

## Task 1.2: Database Setup

Create Database and Table:

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, tabs for 'SQL File 5\*', 'SQL File 6\*', 'SQL File 7\*', 'SQL File 8\*', 'SQL File 9\*', 'lab2\*', and 'SQL File 11\*' are visible. The main pane displays the following SQL script:

```
1 • CREATE DATABASE product_management;
2 • USE product_management;
3
4 • CREATE TABLE products (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    product_code VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    quantity INT DEFAULT 0,
    category VARCHAR(50),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
5
6 -- Insert sample data
7 • INSERT INTO products (product_code, name, price, quantity, category, description) VALUES
8     ('P001', 'Laptop Dell XPS 13', 1299.99, 10, 'Electronics', 'High-performance laptop with Intel i7'),
9     ('P002', 'iPhone 15 Pro', 999.99, 25, 'Electronics', 'Latest iPhone with A17 Pro chip'),
10    ('P003', 'Samsung Galaxy S24', 899.99, 20, 'Electronics', 'Flagship Android smartphone'),
11    ('P004', 'Office Chair Ergonomic', 199.99, 50, 'Furniture', 'Comfortable office chair with lumbar support'),
12    ('P005', 'Standing Desk', 399.99, 15, 'Furniture', 'Adjustable height standing desk')
13
14
15
16
17
18
19
20
21
22
```

The right pane shows a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the message, there are 'Context Help' and 'Snippets' buttons.

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	14:25:01	CREATE DATABASE product_management	1 row(s) affected	0.016 sec
2	14:25:01	USE product_management	0 row(s) affected	0.000 sec
3	14:25:01	CREATE TABLE products ( id BIGINT PRIMARY KEY AUTO_INCREMENT, product_code VARCHAR(20) ...	0 row(s) affected	0.047 sec
4	14:25:01	INSERT INTO products (product_code, name, price, quantity, category, description) VALUES (P001, 'Laptop Del...)	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec

## Task 2.3: Test Repository

The screenshot shows a terminal window with the following command and output:

```
PS C:\Users\84836\Desktop\WebApp\Lab7\product-management> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '@C:\Users\84836\AppData\Local\Temp\cp_2dbngf8oyc2vc998xzy8pg8hu.argfile' 'com.example.product_management.ProductManagementApplication'
```

The output shows the execution of a Java application that queries a database:

```
p1_0.id,  
p1_0.category,  
p1_0.created_at,  
p1_0.description,  
p1_0.name,  
p1_0.price,  
p1_0.product_code,  
p1_0.quantity  
from  
products p1_0  
where  
p1_0.category=?  
  
Electronics: 3  
== Test Complete ==
```

The bottom of the terminal shows the system tray icons and the status bar with 'Screen Reader Optimized', 'Ln 9, Col 23', and 'Tab Size:'.

List all products:

<http://localhost:8080/products>

The screenshot shows a web-based Product Management System. At the top center is the title "Product Management System" with a gear icon. Below it is a search bar with placeholder text "Search products..." and a blue "Search" button. On the left, there is a blue button labeled "Add New Product". The main area displays a table with the following data:

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	\$1299.99	10	Electronics	<button>Edit</button> <button>Delete</button>
2	P002	iPhone 15 Pro	\$999.99	25	Electronics	<button>Edit</button> <button>Delete</button>
3	P003	Samsung Galaxy S24	\$899.99	20	Electronics	<button>Edit</button> <button>Delete</button>
4	P004	Office Chair Ergonomic	\$199.99	50	Furniture	<button>Edit</button> <button>Delete</button>
5	P005	Standing Desk	\$399.99	15	Furniture	<button>Edit</button> <button>Delete</button>

Add new product: <http://localhost:8080/products/new>

The screenshot shows a browser window with the URL "localhost:8080/products/new". The page title is "Add New Product" with a plus sign icon. The form contains the following fields:

- Product Code \***: A text input field with placeholder text "Enter product code (e.g., P001)".
- Product Name \***: A text input field with placeholder text "Enter product name".
- Price (\$)**: A text input field with placeholder text "0.00".
- Quantity**: A text input field with placeholder text "0".
- Category \***: A dropdown menu with placeholder text "Select category".
- Description**: A text area with placeholder text "Enter product description (optional)".

Edit product (ID=1): <http://localhost:8080/products/edit/1>

The screenshot shows a web browser window with the URL [localhost:8080/products/edit/1](http://localhost:8080/products/edit/1). The page title is "Edit Product". The form contains the following fields:

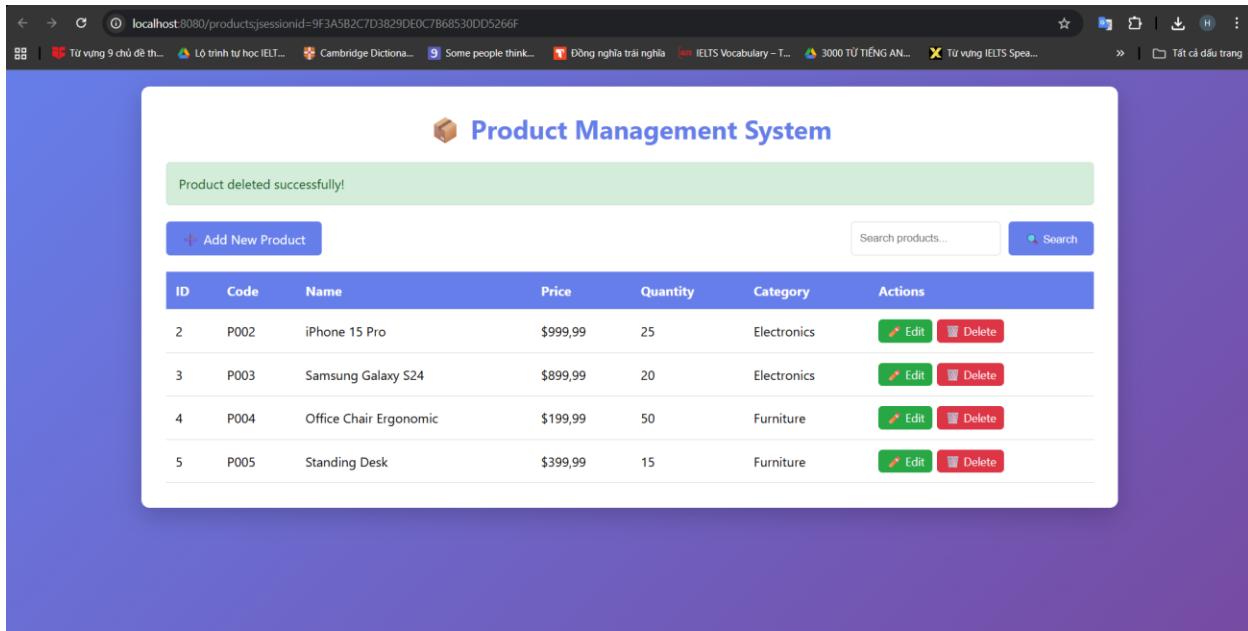
- Product Code \*: P001
- Product Name \*: Laptop Dell XPS 13
- Price (\$) \*: 1299,99
- Quantity \*: 10
- Category \*: Electronics
- Description: High-performance laptop with Intel i7

Search products: <http://localhost:8080/products/search?keyword=laptop>

The screenshot shows a web browser window with the URL [localhost:8080/products/search?keyword=laptop](http://localhost:8080/products/search?keyword=laptop). The page title is "Product Management System". The search bar contains "laptop" and the search button is labeled "Search". The table displays the following data:

ID	Code	Name	Price	Quantity	Category	Actions
1	P001	Laptop Dell XPS 13	\$1299,99	10	Electronics	<a href="#">Edit</a> <a href="#">Delete</a>

Delete product (ID=1): <http://localhost:8080/products/delete/1>



The screenshot shows a web browser window for 'localhost:8080/products;jsessionid=9F3A5B2C7D3829DE0C7B68530DD5266F'. The title bar says 'Product Management System'. A green success message at the top left reads 'Product deleted successfully!'. Below it is a table with columns: ID, Code, Name, Price, Quantity, Category, and Actions. The table contains five rows of data. The 'Actions' column for each row has two buttons: 'Edit' (green) and 'Delete' (red). The data in the table is:

ID	Code	Name	Price	Quantity	Category	Actions
2	P002	iPhone 15 Pro	\$999,99	25	Electronics	<button>Edit</button> <button>Delete</button>
3	P003	Samsung Galaxy S24	\$899,99	20	Electronics	<button>Edit</button> <button>Delete</button>
4	P004	Office Chair Ergonomic	\$199,99	50	Furniture	<button>Edit</button> <button>Delete</button>
5	P005	Standing Desk	\$399,99	15	Furniture	<button>Edit</button> <button>Delete</button>

Explain CRUD flow:

### 1. CREATE flow (Adding a product)

This flow involves two steps: showing the empty form, then processing the submission.

#### Step 1: Showing the form

1. User action: User clicks the "Add New Product" button in product-list.html, which links to @{/products/new}.
2. Controller: The showNewForm method creates a new, empty Product object and binds it to the model, then returns product-form.
3. View: product-form.html renders. The title displays "Add New Product" because the product ID is null.

#### Step 2: Saving the Data

1. User action: The user fills out the form and clicks "Save Product". The form submits a POST request to @{/products/save}.
2. Controller: The saveProduct method receives the form data mapped to a Product object.
3. Service: Calls productService.saveProduct(product).
  - o Inside ProductServiceImpl, productRepository.save(product) is called.

4. Database: Because the Product ID is null, Hibernate performs an INSERT statement.
  - o The @PrePersist method in Product.java automatically sets the createdAt timestamp.

5. Result: The controller redirects the user back to /products with a success message.

## 2. READ flow (Listing products)

The default view when the application starts or the user navigates to the main page.

1. User action: The user accesses /products.
2. Controller: The ProductController.java receives the GET request at the listProducts method.
3. Service: The controller calls productService.getAllProducts().
4. The ProductServiceImpl calls productRepository.findAll().
5. Repository: ProductRepository executes a SQL SELECT \* query via JPA.
6. View: The controller adds the list of products to the Model and returns the product-list view. The product-list.html template uses th:each="product : \${products}" to iterate through the list and generate an HTML table row for every product found.

## 3. UPDATE Flow (Editing a Product)

This flow reuses the Create logic but includes the existing ID.

### Step 1: Pre-filling the Form

1. **User Action:** In the product list, the user clicks the "Edit" button next to a specific product. The link is @{/products/edit/{id}}.
2. **Controller:** The showEditForm method extracts the ID from the URL.
3. **Service:** It calls productService.getProductById(id) to find the existing data.
4. **View:** The controller passes this specific Product object to product-form.html.
  - o Thymeleaf pre-fills the input fields (Name, Price, etc.) using th:field="\*{name}".
  - o Crucially, there is a hidden input field: <input type="hidden" th:field="\*{id}" />. This ensures the ID is sent back during submission.

### Step 2: Saving Changes

1. **User action:** User modifies data and clicks save. The form POSTs to /products/save.

2. **Logic:** The flow is identical to **Create Step 2**, but because the Product object now has an existing ID (from the hidden field), the Repository performs an UPDATE query instead of an INSERT.

#### 4. DELETE Flow

1. **User action:** User clicks the "Delete" button in product-list.html.
  - o A JavaScript confirmation dialog (`onclick="return confirm..."`) asks for verification.
  - o If confirmed, the browser requests `@{/products/delete/{id}}`.
2. **Controller:** The deleteProduct method captures the ID.
3. **Service:** Calls `productService.deleteProduct(id)`.
  - o `ProductServiceImpl` calls `productRepository.deleteById(id)`.
4. **Result:** The product is removed from the database, and the controller redirects the user to `/products` with a flash message "Product deleted successfully!".