Inventory and Supplier System

Final Project Report

Database Technology

COMP6799001

Lecturer

Yenny, S.Kom., M.Kom.

Report by:

Irene Angelina - 2802501060

Jeremy Nathanael Gunawan - 2802522960

Osten Antonio - 2802546115

Class L3BC

Computer Science Program

School of Computing and Creative Arts

BINUS UNIVERSITY INTERNATIONAL

JAKARTA

(2025)

Table of Contents

I. Introduction

For this Database Technology final project, our group was selected to implement an Inventory and Supplier Management System. Modern businesses that operate multiple warehouses require efficient systems to manage inventory, suppliers, and transactions across different locations. Without a centralized system, tracking stock levels, managing orders, and coordinating suppliers becomes very inefficient and prone to errors. This can potentially lead to stock shortages or overstocking.

An Inventory and Supplier Management System provides a centralized platform for monitoring warehouse inventory, managing product information, and recording transactions in real time. By consolidating this into a single database, warehouse managers can gain better visibility into warehouse operations, supplier relationships, and order fulfillment processes. This allows for faster decision-making and improvement in operational efficiency.

To support these requirements, a well-designed database is essential. The database must be able to store and organize data while minimizing redundancy, as well as maintaining data integrity. A proper database design ensures that real-world business rules – like how products are supplied, how orders are structured, and how inventory is tracked – are accurately represented.

This project focuses on the design and implementation of an Inventory and Supplier Management System database. It is designed to support internal business operations for managers and suppliers. This is done by providing structured access to warehouse data, product information, orders, and inventory records.

Objectives

The main objective of an Inventory and Supplier Management System is to allow businesses to oversee the warehouses they have, manage each product and transaction in an organized and efficient manner. Based on this, the objectives of this project are as follows:

- Allowing managers to manage their warehouses, including the products stored in each warehouse
- Allowing the management of orders for each warehouse
- Allowing the management of each warehouse
- Allowing suppliers to create new products
- Provide useful summarized information for managers and suppliers through aggregate functions

These objectives guide the overall system design and determine the structure of the database and its core entities

System Overview

The Inventory and Supplier Management System allows users to view and manage information related to warehouses, products, inventory, orders, and suppliers. The system is designed for internal use and supports the operational needs of warehouse managers and product suppliers.

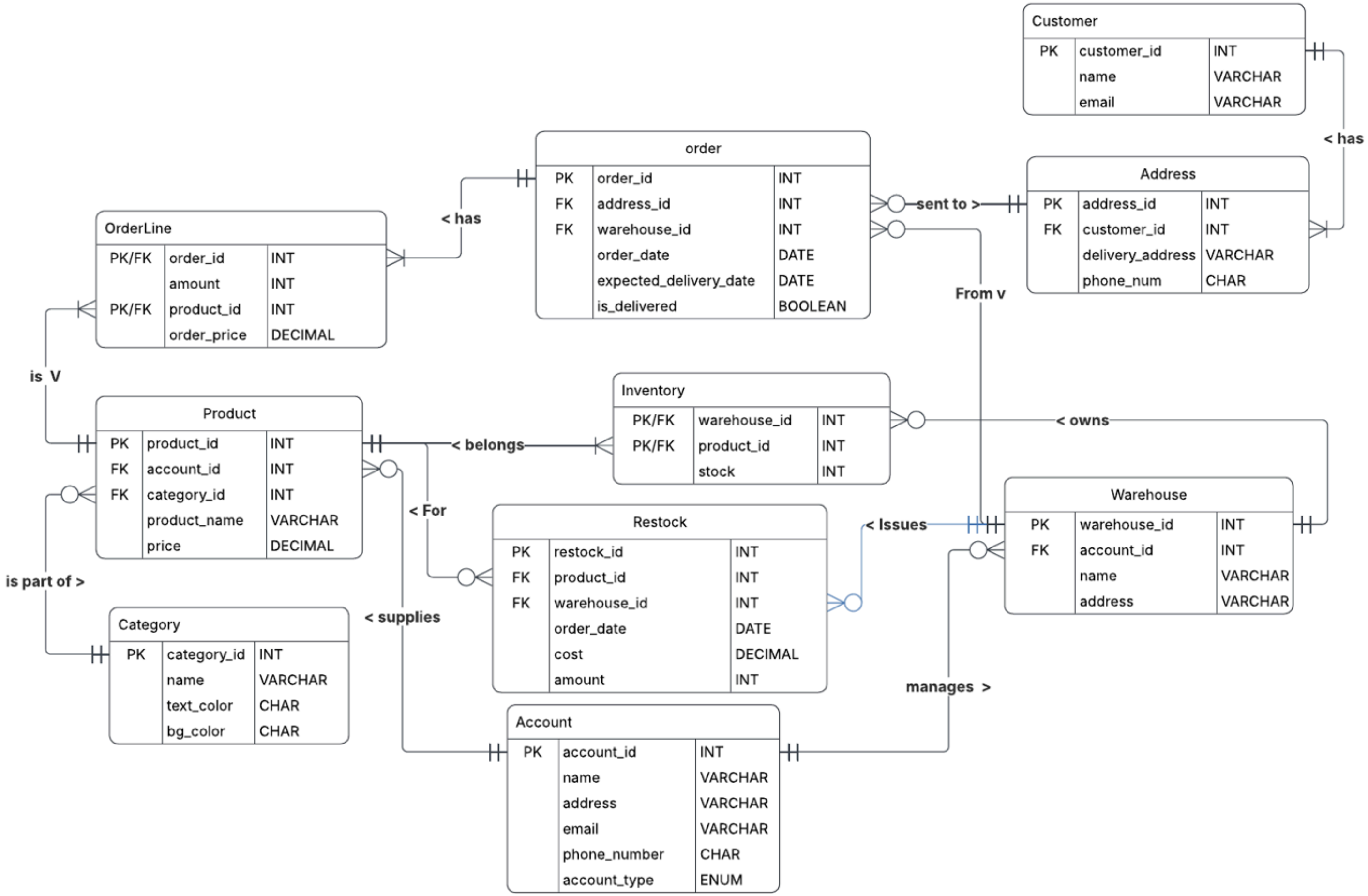
The system is implemented as a web-based application, consisting of a backend server and a frontend interface. The backend handles all database operations, including Create, Read, Update, and Delete (CRUD) actions. The frontend provides an interface for users to interact with the system. This separation allows the database to function independently from the user interface while maintaining data consistency and integrity.

Database Design

Core assumptions

- A customer can have multiple delivery addresses, with each address having its own phone number
- An order:
  - Comes from one warehouse
  - Is delivered to one address
  - Can contain multiple products (via OrderLine)
- A restock order can only contain one product; however, a warehouse can create multiple restock orders for the same product over time
- A warehouse is managed by one manager, while a manager can manage multiple warehouses
- A supplier can provide multiple products, but each product only has one supplier
- There are no partial deliveries for orders; orders are either delivered or not delivered
- The price stored in Product table represents the current list price, while order_price in OrderLine stores the historical selling price at the time of the transaction
- The Category table is used mainly for user interface and grouping purposes; multiple categories may share the same name but differ in visual attributes such as color
- Customers are not considered a part of the system, as the database was designed for internal use by managers and suppliers only

II. ERD Diagram



Customer

PK	customer_id	INT
	name	VARCHAR
	email	VARCHAR

order

PK	order_id	INT
FK	address_id	INT
FK	warehouse_id	INT
	order_date	DATE
	expected_delivery_date	DATE
	is_delivered	BOOLEAN

Address

PK	address_id	INT
FK	customer_id	INT
	delivery_address	VARCHAR
	phone_num	CHAR

OrderLine

PK/FK	order_id	INT
	amount	INT
PK/FK	product_id	INT
	order_price	DECIMAL

Inventory

PK/FK	warehouse_id	INT
PK/FK	product_id	INT
	stock	INT

Product

PK	product_id	INT
FK	account_id	INT
FK	category_id	INT
	product_name	VARCHAR
	price	DECIMAL

Restock

PK	restock_id	INT
FK	product_id	INT
FK	warehouse_id	INT
	order_date	DATE
	cost	DECIMAL
	amount	INT

Warehouse

PK	warehouse_id	INT
FK	account_id	INT
	name	VARCHAR
	address	VARCHAR

Category

PK	category_id	INT
	name	VARCHAR
	text_color	CHAR
	bg_color	CHAR

Account

PK	account_id	INT
	name	VARCHAR
	address	VARCHAR
	email	VARCHAR
	phone_number	CHAR
	account_type	ENUM

III. Entities

I. Main entities

- **Account (Manager/Supplier/Admin)**
  - Accounts for the user, storing account details like name, email, password, and contact information (address and phone number)
- **Customer**
  - Customers that are available in the system are shared globally across all warehouses (since one company), contains name and email
- **Warehouse**
  - Warehouses that are available in the system, contains the name, address, and manager
- **Order**
  - Stores customer orders, including order lines, delivery address, order and delivery dates, and delivery status
  - Used to track sales data such as total revenue and items sold
- **Product**
  - Products that are available in the system, contains details such as name, price, category, and supplier
- **Restock**
  - Represents the order made by the warehouse to a supplier for a new product or restock existing products in the warehouse, containing the cost at which each item is bought and the amount.

II. Supporting entities

- **Address**
  - Customer's addresses are used in order, and contain the delivery address and the phone number (receiver/address phone number)
- **OrderLine**
  - Represents each product in one order, and contains supporting information such as the price bought and the amount bought
- **Inventory**
  - Represents each product owned by a warehouse, used to track stock

III. Relationships

1. **Customer to Address: one to many**

   One customer has multiple address but an address only belongs to one customer

2. **Address to Order: one to many**

   An address can have many order but an order can only be shipped to one address

3. **Order to OrderLine: one to many**

   An order contains many orderline which represent a product. However, an orderline is only assigned to one order

4. **OrderLine to Product: many to one**

   Each orderline represents one product, and each product can have many order lines

5. **Product to Category: one to many**

   Each product has only one category, but each category contains many products

6. **Product to Inventory: one to many**

   A product belongs to many inventory, but each inventory only represents one product

7. **Product to Restock: one to many**

   A product can be ordered by many restock orders, but a restock order only represents one product

8. **Warehouse to Inventory: one to many**

   A warehouse contains many inventory, but an inventory is only owned by one warehouse

9. **Warehouse to Restock: one to many**

   A warehouse can issue many restock orders, but a restock order is only for one warehouse

10. **Warehouse to Order: one to many**

    An order can only come from one warehouse, but a warehouse can have many orders

11. **Account to Warehouse: one to many**

    An account (manager) can manage multiple warehouse but a warehouse can only be managed by one account (manager)

12. **Account to Product: one to many**

    An account (supplier) can supply and manage multiple products but a product can only be managed by one account (supplier)

IV. Database normalization

I. Unnormalized

The central entity in the inventory and supplier system is the warehouse as almost all transactions will end up using the warehouse entity at the end, as such the following is the warehouse table with all necessary details, each row represents one warehouse:

Warehouse

Attribute	Sample data	Description
order_details	"Product A:2:100000:Product:Bob:Jl. Kemang Raya 12, Jakarta Selatan:bob@mail.com:081234567890; Item B:1:50000:Item:Andi:Jl. Sudirman 78, Jakarta Pusat:andi@mail.com:082334455667;Item C:1:50000:Item:Andi:Jl. Sudirman 78, Jakarta Pusat:andi@mail.com:082334455667"	Detail of the order of the warehouse
order_date	2025-10-01:2025-10-02	Date of order
expected_delivery_date	2025-10-05:2025-10-25	Expected delivery date
delivery_status	TRUE:FALSE	Delivery status
customer_details	"Bob:bob@mail.com:Jl. Kemang Raya 12, Jakarta Selatan:81234567890;Andi:andi@mail.com;Jl. Sudirman 78, Jakarta Pusat;82334455667"	Details of the customer that ordered from the warehouse
warehouse_details	"Central Depot:Jl. Industri No.100, Jakarta Barat"	Detail of the warehouse
warehouse_inventory	"Product A:500:1000000:Product; Item B:300:25000:Item"	Inventory of the warehouse
manager_details	"Alice:Jl. Mangga Besar 12, Jakarta:alice@mail.com:081234567890"	Manager detail
supplier_details	"Product A:Rizky:Jl. Kemang Raya 12, Jakarta Selatan:Rizky@mail.com:081234567890; Item B:Rizky:Jl. Kemang Raya 12 Jakarta Pusat:Rizky@mail.com:082334455667; Item D:Rizky:Jl. Sudirman 78, Jakarta Pusat:Rizky@mail.com:082334455667 "	Product and supplier detail
restock_details	"Product A:2:100000:Product:Rizky:Jl. Kemang Raya 12 Jakarta Pusat:Rizky@mail.com:082334455667; Item B:1:50000:Item:Rizky:Jl. Kemang Raya 12 Jakarta Pusat:Rizky@mail.com:082334455667"	Detail of restock

II. 1st Normal Form

The first normal form needs each field/column to be one data type and each cell only containing one data (atomic values):

Warehouse

Attributes	Datatype	Key	Description
order_id	INT	PK	Unique identifier for orders
order_date	DATE		Date of order
order_amount	INT		Amount of item ordered
order_price	DECIMAL		Price of item during time of order
expected_delivery_date	DATE		Expected delivery date
is_delivered	BOOLEAN		Flag for if the order is delivered
address	VARCHAR		Address of customer
address_number	CHAR		Phone number of the address
product	VARCHAR		Product name
customer_name	VARCHAR		Customer name
customer_email	VARCHAR		Customer email
warehouse_name	VARCHAR		Warehouse name
warehouse_address	VARCHAR		Warehouse address
manager_name	VARCHAR		Name of manager
manager_address	VARCHAR		Address of manager
manager_email	VARCHAR		Email of manager
manager_phone_num	VARCHAR		Phone number of the manager
product_id	INT	PK	Unique identifier for products
stock	INT		Stock of product within a warehouse
category	VARCHAR		Product category
price	DECIMAL		Price of the product
supplier_name	VARCHAR		Name of supplier
supplier_address	VARCHAR		Address of supplier
supplier_email	VARCHAR		Email of supplier
supplier_phone_num	VARCHAR		Phone number of the supplier
restock_id	INT		Unique identifier for restock order
restock_product	VARCHAR		Name of the item to be restocked to the warehouse
restock_amount	INT		Amount of product ordered to be restocked
restock_date	DATE		Date of where restock is ordered
restock_cost	DECIMAL		Cost of item at the time of purchase

From the attributes above, there are multiple entities, as such a surrogate (primary) key is made for each entity later; customer, manager, and supplier.

Partial dependencies

order_id -> order_date, expected_delivery_date, is_delivered, address, warehouse_id
product_id->product_name, restock_name, supplier_id
warehouse_id,product_id->stock

Transitive dependencies

warehouse_id -> warehouse_name, warehouse_address, manager_id, restock_id
manager_id-> manager_address, manager_email, manager_phone_num, manager_name
supplier_name->supplier_email,supplier_phone_num, supplier_address, supplier_name
restock_id->product_id,restock_cost,restock_amount,restock_date
customer_id, address-> address_phone
customer_id -> customer_email, address,customer_name

* Since addresses can be changed, it would be better if a new surrogate key is made to identify that address:
address_id-> address_phone,customer_id, address
customer_id -> customer_email, address,customer_name

Full dependencies

order_id, product_id->order_amount,order_price

III. 2nd Normal form

The second normal form involves splitting the table in order to get rid of the partial dependencies, as such:

Order

Attributes	Datatype	Key	Description
order_id	INT	PK	Unique identifier for orders
order_date	DATE		Date of order
expected_delivery_date	DATE		Amount of item ordered
is_delivered	BOOLEAN		Price of item during time of order
address_id	INT		Unique identifier for address
customer_id	INT		Unique identifier for customer
customer_name	VARCHAR		Name of customer
customer_email	VARCHAR		Email of customer
address	VARCHAR		Address
phone_number	VARCHAR		Phone number of address
warehouse_id	INT		Unique identifier for warehouse
warehouse_name	VARCHAR		Warehouse name
warehouse_address	VARCHAR		Warehouse address
manager_id	INT		Unique identifier for manager
manager_name	VARCHAR		Name of manager
manager_address	VARCHAR		Address of manager
manager_email	VARCHAR		Email of manager
manager_phone_num	VARCHAR		Phone number of the manager
restock_id	INT		Unique identifier for restock order
product_id	INT		Unique identifier for restock order
restock_amount	INT		Amount of product ordered to be restocked
restock_date	DATE		Date of where restock is ordered
restock_cost	DECIMAL		Cost of item at the time of purchase

OrderLine

Attributes	Datatype	Key	Description
order_id	INT	PK/FK	Unique identifier for orders
product_id	INT	PK/FK	Unique identifier for product
order_amount	INT		Amount of item ordered
order_price	DECIMAL		Price of item during time of order

Product

Attributes	Datatype	Key	Description
product_id	INT	PK	Unique identifier for products
product_name	VARCHAR		Product name
category	VARCHAR		Product category
price	DECIMAL		Price of the product
supplier_id	INT		Unique identifier for supplier
supplier_name	VARCHAR		Name of supplier
supplier_address	VARCHAR		Address of supplier
supplier_email	VARCHAR		Email of supplier
supplier_phone_num	VARCHAR		Phone number of the supplier

Inventory

Attributes	Datatype	Key	Description
warehouse_id	INT	PK/FK	Unique identifier for warehouse
product_id	INT	PK/FK	Unique identifier for product
stock	INT		Stock of item within a warehouse
warehouse_name	VARCHAR		Warehouse name
warehouse_address	VARCHAR		Warehouse address
manager_id	INT		Unique identifier for manager
manager_name	VARCHAR		Name of manager
manager_address	VARCHAR		Address of manager
manager_email	VARCHAR		Email of manager
manager_phone_num	VARCHAR		Phone number of the manager
restock_id	INT	PK	Unique identifier for restock order
product_id	INT	FK	Unique identifier for restock order
restock_amount	INT		Amount of product ordered to be restocked
restock_date	DATE		Date of where restock is ordered
restock_cost	DECIMAL		Cost of item at the time of purchase

Resolved dependencies

order_id -> order_date, expected_delivery_date, is_delivered, address, warehouse_id

product_id->product_name, restock_name, supplier_id

warehouse_id,product_id->stock

order_id, product_id->order_amount,order_price

Transitive dependencies

warehouse_id -> warehouse_name, warehouse_address, manager_id, restock_id

manager_id-> manager_address, manager_email, manager_phone_num, manager_name

supplier_id->supplier_email,supplier_phone_num, supplier_address, supplier_name

restock_id->product_id,restock_cost,restock_amount,restock_date

address_id->address, address_phone,customer_id

customer_id -> customer_email, address,customer_name

IV. 3rd Normal form

The third normal form resolves the remaining dependencies which are not resolved by the second normal form (transitive):

Manager

Attributes	Datatype	Key	Description
manager_id	INT	PK	Unique identifier for manager
manager_name	VARCHAR		Name of manager
manager_address	VARCHAR		Address of manager
manager_email	VARCHAR		Email of manager
manager_phone_num	VARCHAR		Phone number of the manager

Supplier

Attributes	Datatype	Key	Description
supplier_id	INT	PK	Unique identifier for supplier
supplier_name	VARCHAR		Name of supplier
supplier_address	VARCHAR		Address of supplier
supplier_email	VARCHAR		Email of supplier
supplier_phone_num	VARCHAR		Phone number of the supplier

Order

Attributes	Datatype	Key	Description
order_id	INT	PK	Unique identifier for orders
order_date	DATE		Date of order
expected_delivery_date	DATE		Amount of item ordered
is_delivered	BOOLEAN		Price of item during time of order
address_id	INT	FK	Unique identifier for address
warehouse_id	INT		Unique identifier for warehouse

OrderLine

Attributes	Datatype	Key	Description
order_id	INT	PK/FK	Unique identifier for orders
product_id	INT	PK/FK	Unique identifier for product
order_amount	INT		Amount of item ordered
order_price	DECIMAL		Price of item during time of order

Warehouse

Attributes	Datatype	Key	Description
warehouse_id	INT	PK	Unique identifier for manager
manager_id	VARCHAR	FK	Unique identifier for account
warehouse_address	VARCHAR		Address of manager
warehouse_name	VARCHAR		Email of manager

Inventory

Attributes	Datatype	Key	Description
warehouse_id	INT	PK/FK	Unique identifier for warehouse
product_id	INT	PK/FK	Unique identifier for product
stock	INT		Stock of item within a warehouse

Product

Attributes	Datatype	Key	Description
product_id	INT	PK	Unique identifier for products
product_name	VARCHAR		Product name
category	VARCHAR		Product category
price	DECIMAL		Price of the product
supplier_id	INT	FZK	Unique identifier for supplier

Restock

Attributes	Datatype	Key	Description
restock_id	INT	PK	Unique identifier for restock order
warehouse_id	INT	FK	Unique identifier for restock order
product_id	INT	FK	Unique identifier for restock order
restock_amount	INT		Amount of product ordered to be restocked
restock_date	DATE		Date of where restock is ordered
restock_cost	DECIMAL		Cost of item at the time of purchase

Address

Attributes	Datatype	Key	Description
address_id	INT	PK	Unique identifier for address
customer_id	INT	FK	Unique identifier for customer
address	VARCHAR		Address
phone_number	VARCHAR		Phone number of address

Customer

Attributes	Datatype	Key	Description
customer_id	INT	PK	Unique identifier for customer
customer_name	VARCHAR		Name of customer
customer_email	VARCHAR		Email of customer

The manager and supplier table will be merged together as they both act as the same entity, but with different permission levels, and to accommodate the Admin role, an account table is made. In the future, specialization/generalization can be used to create a separate table for each role with their new attributes that are specifically needed for them (not within this project's scope).

Adapting to the application, a new table derived from category_name (product table) will be made which is used to store the attributes for the UI, this can also be helpful in the future as it reduces redundancy on tasks such as renaming a category name. As such the ERD presented earlier is the representation of these implementations. The following are also additional constraints (other than FK):

- All foreign key will have ON DELETE CASCADE to reinforce referential integrity (as all entity that has FK needs the other entity), except for strong entities such as warehouse and product
- All numerics, specifically stock, cost/price and amount has constraint of not being negative:

  CONSTRAINT not_negative (______>0)

V. Table Structure

Account

Attributes	Datatype	Key	Constraints
account_id	INT	PK	AUTO_INCREMENT
account_type	ENUM		NOT NULL
password	VARCHAR(255)		NOT NULL
name	VARCHAR(255)		NOT NULL
address	VARCHAR(255)		NOT NULL
email	VARCHAR(255)	UNIQUE	NOT NULL
phone_num	CHAR(15)		NOT NULL

Category

Attributes	Datatype	Key	Constraints
category_id	INT	PK	AUTO_INCREMENT
name	VARCHAR(255)		NOT NULL
text_color	CHAR(7)		DEFAULT (“#FFFFFF”)
bg_color	CHAR(7)		DEFAULT (“#000000”)

Order

Attributes	Datatype	Key	Constraints
order_id	INT	PK	AUTO_INCREMENT
order_date	DATE		NOT NULL
expected_delivery_date	DATE		NOT NULL
is_delivered	BOOLEAN		DEFAULT (FALSE)
address_id	INT	FK	REFERENCES Address (address_id) ON DELETE CASCADE
warehouse_id	INT	FK	REFERENCES Warehouse(warehouse_id) ON DELETE CASCADE

OrderLine

Attributes	Datatype	Key	Constraints
order_id	INT	PK/FK	REFERENCES `Order` (order_id) ON DELETE CASCADE
product_id	INT	PK/FK	REFERENCES Product (product_id) ON DELETE CASCADE
order_amount	INT		DEFAULT(0), CHECK (order_amount >=0)
order_price	DECIMAL		DEFAULT(0), CHECK (order_price >=0)

Warehouse

Attributes	Datatype	Key	Constraints
warehouse_id	INT	PK	AUTO_INCREMENT
account_id	VARCHAR(255)	FK	REFERENCES Account(account_id)
warehouse_address	VARCHAR(255)		NOT NULL
warehouse_name	VARCHAR(255)		NOT NULL

Inventory

Attributes	Datatype	Key	Constraints
warehouse_id	INT	PK/FK	REFERENCES Warehouse(warehouse_id) ON DELETE CASCADE
product_id	INT	PK/FK	REFERENCES Product(produce_id) ON DELETE CASCADE
stock	INT		DEFAULT(0), CHECK(stock>=0)

Product

Note: category_id of 0 is ‘uncategorized’

Attributes	Datatype	Key	Constraints
product_id	INT	PK	AUTO_INCREMENT
product_name	VARCHAR(255)		NOT NULL
category_id	INT	FK	REFERENCES Category(category_id), DEFAULT (0)
price	DECIMAL(15,3)		DEFAULT (0), CHECK (price>=0)
account_id	INT	FK	REFERENCES Account(account_id) ON DELETE CASCADE

Restock

Attributes	Datatype	Key	Constraints
restock_id	INT	PK	AUTO_INCREMENT
warehouse_id	INT	FK	REFERENCES Warehouse(warehouse_id) ON DELETE CASCADE
product_id	INT	FK	REFERENCES Product(product_id) ON DELETE CASCADE
restock_amount	INT		DEFAULT(0), CHECK (restock_amount>=0)
restock_date	DATE		NOT NULL
restock_cost	DECIMAL(15,3)		DEFAULT(0), CHECK (restock_cost>=0)

Address

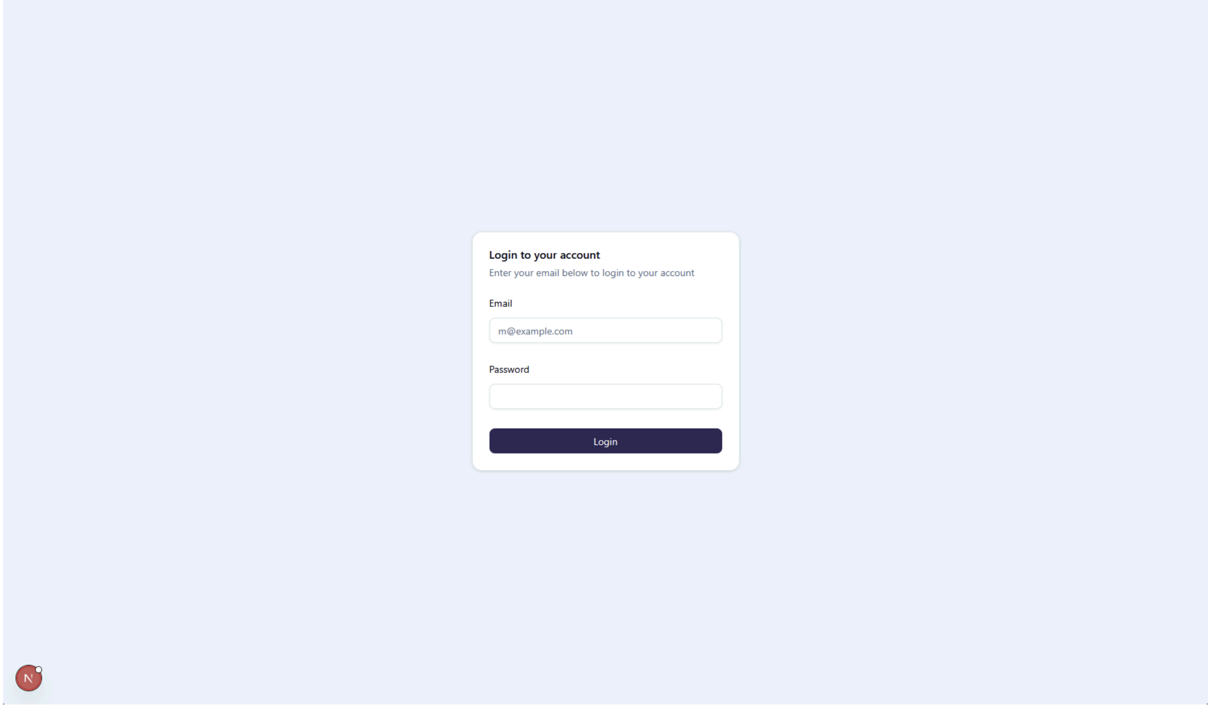
Attributes	Datatype	Key	Constraints
address_id	INT	PK	AUTO_INCREMENT
customer_id	INT	FK	REFERENCES Customer(customer_id) ON DELETE CASCADE
address	VARCHAR(255)		NOT NULL
phone_number	VARCHAR(255)		NOT NULL

Customer

Attributes	Datatype	Key	Constraints
customer_id	INT	PK	AUTO_INCREMENT
customer_name	VARCHAR(255)		NOT NULL
customer_email	VARCHAR(255)	UNIQUE	NOT NULL

VI. User Interface

Login screen



Login to your account

Enter your email below to login to your account

Email

m@example.com

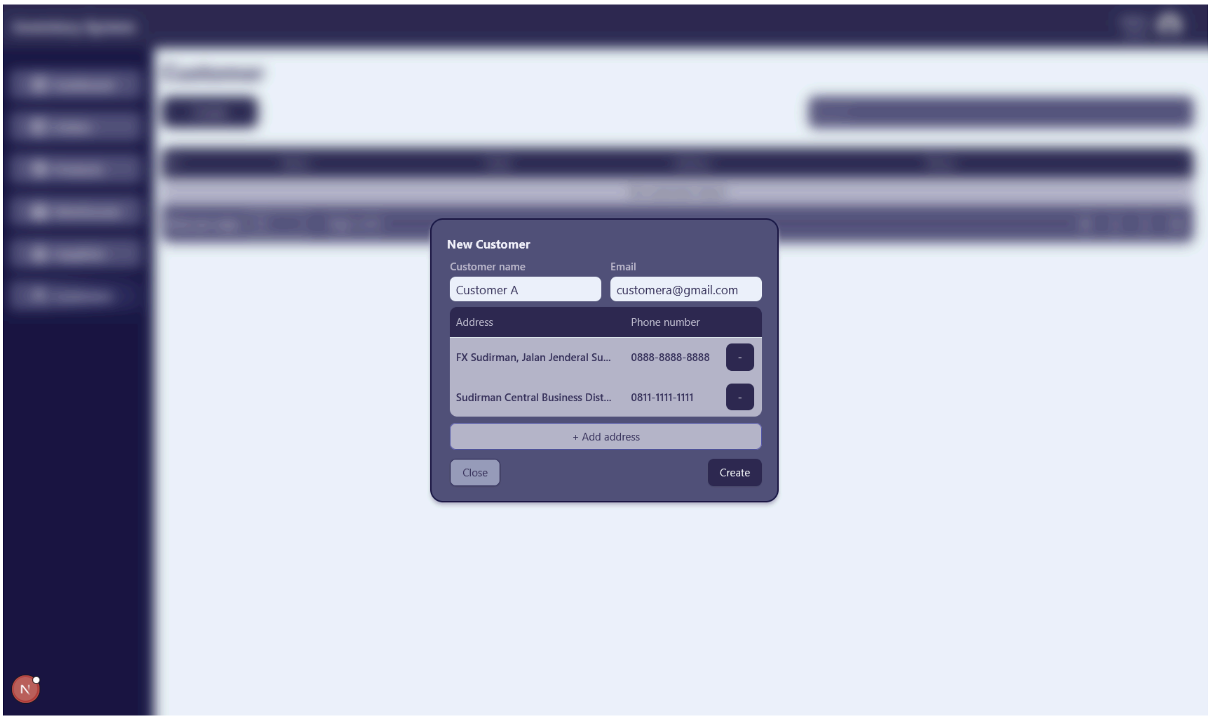
Password

Only admins can create user, users are given the account as it is an in company software

Customer

Creation/Editing screen

For editing, creation screen is used but prefilled with the data that needs to be edited, the primary key of the main entity is stored in the background (not shown in the UI)



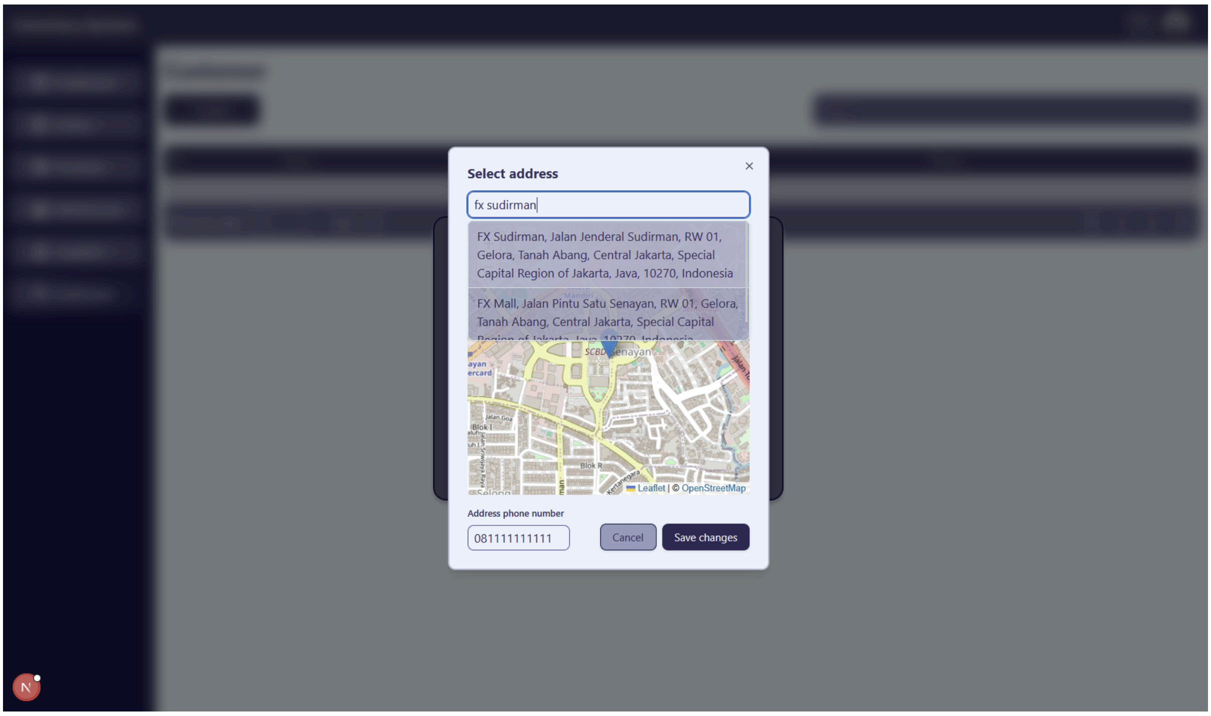
New Customer

Customer name: Customer A

Email: customera@gmail.com

Address	Phone number	
FX Sudirman, Jalan Jenderal Su...	0888-8888-8888	-
Sudirman Central Business Dist...	0811-1111-1111	-

Add address screen



Select address

fx sudirman

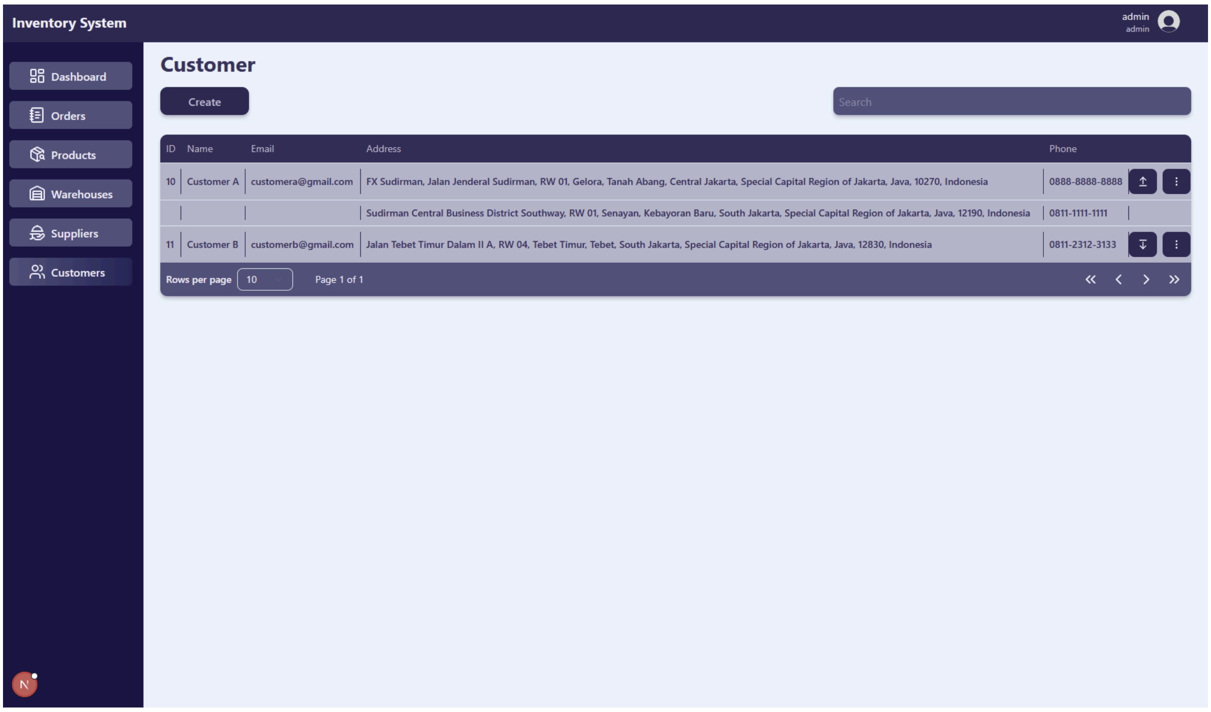
FX Sudirman, Jalan Jenderal Sudirman, RW 01, Gelora, Tanah Abang, Central Jakarta, Special Capital Region of Jakarta, Java, 10270, Indonesia

FX Mall, Jalan Pintu Satu Senayan, RW 01, Gelora, Tanah Abang, Central Jakarta, Special Capital Region of Jakarta, Java, 10270, Indonesia

Address phone number: 081111111111

Customer summary

Each row is an accordion for each of the customers if they have multiple addresses, the search bar supports parameter handling if specific columns need to be searched such as ?customer_name=aaaa?address=somewhere

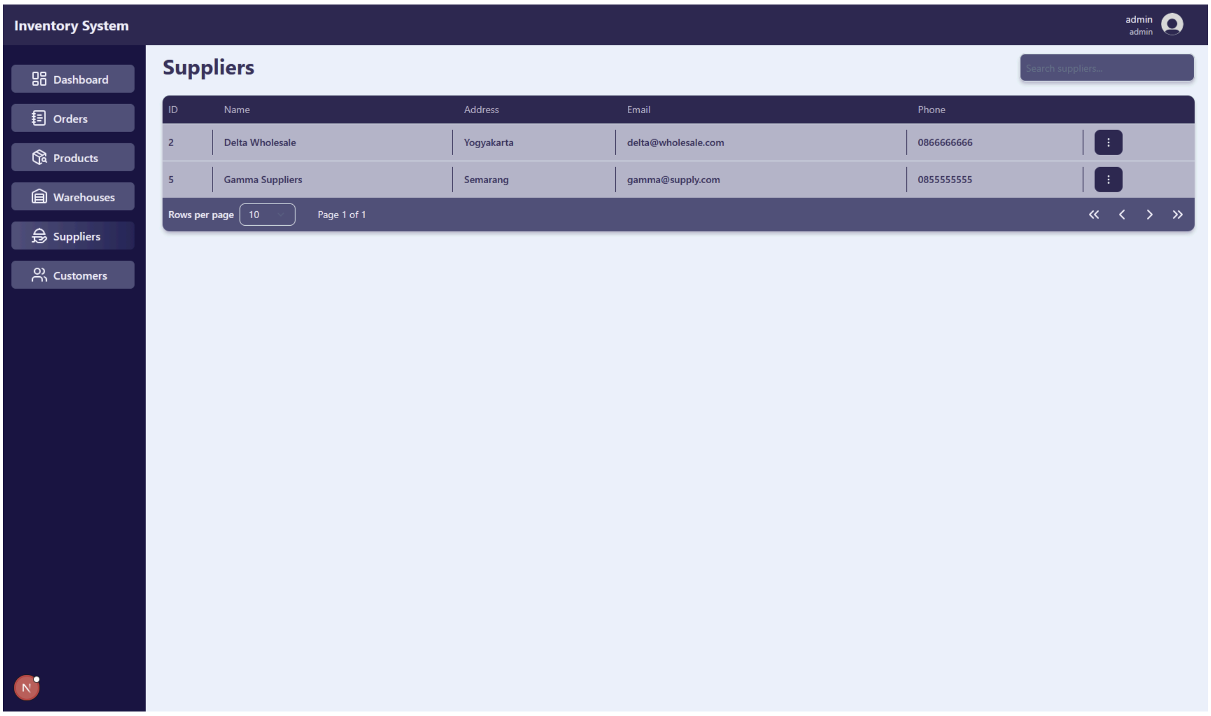


ID	Name	Email	Address	Phone
10	Customer A	customera@gmail.com	FX Sudirman, Jalan Jenderal Sudirman, RW 01, Gelora, Tanah Abang, Central Jakarta, Special Capital Region of Jakarta, Java, 10270, Indonesia	0888-8888-8888
			Sudirman Central Business District Southway, RW 01, Senayan, Kebayoran Baru, South Jakarta, Special Capital Region of Jakarta, Java, 12190, Indonesia	0811-1111-1111
11	Customer B	customerb@gmail.com	Jalan Tebet Timur Dalam II A, RW 04, Tebet Timur, Tebet, South Jakarta, Special Capital Region of Jakarta, Java, 12830, Indonesia	0811-2312-3133

Supplier

Supplier summary

Since supplier is linked to account, the screen only shows who are the suppliers, and a simple search bar



Suppliers

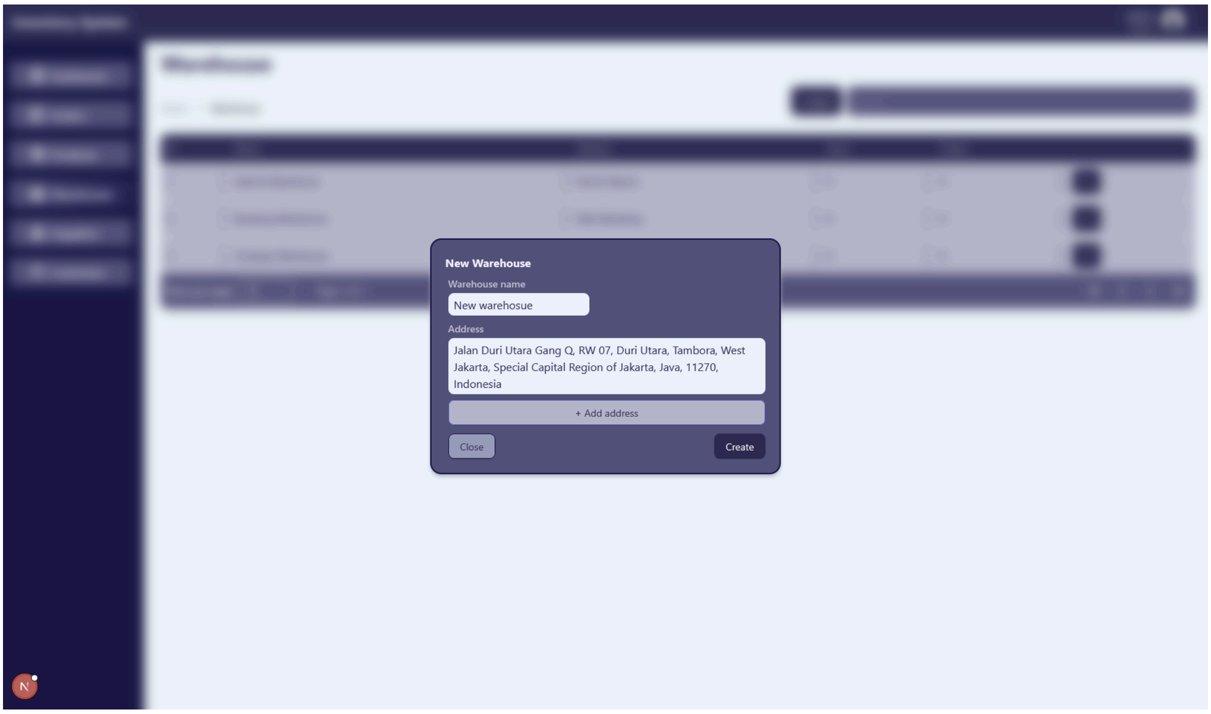
ID	Name	Address	Email	Phone	
2	Delta Wholesale	Yogyakarta	delta@wholesale.com	0866666666	⋮
5	Gamma Suppliers	Semarang	gamma@supply.com	0855555555	⋮

Rows per page 10 Page 1 of 1

Warehouse

Creation/Editing screen

Add address reuses the same address selection screen as customer without the unique phone number

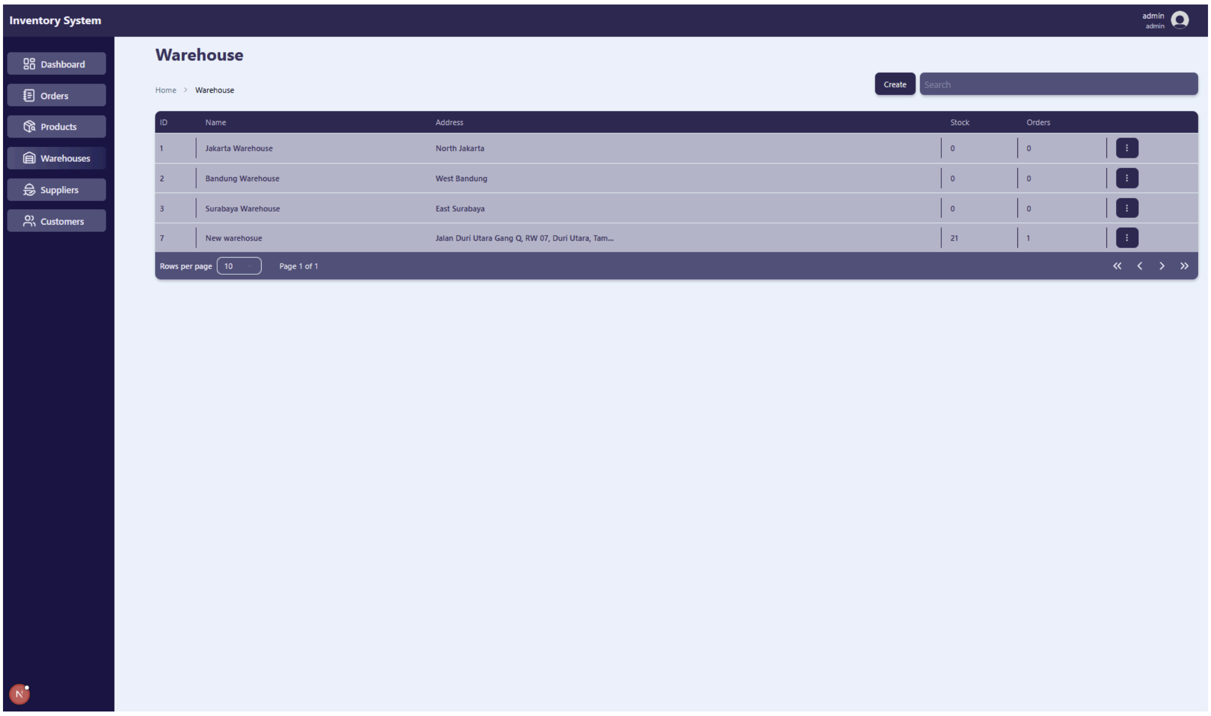


New Warehouse

Warehouse name: New warehosue

Address: Jalan Duri Utara Gang Q, RW 07, Duri Utara, Tambora, West Jakarta, Special Capital Region of Jakarta, Java, 11270, Indonesia

Warehouse summary



Warehouse

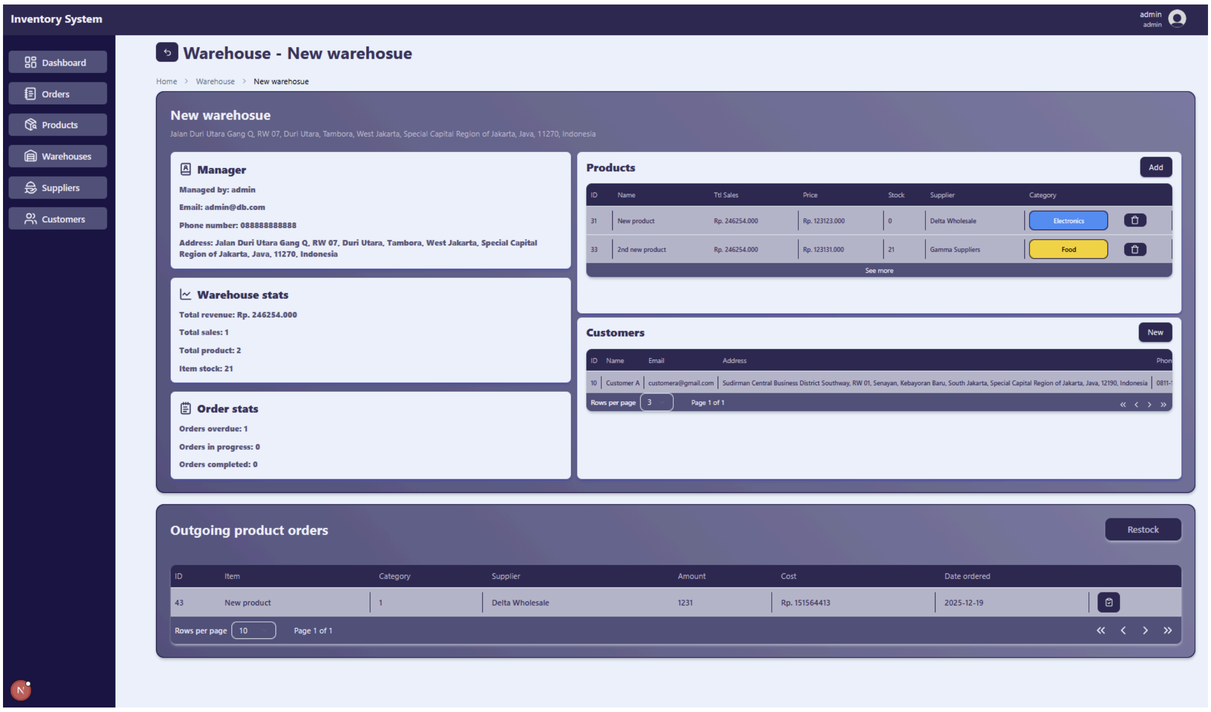
Home > Warehouse

ID	Name	Address	Stock	Orders	
1	Jakarta Warehouse	North Jakarta	0	0	⋮
2	Bandung Warehouse	West Bandung	0	0	⋮
3	Surabaya Warehouse	East Surabaya	0	0	⋮
7	New warehosue	Jalan Duri Utara Gang Q, RW 07, Duri Utara, Tam...	21	1	⋮

Rows per page 10 Page 1 of 1

Specific warehouse page

Accessed by clicking a row in the previous screen, shows a summary of the warehouse (managed by who, how many orders are completed, etc). Managers can add a product to add a new product that is not in the warehouse's inventory, and restock for items which are already in the warehouse inventory. The customer card shows all of the customers which have bought from the warehouse



Warehouse - New warehosue

New warehosue

Jalan Duri Utara Gang Q, RW 07, Duri Utara, Tambora, West Jakarta, Special Capital Region of Jakarta, Java, 11270, Indonesia

Manager

Managed by: admin

Email: admin@db.com

Phone number: 088888888888

Address: Jalan Duri Utara Gang Q, RW 07, Duri Utara, Tambora, West Jakarta, Special Capital Region of Jakarta, Java, 11270, Indonesia

Warehouse stats

Total revenue: Rp. 246254.000

Total sales: 1

Total product: 2

Item stock: 21

Order stats

Orders overdue: 1

Orders in progress: 0

Orders completed: 0

Products

ID	Name	Ttl Sales	Price	Stock	Supplier	Category
31	New product	Rp. 246254.000	Rp. 123123.000	0	Delta Wholesale	Electronics
33	2nd new product	Rp. 246254.000	Rp. 123131.000	21	Gamma Suppliers	Food

Customers

ID	Name	Email	Address	Phon
10	Customer A	customera@gmail.com	Sudirman Central Business District Southway, RW 01, Senayan, Kebayoran Baru, South Jakarta, Special Capital Region of Jakarta, Java, 12190, Indonesia	0811-1

Rows per page 3 Page 1 of 1

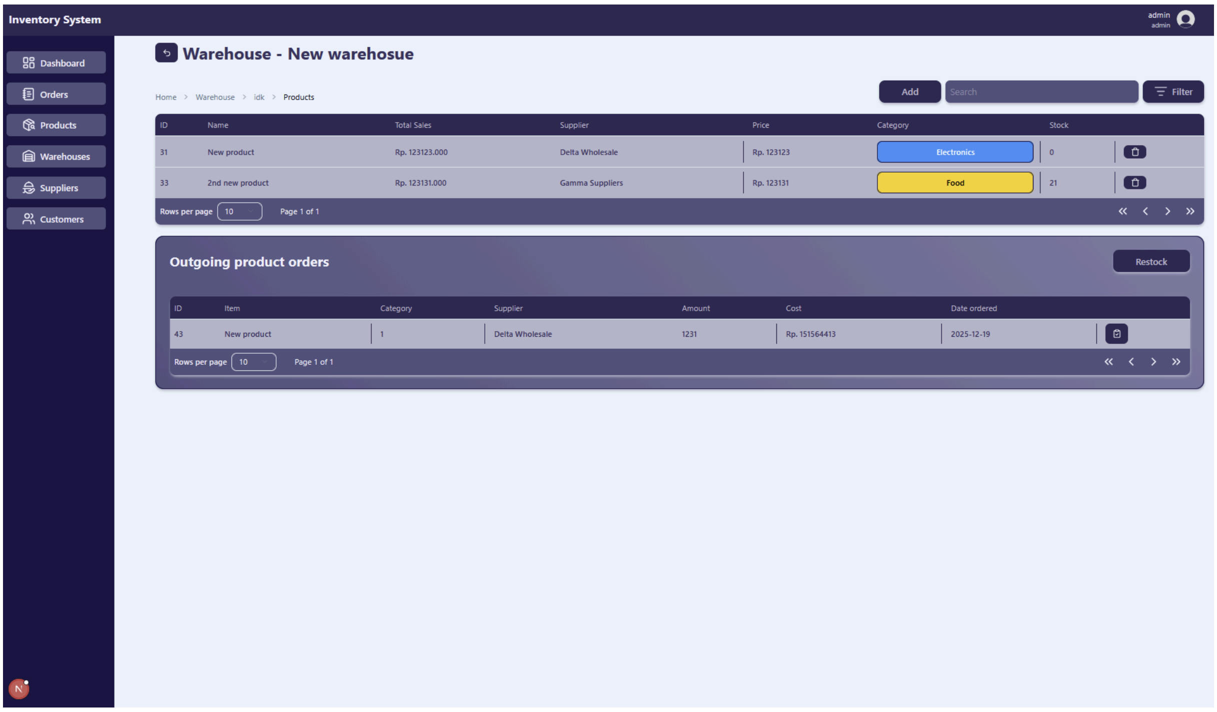
Outgoing product orders

ID	Item	Category	Supplier	Amount	Cost	Date ordered
43	New product	1	Delta Wholesale	1231	Rp. 151564413	2025-12-19

Rows per page 10 Page 1 of 1

Warehouse inventory page

Shows products in a warehouse, reuses the product screen but filtered by warehouse, with support for stock and restocking functions.



ID	Name	Total Sales	Supplier	Price	Category	Stock
31	New product	Rp. 123123.000	Delta Wholesale	Rp. 123123	Electronics	0
33	2nd new product	Rp. 123131.000	Gamma Suppliers	Rp. 123131	Food	21

Rows per page 10 Page 1 of 1

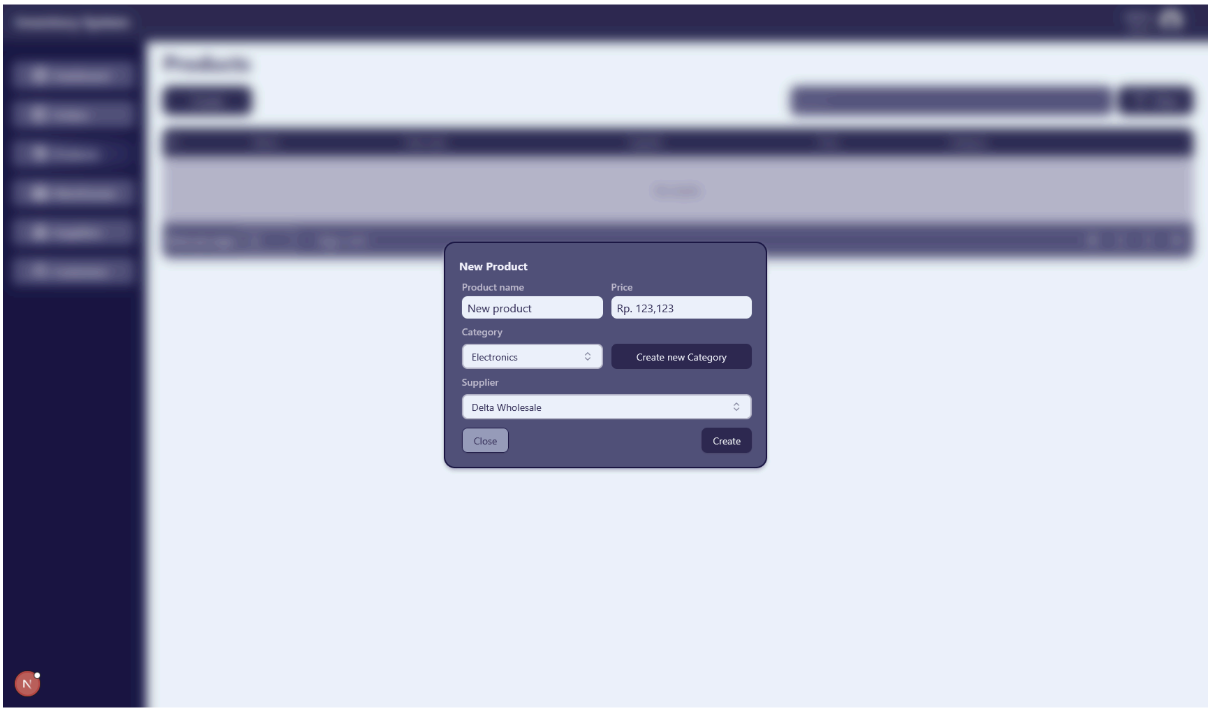
ID	Item	Category	Supplier	Amount	Cost	Date ordered
43	New product	1	Delta Wholesale	1231	Rp. 151564413	2025-12-19

Rows per page 10 Page 1 of 1

Products

Creation/Editing screen

For admins, they can choose what supplier supplies the new product, however, in practice it will be determined from the logged in account



New Product

Product name: New product

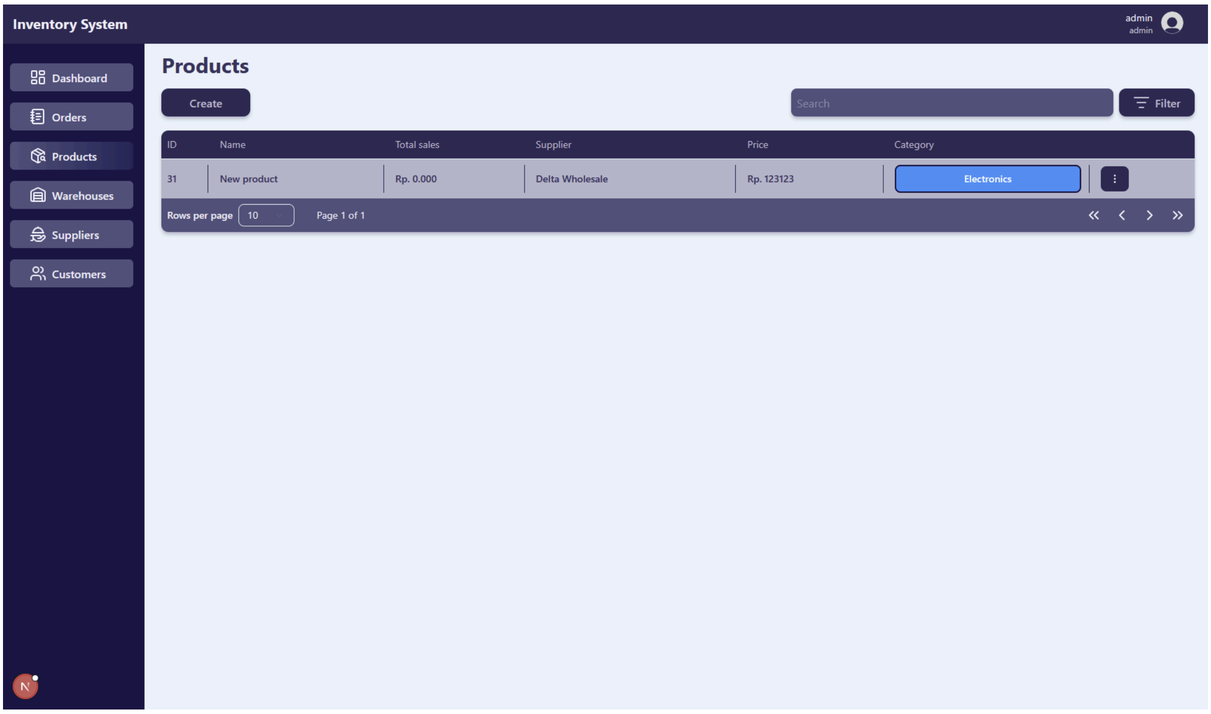
Price: Rp. 123,123

Category: Electronics

Supplier: Delta Wholesale

Product summary page

Search bar supports param search, ?supplier=bgifodgbosdgdfbgosdbgv



Inventory System

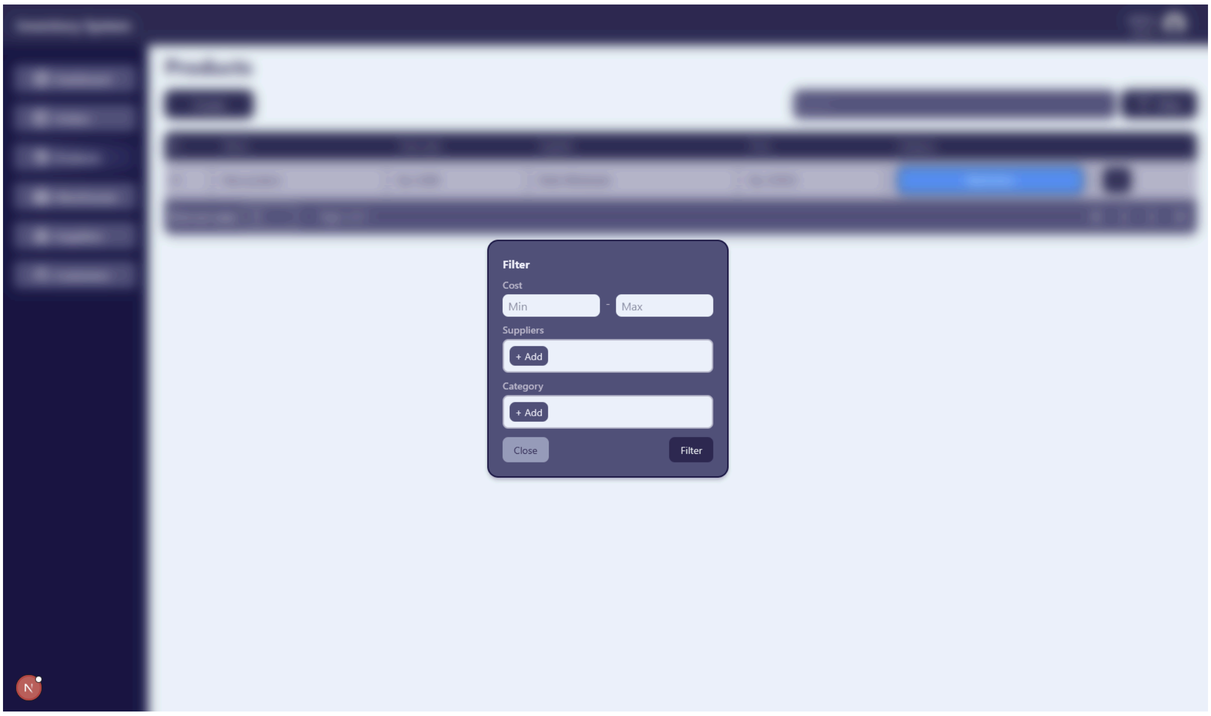
admin
admin

Products

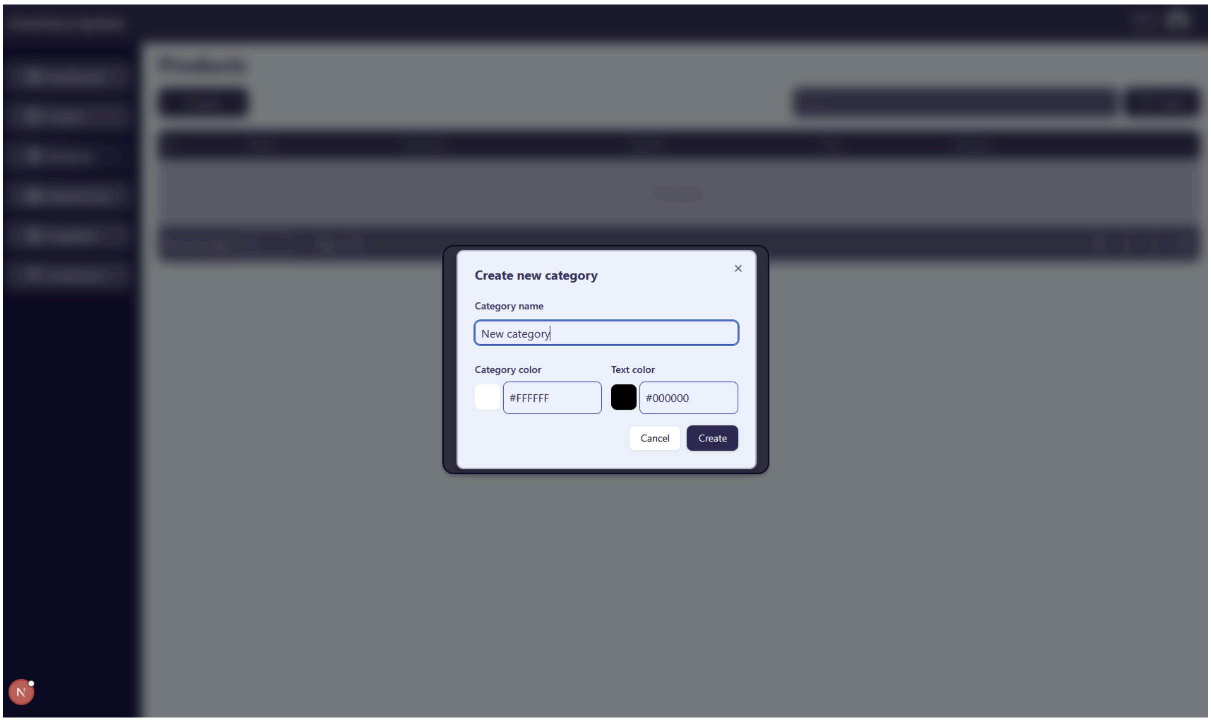
ID	Name	Total sales	Supplier	Price	Category
31	New product	Rp. 0.000	Delta Wholesale	Rp. 123123	Electronics

Rows per page 10   Page 1 of 1

Product filter screen



New category



Create new category

Category name

New category

Category color

#FFFFFF

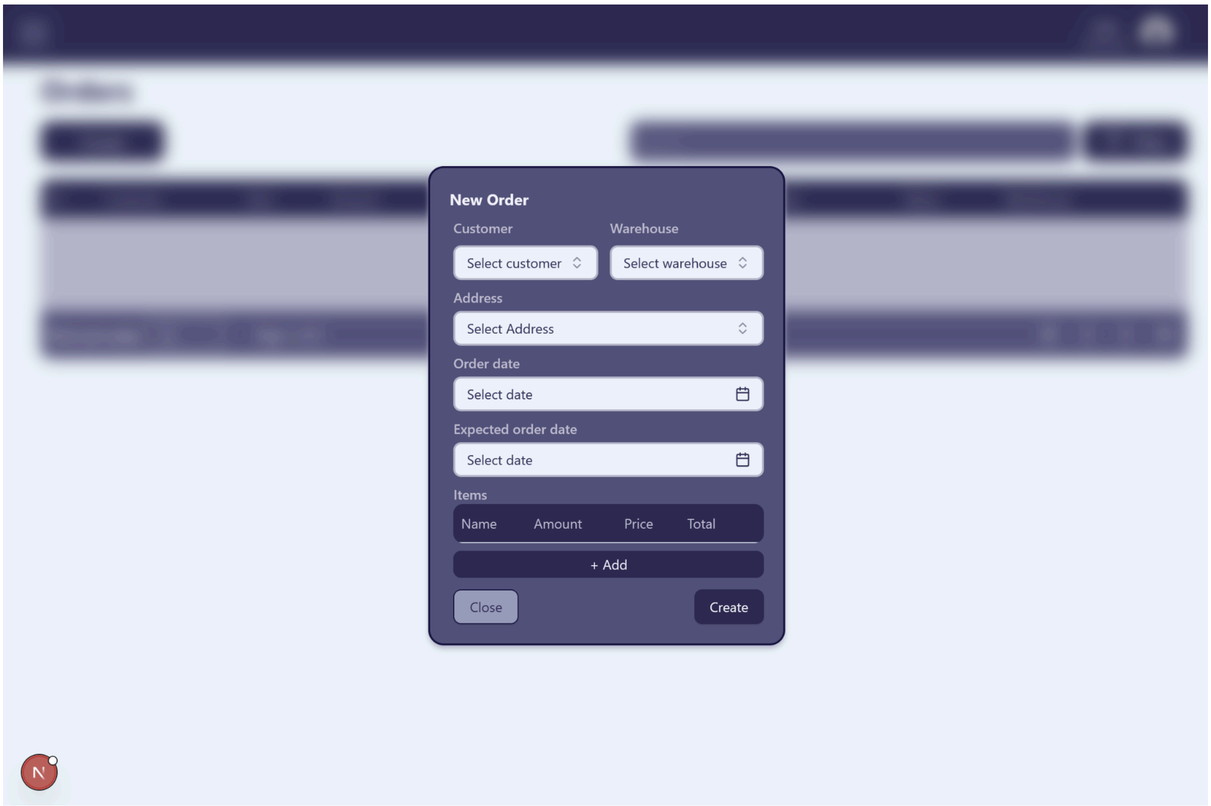
Text color

#000000

Order

Creation/editing screen

The order will take the stock of the product in the warehouse, if there is not enough stock the transaction will not proceed.



New Order

Customer: Select customer

Warehouse: Select warehouse

Address: Select Address

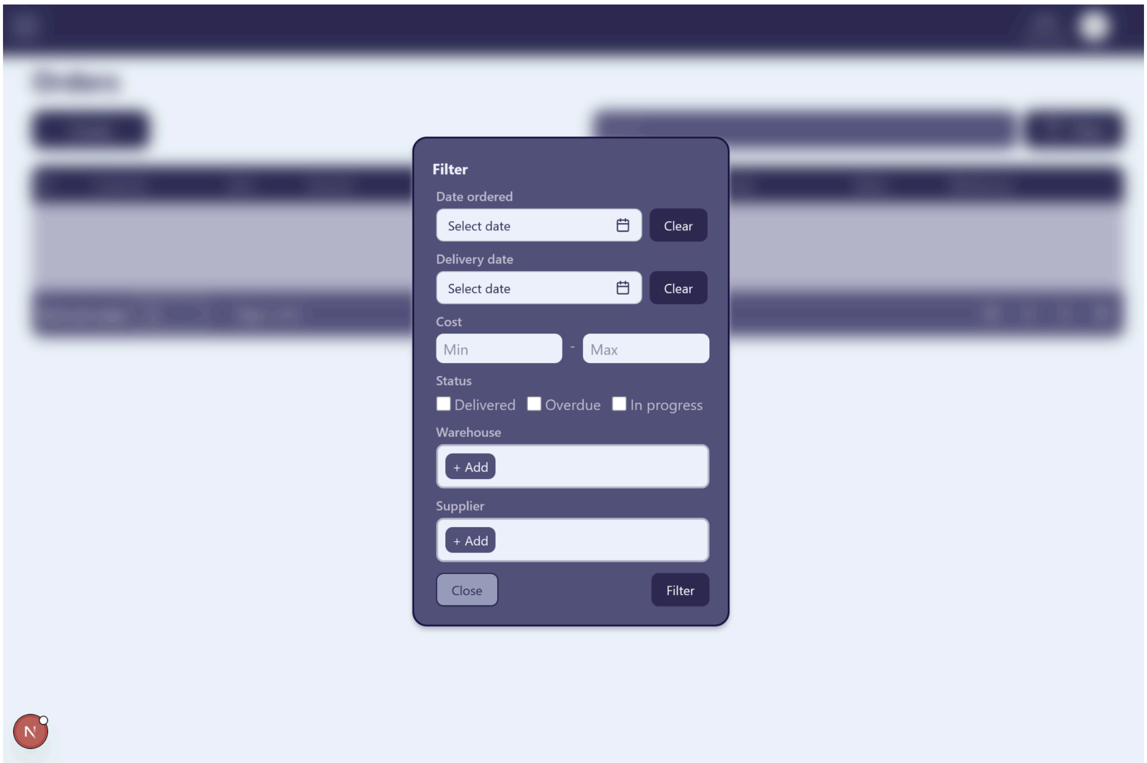
Order date: Select date

Expected order date: Select date

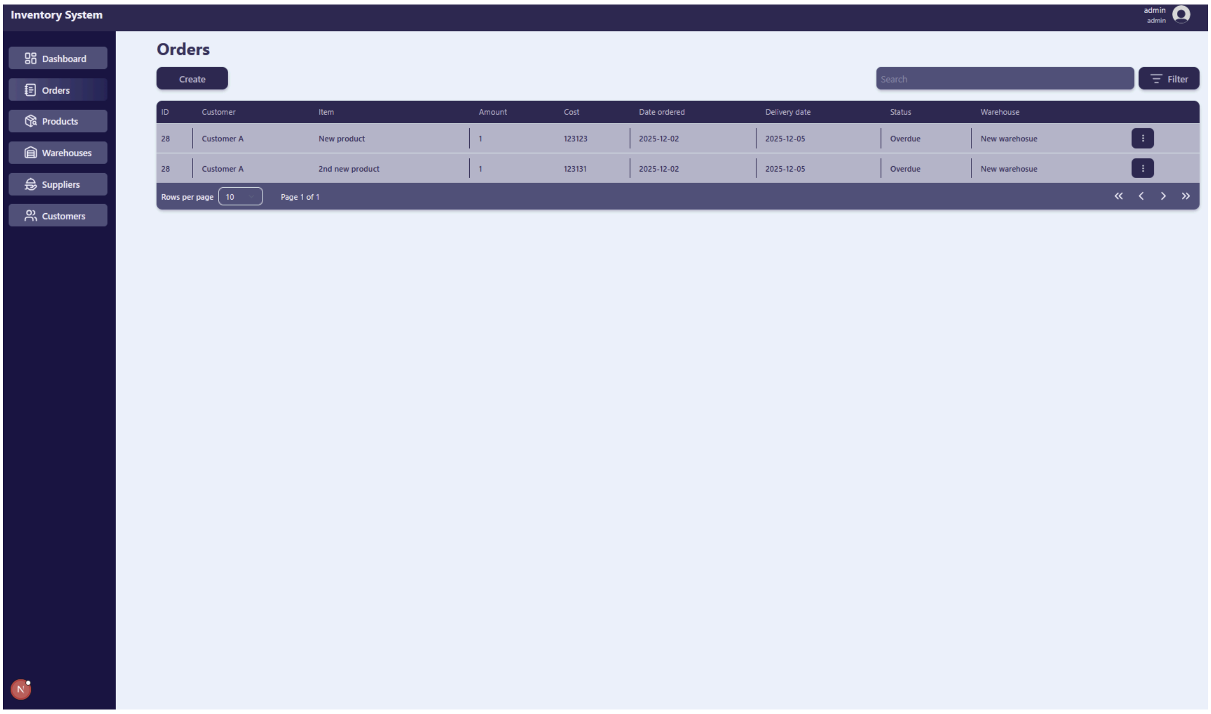
Items

Name	Amount	Price	Total

Filter screen



Summary screen

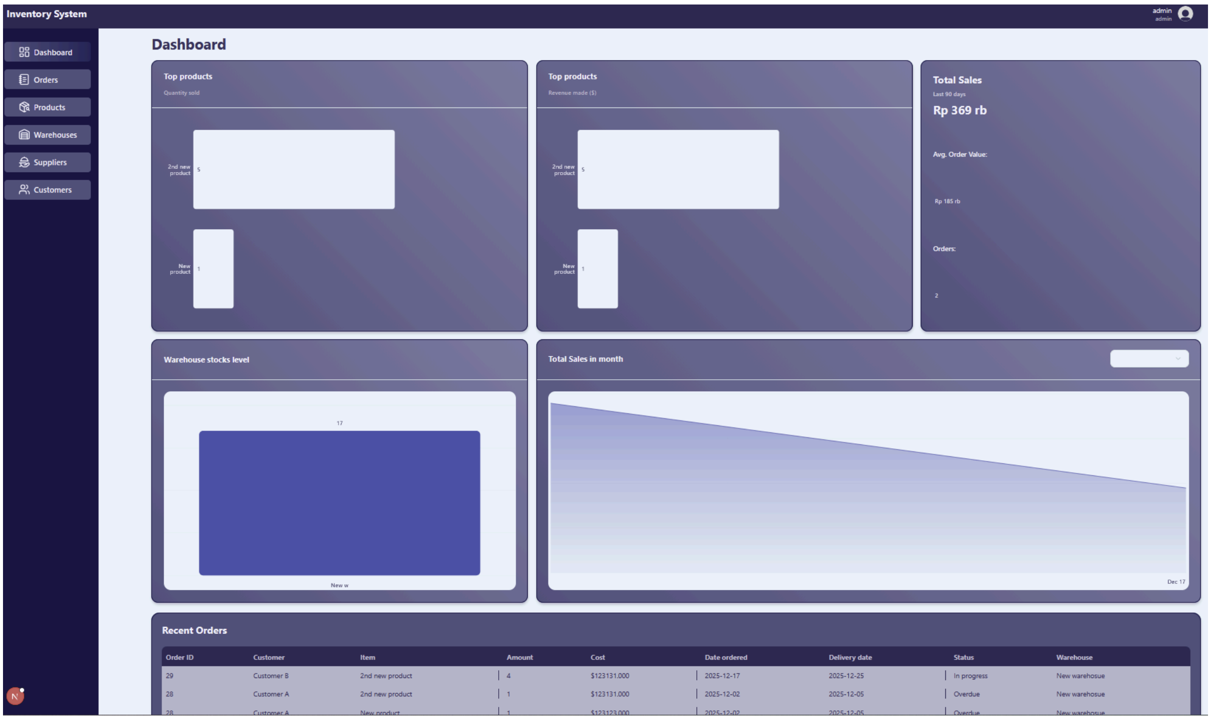


ID	Customer	Item	Amount	Cost	Date ordered	Delivery date	Status	Warehouse	
28	Customer A	New product	1	123123	2025-12-02	2025-12-05	Overdue	New warehosue	⋮
28	Customer A	2nd new product	1	123131	2025-12-02	2025-12-05	Overdue	New warehosue	⋮

Rows per page 10 Page 1 of 1 « ‹ › »

Dashboard

Shows a summary for all data in the inventory system, recent orders, top products that are bought by quantity/revenue, total sales per month, warehouse stock levels. For suppliers in practice it will show a different screen.



Total Sales
Last 90 days
Rp 369 rb

Avg. Order Value:

Rp 185 rb

Orders:

2

Recent Orders

Order ID	Customer	Item	Amount	Cost	Date ordered	Delivery date	Status	Warehouse
29	Customer B	2nd new product	4	$123131.000	2025-12-17	2025-12-25	In progress	New warehosue
28	Customer A	2nd new product	1	$123131.000	2025-12-02	2025-12-05	Overdue	New warehosue
28	Customer A	New product	1	$123123.000	2025-12-02	2025-12-05	Overdue	New warehosue

VII. Conclusion

Achievements

The Inventory and Supplier Management System was successfully designed and implemented to support internal business operations involving multiple warehouses, suppliers, and products. A normalized relational database was created to represent real-world inventory and supply workflows while minimizing redundancy and maintaining data integrity. This system supports core functionalities like inventory tracking, product management, order handling, and supplier coordination.

Through the use of well-defined relationships and transactional tables such as Order and OrderLine, the system is able to handle complex scenarios, including orders with multiple items and historical pricing. Furthermore, aggregate functions were utilized to generate insights such as product performance, sales quantity, revenue contribution, and overall warehouse activity. A functional user interface connected to a real database was developed, fulfilling the requirement for real database interaction.

Challenges

We faced a lot of issues due to overscoping during the early stages of design. There was a lack of clarity regarding what functionalities truly belonged in an inventory and supplier management system. As a result, we attempted to include too many features and entities, which made the database more complex than necessary and harder to manage.

Additionally, we had trouble understanding and applying Second Normal Form (2NF). First Normal Form was pretty straightforward, but identifying partial dependencies – especially for tables with composite keys like OrderLine and Inventory – proved to be more difficult than we initially imagined. Determining which attributes depended on the entire primary key required multiple revisions and often caused confusion among the team during the normalization process. Furthermore, there are improvements that could have been made such as using EERD specialization for the account roles, however we thought that it was not necessary for this project as there are no extra attributes that are needed for each specific role, and it would only introduce more complexity.

Thus, these challenges forced our team to repeatedly revisit the system scope, reassess assumptions, and refine the database design to better reflect how real-world inventory and supplier systems operate.

Lessons Learned

This project highlights the importance of a strong database design as the foundation of any data-driven system. A well-organized and normalized database not only ensures data integrity but also makes the system easier to maintain and extend in the future. Proper design decisions, like defining clear entity relationships, primary and foreign keys, and normalization levels, significantly influence the scalability and flexibility of the system. For example, role-specific tables can be created using specialization and generalization without introducing data inconsistency and structural errors.

We also learned that fully understanding the real-world system being modeled before we try to design the database is crucial. Our lack of understanding of what an actual inventory and supplier management system entails led to overscoping and unnecessary complexity. Refining the system scope could have helped ensure that the database focused on the essential operations and better reflected real-world workflows.

Through this project, we also personally learnt how important normalization is. We had struggled to figure out partial dependencies and applying 2NF correctly. But in the end, it improved our understanding of functional dependencies as well as truly highlighted how incorrect normalization can affect the data's consistency and long-term maintainability.

This project showed several limitations imposed by time and system complexity when implementing advanced features. We had planned for the restock function to be able to order multiple products within a single transaction. In real-world scenarios, it would be inefficient to place separate restock orders for each item. But we failed to implement it since restock was a last-minute feature, and doing that would require redesigning the entire restock structure to support multiple products per stock. Due to time constraints, this enhancement was not pursued further.

Our lecturer had also suggested a feature we had overlooked: automatic restocking. This feature would allow the system to generate restock orders automatically when inventory levels fall below a configurable threshold set by warehouse managers, ensuring timely replenishment for high-demand products.

In summary, this project strengthened our practical understanding of relational database design and emphasized the importance of careful planning, realistic scoping, and iterative refinement. While not all intended features were implemented, the system provides a solid database foundation that can support future enhancements and more advanced functionality.

VIII. Appendix

DDL

```
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL
);
CREATE TABLE Address (
    address_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    delivery_address VARCHAR(255) NOT NULL,
    phone_num CHAR(15) NOT NULL,
        FOREIGN KEY (customer_id) REFERENCES Customer(customer_id) ON DELETE CASCADE
);
CREATE TABLE Account (
    account_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone_number CHAR(15) NOT NULL,
    password VARCHAR(255) NOT NULL,
    account_type ENUM('supplier', 'manager', 'admin')
);
CREATE TABLE Category (
    category_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    text_color CHAR(7) DEFAULT('#000000'),
    bg_color CHAR(7) DEFAULT('#FFFFFF')
);
```

```
CREATE TABLE Product (
   product_id INT PRIMARY KEY AUTO_INCREMENT,
   account_id INT NOT NULL,
   category_id INT DEFAULT(0),
   product_name VARCHAR(255) NOT NULL,
   price DECIMAL(15,3) NOT NULL,
   CONSTRAINT not_negative CHECK (price>=0),
      FOREIGN KEY (account_id) REFERENCES Account(account_id) ON DELETE
CASCADE,
   FOREIGN KEY (category_id) REFERENCES Category(category_id)
);
CREATE TABLE Warehouse (
   warehouse_id INT PRIMARY KEY AUTO_INCREMENT,
   account_id INT NOT NULL,
   name VARCHAR(255) NOT NULL,
   address VARCHAR(255) NOT NULL,
   FOREIGN KEY (account_id) REFERENCES Account(account_id)
);
CREATE TABLE Inventory (
   warehouse_id INT NOT NULL,
   product_id INT NOT NULL,
   stock INT DEFAULT(0),
   CONSTRAINT not_negative CHECK (stock>=0),
   PRIMARY KEY (warehouse_id, product_id),
    FOREIGN KEY (warehouse_id) REFERENCES Warehouse(warehouse_id) ON DELETE
CASCADE,
      FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE
);
```

```
CREATE TABLE `Order` (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    address_id INT NOT NULL,
    warehouse_id INT NOT NULL,
    order_date DATE NOT NULL,
    expected_delivery_date DATE NOT NULL,
    is_delivered BOOLEAN DEFAULT(FALSE),
        FOREIGN KEY (address_id) REFERENCES Address(address_id) ON DELETE CASCADE,
     FOREIGN KEY (warehouse_id) REFERENCES Warehouse(warehouse_id) ON DELETE CASCADE
);
CREATE TABLE OrderLine (
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    amount INT NOT NULL,
    order_price DECIMAL(15,3) NOT NULL,
    CONSTRAINT not_negative CHECK (amount>=0 AND order_price>=0),
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES `Order`(order_id) ON DELETE CASCADE,
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

```
CREATE TABLE Restock(
   restock_id INT PRIMARY KEY AUTO_INCREMENT,
   product_id INT NOT NULL,
   warehouse_id INT NOT NULL,
   order_date DATE NOT NULL,
   cost DECIMAL(15,3) NOT NULL,
   amount INT NOT NULL,
   CONSTRAINT not_negative CHECK (amount>=0 AND cost>=0),
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY (warehouse_id) REFERENCES Warehouse(warehouse_id) ON DELETE
CASCADE
);
```

DML/Queries

* Some needs consecutive SQL and some processes outside SQL, these processes is written in pseudo code

Inserts

- Register account

```
INSERT INTO Account (name,email,phone_number,password,account_type)
VALUES (?,?,?,?,?)
```

- Create warehouse

```
INSERT INTO Warehouse (name, address, account_id)
VALUES (?, ?, ?)
```

- Create product

```
INSERT INTO Product (product_name, price, category_id, account_id)
VALUES (?, ?, ?, ?)
```

- Create customer

```
INSERT INTO Customer (name, email) VALUES (?, ?)
```

```
For each address in addresses:
INSERT INTO Address (customer_id, delivery_address, phone_num)
VALUES (?, ?, ?)
```

- Create category

```
INSERT INTO Category (name, bg_color, text_color) VALUES (?, ?, ?)
```

- Create restock order

```
INSERT INTO Restock(product_id,warehouse_id,amount,cost,order_date)
VALUES (?,?,?,?,?)
```

- Create order

```
BEGIN TRANSACTION;
INSERT INTO `Order` (address_id, warehouse_id, order_date, expected_delivery_date,
is_delivered)
VALUES (?, ?, ?, ?, FALSE);

For each item in items:
     SELECT stock
    FROM Inventory
    WHERE product_id = ? AND warehouse_id = ?;

     If !(stock > amount): ROLLBACK;

     UPDATE Inventory
     SET stock = stock - ?
     WHERE product_id = ? AND warehouse_id = ?;

     INSERT INTO OrderLine
     (order_id, product_id, amount, order_price)
     VALUES (?, ?, ?, ?);

COMMIT;
```

Selects

- Select warehouse summary

```
SELECT
   w.warehouse_id,
   w.name,
   w.address,
   inv.total_stock AS total_stock,
   ord.total_orders AS total_orders
FROM Warehouse w
JOIN Account a ON w.account_id = a.account_id
LEFT JOIN (
   SELECT warehouse_id, SUM(stock) AS total_stock
   FROM Inventory
   GROUP BY warehouse_id
) inv ON w.warehouse_id = inv.warehouse_id
LEFT JOIN (
   SELECT warehouse_id, COUNT(order_id) AS total_orders
   FROM `Order`
   GROUP BY warehouse_id
) ord ON w.warehouse_id = ord.warehouse_id;
```

- Select suppliers

```
SELECT account_id as id, name, address, email, phone_number
FROM Account
WHERE account_type = 'supplier'
```

- Select restock summary

```
SELECT r.restock_id, p.product_name, a.name, r.amount, r.cost, r.order_date,
p.category_id
         FROM Restock r
         JOIN Product p ON r.product_id = p.product_id
         JOIN Account a ON p.account_id = a.account_id
         WHERE r.warehouse_id = ?
```

- Login user

```
SELECT account_id, name, password, account_type FROM Account WHERE email = ?
```

- Select orders

```
SELECT
   o.order_id,  p.product_name AS item,
   ol.amount,
   ol.order_price AS cost,
   c.name AS customer_name,
   a.delivery_address,
   o.order_date,
   o.expected_delivery_date,
   o.is_delivered,
   w.name AS warehouse_name,
   c.customer_id,
   w.warehouse_id,
   a.address_id,
   ol.product_id
FROM `Order` o
JOIN Address a ON o.address_id = a.address_id
JOIN Customer c ON a.customer_id = c.customer_id
JOIN OrderLine ol ON o.order_id = ol.order_id
JOIN Product p ON ol.product_id = p.product_id
JOIN Warehouse w ON o.warehouse_id = w.warehouse_id
ORDER BY o.order_id DESC
```

- Select product summary

```
          SELECT p.product_id, p.product_name, p.price, p.category_id, a.name AS supplier_name, a.email AS supplier_email, SUM(ol.amount * ol.order_price) AS total_sales, a.account_id as account_id
     FROM Product p
     LEFT JOIN Account a ON p.account_id = a.account_id
     LEFT JOIN OrderLine ol ON p.product_id = ol.product_id
     GROUP BY p.product_id, p.product_name, p.price, p.category_id, a.name, a.email
```

- Get all categories

```
     SELECT category_id, name, bg_color, text_color FROM Category
```

*** For search and filter functionality it is generally the same with extra WHERE clauses for example (product)**

- Search product

```
sql=  SELECT p.product_id, p.product_name, p.price, p.category_id, a.name AS supplier_name, SUM(ol.amount * ol.order_price AS total_sales, a.account_id as account_id
     FROM Product p
     LEFT JOIN Account a ON p.account_id = a.account_id
     LEFT JOIN OrderLine ol ON p.product_id = ol.product_id
     WHERE 1 = 1

// WHERE 1=1 is needed to add extra AND conditions
   if name:
     query += " AND p.product_name LIKE ?"
   if supplier:
     query += " AND a.name LIKE ?"

   query += """
     GROUP BY p.product_id, p.product_name, p.price, p.category_id, a.name
   """
```

- Filter

```
    SELECT
        p.product_id,
        p.product_name,
        p.price,
        p.category_id,
        a.name AS supplier_name,
        a.email AS supplier_email,
        SUM(ol.amount * ol.order_price) AS total_sales,
        a.account_id AS account_id
    FROM Product p
    LEFT JOIN Account a ON p.account_id = a.account_id
    LEFT JOIN OrderLine ol ON p.product_id = ol.product_id
    WHERE p.price BETWEEN ? AND ?
```

// In the code, there is a default parameter for max price and min price, hence no1=1

```
  if suppliers:
    query += " AND a.account_id IN (" + ", ".join(["?"] * len(suppliers)) + ")"
    params.extend(suppliers)

  if category_id:
    query += " AND p.category_id IN (" + ", ".join(["?"] * len(category_id)) + ")"
    params.extend(category_id)

  query += """
    GROUP BY
        p.product_id,
        p.product_name,
        p.price,
        p.category_id,
        a.name,
        a.email,
        a.account_id
  """
```

- Dashboard total sales

```
SELECT
   SUM(ol.order_price) AS total_sales,
   COUNT(DISTINCT o.order_id) AS orders
FROM OrderLine ol
JOIN `Order` o ON ol.order_id = o.order_id
WHERE o.order_date >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
```

- Dashboard total sales per date

```
SELECT
   o.order_date,
   SUM(ol.order_price) AS daily_sales
FROM OrderLine ol
JOIN `Order` o ON ol.order_id = o.order_id
WHERE o.order_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
GROUP BY o.order_date
ORDER BY o.order_date
```

- Dashboard warehouse stocks summary

```
SELECT
   w.name,
   SUM(i.stock) AS total_stock
FROM Warehouse w
JOIN Inventory i ON w.warehouse_id = i.warehouse_id
GROUP BY w.warehouse_id
ORDER BY total_stock DESC
LIMIT 5
```

- Dashboard top products by quantity

```
SELECT
   p.product_name,
   SUM(ol.amount) AS total_qty
FROM OrderLine ol
JOIN Product p ON ol.product_id = p.product_id
GROUP BY ol.product_id
ORDER BY total_qty DESC
LIMIT 5
```

- Warehouse customers

```
SELECT c.customer_id, c.name, c.email,
     a.address_id, a.delivery_address, a.phone_num
FROM Customer c
LEFT JOIN Address a ON c.customer_id = a.customer_id
WHERE c.customer_id IN (
          SELECT a.customer_id FROM `Order` o JOIN Address a ON o.address_id = a.address_id
        WHERE warehouse_id = ?
        )
```

- Warehouse manager info

```
  SELECT w.warehouse_id, w.name, w.address, a.name as manager_name, a.email as manager_email, a.phone_number as phone_num
FROM Warehouse w
LEFT JOIN Account a ON w.account_id = a.account_id
WHERE w.warehouse_id = ?;
```

- Warehouse products

```
SELECT
   i.product_id,
   p.product_name,
   p.category_id,
   a.name AS supplier,
   p.price,
   i.stock,
   SUM(ol.order_price) AS ttl_sales
FROM Inventory i
JOIN Product p ON i.product_id = p.product_id
JOIN Warehouse w ON i.warehouse_id = w.warehouse_id
LEFT JOIN `Order` o ON o.warehouse_id = w.warehouse_id
LEFT JOIN OrderLine ol ON o.order_id = ol.order_id
JOIN Account a ON p.account_id = a.account_id
WHERE i.warehouse_id = ?
GROUP BY i.product_id, p.product_name, p.category_id, a.name, p.price, i.stock;
```

Updates

- Update customer

```
BEGIN TRANSACTION;
UPDATE Customer SET name=?, email=? WHERE customer_id=?;
DELETE FROM Address WHERE customer_id=?;

For each addresses:
    INSERT INTO Address (customer_id, delivery_address, phone_num)
    VALUES (?, ?, ?);
COMMIT;
```

- Update order

```
BEGIN TRANSACTION;
UPDATE `Order`
SET address_id = ?,
   warehouse_id = ?,
   order_date = ?,
   expected_delivery_date = ?,
   is_delivered = FALSE
WHERE order_id = ?;
DELETE FROM OrderLine WHERE order_id = ?;

For each items:
    INSERT INTO OrderLine (order_id, product_id, amount, order_price)
   VALUES (?, ?, ?, ?);
COMMIT;
```

- Update product

```
UPDATE Product
SET product_name = ?, price = ?, category_id = ?, account_id = ?
WHERE product_id = ?
```

- Update warehouse

```
UPDATE Warehouse
SET name=?, address=?
WHERE warehouse_id=?
```

Deletes

- Complete restock order

```
SELECT product_id, warehouse_id, amount, cost
FROM Restock
WHERE restock_id = ?;

SELECT stock
FROM Inventory
WHERE product_id = ? AND warehouse_id = ?;

If returns a row:
    SELECT stock
    FROM Inventory
    WHERE product_id = ? AND warehouse_id = ?      ;
Else:
    INSERT INTO Inventory (product_id, warehouse_id, stock)
    VALUES (?, ?, ?);

DELETE FROM Restock WHERE restock_id = ?;
```

- Delete warehouse

```
DELETE FROM Inventory WHERE warehouse_id=? AND product_id=?;
```

- Delete customers

```
DELETE FROM Customer WHERE customer_id=?
```

- Delete order (the flow from the UI is that each orderline will be deleted first)

```
BEGIN TRANSACTION;
DELETE FROM `Order` WHERE order_id = ?;
SELECT COUNT(*) FROM OrderLine WHERE order_id = ?;

If count == 0:
    DELETE FROM `Order` WHERE order_id = ?;
COMMIT;
```

- Delete product

```
DELETE FROM Product WHERE product_id = ?;
```

Github

https://github.com/osten-antonio/Database-FP/tree/main