# Cours MySQL

Formateur: Boris Rose

21 septembre 2025

## 1. Introduction à MySQL

**Qu'est-ce que MySQL ?**

MySQL est un **Système de Gestion de Base de Données Relationnelle** (SGBDR). Il permet de stocker, organiser et interroger des données.
Il utilise le langage **SQL** (Structured Query Language) pour interagir avec les données.

**Différences avec d'autres SGBD**

- **MySQL** : rapide, open source, largement utilisé pour le web.
- **MariaDB** : fork de MySQL, 100% compatible.
- **PostgreSQL** : plus avancé pour les fonctions complexes.

**Installation**

- **Linux (Debian/Ubuntu)** :

```
sudo apt update
sudo apt install mysql-server
```

- **macOS (Homebrew)** :

```
brew install mysql
```

- **Windows** : via l'installeur MySQL Workbench.

**Connexion à MySQL**

Pour se connecter en ligne de commande :

```
mysql -u root -p
```

Puis entrer le mot de passe de l'utilisateur `root`.

## 1. Introduction à MySQL

**Qu'est-ce que MySQL ?**

MySQL est un **Système de Gestion de Base de Données Relationnelle** (SGBDR). Il utilise le langage **SQL** (Structured Query Language) pour interagir avec les données :

- Stockage structuré dans des tables.
- Relations entre tables via des clés.
- Gestion multi-utilisateurs.

Connexion à MySQL

```
mysql -u root -p
```

## 2. Création et gestion des bases de données

Créer une base de données

```
CREATE DATABASE entreprise;
```

Afficher les bases existantes

```
SHOW DATABASES;
```

Utiliser une base de données

```
USE entreprise;
```

Supprimer une base de données

```
DROP DATABASE entreprise;
```

## 2. Création de tables et manipulation des données (CRUD)

Créer une table

```
CREATE TABLE employes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    poste VARCHAR(50),
    salaire DECIMAL(10,2)
);
```

Insérer des données (INSERT)

```
INSERT INTO employes (nom, prenom, poste, salaire)
VALUES ('Dupont', 'Jean', 'Développeur', 2500.00);

INSERT INTO employes (nom, prenom, poste, salaire)
VALUES ('Martin', 'Sophie', 'Chef de projet', 3200.00);
```

### Lire les données (SELECT)

```sql
-- Tous les enregistrements
SELECT * FROM employes;

-- Avec condition
SELECT nom, prenom, salaire
FROM employes
WHERE salaire > 3000;
```

### Modifier les données (UPDATE)

```sql
UPDATE employes
SET salaire = 2800
WHERE nom = 'Dupont';
```

### Supprimer des données (DELETE)

```sql
DELETE FROM employes
WHERE nom = 'Martin';
```

## 3. Requêtes avancées (jointures, regroupements, sous-requêtes)

### Jointures internes (INNER JOIN)

```sql
-- Exemple avec deux tables : employes et departements
SELECT e.nom, e.prenom, d.nom AS departement
FROM employes e
INNER JOIN departements d ON e.departement_id = d.id;
```

### Jointures externes (LEFT / RIGHT JOIN)

```sql
-- LEFT JOIN : tous les employés, même sans département
SELECT e.nom, e.prenom, d.nom AS departement
FROM employes e
LEFT JOIN departements d ON e.departement_id = d.id;

-- RIGHT JOIN : tous les départements, même sans employés
SELECT e.nom, e.prenom, d.nom AS departement
FROM employes e
RIGHT JOIN departements d ON e.departement_id = d.id;
```

### Regroupement et agrégats (GROUP BY/ HAVING)

```sql
-- Salaire moyen par département
SELECT d.nom AS departement, AVG(e.salaire) AS salaire_moyen
FROM employes e
INNER JOIN departements d ON e.departement_id = d.id
GROUP BY d.nom;

-- Filtrer les départements dont le salaire moyen > 3000
SELECT d.nom AS departement, AVG(e.salaire) AS salaire_moyen
FROM employes e
INNER JOIN departements d ON e.departement_id = d.id
GROUP BY d.nom
HAVING AVG(e.salaire) > 3000;
```

### Sous-requêtes (Subqueries)

```sql
-- Employés dont le salaire est supérieur à la moyenne
SELECT nom, prenom, salaire
FROM employes
WHERE salaire > (SELECT AVG(salaire) FROM employes);

-- Départements ayant plus de 5 employés
SELECT nom
FROM departements
WHERE id IN (
    SELECT departement_id
    FROM employes
    GROUP BY departement_id
    HAVING COUNT(*) > 5
);
```

### Tri et limitation (ORDER BY / LIMIT)

```sql
-- Trier par salaire décroissant
SELECT nom, prenom, salaire
FROM employes
ORDER BY salaire DESC;

-- Obtenir les 3 plus hauts salaires
SELECT nom, prenom, salaire
FROM employes
ORDER BY salaire DESC
LIMIT 3;
```

## 4. Contraintes, clés étrangères et index

### Contraintes de base (PRIMARY KEY | NOT NULL | UNIQUE | DEFAULT)

```sql
-- Création d'une table avec contraintes
CREATE TABLE departements (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(100) NOT NULL UNIQUE,
  actif TINYINT(1) NOT NULL DEFAULT 1
);
```

## Clés étrangères (FOREIGN KEY) avec actions

```sql
-- InnoDB requis pour les FKs
CREATE TABLE employes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(50) NOT NULL,
  prenom VARCHAR(50) NOT NULL,
  salaire DECIMAL(10,2) NOT NULL,
  departement_id INT NULL,
  CONSTRAINT fk_employes_dept
    FOREIGN KEY (departement_id)
    REFERENCES departements(id)
    ON DELETE SET NULL   -- alternatives: RESTRICT | CASCADE
    ON UPDATE CASCADE
) ENGINE=InnoDB;
```

```sql
-- Ajouter une contrainte après création
ALTER TABLE employes
ADD CONSTRAINT fk_employes_dept
FOREIGN KEY (departement_id) REFERENCES departements(id)
ON DELETE SET NULL ON UPDATE CASCADE;

-- Supprimer une contrainte de clé étrangère
ALTER TABLE employes
DROP FOREIGN KEY fk_employes_dept;
```

## Clés primaires composées et contraintes composées

```sql
-- Exemple de PK composite
CREATE TABLE affectations (
  employe_id INT NOT NULL,
  projet_id  INT NOT NULL,
  role VARCHAR(40) NOT NULL,
  PRIMARY KEY (employe_id, projet_id),          -- composite
  CONSTRAINT fk_aff_emp FOREIGN KEY (employe_id) REFERENCES employes(id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_aff_proj FOREIGN KEY (projet_id)  REFERENCES projets(id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT uq_role_par_projet UNIQUE (projet_id, role)
) ENGINE=InnoDB;
```

## CHECK (MySQL 8.0+)

```sql
-- Les CHECK sont appliqués depuis MySQL 8.0.16+
ALTER TABLE employes
ADD CONSTRAINT chk_salaire_positif CHECK (salaire >= 0);

-- Exemple multi-condition
ALTER TABLE employes
ADD CONSTRAINT chk_noms CHECK (CHAR_LENGTH(nom) >= 2 AND CHAR_LENGTH(prenom) >= 2);
```

### Index : créer/ supprimer /lister

```sql
-- Index simple (BTREE par défaut)
CREATE INDEX idx_emp_salaire ON employes (salaire);

-- Index composite (l'ordre des colonnes compte pour les recherches)
CREATE INDEX idx_emp_nom_prenom ON employes (nom, prenom);

-- Index préfixe (utile sur VARCHAR longs)
CREATE INDEX idx_emp_nom_prefix ON employes (nom(20));

-- Supprimer un index
DROP INDEX idx_emp_salaire ON employes;

-- Voir les index d'une table
SHOW INDEX FROM employes;
```

### Index spécialisés : FULLTEXT et UNIQUE

```sql
-- Index UNIQUE (empêche les doublons)
CREATE UNIQUE INDEX uq_dept_nom ON departements(nom);

-- FULLTEXT (InnoDB/MyISAM, colonnes TEXT/VARCHAR)
CREATE FULLTEXT INDEX ft_emp_nom_prenom ON employes (nom, prenom);

-- Recherche plein texte (mode naturel)
SELECT id, nom, prenom
FROM employes
WHERE MATCH(nom, prenom) AGAINST ('"Jean Dupont"' IN NATURAL LANGUAGE MODE);

-- Mode booléen (opérateurs +, -, *)
SELECT id, nom, prenom
FROM employes
WHERE MATCH(nom, prenom) AGAINST ('+Jean +dev* -stagiaire' IN BOOLEAN MODE);
```

### Bonnes pratiques express

- Utiliser `ENGINE=InnoDB` pour le support ACID et les clés étrangères.
- Indexer les colonnes utilisées dans les `JOIN`, `WHERE`, `ORDER BY`.
- Préférer des index composés alignés avec les prédicats les plus sélectifs en premier.
- Éviter de multiplier les index redondants (coût d'écriture, stockage).
- Vérifier l'impact avec `EXPLAIN` (vu dans la partie Performance).

## 5. Transactions et gestion de la concurrence

### Notion de transaction

Une **transaction** est une suite d'opérations SQL exécutées comme une unité logique. Les transactions respectent les propriétés **ACID** :

- **Atomicité** : tout ou rien.
- **Cohérence** : l'état final doit être valide.
- **Isolation** : les transactions ne se perturbent pas mutuellement.
- **Durabilité** : une fois validée, une transaction survit aux pannes.

### Syntaxe de base des transactions

```sql
START TRANSACTION;

UPDATE comptes SET solde = solde - 100 WHERE id = 1;
UPDATE comptes SET solde = solde + 100 WHERE id = 2;

COMMIT;   -- Valider
-- ou
ROLLBACK; -- Annuler
```

### Points de sauvegarde (SAVEPOINT)

```sql
START TRANSACTION;

UPDATE comptes SET solde = solde - 100 WHERE id = 1;
SAVEPOINT apres_debit;

UPDATE comptes SET solde = solde + 100 WHERE id = 2;

ROLLBACK TO apres_debit; -- Annule le crédit mais conserve le débit
COMMIT;
```

### Niveaux d'isolation

MySQL propose plusieurs niveaux d'isolation pour contrôler la visibilité des changements :

- **READ UNCOMMITTED** : peut lire des données non validées (*dirty reads*).
- **READ COMMITTED** : lit uniquement les données validées.
- **REPEATABLE READ** (par défaut InnoDB) : garantit des lectures cohérentes pendant la transaction.
- **SERIALIZABLE** : le plus strict, exécute les transactions comme si elles étaient séquentielles.

```sql
-- Vérifier le niveau actuel
SELECT @@transaction_isolation;

-- Changer le niveau
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

### Verrous (Locks)

- **Shared locks (S)** : plusieurs lectures simultanées.
- **Exclusive locks (X)** : écriture exclusive.
- **Row-level locking** : InnoDB verrouille au niveau ligne (meilleure concurrence).
- **Table-level locking** : MyISAM verrouille toute la table.

```sql
-- Verrouiller une table
LOCK TABLES employes WRITE;

-- Opérations critiques ici

UNLOCK TABLES;
```

**Bonnes pratiques transactions**

- Garder les transactions **courtes** pour réduire la contention.
- Toujours gérer explicitement `COMMIT` ou `ROLLBACK`.
- Préférer l'**isolation par défaut REPEATABLE READ** pour un bon compromis.
- Surveiller les verrous bloquants via `SHOW ENGINE INNODB STATUS`.

## 6. Fonctions, vues et procédures stockées

**Fonctions intégrées utiles**

MySQL propose de nombreuses fonctions pour manipuler les données :

- **Chaînes** : `CONCAT()`, `UPPER()`, `LOWER()`, `SUBSTRING()`.
- **Numériques** : `ROUND()`, `CEIL()`, `FLOOR()`, `MOD()`.
- **Dates** : `NOW()`, `CURDATE()`, `DATE_ADD()`, `DATEDIFF()`.
- **Agrégats** : `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`.

```sql
-- Exemple d'utilisation
SELECT nom, UPPER(prenom), ROUND(salaire, 0)
FROM employes
WHERE YEAR(CURDATE()) - YEAR(date_embauche) > 5;
```

**Créer une vue**

Une **vue** est une requête enregistrée comme une table virtuelle.

```sql
CREATE VIEW vue_employes_high_salary AS
SELECT nom, prenom, salaire
FROM employes
WHERE salaire > 3000;

-- Utilisation
SELECT * FROM vue_employes_high_salary;
```

**Note** : Les vues peuvent être utilisées pour simplifier les requêtes ou restreindre l'accès aux colonnes.

**Procédures stockées**

Une **procédure stockée** est un bloc SQL enregistré dans le serveur.

```sql
DELIMITER $$

CREATE PROCEDURE augmenter_salaire(IN emp_id INT, IN pourcentage DECIMAL(5,2))
BEGIN
  UPDATE employes
  SET salaire = salaire * (1 + pourcentage/100)
  WHERE id = emp_id;
END $$

DELIMITER ;

-- Appel de la procédure
CALL augmenter_salaire(1, 10);
```

### Fonctions stockées

Une **fonction stockée** retourne une valeur unique.

```
DELIMITER $$

CREATE FUNCTION salaire_annuel(emp_id INT) RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE annuel DECIMAL(10,2);
  SELECT salaire * 12 INTO annuel
  FROM employes
  WHERE id = emp_id;
  RETURN annuel;
END $$

DELIMITER ;

-- Utilisation
SELECT nom, prenom, salaire_annuel(id) AS revenu
FROM employes;
```

### Déclencheurs (TRIGGERS)

Les **triggers** permettent d'exécuter automatiquement une action lors d'un événement.

```
DELIMITER $$

CREATE TRIGGER avant_insert_employe
BEFORE INSERT ON employes
FOR EACH ROW
BEGIN
  IF NEW.salaire < 0 THEN
    SET NEW.salaire = 0;
  END IF;
END $$

DELIMITER ;
```

**Exemple** : Ici, on empêche l'insertion d'un salaire négatif.

## 7. Sécurité et gestion des utilisateurs

### Création et gestion des utilisateurs

En MySQL, chaque utilisateur est défini par un couple `utilisateur@hôte`.

```
-- Créer un utilisateur
CREATE USER 'alice'@'localhost' IDENTIFIED BY 'motdepasse';

-- Créer un utilisateur accessible depuis n'importe quelle machine
CREATE USER 'bob'@'%' IDENTIFIED BY 'motdepasse';
```

### Accorder des privilèges (GRANT)

```
-- Donner tous les droits sur une base
GRANT ALL PRIVILEGES ON entreprise.* TO 'alice'@'localhost';

-- Donner uniquement SELECT et INSERT sur une table
GRANT SELECT, INSERT ON entreprise.employes TO 'bob'@'%';

-- Appliquer les changements
FLUSH PRIVILEGES;
```

### Révoquer des privilèges (REVOKE)

```
-- Retirer le droit INSERT à Bob
REVOKE INSERT ON entreprise.employes FROM 'bob'@'%';
```

### Gestion des mots de passe

```
-- Changer le mot de passe
ALTER USER 'alice'@'localhost' IDENTIFIED BY 'nouveauMDP';

-- Forcer la réinitialisation du mot de passe à la prochaine connexion
ALTER USER 'bob'@'%' PASSWORD EXPIRE;
```

### Rôles (MySQL 8.0+)

```
-- Créer un rôle
CREATE ROLE lecteur;

-- Accorder des droits au rôle
GRANT SELECT ON entreprise.* TO lecteur;

-- Attribuer un rôle à un utilisateur
GRANT lecteur TO 'bob'@'%';

-- Activer un rôle par défaut
SET DEFAULT ROLE lecteur TO 'bob'@'%';
```

### Bonnes pratiques sécurité

- Ne jamais utiliser `root` pour les applications.
- Créer des utilisateurs spécifiques avec le minimum de droits nécessaires.
- Utiliser des mots de passe forts et, si possible, l'authentification par plugin (`caching_sha2_password`).
- Surveiller les connexions via les logs MySQL.

## 8. Sauvegarde, restauration et optimisation

### Sauvegarde avec mysqldump

L'outil `mysqldump` permet de sauvegarder une base ou l'ensemble des bases.

```
# Sauvegarder une base
mysqldump -u root -p entreprise > entreprise.sql

# Sauvegarder toutes les bases
mysqldump -u root -p --all-databases > backup_all.sql

# Sauvegarder uniquement la structure sans données
mysqldump -u root -p --no-data entreprise > structure.sql
```

### Restauration

```
# Restaurer une base
mysql -u root -p entreprise < entreprise.sql

# Restaurer toutes les bases
mysql -u root -p < backup_all.sql
```

### Optimisation des requêtes avec EXPLAIN

La commande `EXPLAIN` permet de comprendre comment MySQL exécute une requête.

```
EXPLAIN SELECT nom, prenom
FROM employes
WHERE salaire > 3000
ORDER BY nom;
```

Elle affiche l'utilisation des index, les scans de table et les jointures.

### Analyse et optimisation des tables

```
-- Vérifier et réparer une table
CHECK TABLE employes;
REPAIR TABLE employes;

-- Optimiser une table (reconstruction, libération d'espace)
OPTIMIZE TABLE employes;
```

### Index et performance

- Utiliser des index sur les colonnes de filtrage et jointure.
- Éviter les colonnes avec faible sélectivité (ex : booléens).
- Préférer les index composés alignés avec les clauses WHERE.
- Surveiller les requêtes lentes via `slow query log`.

**Bonnes pratiques de sauvegarde**

- Automatiser les sauvegardes avec des scripts (cron, systemd timer).
- Stocker les sauvegardes sur un autre serveur ou dans le cloud.
- Tester régulièrement la restauration (sauvegarde inutile si non testée).
- Utiliser `-single-transaction` avec InnoDB pour des dumps cohérents sans bloquer les tables.

## 9. Réplication et haute disponibilité

**Principe de la réplication MySQL**

La réplication permet de copier automatiquement les données d'un **serveur maître** vers un ou plusieurs **serveurs esclaves** :

- **Master-Slave** : le maître reçoit les écritures, les esclaves répliquent en lecture seule.
- **Master-Master** : chaque serveur peut écrire, mais nécessite une bonne gestion des conflits.
- **Asynchrone** (par défaut) ou **semi-synchrone**.

Avantages : tolérance de panne, montée en charge (lecture sur les esclaves), sauvegarde sans impacter la production.

**Configuration côté maître**

```
-- Éditer my.cnf (ou mysqld.cnf) et activer :
[mysqld]
server-id = 1
log_bin = /var/log/mysql/mysql-bin.log

-- Redémarrer MySQL puis créer un utilisateur dédié :
CREATE USER 'repl'@'%' IDENTIFIED BY 'motdepasse';
GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%';
FLUSH PRIVILEGES;

-- Verrouiller les tables pour snapshot cohérent
FLUSH TABLES WITH READ LOCK;

-- Vérifier les infos de binlog
SHOW MASTER STATUS;
```

### Configuration côté esclave

```
-- Dans my.cnf
[mysqld]
server-id = 2

-- Lancer la réplication
CHANGE MASTER TO
  MASTER_HOST='ip_master',
  MASTER_USER='repl',
  MASTER_PASSWORD='motdepasse',
  MASTER_LOG_FILE='mysql-bin.000001',
  MASTER_LOG_POS=154;

START SLAVE;

-- Vérifier l'état
SHOW SLAVE STATUS\G
```

### Haute disponibilité avec MySQL

- **Réplication + failover** : basculer vers un esclave en cas de panne du maître.
- **MySQL Router** ou **ProxySQL** pour rediriger les connexions vers les nœuds disponibles.
- **Group Replication** (MySQL 5.7+) : cluster multi-maîtres avec consensus automatique.
- **InnoDB Cluster** : solution clé en main de haute disponibilité (MySQL Shell + Group Replication + Router).

### Bonnes pratiques réplication et HA

- Sécuriser le compte de réplication avec des privilèges minimaux.
- Surveiller la latence de réplication (`Seconds_Behind_Master`).
- Utiliser des sauvegardes physiques (Percona XtraBackup) pour initialiser rapidement les esclaves.
- Mettre en place une supervision (Zabbix, Prometheus, etc.) pour détecter les pannes.