INT`	—	non	INDEX	—
6	`evenement_id`	`INT`	—	non	INDEX	—

> 💡 Sur les colonnes FK, choisis l'index **INDEX** (pas PRIMARY). MySQL exige un index sur les colonnes qui serviront de clé étrangère — phpMyAdmin te demandera de toute façon de l'ajouter au moment de créer la contrainte si tu l'oublies maintenant.

Partie 6 — Ajouter les clés étrangères

Une fois la table `ticket` créée, ouvre-la (clic dans le panneau gauche) et va sur l'onglet **Structure**, puis sous-onglet **Vue relationnelle** (lien en bas de l'écran, parfois libellé *Relation view* ou *Gestion des relations*).

Tu obtiens un tableau pour configurer les FK. Pour chacune :

FK 1 — `client_id → client.id`

Champ	Valeur
Contrainte de clé étrangère	`fk_ticket_client`
Colonne	`client_id`

Champ	Valeur
Base de données	`billetterie_phpma`
Table	`client`
Colonne	`id`
ON DELETE	`RESTRICT`
ON UPDATE	`CASCADE`

FK 2 — `evenement_id → evenement.id`

Champ	Valeur
Contrainte de clé étrangère	`fk_ticket_evenement`
Colonne	`evenement_id`
Base de données	`billetterie_phpma`
Table	`evenement`
Colonne	`id`
ON DELETE	`RESTRICT`
ON UPDATE	`CASCADE`

Clique **Enregistrer**. Si tu reçois une erreur :

- vérifie que les colonnes FK ont **exactement le même type** que les PK référencées (toutes en `INT NOT NULL`)
- vérifie que l'**index** est bien défini sur les colonnes `client_id` et `evenement_id`
- vérifie que tu utilises bien le moteur **InnoDB** (par défaut sur les versions récentes — MyISAM, lui, ne supporte pas les FK)

Partie 7 — Tester les contraintes

Onglet **SQL** (en haut, après avoir sélectionné `billetterie_phpma`). Exécute :

```sql
-- Insérer un client et un événement valides
INSERT INTO client (nom, prenom) VALUES ('Test', 'Démo');
INSERT INTO evenement (titre, date, prix_base)
VALUES ('Démo phpMyAdmin', '2026-12-01 20:00:00', 10.00);

-- Récupérer les ids
```

```sql
SELECT * FROM client;
SELECT * FROM evenement;
```

Puis, avec les ids retournés (probablement 1 et 1) :

```sql
INSERT INTO ticket (prix, statut, date_achat, client_id, evenement_id)
VALUES (10.00, 'payé', '2026-11-25 18:00:00', 1, 1);
```

→ succès.

```sql
INSERT INTO ticket (prix, statut, date_achat, client_id, evenement_id)
VALUES (10.00, 'payé', '2026-11-25 18:00:00', 999, 1);
```

→ **Erreur** :

```
#1452 - Cannot add or update a child row: a foreign key constraint fails
```

Même erreur, même comportement qu'avec HeidiSQL. Confirmation : les contraintes vivent **dans le serveur MySQL**, pas dans le client.

Partie 8 — Importer un dump SQL

C'est le cas d'usage typique du transfert vers un hébergement : tu as développé en local, tu veux mettre la même structure (et éventuellement les mêmes données) sur le serveur de production.

Exporter depuis ta base locale

Reste pour l'instant dans phpMyAdmin sur la base `billetterie` (la première, celle du cours 08). Onglet **Exporter** :

- Méthode : **Personnalisée**
- Format : **SQL**
- Coche **Ajouter** `DROP TABLE` si tu veux écraser une base cible existante
- Décoche les options dont tu ne veux pas

Clique **Exécuter**. Un fichier `billetterie.sql` est téléchargé.

> 💡 Ce fichier contient des `CREATE TABLE …` et des `INSERT INTO …`. C'est du SQL pur, portable d'un serveur à l'autre.

Importer dans une autre base

Crée une nouvelle base `billetterie_import` (Partie 3, mais avec ce nom). Sélectionne-la, onglet **Importer** :

- Fichier à importer : sélectionne `billetterie.sql`
- Laisse les autres options par défaut
- Clique **Exécuter**

phpMyAdmin reproduit toutes les tables, contraintes et données. C'est exactement la procédure que tu utiliseras le jour où tu déploieras un projet sur un hébergement réel.

Partie 9 — HeidiSQL vs phpMyAdmin : quand utiliser quoi ?

Critère	HeidiSQL	phpMyAdmin
Type	Application desktop (Windows)	Application web (PHP)
Installation	À installer sur ta machine	Déjà fourni par presque tous les hébergeurs
Performance	Très rapide sur grosses tables	Plus lent (chaque action = requête HTTP)
Édition de données	Fluide, façon tableur	Correcte mais formulaires lourds
Requêtes SQL	Confortable, suggestions	OK, mais on quitte la page facilement
Accès distant	Possible (avec tunnel SSH)	Natif (URL publique)
Outils intégrés	Export/import, profileur, snapshots	Export/import, comptes utilisateurs, état serveur

En pratique

- **Sur ta machine de dev** : HeidiSQL pour le confort quotidien.
- **Sur un hébergement mutualisé** : phpMyAdmin parce que c'est tout ce que tu as.
- **Sur un serveur dédié distant** : HeidiSQL via un tunnel SSH si c'est possible, sinon phpMyAdmin.

L'important : **les deux outils sont équivalents pour ce qu'ils font** — créer des bases, des tables, des contraintes, des requêtes. La différence est ergonomique.

🧪 Exercices

Exercice 1 — Vérifier l'équivalence

Ouvre HeidiSQL. Connecte-toi au même serveur. Constate que la base `billetterie_phpma` (créée via phpMyAdmin) y est présente, avec ses tables et ses contraintes. Insère un client via HeidiSQL ; vérifie qu'il apparaît dans

phpMyAdmin. Conclus.

Exercice 2 — Import dans phpMyAdmin

Exporte une base depuis HeidiSQL (clic droit → *Exporter la base*). Importe le fichier obtenu dans phpMyAdmin (nouvelle base vierge). Vérifie que tout est là.

Exercice 3 — Le formulaire de connexion

Sur l'écran d'accueil de phpMyAdmin, repère ces informations :

- Quel utilisateur MySQL es-tu actuellement ?
- Quelle est la version de MariaDB / MySQL du serveur ?
- Combien de bases sont présentes sur le serveur (y compris les bases système) ?

Exercice 4 — Ton projet

Refais l'ensemble de **ton projet personnel** dans phpMyAdmin, dans une nouvelle base. Toutes les tables et toutes les clés étrangères. Compare le SQL généré avec celui qu'avait produit HeidiSQL au cours 09.

✍️ À retenir

- phpMyAdmin est un **client web** pour MySQL — exactement comme HeidiSQL est un client desktop.
- Il tourne grâce à **Apache** qui sert son code PHP ; les deux services doivent être démarrés dans Laragon.
- L'interface diffère, mais le **SQL généré est le même** et les **données sont les mêmes** (un seul serveur).
- Les contraintes de clé étrangère se déclarent dans la **Vue relationnelle** d'une table.
- **Importer / exporter** un dump SQL est la procédure standard pour déplacer une base d'un serveur à un autre.

Suite

Tu as maintenant toutes les compétences pour créer une base relationnelle bien construite, en local ou sur un hébergement. La prochaine étape — qui dépasse cette UAA — sera d'apprendre à **interroger et manipuler les données par SQL pur** : `SELECT`, `INSERT`, `UPDATE`, `DELETE`, puis les jointures pour exploiter les relations. Les articles existent déjà dans la section SQL du site.