

# W6 PRACTICE


## My SQL

 *At the end of his practice, you should be able to...*


- ✓ Establish a **MySQL connection** on the back-end app
- ✓ Implement a **repository** layer using **MySQL queries**
- ✓ **Test the endpoints** (REST API client + front-end app)
- ✓ **Extends the project** to handle **4 tables** in the database

 *How to start?*

- ✓ Download **start code** from related MS Team assignment
- ✓ Run `npm install` on both front and back projects
- ✓ Run `npm run dev` on both front and back projects to run the client and the server

 *How to submit?*

- ✓ Submit your **code** on MS Team assignment

 *Are you lost?*

*To review MySQL queries syntax*

[https://www.w3schools.com/mysql/mysql\\_sql.asp](https://www.w3schools.com/mysql/mysql_sql.asp)

*To install My SQL Server (if needed)*

<https://dev.mysql.com/doc/refman/8.4/en/windows-installation.html>

<https://dev.mysql.com/downloads/>

*To connect Node back end to MySQL*

[https://www.w3schools.com/nodejs/nodejs\\_mysql.asp](https://www.w3schools.com/nodejs/nodejs_mysql.asp)

<https://sidorares.github.io/node-mysql2/docs/documentation>

# EXERCISE 1 – MySQL Manipulation

Before starting !

You should have a MySQL Server running. Check it out with bellow command:

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

You should see MySQL monitor run properly:

```
C:\Users\PC>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 9.3.0 MySQL Community Server - GPL
```

If not, you need to install and configure MySQL server properly.

<https://dev.mysql.com/doc/refman/8.4/en/windows-installation.html>

## Q1 - Create the database and the table of articles

- Open the terminal and launch MySQL monitor:

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

- Create a new database (e.g. **week6Db**) using the command line
- Create a new table (articles) with the columns below:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
title	varchar(255)	YES		NULL	
content	text	YES		NULL	
journalist	varchar(100)	YES		NULL	
category	varchar(50)	YES		NULL	

## Q2 - Review My SQL queries

- Complete the bellow table with the appropriate MySQL query

Use case	My SQL Query
Get all articles	<code>SELECT * FROM articles</code>
Get articles written by the journalist 'RONAN'	<code>SELECT * FROM articles WHERE journalist = 'RONAN';</code>
Add an article	<code>INSERT INTO articles(title, content, journalist, category) VALUES ( 'Spirited Away',</code>

	<pre>'Spirited Away is a 2001 Japanese animated fantasy film written and directed by Hayao Miyazaki. It was produced by Toshio Suzuki, animated by Studio Ghibli, and distributed by Toho. The film stars Rumi Hiiragi, alongside Miyu Irino, Mari Natsuki, Takashi Naito, Yasuko Sawaguchi, Tsunehiko Kamijō, Takehiko Ono, and Bunta Sugawara. In Spirited Away, Chihiro "Sen" Ogino moves to a new neighborhood and inadvertently enters the world of kami (spirits of Japanese Shinto folklore). After her parents are turned into pigs by the witch Yubaba, Chihiro takes a job working in Yubaba\'s bathhouse to find a way to free herself and her parents and return to the human world.',     'Hayao Miyazaki',     'fantasy' );</pre>
Delete all articles whose title starts with "R"	<pre>DELETE FROM articles WHERE title LIKE 'R%';</pre>

## EXERCISE 2 – MySQL on Backend

For this exercise, you start with a start frontend and a backend code.

The goal for this exercise is to replace the provided mock repository with a MySQL repository.

### Q1 - Run Frontend & Backend

Open a dedicated terminal to run the server:

```
cd back  
npm i  
npm run dev
```

Open a dedicated terminal to run the client:

```
cd front  
npm i  
npm run dev
```

Open the browser and check the front end is correctly **connected with the back end** :



The project already works as we provide fake data (mock repository).

Let's understand in detail the back and front ends.

## Q2 - Look at ArticleForm

How does the component know whether to create a new article or update an existing one?

```
The component using the operation  if (isEdit) {  
                                   await updateArticle(id, formData);  
                                   } else {  
                                   await createArticle(formData);  
                                   }
```

to perform a logic handle which mean when the isEdit IF TRUE performs the Edit Article updateArticle(id, formData) (i.e., update the article with that id). IF FALSE perform Create Article createArticle(formData) (i.e., create a new article).

Why is the **useParams** hook used in this component? What value does it provide when isEdit is true?

The **useParams** hook is used to extract route parameters from the URL. In this component, it retrieves the id of the article from the URL when editing an existing article.

When isEdit is true, the component needs to know which article to update, and useParams() provides the id value needed to:

Fetch the article's data from the server (getArticleById(id))

Send the correct article ID when calling updateArticle(id, formData)

Example:

If the URL is /articles/edit/7, then useParams() will return { id: "7" }. This tells the component: "Hey, we're editing article #7."

Explain what happens inside the **useEffect** hook. When does it run, and what is its purpose?

The **useEffect** hook runs once when the component mounts because its dependency array is empty ([]), meaning it won't run again unless the component is re-mounted.

Inside the **useEffect**, it checks:

```
if (isEdit && id) {  
  fetchArticle(id);  
}
```

This means:

If the form is in edit mode (isEdit === true) and there's an id in the URL,

Then it calls the fetchArticle(id) function, which:

Sets the loading state,

Fetches the article's data from the backend (getArticleById(id)),

Sets the form fields (formData) with the loaded article data.

Purpose: To pre-fill the form with existing data so the user can edit the article

### Q3 - Look at the ArticleList

How are the three promise states (loading, success, and error) handled in the fetchArticles function?

Before the request start `setIsLoading(true)` and `setError("")` are called, then the UI start to show the spinner loading and show with no error, Then it's start to check the `try & catch` operation.

If the request Success it's start fetching the data  
`const data = await getArticles()` and `setArticles(data)`  
the loading is cleared.

```
    } finally {  
      setIsLoading(false);  
    }  
  }
```

But If the request Failed it's show the error messages

```
    setError("Failed to load articles. Please try again.");  
And the loading is also cleared.
```

What is the role of the ArticleCard component, how does it communicate with the parent ArticleList?

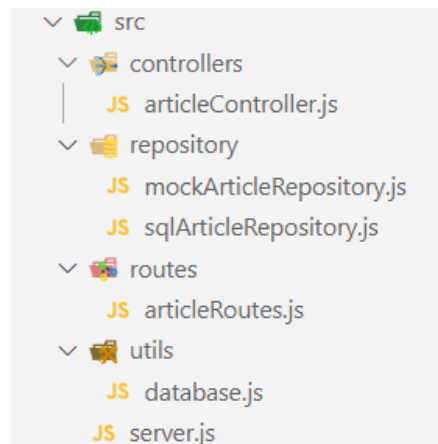
The role of ArticleCard is to display each article's title and author, and action.

and it's communicate with the parent ArticleList component to perform actions like viewing, editing, and deleting articles. The `ArticleList` component manages the state of the article list, handles fetching articles from the API, and provides the necessary functions to interact with each article.

### BACK-END

#### Q4 - Why 3 layers ?

The backend is composed of the below 3 layers : routes, controllers and repository :



Describe the **responsibility** of each **layer** by completing the table below:

LAYER	RESPONSABILITIES
Routes	Handle the specific endpoint route for each request operation that application can response to. It represents a specific URL path and HTTP method (GET, POST, PUT, DELETE)
Controller	Responsible for managing the flow of data between the model(BUSINESS logic and DATABASE) and the view (Client interact with) after route receives a request from a client because it's needs to handle that request and send a response back to the client.
Repository	It's working like a bridge between database and the logic required to access the database. They act as an abstraction layer between the services and the DAL (data access layer). By adopting this separation, we achieve more maintainability and scalability, making it easier to modify the database or switch to a different database.

#### Q5 - Implement the database connection

Here are the files you need to update to **connect the backend to the database**:

FILE	RESPONSABILITIES
/.env	securely store your MySQL database credentials
/utils/ database.js	Holds the <b>MySQL connection setup</b> logic  <i>Responsible for creating and exporting a <b>connection pool</b> that other parts of the application can use.</i>
/repository/sqlArticleRepository.js	Provides a <b>clean, reusable interface</b> to interact with the articles table in your <b>MySQL database</b> .  <i>Encapsulates all the SQL queries related to articles and exposes them as functions that the rest of your application can call.</i>

Here is what you need to do:

- **.env file**

Create a .env file to securely store your MySQL database credentials.

See <https://www.npmjs.com/package/dotenv>

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=complete this line
DB_NAME=complete this line
PORT=4000
```

- **utils/database.js**

- Create a **MySQL connection pool** using the credentials from the .env file.  
<https://sidorares.github.io/node-mysql2/docs#using-connection-pools>
- Export this connection pool so it can be used by other modules in the project.

- **repositories/sqlArticleRepository.js**

Implement the following functions to interact with the articles table in the database:

```
getAll() - fetch all articles
getById(id) - fetch one article by ID
create(article) - insert a new article
update(id, article) - update an existing article
remove(id) - delete an article by ID
```

Use the connection pool from database.js to **execute the SQL queries** inside these functions.

*As an example, to implement getAll() :*

```
export async function getArticles() {
  const [rows] = await pool.query("SELECT * FROM articles");
  return rows;
}
```

## Q6 - Test the endpoints

To test the implementation of MySQL repository (*create, update, remove, get articles...*)

- First, perform tests using a **REST API client** (thunder or postman)
- Then, run the **front-end project** and asset the views work properly



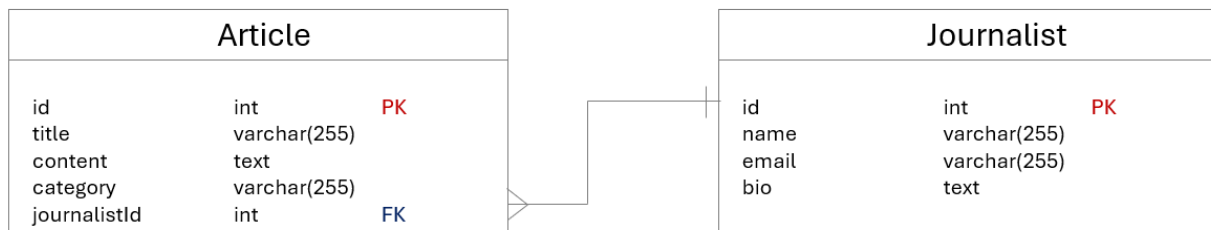
## EXERCISE 3 – Handle Journalists

For this exercise, you continue on the previous exercise code.

Now, users want to see **who wrote each article** to better understand the source.

- You will need to update the app, so the article page shows the **journalist's name and info**.
- You will need to provide a journalist view, showing all articles written by a specific journalist.

### Database



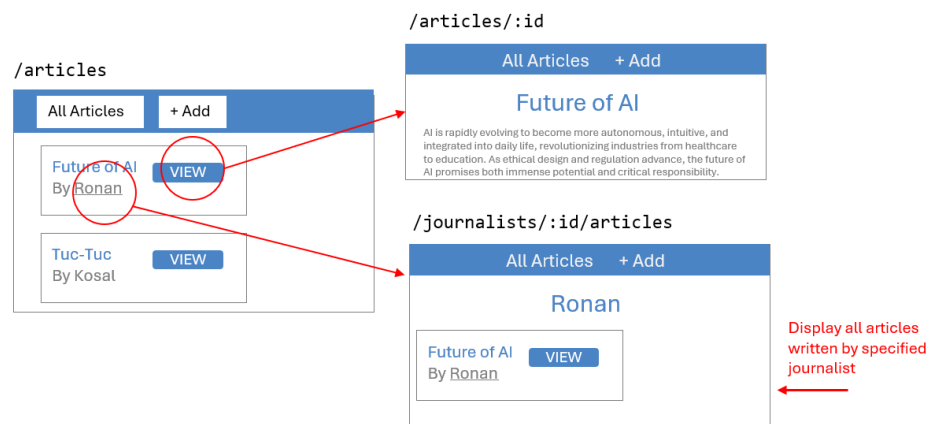
Update your database structure to handle the journalist table

- Create **journalists** table with fields: id, name, email, bio
- Update **articles** table to include **journalist\_id** foreign key
- **Populate the database** with some fake data

### Back End

- Implement **repository** methods:
  - Fetch articles with joined journalist name (using **SQL JOIN**)
  - Fetch all articles written by a specific journalist name (using **SQL JOIN**)
- Add **controller** functions:
  - Get all articles by journalist ID
- Define **new API routes**:
  - GET `/api/articles/:id` article + journalist name.
  - GET `/api/journalists/:id/articles` articles list by journalist.

### Front End



An additional view shall display all articles written by the specific journalist

- Update **Article Details page**:
  - Display journalist name alongside the article.
- Create **Journalist Articles List page**:
  - Display all articles by selected journalist.
- Add navigation:
  - From Article Details page, allow users to click journalist name to view **that journalist's articles**.
- Update API calls:
  - Fetch combined article + journalist data.
  - Fetch articles filtered by journalist ID.

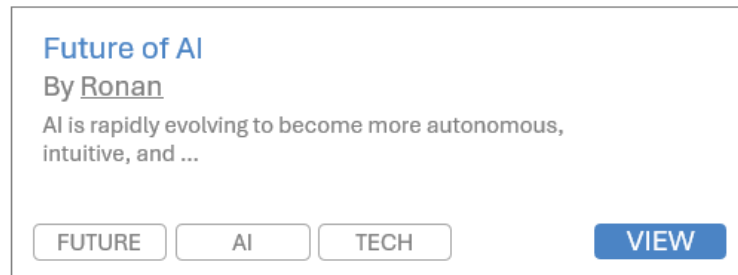
## EXERCISE 4 – Handle Tags

BONUS

For this exercise, you continue on the previous exercise code.

Now, users want to easily **assign tags to articles**.

The users can then filter articles by selecting different tags.



You will need to add categories to articles and let users filter the article list by selecting a category.

Database

- Create a new **table** category (id, name).
  - *What kind of relationships do we have between articles and categories?*
- **Relationship:**
- One article can have many tags

Back End:

- Implement repository methods to:
  - Retrieve all categories.
  - Retrieve all articles filtered by category, using JOIN to include category name.
- Add a new API endpoint to get articles by category ID.

Front End:

- Add a **multiple categories filter UI component** on the **article list page** (multiple choice dropdown, chipset selector).
- When categories are selected, fetch and display only articles in those categories.
- Display categories names alongside articles in the list.