COMPOSANTS ACCES BDD

JOUR 1 : COMPOSANTS ACCES SQL

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE SEMAINE

- **Composants d’accès SQL**
- Validation des données
- MongoDB
- Composants d’accès aux données NoSQL

COMPOSANTS ACCES SQL

LES OBJECTIFS

Comprendre les bases de l'accès aux bases de données SQL en Node.js

Mettre en place une base de données et s'y connecter via Node.js

Comprendre les requêtes préparées

RAPPEL SQL

NUAGE DE MOTS

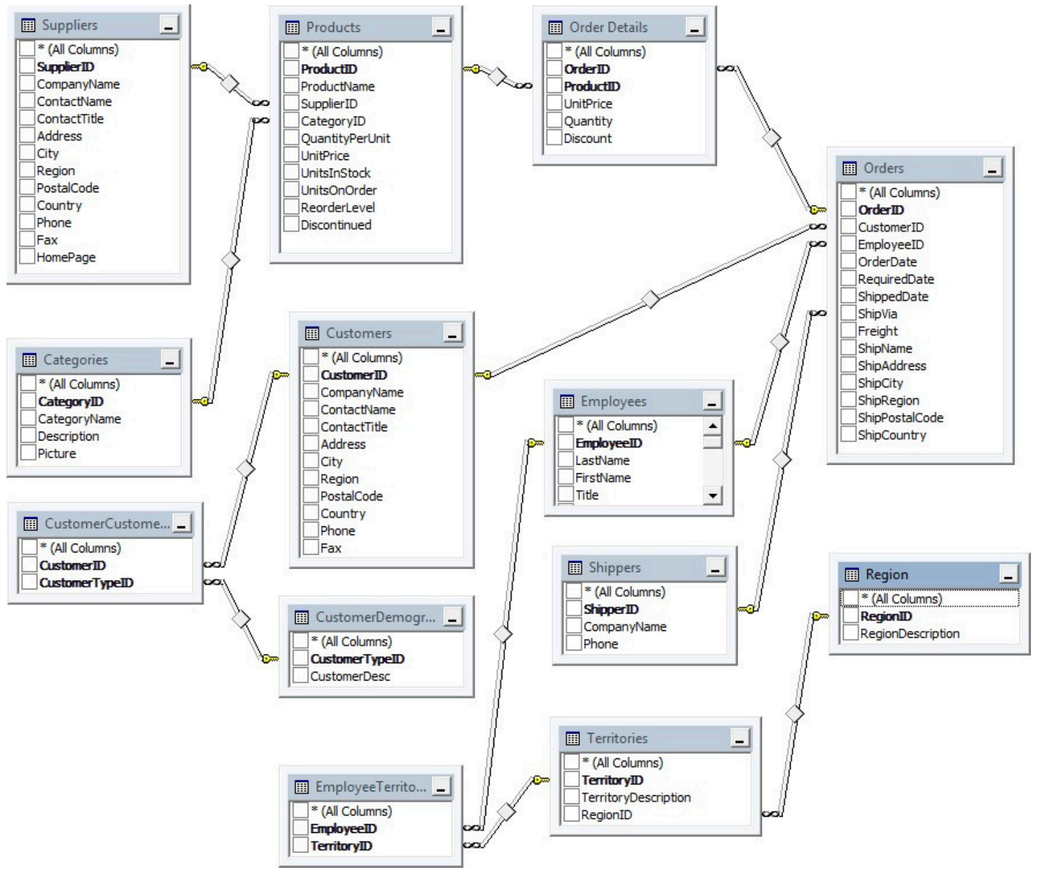
Que vous souvenez vous comme requêtes SQL ?

https://www.menti.com/aldpp4773piz

QU'EST-CE QU'UNE BASE DE DONNÉES RELATIONNELLE ?

- Un Système de Gestion de Base de Données (SGBD) **stocke** et **organise** des données structurées.

- Le modèle relationnel repose sur des tables **interconnectées** par des **relations**, garantissant l'**intégrité** et la **cohérence**.



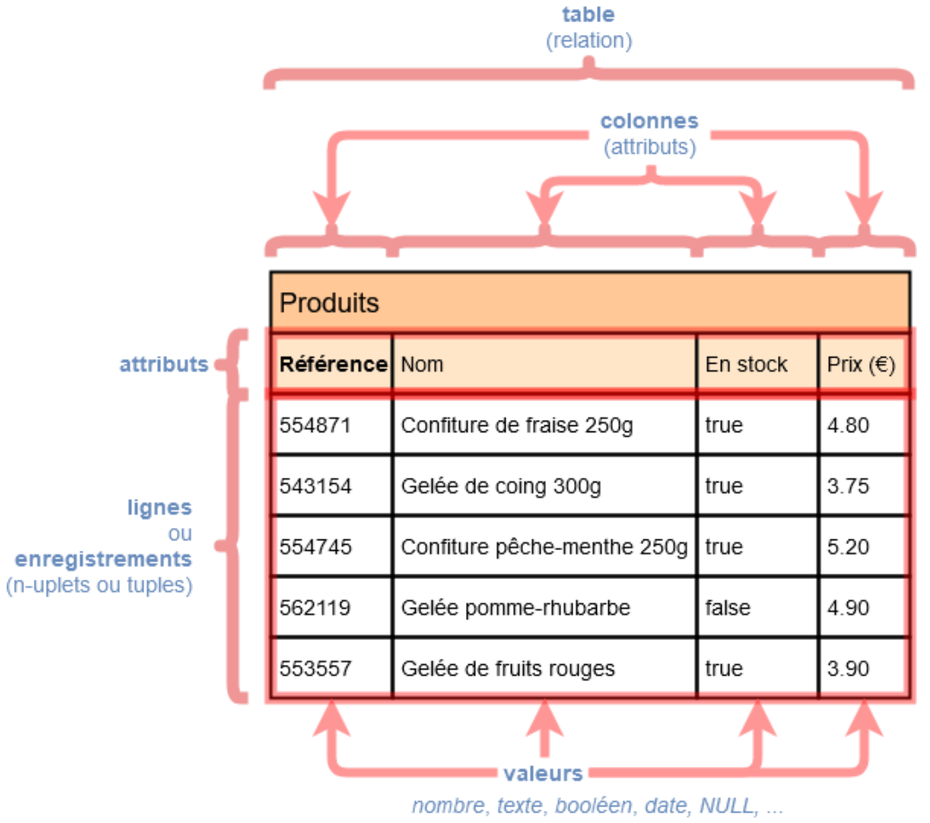
CONCEPTS CLÉS

- **Table** (entité) : structure bidimensionnelle composée de lignes (enregistrements) et de colonnes (attributs). Chaque table représente une entité métier (ex. clients, produits).

- **Colonne** (attribut) : définition d'un attribut, avec un type de données (entier, chaîne de caractères, date, booléen, etc.) et des contraintes (NOT NULL, UNIQUE, DEFAULT, CHECK).

- **Ligne** (entrée ou tuple) : instance d'une table, contenant une valeur pour chaque colonne. Chaque ligne est identifiée par une clé primaire garantissant l'unicité.

- **Valeurs** : Croisement entre une ligne et une colonne



Produits			
Référence	Nom	En stock	Prix (€)
554871	Confiture de fraise 250g	true	4.80
543154	Gelée de coing 300g	true	3.75
554745	Confiture pêche-menthe 250g	true	5.20
562119	Gelée pomme-rhubarbe	false	4.90
553557	Gelée de fruits rouges	true	3.90

CLES PRIMAIRES ET SECONDAIRES

- Clé primaire : identifiant unique, souvent auto-incrémenté ou UUID.
- Clé étrangère : lien vers la clé primaire d'une autre table, permet d'assurer la référentialité.

Table "Commande"

Clé étrangère

Id commande	Produit	Prix	Id client
100	PC 16'	599	1
101	Souris	29	2
102	Lampe de bureau	30	2
103	Clavier	55	2

Clé primaire

Table "Client"

Id client	Prenom	Nom	Adresse
1	Jean	Bonneau	50 av St Marc
2	Ambre	Cerna	13 Grande Rue

Clé primaire

CREER SA PREMIERE BASE DE DONNEES

Créer sa base de données

```
CREATE DATABASE librairie;
USE librairie;
```

Vérifier la database

```
SELECT DATABASE();
```

Créer la table livres

```
CREATE TABLE livres (
 id INT AUTO_INCREMENT PRIMARY KEY,
 titre VARCHAR(255) NOT NULL,
 auteur VARCHAR(100) NOT NULL,
 date_publication DATE
);
```

Vérifier la structure

```
DESCRIBE livres;
```

AJOUTER DES VALEURS

Créer la table livres

```
INSERT INTO livres (titre, auteur, date_publication) VALUES
 ('L\'Étranger', 'Albert Camus', '1942-05-19'),
 ('1984', 'George Orwell', '1949-06-08'),
 ('Le Petit Prince', 'Antoine de Saint-Exupéry', '1943-04-06');
```

Voir les livres

```
SELECT * FROM livres;
```

AJOUTER UNE COLONNE

Modifier la table

```
ALTER TABLE livres
ADD COLUMN age_livre INT ;
```

```
UPDATE livres
SET age_livre = YEAR(CURDATE() - YEAR(date_publication)
```

INTERROGER LES DONNEES

SELECT

```
SELECT *
FROM  livres ;
```

```
SELECT auteur, titre
FROM  livres ;
```

```
SELECT *
FROM livres
WHERE date_publication > '1945-01-01'
```

TRI ET PAGINATION

```
SELECT titre
FROM livres
ORDER BY date_publication DESC
LIMIT 2;
```

```
SELECT titre
FROM livres
ORDER BY date_publication ASC
LIMIT 2;
```

INSTALLATION

PACKAGE MYSQL2

```
npm install mysql2
```

Exemple avec driver natif

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'my_database'
});

connection.connect(err => {
  if (err) throw err;
  console.log('Connected to MySQL!');
});

```

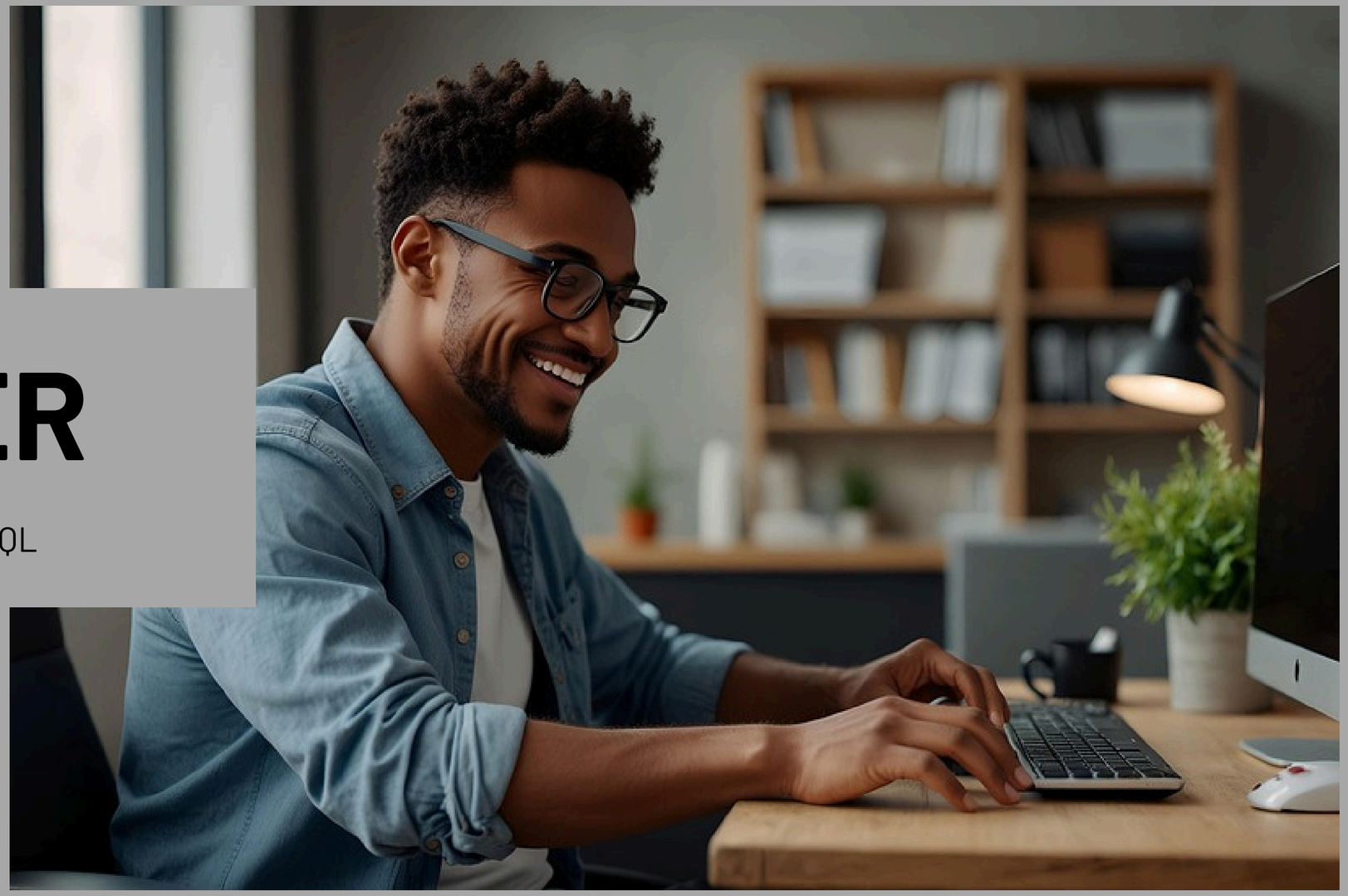
EXECUTION DE REQUETE SIMPLE

Exemple avec driver natif

```
connection.query('SELECT * FROM users', (err, results)
 => {
  if (err) throw err;
  console.log(results);
});
```

ATELIER

Créer une connexion SQL



CONSIGNES

Etape 1 : Installer package mysql2

npm install mysql2

Etape 2 : Créer la connexion

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'my_database'
});

connection.connect(err => {
  if (err) throw err;
  console.log('Connected to MySQL!');
});
```

Etape 3 : choisir une database et modifier les identifiants de connexion

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'my_database'
});

connection.connect(err => {
  if (err) throw err;
  console.log('Connected to MySQL!');
});
```

Etape 4 : Créer une requête simple

```
connection.query('SELECT * FROM
users', (err, results) => {
  if (err) throw err;
  console.log(results);
});
```

SECURITE BASE DE DONNEES

LES INJECTIONS SQL

Définition

Une injection SQL (ou SQL Injection) est une faille de sécurité qui permet à un utilisateur malveillant d'injecter du code SQL dans une requête, afin de lire, modifier ou supprimer des données de manière non autorisée

Comment se produit elle ?

```
1 const email = req.query.email;
2 const query = `SELECT * FROM users WHERE email =
  '${email}'`;
3
4 connection.query(query, (err, results) => {
5   console.log(results);
6 });
```

Si un utilisateur saisit DROP TABLE
à la place du mail on perd toutes les données !!

LES DIFFERENTS TYPES D'INJECTIONS

Authentification contournée

```
1 SELECT * FROM users WHERE username = 'admin' AND
  password = '' OR '1'='1'
2
```

Autorise n'importe quel mot de passe

Vol de données sensibles

```
1 ' UNION SELECT null, password FROM users --
2
```

Combine les données sensibles à la requête initiale

Suppression de données

```
1 email = 'anything'; DELETE FROM users; --
2
```

Supprime toutes les lignes si la requêtes SQL autorise plusieurs statements.

SOLUTION : LES REQUÊTES PRÉPARÉES

Une requête préparée (ou prepared statement) sépare :

Étape	Action	Exemple
Préparation	Le SGBD **précompile la structure SQL**, avec des **emplacements pour les valeurs**	SELECT * FROM users WHERE email = ?
Exécution	Le SGBD reçoit les **valeurs séparément**, et les **associe aux emplacements**	[email]

Le SQL est **compilé avant** de **savoir** quelles seront les **valeurs**, donc aucun code malicieux ne peut être injecté dans la structure.

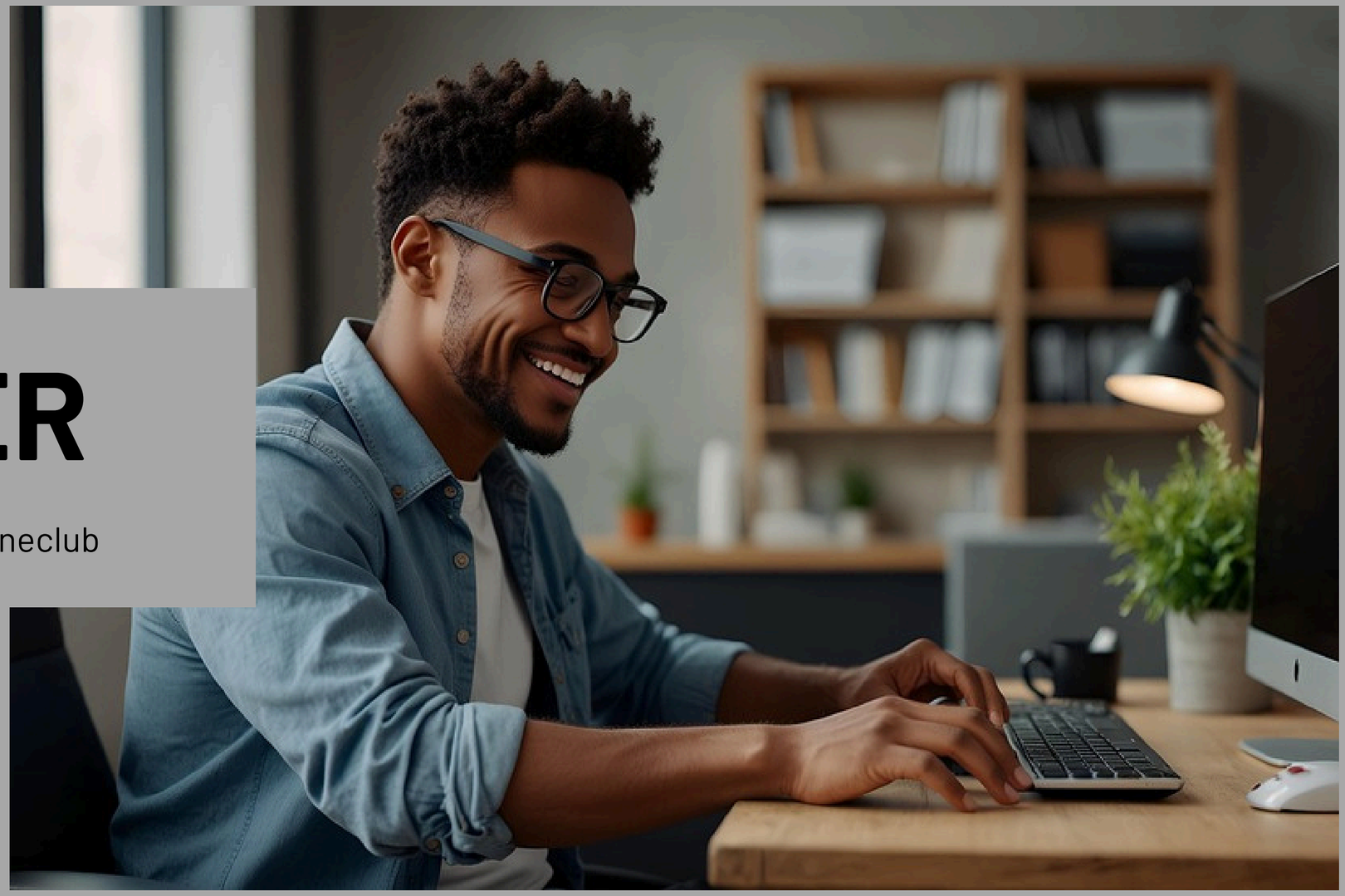
EXEMPLE

Avec la requête préparée, on évite que le code ne s'execute dans la requête mais on l'execute en tant que valeur

```
1 const email = req.query.email;
2 const sql = "SELECT * FROM users WHERE email = ?";
3 connection.execute(sql, [email], (err, results) => {
4   console.log(results);
5 });
```

ATELIER

Connexion à la BDD pour cineclub



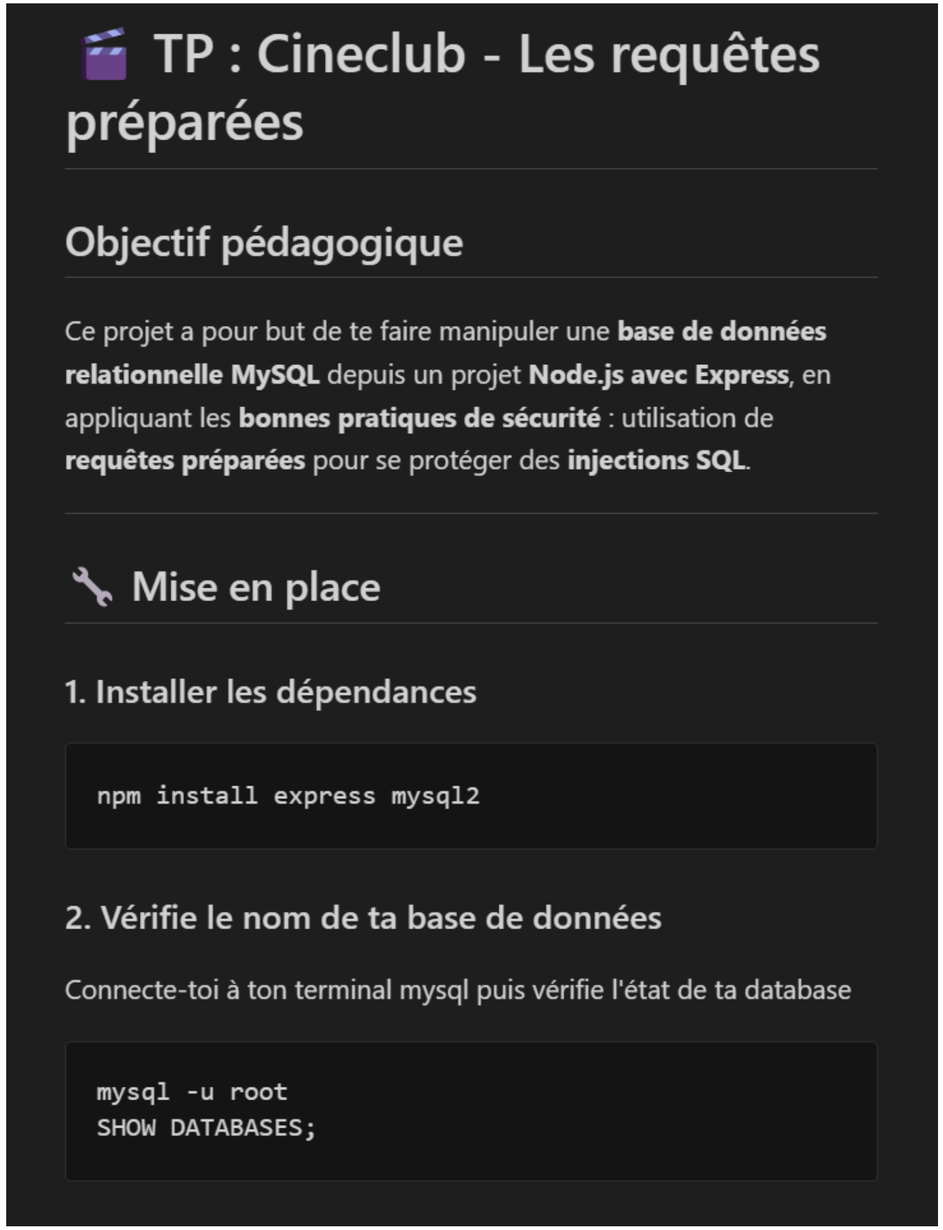
CONSIGNES

https://github.com/dessna12/01-Components-SQL

Jusqu'alors nous avons réalisé une persistance de données à l'aide de fichier json.

Nous allons maintenant nous connecter à notre vraie base de données SQL.

Réalisez les requêtes préparées du projet.



🎬 TP : Cineclub - Les requêtes préparées

Objectif pédagogique

Ce projet a pour but de te faire manipuler une **base de données relationnelle MySQL** depuis un projet **Node.js avec Express**, en appliquant les **bonnes pratiques de sécurité** : utilisation de **requêtes préparées** pour se protéger des **injections SQL**.

🔧 Mise en place

1. Installer les dépendances

```
npm install express mysql2
```

2. Vérifie le nom de ta base de données

Connecte-toi à ton terminal mysql puis vérifie l'état de ta database

```
mysql -u root
SHOW DATABASES;
```

QUIZZ