

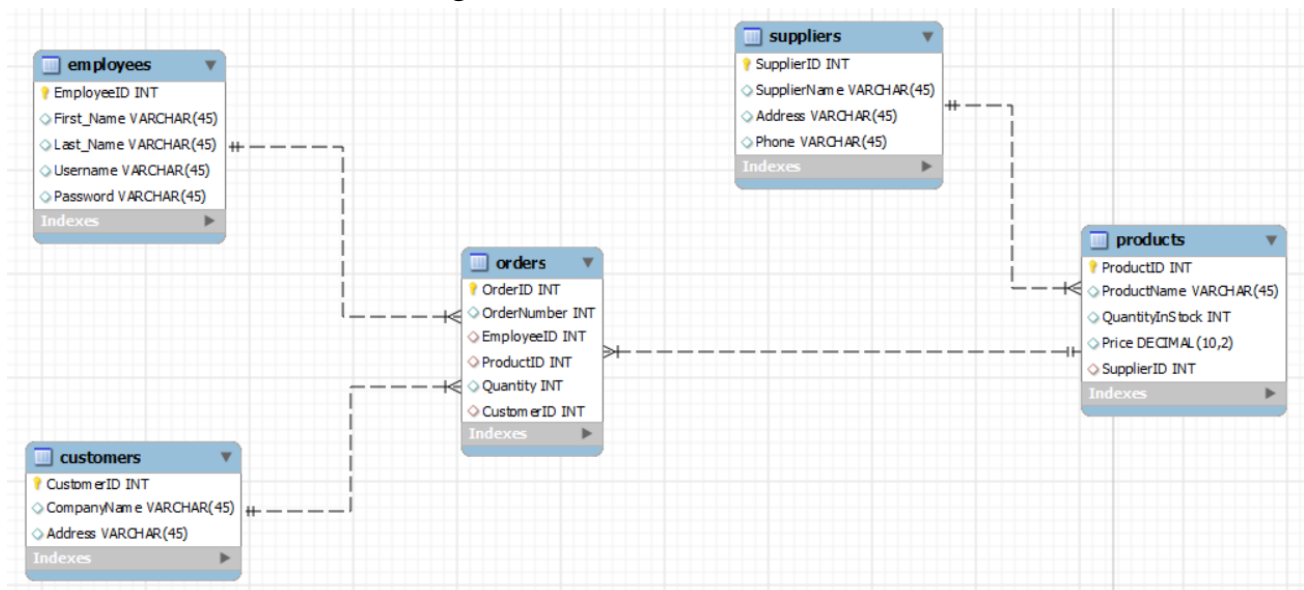
Inventory Management System

Database – Mini Project

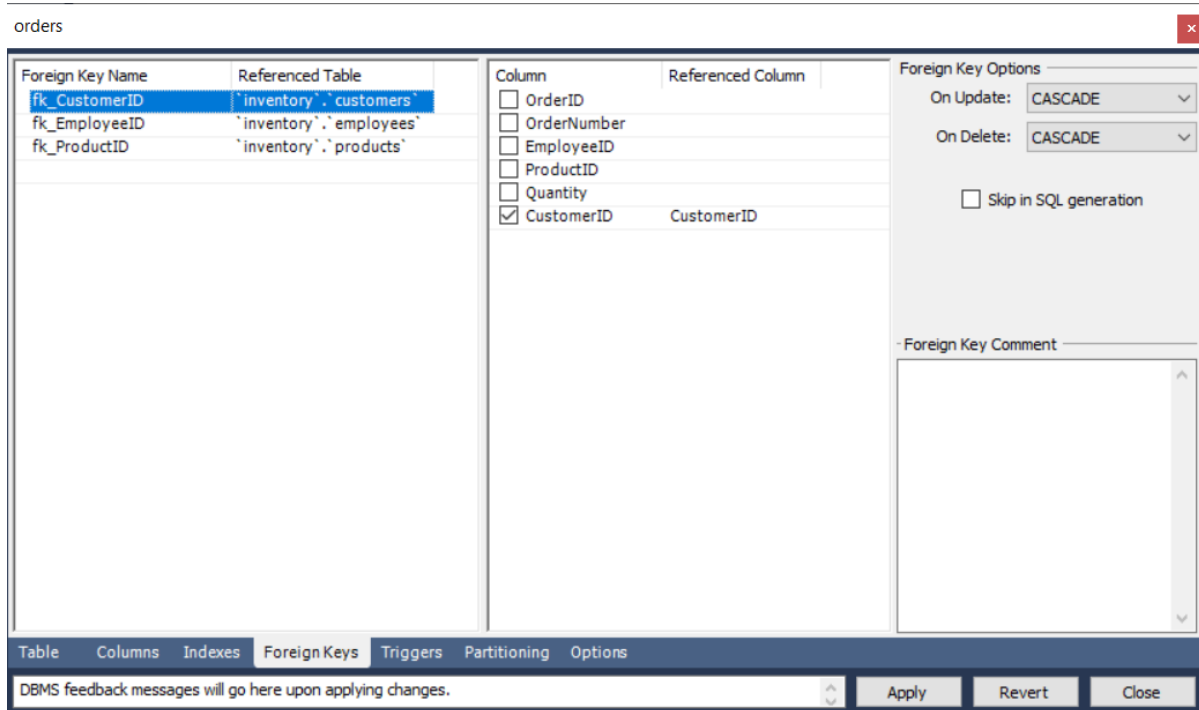
The inventory control management database is designed for employees to keep track of product inventory, orders, suppliers, and customers. It includes several tables that are related to each other, allowing users to query data and generate reports based on various criteria.

There are six tables in the data base: *Products* that has *ProductID* as a primary key, *ProductName*, *QuantityInStock*, *Price* and *SupplierID* as a foreign key with *Suppliers* as the referenced table, *Orders* that has *OrderID* as a primary key, *OrderNumber*, *EmployeeID* as a foreign key with *Employees* as the referenced table, *ProductID* as a foreign key with *Products* as the referenced table, *Quantity* and *CustomerID* as a foreign key with *Customers* as the referenced table, *Suppliers* that has *SupplierID* as a primary key, *SupplierName*, *Address* and *Phone*, *Employees* that has *EmployeeID* as a primary key, *First_Name*, *Last_Name*, *Username* and *Password* and *Customers* that has *CustomerID* as a primary key, *CompanyName* and *Address*. The names of the tables are intuitive and describe the information contained in them.

This is the database diagram:

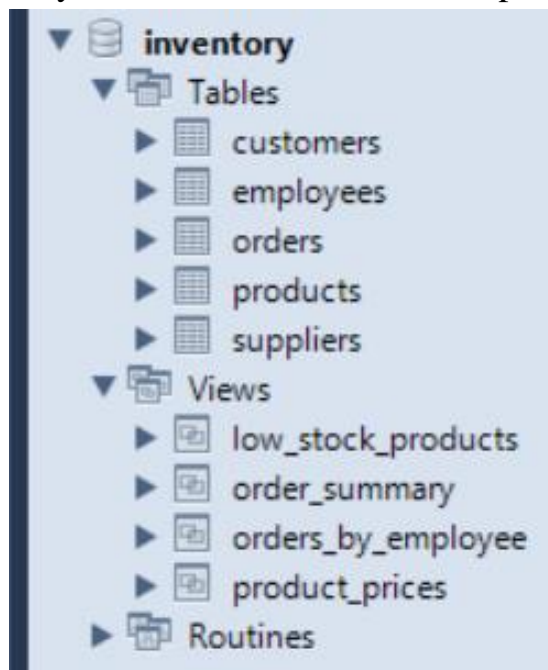


Constraints were established between the tables, for each foreign key in a table the options for update and delete are **CASCADE**. This can be useful in certain scenarios to maintain referential integrity and automate cascading changes.



There are four views: *low_stock_products* that selects all products that have a quantity in stock less than 50, it can be useful for quickly identifying products that need restocking, *order_summary* that provides a summary of each order, including the total number of products and the total quantity, it uses the **GROUP BY** clause to group the orders by **OrderNumber** and it can be useful for obtaining an overview of the orders and their aggregated data, *orders_by_employee* that displays all orders, sorted by the **EmployeeID** and **OrderNumber** and it can be useful for analyzing orders grouped by employees and *product_prices* that selects the **ProductID**, **ProductName**, and **Price** columns from the **products** table, excluding the **QuantityInStock** and **SupplierID** and it can be helpful when you

only need information related to product prices.



This is the query where I created the views:

```
inventory_query.sql  views_query.sql X
1 • USE Inventory;
2 • CREATE VIEW order_summary AS SELECT OrderID, COUNT(ProductID) AS TotalProducts, SUM(Quantity) AS TotalQuantity
3 FROM orders GROUP BY OrderNumber;
4
5 • CREATE VIEW orders_by_employee AS SELECT *
6 FROM orders ORDER BY EmployeeID, OrderNumber;
7
8 • CREATE VIEW product_prices AS SELECT ProductID, ProductName, Price
9 FROM products;
10
11 • CREATE VIEW low_stock_products AS SELECT *
12 FROM products WHERE QuantityInStock < 50;
```

Here are the tables with data:

	ProductID	ProductName	QuantityInStock	Price	SupplierID
▶	1	APPLE iPhone 14 PRO	27	5700.00	1
	2	SAMSUNG Galaxy A52	70	650.00	2
	3	LENOVO IdeaPad Slim 3	122	2999.00	6
	4	SAMSUNG Galaxy Tab S8	99	3450.00	2
	5	APPLE Macbook Air13	88	6600.00	1
	6	ASUS Vivobook PRO15	30	3300.00	3
	9	Philips Prestige	46	350.00	5
	10	HP LaserJet	112	555.00	4
	11	Philips Lumea	68	3200.00	5
•	NULL	NULL	NULL	NULL	NULL

	SupplierID	SupplierName	Address	Phone
▶	1	APPLE	Cupertino, California, US	800-692-7753
	2	SAMSUNG	Suwon-Si, South Korea	800-726-7864
	3	ASUS	Fremont, California, US	510-739-3777
	4	HP	Palo Alto, California, US	800-407-4005
	5	Philips	Cambridge, Massachusetts, US	888-427-9279
	6	Lenovo	Morrisville, North Carolina, US	855-253-6686
•	NULL	NULL	NULL	NULL

	OrderID	OrderNumber	EmployeeID	ProductID	Quantity	CustomerID
▶	1	1	1	1	10	1
	2	1	1	2	15	1
	3	2	1	3	10	2
	4	2	1	4	45	2
	5	2	1	5	50	2
•	NULL	NULL	NULL	NULL	NULL	NULL

	EmployeeID	First_Name	Last_Name	Username	Password
▶	1	John	Doe	user1	pass1
	2	Jane	Doe	user2	pass2
•	NULL	NULL	NULL	NULL	NULL

This is the query where I tested **JOIN operation, NULL, SELECT** and aggregate funtions and also, the SQL statements: **INSERT, UPDATE AND DELETE**:

```

1  USE inventory;
2  SELECT ProductID, ProductName, QuantityInStock, Price, SupplierID FROM products ORDER BY QuantityInStock DESC;
3
4  SELECT customers.CustomerID, customers.CompanyName, orders.OrderNumber FROM customers
5  JOIN orders ON customers.CustomerID = orders.CustomerID GROUP BY customers.CustomerID;
6
7  SELECT customers.CustomerID, customers.CompanyName, orders.OrderNumber FROM customers
8  LEFT JOIN orders ON customers.CustomerID = orders.CustomerID WHERE orders.CustomerID IS NULL;
9
10 SELECT orders.OrderNumber, products.ProductID, products.ProductName, products.Price, orders.Quantity, orders.CustomerID
11 FROM orders INNER JOIN products ON orders.ProductID = products.ProductID WHERE orders.OrderNumber = '1';
12
13 INSERT INTO products (ProductName, QuantityInStock, Price, SupplierID)
14 VALUES ('HP Z3700', '160', '110', '3');
15
16 UPDATE products SET ProductName = 'HP Z3700', QuantityInStock = '160', Price = '110', SupplierID = '4' WHERE ProductID = '12';
17
18 DELETE FROM products WHERE ProductID = '12';
  
```

And here are the results in the same order as the statements are made:

	ProductID	ProductName	QuantityInStock	Price	SupplierID
▶	3	LENOVO IdeaPad Slim 3	122	2999.00	6
	10	HP LaserJet	112	555.00	4
	4	SAMSUNG Galaxy Tab S8	99	3450.00	2
	5	APPLE Macbook Air13	88	6600.00	1
	2	SAMSUNG Galaxy A52	70	650.00	2
	11	Philips Lumea	68	3200.00	5
	9	Philips Prestige	46	350.00	5
	6	ASUS Vivobook PRO15	30	3300.00	3
	1	APPLE iPhone 14 PRO	27	5700.00	1

	CustomerID	CompanyName	OrderNumber
▶	1	Altex	1
	2	Flanco	2

	OrderNumber	ProductID	ProductName	Price	Quantity	CustomerID
▶	1	1	APPLE iPhone 14 PRO	5700.00	10	1
	1	2	SAMSUNG Galaxy A52	650.00	15	1

The *Products* table after the **INSERT** statement:

	ProductID	ProductName	QuantityInStock	Price	SupplierID
	1	APPLE iPhone 14 PRO	27	5700.00	1
	2	SAMSUNG Galaxy A52	70	650.00	2
	3	LENOVO IdeaPad Slim 3	122	2999.00	6
	4	SAMSUNG Galaxy Tab S8	99	3450.00	2
	5	APPLE Macbook Air13	88	6600.00	1
	6	ASUS Vivobook PRO15	30	3300.00	3
	9	Philips Prestige	46	350.00	5
	10	HP LaserJet	112	555.00	4
	11	Philips Lumea	68	3200.00	5
✎	12	HP Z3700	160	110.00	3
•	NULL	NULL	NULL	NULL	NULL

The *Products* table after the **UPDATE** statement:

	ProductID	ProductName	QuantityInStock	Price	SupplierID
▶	1	APPLE iPhone 14 PRO	27	5700.00	1
	2	SAMSUNG Galaxy A52	70	650.00	2
	3	LENOVO IdeaPad Slim 3	122	2999.00	6
	4	SAMSUNG Galaxy Tab S8	99	3450.00	2
	5	APPLE Macbook Air13	88	6600.00	1
	6	ASUS Vivobook PRO15	30	3300.00	3
	9	Philips Prestige	46	350.00	5
	10	HP LaserJet	112	555.00	4
	11	Philips Lumea	68	3200.00	5
	12	HP Z3700	160	110.00	4
•	NULL	NULL	NULL	NULL	NULL

The *Products* table after the **DELETE** statement:

	ProductID	ProductName	QuantityInStock	Price	SupplierID
	1	APPLE iPhone 14 PRO	27	5700.00	1
	2	SAMSUNG Galaxy A52	70	650.00	2
	3	LENOVO IdeaPad Slim 3	122	2999.00	6
	4	SAMSUNG Galaxy Tab S8	99	3450.00	2
	5	APPLE Macbook Air13	88	6600.00	1
	6	ASUS Vivobook PRO15	30	3300.00	3
	9	Philips Prestige	46	350.00	5
	10	HP LaserJet	112	555.00	4
▶	11	Philips Lumea	68	3200.00	5
•	NULL	NULL	NULL	NULL	NULL

Regarding the website:

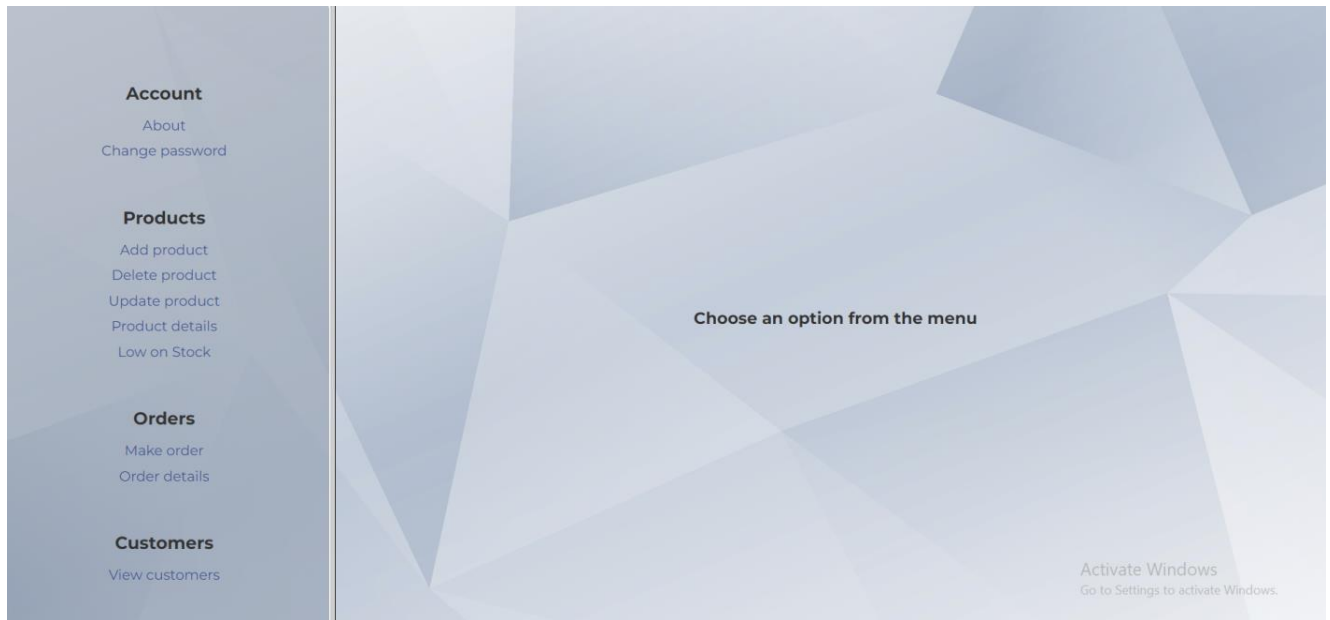
index.php

- This file serves as the login page for the Inventory Management System.
- It includes a login form that allows users to enter their username and password.
- When the form is submitted, the PHP code validates the credentials against a database.
- If the credentials are valid, the user is redirected to the **home.html** page.
- If the credentials are invalid, an error message is displayed.



home.html

- This file represents the main page of the Inventory Management System.
- It utilizes framesets to divide the page into two frames: a sidebar frame and a main content frame.
- The sidebar frame is sourced from **sidebar.php** and contains navigation links.
- The main content frame is sourced from **main.php** and initially displays a default message.



sidebar.php

- This file defines the HTML structure for the sidebar displayed in **home.html**.
- It includes headings for different sections, such as "Account," "Products," "Orders," and "Customers".
- Each section contains a list of links that serve as navigation items.
- Clicking a link loads the corresponding content in the main content frame.

main.php

- This file represents the main content area displayed in **home.html**.
- It includes a container element with a heading that suggests choosing an option from the menu.
- Initially, this page serves as a placeholder for the actual content loaded from the sidebar navigation.

about.php

- The **about.php** file is responsible for displaying employee information.
- It starts by checking if the user is logged in. If not, it redirects them to the login page (**index.php**).
- The file includes the **connect.php** file to establish a database connection.

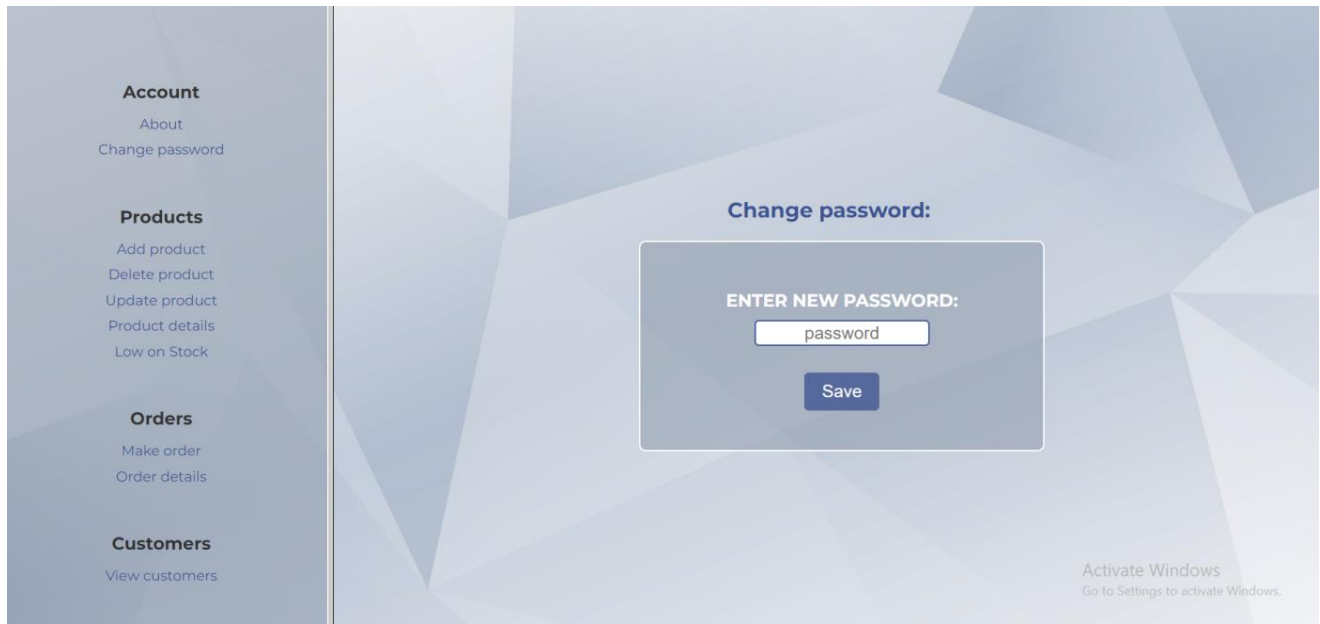
- It retrieves the logged-in username from the session and executes a SQL query to fetch the corresponding employee data.
- If a matching employee record is found, the data is stored in the **\$employee** variable.
- If no matching record is found, an error message is stored in the **\$error** variable.
- After closing the database connection, the HTML section displays the employee details if the **\$employee** variable is set, or displays the error message if it's not.



password.php

- The **password.php** file allows users to change their password.
- It also starts by checking if the user is logged in. If not, it redirects them to the login page (**index.php**).
- The file includes the **connect.php** file to establish a database connection.
- If the form is submitted (via POST method), it retrieves the new password from the form and updates the password in the database for the logged-in user.
- A SQL query is executed to update the password in the database.
- If the query is successful, a success message is displayed. Otherwise, an error message is displayed.

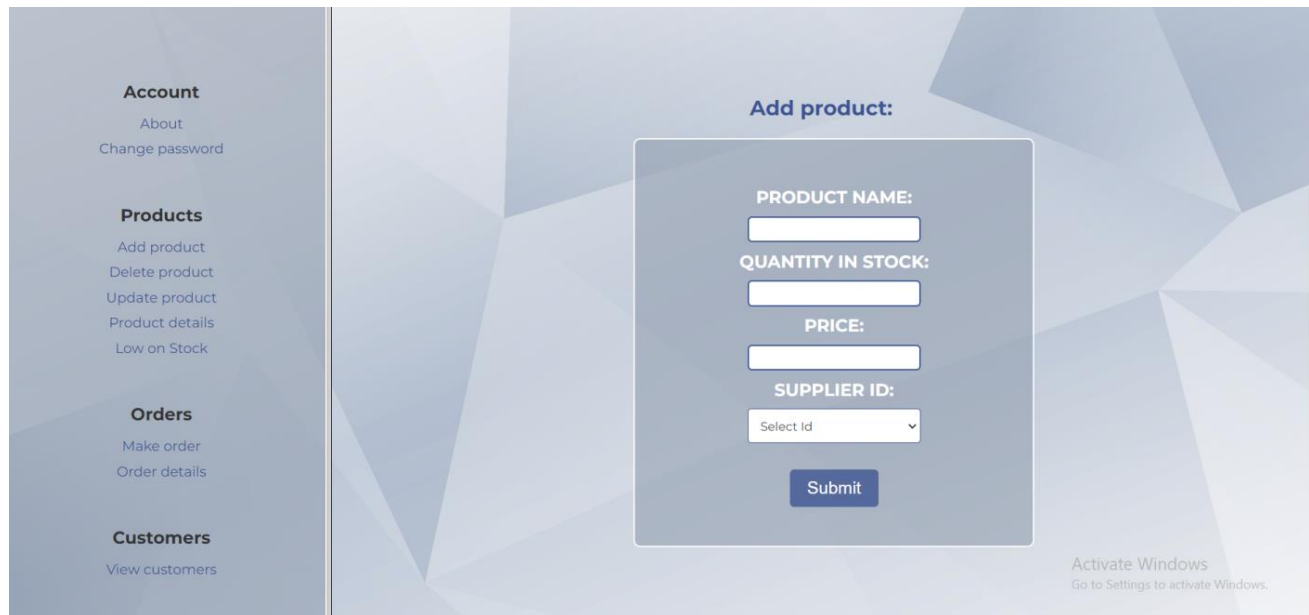
- After closing the database connection, the HTML section displays a form where users can enter a new password.
- When the form is submitted, the new password is sent to the same file (**password.php**) for processing.



add_product.php

- The **add_product.php** file allows users to add a new product to the inventory.
- It starts by including the **connect.php** file to establish a database connection.
- The **session_start()** function is called to start the session.
- If the request method is POST, indicating that the form has been submitted, the code retrieves the product details from the form fields.
- A SQL query is then executed to insert the product into the **products** table in the database.
- If the query is successful, a success message is stored in the session variable **\$_SESSION['message']**, and the page is redirected back to itself using the **header()** function.
- If the query fails, an error message is stored in the session variable **\$_SESSION['message']**.
- The code then displays a form where users can enter the product details, including product name, quantity in stock, price, and supplier ID.

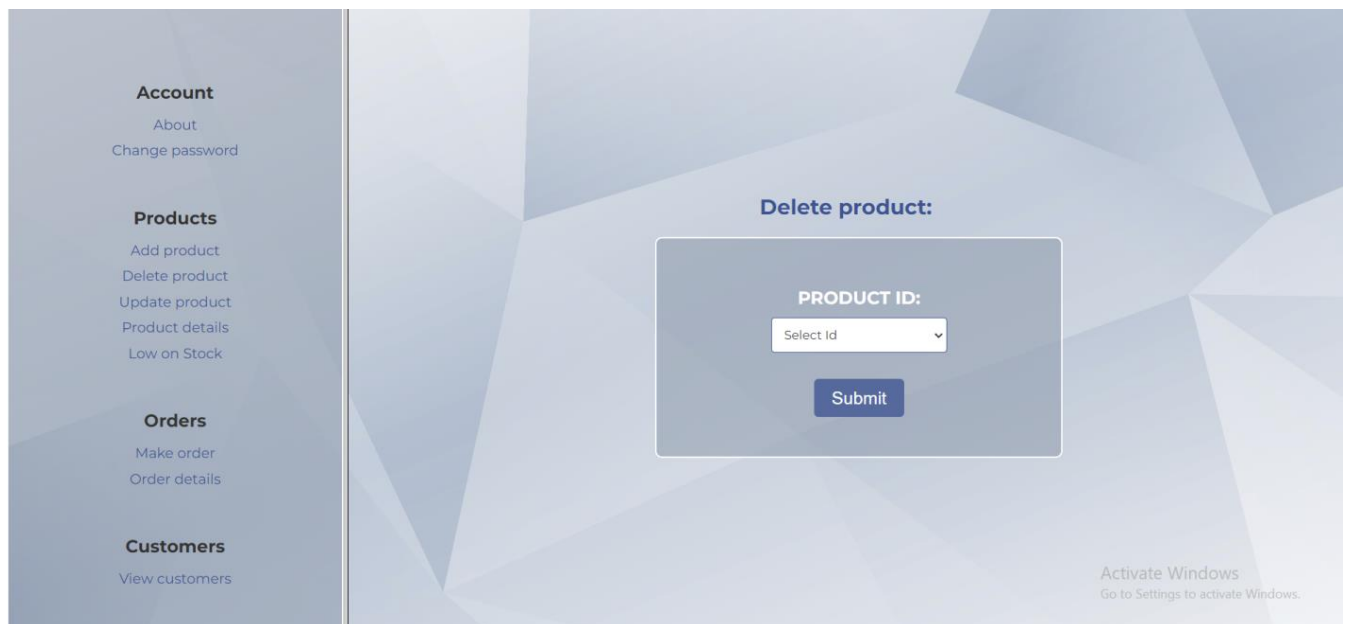
- The supplier ID is fetched from the database using a SQL query that retrieves the available supplier IDs from the **suppliers** table.
- The supplier IDs are dynamically populated in a select dropdown field within the form.
- When the form is submitted, the product details are sent to the same file (**add_product.php**) for processing.



delete_product.php

- The **delete_product.php** file allows users to delete a product from the inventory.
- It starts by including the **connect.php** file to establish a database connection.
- The **session_start()** function is called to start the session.
- If the request method is POST, indicating that the form has been submitted, the code retrieves the product ID from the form field.
- A SQL query is then executed to delete the product with the specified ID from the **products** table in the database.
- If the query is successful, a success message is stored in the session variable **\$_SESSION['message']**, and the page is redirected back to itself using the **header()** function.
- If the query fails, an error message is stored in the session variable **\$_SESSION['message']**.

- The code then displays a form where users can select the product ID to delete from a select dropdown field.
- The product IDs are fetched from the database using a SQL query that retrieves the available product IDs from the **products** table.
- The product IDs are dynamically populated in the select dropdown field within the form.
- When the form is submitted, the selected product ID is sent to the same file (**delete_product.php**) for processing.



update_product.php

- The **update_product.php** file allows users to update the information of a product in the inventory.
- It starts by including the **connect.php** file to establish a database connection.
- The **session_start()** function is called to start the session.
- If the request method is POST, indicating that the form has been submitted, the code retrieves the product ID, product name, quantity, price, and supplier ID from the form fields.
- A SQL query is then executed to update the corresponding fields of the product with the specified ID in the **products** table in the database.

- If the query is successful, a success message is stored in the session variable **\$_SESSION['message']**, and the page is redirected back to itself using the **header()** function.
- If the query fails, an error message is stored in the session variable **\$_SESSION['message']**.
- The code then displays a form where users can select the product ID to update from a select dropdown field.
- The product IDs are fetched from the database using a SQL query that retrieves the available product IDs from the **products** table.
- The product IDs are dynamically populated in the select dropdown field within the form.
- Users can enter the new information for the product, including the product name, quantity, price, and supplier ID.

The screenshot displays a web application interface for updating a product. On the left, a sidebar contains navigation links categorized under 'Account' (About, Change password), 'Products' (Add product, Delete product, Update product, Product details, Low on Stock), 'Orders' (Make order, Order details), and 'Customers' (View customers). The main content area is titled 'Update product:' and features a form with the following elements: a dropdown menu labeled 'ID FOR THE PRODTUCT TO BE UPDATED:' with 'Select Id' as the placeholder; input fields for 'PRODUCT NAME:', 'QUANTITY IN STOCK:', 'PRICE:', and 'SUPPLIER ID:'; and a 'Submit' button. The 'SUPPLIER ID' field also has a dropdown menu with 'Select Id' as the placeholder. The background is a light blue geometric pattern. At the bottom right, there is a watermark for 'Activate Windows' with the text 'Go to Settings to activate Windows.'

view_products.php

- The **view_products.php** file retrieves and displays the products from the inventory in a tabular format, in descending order by quantity in stock.
- It starts by including the **connect.php** file, which establishes a connection to the database.
- The code then executes a SELECT query to retrieve the **ProductID**, **ProductName**, **QuantityInStock**, **Price**, and **SupplierID** of all products from the **products** table. The query results are stored in the **\$result** variable.

- If the query returns more than zero rows, the code fetches all the rows using the **fetch_all()** function and stores them in the **\$products** array. Each row represents a product and contains the aforementioned product details.
- If no products are found (resulting in zero rows), an error message is stored in the **\$error** variable.
- The retrieved products (if any) are then displayed in an HTML table. The table headers include "Product ID," "Product Name," "Quantity in Stock," "Price," and "Supplier ID."
- Inside the table, a PHP foreach loop is used to iterate over each product in the **\$products** array. For each product, a table row is generated, and the respective product details are displayed in the corresponding table cells.
- If there are no products to display, the **\$error** message is shown instead.
- The styling of the table and error message is handled through CSS classes defined in the **data.css** file, which is included in the HTML head section.
- The resulting HTML page presents the products in a table format, or the error message if there are no products found.

Account

About

Change password

Products

Add product

Delete product

Update product

Product details

Low on Stock

Orders

Make order

Order details

Customers

View customers

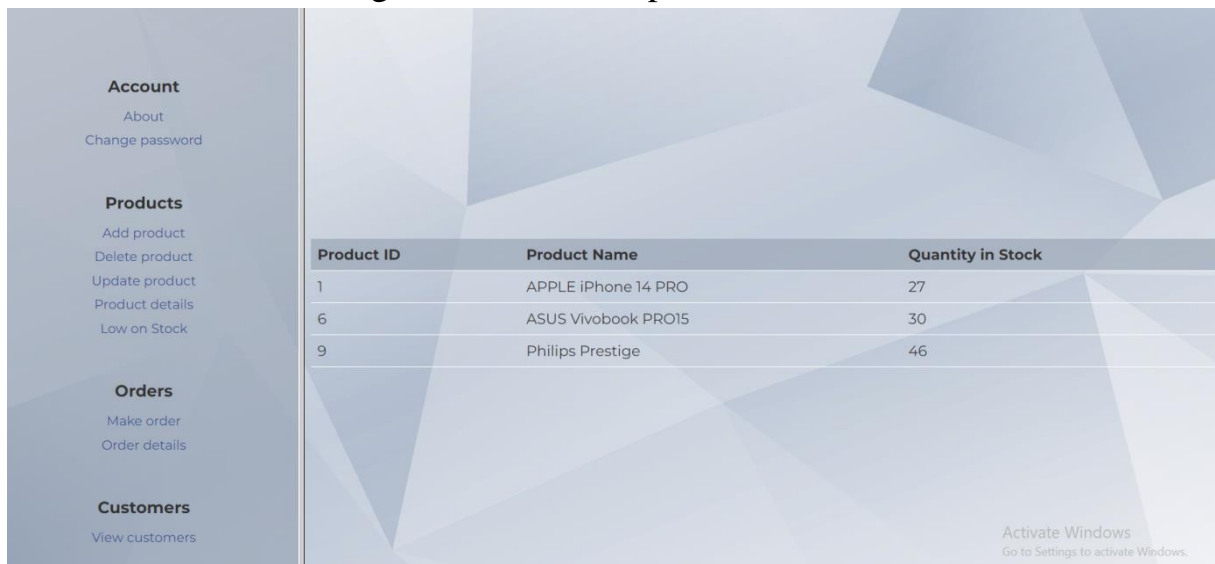
Product ID	Product Name	Quantity in Stock	Price	Supplier ID
3	LENOVO IdeaPad Slim 3	122	2999.00	6
10	HP LaserJet	112	555.00	4
4	SAMSUNG Galaxy Tab S8	99	3450.00	2
5	APPLE Macbook Air13	88	6600.00	1
2	SAMSUNG Galaxy A52	70	650.00	2
11	Philips Lumea	68	3200.00	5
9	Philips Prestige	46	350.00	5
6	ASUS Vivobook PRO15	30	3300.00	3
1	APPLE iPhone 14 PRO	27	5700.00	1

Activate Windows
Go to Settings to activate Windows.

low_stock.php

- The **low_stock.php** file retrieves and displays the products with low stock from the database (if the quantity for a product is lower than 50).
- It starts by including the **connect.php** file, which establishes a connection to the database.

- The code then executes a **SELECT** query to fetch all the columns from the **low_stock_products** table. The query results are stored in the **\$result** variable.
- If the query returns more than zero rows, the code fetches all the rows using the **fetch_all()** function and stores them in the **\$products** array. Each row represents a low stock product and contains the product details.
- If no low stock products are found (resulting in zero rows), an error message is stored in the **\$error** variable.
- The retrieved low stock products (if any) are then displayed in an **HTML** table. The table headers include "Product ID," "Product Name," and "Quantity in Stock."
- Inside the table, a **PHP** foreach loop is used to iterate over each low stock product in the **\$products** array. For each product, a table row is generated, and the respective product details are displayed in the corresponding table cells.
- If there are no low stock products to display, the **\$error** message is shown instead.
- The resulting **HTML** page presents the low stock products in a table format, or the error message if no low stock products are found.



The screenshot shows a web application interface. On the left is a sidebar menu with the following sections:

- Account**
 - About
 - Change password
- Products**
 - Add product
 - Delete product
 - Update product
 - Product details
 - Low on Stock
- Orders**
 - Make order
 - Order details
- Customers**
 - View customers

The main content area displays a table with the following data:

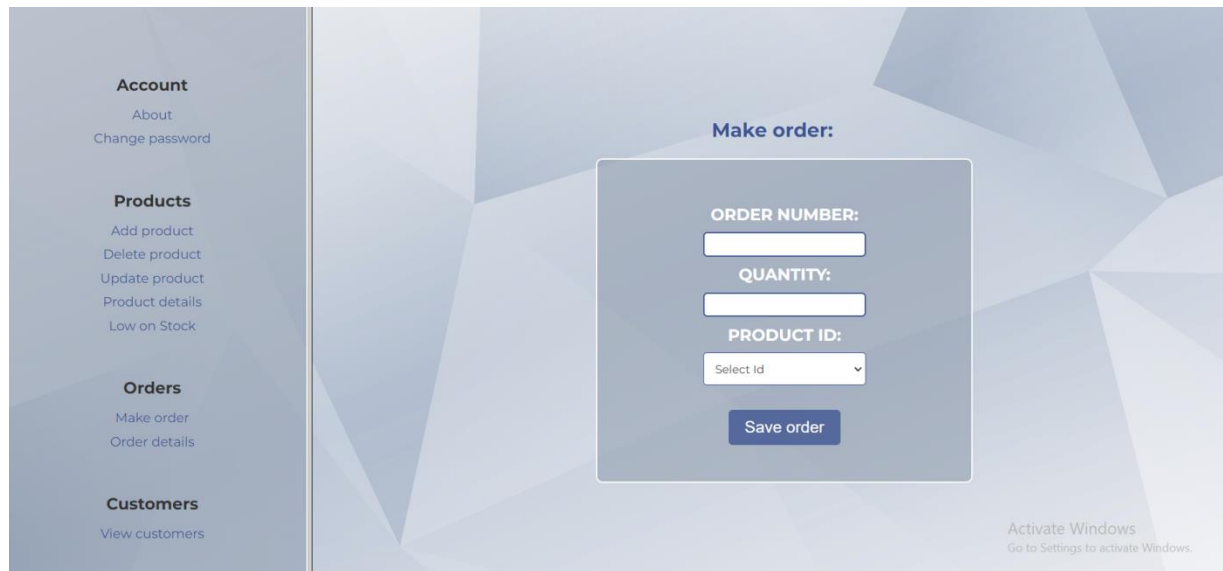
Product ID	Product Name	Quantity in Stock
1	APPLE iPhone 14 PRO	27
6	ASUS Vivobook PRO15	30
9	Philips Prestige	46

In the bottom right corner, there is a watermark that says "Activate Windows Go to Settings to activate Windows."

make_order.php

- The **make_order.php** file allows users to create new orders in the inventory management system.

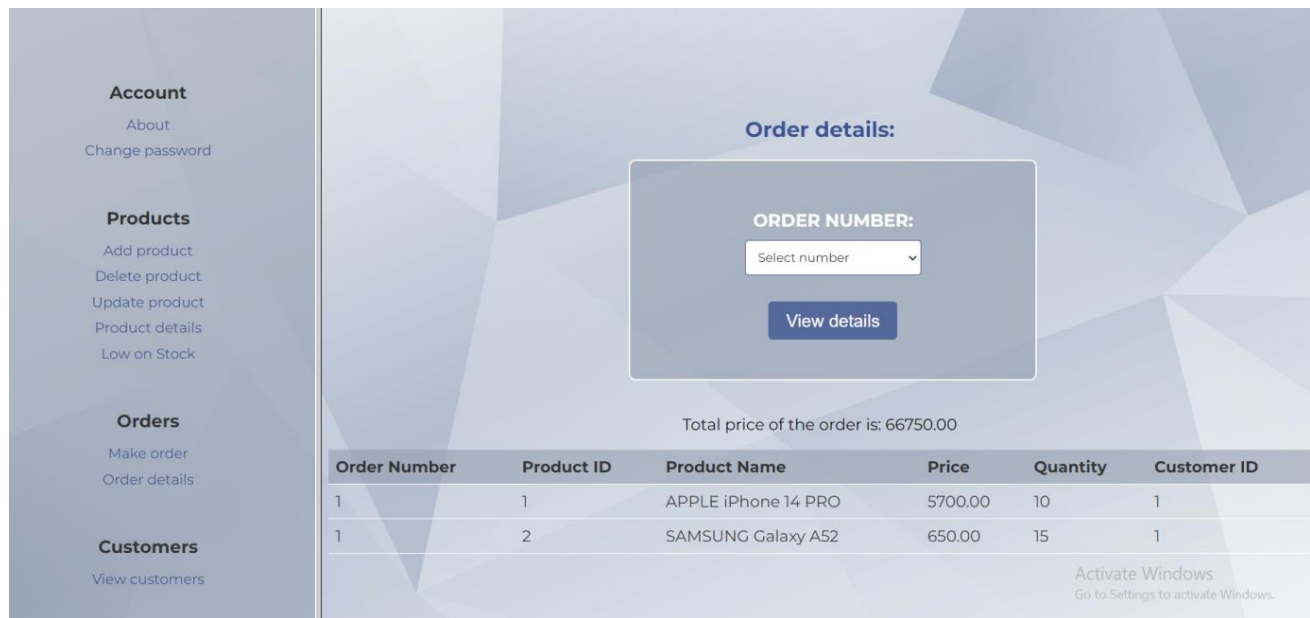
- It starts by including the **connect.php** file, which establishes a connection to the database.
- The code then starts a session using **session_start()** to enable session-based data storage.
- If the request method is POST (indicating a form submission), the code retrieves the necessary form data: **orderNr**, **productId**, **quantity**, and **employeeId** (retrieved from the session).
- A SQL INSERT query is constructed to insert the order details into the **orders** table. The order number, employee ID, product ID, and quantity are inserted into the respective columns.
- The query is executed using **\$conn->query(\$query)** and the result is stored in the **\$result** variable.
- If the query execution is successful, a success message is stored in the session variable **\$_SESSION['message']**. The user is then redirected to the same page using **header("Location: ".\$_SERVER['PHP_SELF'])** to prevent duplicate form submissions.
- The HTML form allows users to enter the order number, quantity, and select a product ID from a dropdown menu.
- The product IDs are fetched from the **products** table in the database using a SELECT query.
- Inside the dropdown menu, a PHP while loop is used to iterate over the query results and generate an option for each product ID.
- The resulting HTML page displays the message (success or error) at the top, followed by a form where users can enter the order details.
- Users can enter the order number, quantity, and select a product ID to create a new order.



order_details.php

- The **order_details.php** file allows users to view details of a specific order in the inventory management system.
- It starts by including the **connect.php** file, which establishes a connection to the database.
- If the request method is POST and the **orderNr** parameter is set (indicating a form submission), the code retrieves the selected order number from **\$_POST["orderNr"]**.
- A SQL SELECT query is constructed to fetch the details of the selected order. The query joins the **orders** table with the **products** table on the **ProductId** column and filters the results based on the selected order number.
- The query is executed using **\$conn->query(\$query)** and the result is stored in the **\$result** variable.
- If the query returns rows (indicating the order is found), the order details are fetched into the **\$orders** variable using **\$result->fetch_all(MYSQLI_ASSOC)**.
- To calculate the total price of the order, a separate query is executed. The query uses the SUM function to multiply the price of each product by its quantity and calculates the total price. The result is stored in the **\$totalPrice** variable.

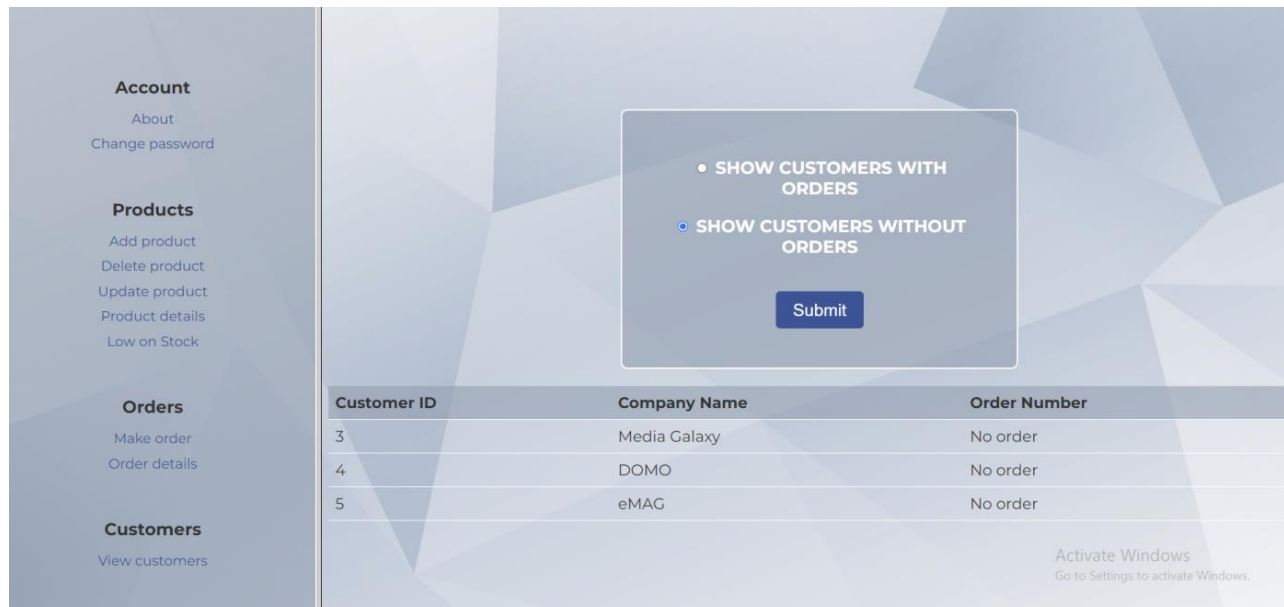
- If the query doesn't return any rows, an error message is stored in the **\$error** variable.
- The HTML form allows users to select an order number from a dropdown menu and view its details by clicking the "View details" button.
- The order numbers are fetched from the **orders** table using a SELECT query. Inside the dropdown menu, a PHP while loop is used to iterate over the query results and generate an option for each order number.
- When the user submits the form, the selected order number is sent via POST to the same page (**\$_SERVER['PHP_SELF']**).
- The resulting HTML page displays the order details in a table if they exist.
- The table headers include columns for Order Number, Product ID, Product Name, Price, Quantity, and Customer ID.
- Inside the table body, a PHP foreach loop is used to iterate over the **\$orders** array and display the details of each order in a row.
- After the table, the total price of the order is displayed if it exists.



customer_list.php

- The **customer_list.php** file displays a list of customers in the inventory management system based on user-selected options.
- It starts by including the **connect.php** file, which establishes a connection to the database.

- If the form is submitted and the **option** parameter is set, the code retrieves the selected option from **\$_POST['option']**.
- Depending on the selected option, a SQL query is constructed to fetch the customers.
 - If the option is 'with_orders', the query joins the **customers** table with the **orders** table on the **CustomerID** column and groups the results by **CustomerID**.
 - If the option is 'without_orders', the query uses a left join to include all customers from the **customers** table where the **CustomerID** in the **orders** table is null (indicating no orders).
 - If neither of the above options is selected, an error message is stored in the **\$error** variable.
- The query is executed using **mysqli_query(\$conn, \$query)**, and the result is stored in the **\$result** variable.
- If the query returns rows (indicating customers are found), the customer details are fetched into the **\$customers** variable using **\$result->fetch_all(MYSQLI_ASSOC)**.
- The HTML form allows users to select an option to either show customers with orders or customers without orders.
- Inside the form, two radio buttons are provided for each option, and the previously selected option is checked using PHP if conditions.
- When the user submits the form, the selected option is sent via POST to the same page.
- The resulting HTML page displays the customer list in a table if customers exist.
- The table headers include columns for Customer ID, Company Name, and Order Number.
- Inside the table body, a PHP foreach loop is used to iterate over the **\$customers** array and display the details of each customer in a row.
- If the customer has an associated order, the order number is displayed. Otherwise, the cell displays "No order".



Bibliography:

- https://www.youtube.com/watch?v=48VkVOHYGWw&t=31s&ab_channel=LifeBeyondCode
- <https://www.w3schools.com/html/>
- <https://www.w3schools.com/php/default.asp>
- <https://www.w3schools.com/css/>
- <https://www.tutorialspoint.com/php/index.html>
- <https://www.phptutorial.net/>
- <https://openai.com/blog/chatgpt>