SQL — Premières requêtes (SELECT, FROM, WHERE, ORDER BY)

Premiers pas pratiques en SQL sur une vraie base de données : 20 clients, 12 voitures et 200 locations à interroger. Tu apprends à importer la base dans HeidiSQL et phpMyAdmin, puis tu écris tes premières requêtes pour chercher, filtrer et trier.

4TTR | Découverte

*Tu as vu **comment** une base relationnelle est structurée. Maintenant on l'utilise pour de vrai : on importe une base existante et on commence à l'interroger avec SQL. À la fin de ce cours, tu sauras répondre à des questions concrètes comme « quelles locations sont en cours ? » ou « quels sont les modèles à moins de 50 € par jour, triés du moins cher au plus cher ? ».*

🎯 Objectifs

À la fin de ce cours, tu seras capable de :

1. **Importer** un fichier `.sql` dans HeidiSQL et dans phpMyAdmin.
2. Lire le **schéma** d'une base (tables, colonnes, types).
3. Écrire une requête `SELECT … FROM …` pour afficher des lignes.
4. **Filtrer** les résultats avec `WHERE` et les opérateurs `=`, `<>`, `<`, `>`, `LIKE`.
5. **Combiner** plusieurs conditions avec `AND` et `OR`.
6. **Trier** les résultats avec `ORDER BY` (croissant, décroissant, multi-colonnes).

Partie 0 — Préparer la base de données

Avant de pouvoir écrire des requêtes, il faut **importer** la base dans ton SGBD. Télécharge d'abord le fichier `location_voitures_4e_ttinfo.sql` via le lien de téléchargement à droite de cette page.

ℹ️ **C'EST QUOI UN FICHIER `.SQL` ?** C'est un script texte qui contient des instructions SQL : `CREATE TABLE`, `INSERT INTO`, etc. Quand tu l'« exécutes » dans un SGBD, il rejoue toutes ces instructions et reconstruit la base de zéro.

Méthode A — Import dans HeidiSQL (client desktop)

1. Démarre **Laragon**, puis clique sur « Démarrer tout » pour activer MySQL.
2. Ouvre **HeidiSQL** et connecte-toi à la session locale (`Hôte: 127.0.0.1`, `Utilisateur: root`, mot de passe vide).
3. Dans l'arborescence à gauche, **clic droit sur la session** → *Créer un nouveau* → *Base de données*. Nomme-la `location_voitures`, valide.
4. **Double-clique sur cette nouvelle base** dans l'arbre pour la « sélectionner » (son nom passe en gras).
5. Menu **Fichier** → **Charger un fichier SQL…** → choisis `location_voitures_4e_ttinfo.sql` dans tes téléchargements. Une nouvelle requête s'ouvre.
6. Appuie sur **F9** (ou clique sur le bouton bleu « ▶ Exécuter ») pour lancer le script.
7. Dans l'arbre à gauche, déplie `location_voitures`. Tu dois voir **3 tables** : `client`, `voiture`, `location`.

> ⚠️ **SI HEIDISQL REFUSE :** vérifie que tu as bien double-cliqué sur la base `location_voitures` (et pas une autre) avant d'exécuter le script — les `CREATE TABLE` se font dans la base active.

Méthode B — Import dans phpMyAdmin (client web)

1. Démarre **Laragon**, active MySQL et Apache.
2. Ouvre `http://localhost/phpmyadmin` dans ton navigateur.
3. Onglet **Bases de données** → tape `location_voitures` dans le champ « Créer une base de données » → clique **Créer**.
4. **Clique sur** `location_voitures` à gauche pour t'y placer.
5. Onglet **Importer** (en haut) → bouton *Parcourir…* → sélectionne `location_voitures_4e_ttinfo.sql`.
6. Tout en bas, clique **Importer**.
7. Tu dois voir un message vert *« L'importation a été exécutée avec succès »*. Les 3 tables apparaissent dans la barre latérale gauche.

> 💡 **HEIDISQL VS PHPMYADMIN** : tu obtiens **exactement la même base**, parce que les deux parlent au même serveur MySQL. Tu peux ouvrir les deux clients en même temps : ce que tu fais d'un côté apparaît immédiatement de l'autre.

Vérification rapide

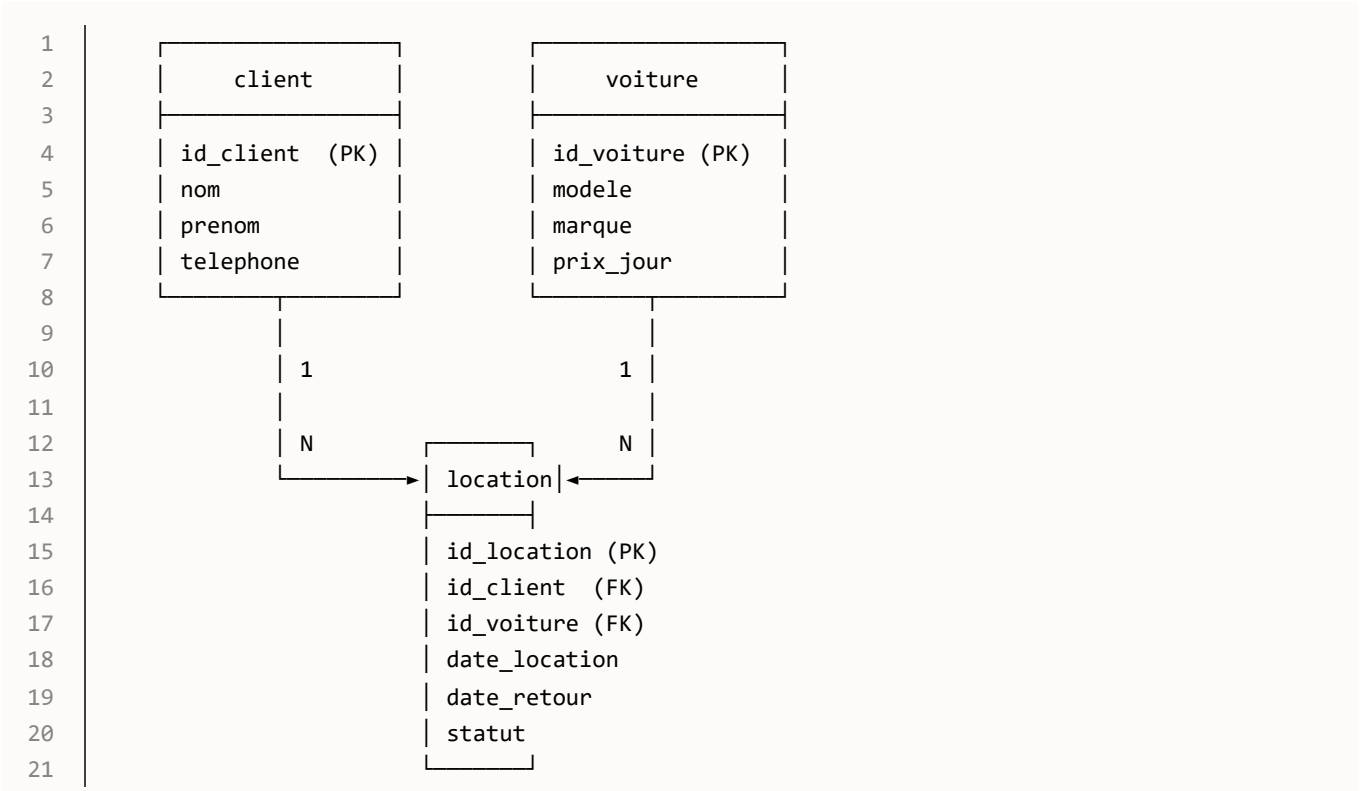
Quel que soit le client utilisé, ouvre une nouvelle requête SQL et tape :

```sql
SELECT COUNT(*) FROM location;
```

Tu dois obtenir **200**. Si oui, ta base est prête.

Partie 1 — Le schéma de la base

La base contient **3 tables** liées par des clés étrangères :



Un client peut faire **plusieurs** locations. Une voiture peut être louée **plusieurs** fois. Chaque ligne de `location` fait le lien entre **un** client et **une** voiture, à une date donnée.

Détail des colonnes

Table `client`

Colonne	Type	Description
`id_client`	`INT` (PK)	Identifiant interne (auto-incrément)
`nom`	`VARCHAR(50)`	Nom de famille
`prenom`	`VARCHAR(50)`	Prénom
`telephone`	`VARCHAR(20)`	Numéro (unique)

Table `voiture`

Colonne	Type	Description
`id_voiture`	`INT` (PK)	Identifiant interne
`modele`	`VARCHAR(50)`	Modèle (Clio, Golf, A3…)
`marque`	`VARCHAR(50)`	Marque (Renault, VW, Audi…)
`prix_jour`	`DECIMAL(6,2)`	Prix de location par jour, en €

Table `location`

Colonne	Type	Description
`id_location`	`VARCHAR(5)` (PK)	Code lisible (`L001` , `L002`)
`id_client`	`INT` (FK → client)	Qui loue
`id_voiture`	`INT` (FK → voiture)	Quelle voiture
`date_location`	`DATE`	Date de début
`date_retour`	`DATE`	Date de retour
`statut`	`ENUM('terminée','en cours','réservée','annulée')`	État de la location

Partie 2 — `SELECT` et `FROM` : lire des lignes

C'est la requête la plus simple. Elle dit *« lis cette table »*.

2.1 — Tout afficher avec `SELECT *`

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

L'étoile `*` veut dire **toutes les colonnes**. Tu obtiens les 20 clients avec toutes leurs infos :

id_client	nom	prenom	telephone
1	Dupont	Marie	0471 12 34 56
2	Martin	Lucas	0485 22 11 00
3	Lambert	Sophie	0492 44 55 66
…	…	…	…

(20 lignes au total)

2.2 — Choisir des colonnes précises

Quand tu n'as besoin que de certaines colonnes, **liste-les** explicitement après `SELECT`, séparées par des virgules :

```sql
SELECT nom, prenom
FROM client;
```

Tu obtiens seulement 2 colonnes (et toujours 20 lignes) :

nom	prenom
Dupont	Marie
Martin	Lucas
Lambert	Sophie
…	…

🧠 **BONNE PRATIQUE** : préfère lister les colonnes plutôt que `SELECT *`. C'est plus lisible, plus rapide, et ça évite les surprises si quelqu'un ajoute une colonne à la table.

2.3 — Limiter le nombre de lignes affichées

Quand une table contient des centaines ou des milliers de lignes (comme `location`, avec ses 200 entrées), `SELECT *` t'inonde la console. La clause `LIMIT n` dit *« ne montre que les n premières lignes »*.

```sql
SELECT *
FROM location
LIMIT 5;
```

Très pratique pour **jeter un œil** à une grande table sans tout charger.

Partie 3 — `WHERE` : filtrer les lignes

`SELECT … FROM …` lit **toutes** les lignes. Si tu veux uniquement celles qui remplissent une condition, ajoute une clause `WHERE`.

3.1 — L'égalité `=`

> *Quelles sont les locations dont le statut est « en cours » ?*

```sql
SELECT id_location, id_client, date_location, date_retour
FROM location
WHERE statut = 'en cours';
```

Tu obtiens uniquement les lignes où `statut` vaut exactement `'en cours'`. Une trentaine de lignes au lieu de 200.

⚠️ **LES CHAÎNES VONT ENTRE GUILLEMETS SIMPLES** : `'en cours'` . Les nombres, eux, s'écrivent sans guillemets : `id_client = 5` . Pas `id_client = '5'` (ça marcherait par chance, mais c'est faux conceptuellement).

3.2 — Les opérateurs de comparaison

Opérateur	Sens	Exemple
`=`	égal	`prix_jour = 50`
`<>` ou `!=`	différent de	`statut <> 'annulée'`
`<`	strictement inférieur	`prix_jour < 50`
`>`	strictement supérieur	`prix_jour > 60`
`<=`	inférieur ou égal	`prix_jour <= 50`
`>=`	supérieur ou égal	`id_client >= 10`

Exemple — Les voitures à plus de 60 €

```sql
SELECT modele, marque, prix_jour
FROM voiture
WHERE prix_jour > 60;
```

Résultat :

modele	marque	prix_jour
A3	Audi	75.00
Civic	Honda	62.00

3.3 — La recherche partielle avec `LIKE`

Pour chercher dans une chaîne sans donner sa valeur exacte, on utilise `LIKE` combiné avec le **joker** `%` (qui remplace zéro ou plusieurs caractères).

> *Quels sont les clients dont le nom commence par « L » ?*

```sql
SELECT nom, prenom
FROM client
WHERE nom LIKE 'L%';
```

Résultat :

nom	prenom
Lambert	Sophie
Leroy	Emma
Laurent	Adam
Leclercq	Tom

💡 **QUELQUES MOTIFS UTILES :**

- `'L%'` → commence par L (Lambert, Leroy…)
- `'%n'` → finit par n (Dupont, Martin, Simon…)
- `'%art%'` → contient « art » (Martin, Bernard…)
- `'D____'` → exactement 5 caractères, commence par D (`_` remplace UN caractère)

3.4 — Combiner plusieurs conditions : `AND` et `OR`

`AND` — toutes les conditions doivent être vraies

*Quelles locations sont **en cours** ET concernent le **client 4** ?*

```sql
SELECT id_location, date_location, date_retour
FROM location
WHERE statut = 'en cours' AND id_client = 4;
```

`OR` — au moins une condition doit être vraie

*Quelles voitures sont des **Renault** OU des **VW** ?*

```sql
SELECT modele, marque, prix_jour
FROM voiture
WHERE marque = 'Renault' OR marque = 'VW';
```

⚠️ **ATTENTION À L'ORDRE DES CONDITIONS** quand tu mélanges `AND` et `OR`. Comme en mathématiques, `AND` a priorité sur `OR`. En cas de doute, **mets des parenthèses** :

```sql
WHERE (marque = 'Renault' OR marque = 'VW') AND prix_jour < 55
```

Partie 4 — `ORDER BY` : trier les résultats

Par défaut, les lignes reviennent dans l'ordre où la base les a en stock — c'est-à-dire **dans n'importe quel ordre**. Pour les trier, on ajoute une clause `ORDER BY`.

4.1 — Trier dans l'ordre croissant

```sql
SELECT modele, marque, prix_jour
FROM voiture
ORDER BY prix_jour;
```

Les voitures sortent triées **du moins cher au plus cher** :

modele	marque	prix_jour
500	Fiat	40.00
Corsa	Opel	42.00
Fiesta	Ford	43.00
Clio	Renault	45.00
Yaris	Toyota	48.00
208	Peugeot	50.00
…	…	…

ℹ️ **`ASC` EST IMPLICITE** : `ORDER BY prix_jour` et `ORDER BY prix_jour ASC` font la même chose. Le mot-clé `ASC` (*ascending*) est juste là pour être explicite.

4.2 — Trier dans l'ordre décroissant avec `DESC`

> *Les voitures les **plus chères** d'abord :*

```sql
SELECT modele, marque, prix_jour
FROM voiture
ORDER BY prix_jour DESC;
```

4.3 — Trier sur plusieurs colonnes

Tu peux trier sur **plusieurs critères** : la première colonne est le critère principal, la suivante départage les égalités.

> *Les clients **par nom**, puis par **prénom** en cas d'homonymie :*

```sql
SELECT nom, prenom, telephone
FROM client
ORDER BY nom ASC, prenom ASC;
```

Partie 5 — Tout assembler

Les quatre clauses `SELECT / FROM / WHERE / ORDER BY` se combinent dans cet ordre :

```sql
SELECT  colonnes              -- QUOI afficher
FROM    table                 -- OÙ chercher
WHERE   conditions            -- QUELLES lignes garder
ORDER BY colonne(s)           -- COMMENT trier
LIMIT n;                      -- (optionnel) Combien
```

Exemple complet

> *Les **10 locations les plus anciennes** dont le **statut est terminée**, triées de la plus ancienne à la plus récente :*

```sql
SELECT id_location, id_client, date_location, date_retour
FROM location
WHERE statut = 'terminée'
ORDER BY date_location ASC
LIMIT 10;
```

🧪 Exercice guidé — Locations chères, en cours

Énoncé. Tu veux la liste des **locations en cours**, **triées de la plus récente à la plus ancienne**, mais seulement **les 5 premières**.

Au lieu d'écrire la requête finale d'un coup, on la **construit par couches** — c'est la bonne façon de faire en SQL.

Étape 1 — Lecture brute

Premier brouillon : afficher toutes les locations, sans filtre.

```sql
SELECT id_location, date_location, statut
FROM location;
```

→ 200 lignes. C'est beaucoup. On commence à filtrer.

Étape 2 — Filtrer par statut

```sql
SELECT id_location, date_location, statut
FROM location
WHERE statut = 'en cours';
```

→ Une trentaine de lignes. On a déjà bien réduit.

Étape 3 — Trier de la plus récente à la plus ancienne

```sql
SELECT id_location, date_location, statut
FROM location
WHERE statut = 'en cours'
ORDER BY date_location DESC;
```

→ Les locations apparaissent maintenant **du plus récent au plus ancien**.

Étape 4 — Garder seulement les 5 premières

```sql
SELECT id_location, date_location, statut
FROM location
WHERE statut = 'en cours'
ORDER BY date_location DESC
LIMIT 5;
```

→ **5 lignes**, exactement comme demandé.

> 🧠 **LE RÉFLEXE À ACQUÉRIR** : ne tape **jamais** une requête complexe d'un seul coup. Commence par `SELECT * FROM table LIMIT 5` pour voir la donnée, puis ajoute **une clause à la fois** en vérifiant le résultat à chaque étape.

🧪 Exercices supplémentaires

Pour chacun, écris la requête, exécute-la dans HeidiSQL ou phpMyAdmin, et **vérifie le nombre de lignes** attendu.

1. Affiche le **nom et prénom** de tous les clients, triés **par nom alphabétique**.
2. Affiche **toutes les colonnes** des voitures dont la marque est **Renault**.
3. Affiche les voitures à **moins de 50 €** par jour, triées **du moins cher au plus cher**.
4. Affiche les voitures dont la marque est **Renault** OU **VW**, triées **par prix décroissant**.
5. Affiche les **locations annulées** (toutes les colonnes).
6. Affiche les **id et téléphone** des clients dont le nom **finit par « t »**.
7. Affiche les locations du **client 4** dont le statut est **réservée**.
8. Affiche les **10 voitures** les **moins chères** (modèle, marque, prix). Indice : `ORDER BY` + `LIMIT`.
9. Affiche les locations dont la **date de location** est en **août 2026** (utilise `BETWEEN '2026-08-01' AND '2026-08-31'`).
10. **Défi** — Affiche les clients dont le **prénom commence par une voyelle**. Indice : `LIKE 'A%' OR LIKE 'E%' OR …`

✍️ À retenir

- Une requête SQL minimale s'écrit toujours dans l'ordre : `SELECT` colonnes, `FROM` table, *(`WHERE`, `ORDER BY`, `LIMIT` optionnels)*.
- `SELECT *` prend toutes les colonnes. `SELECT col1, col2` ne prend que celles listées (à préférer).
- `WHERE` filtre les lignes. Opérateurs : `=`, `<>`, `<`, `>`, `<=`, `>=`, `LIKE`, combinés avec `AND` / `OR`.
- **Les chaînes** vont entre **guillemets simples** : `WHERE statut = 'terminée'`.
- `LIKE` + le joker `%` cherche dans une chaîne (commence par, finit par, contient).
- `ORDER BY` trie. `ASC` = croissant (par défaut), `DESC` = décroissant. Plusieurs colonnes possibles.
- `LIMIT n` garde uniquement les `n` premières lignes — utile pour explorer ou pour un *top N*.
- **Bonne méthode** : construire la requête **couche par couche**, en vérifiant le résultat à chaque étape.

Suite

Tu sais maintenant interroger **une seule table à la fois**. Mais regarde tes requêtes sur `location` : tu vois `id_client = 4` au lieu de « Dubois Nora », et `id_voiture = 11` au lieu de « Fiat 500 ». Pour afficher les **vrais noms** à la place des identifiants, il faut **combiner plusieurs tables** dans une seule requête — c'est le rôle des **jointures**, sujet du cours suivant.