**Supply Chain Management System**

**Abstract :**

The Supply Chain Management (SCM) System is an integrated framework that optimizes the flow of goods, information, and finances across the supply chain. It encompasses various components including suppliers, products, orders, shipments, warehouses, inventory, customers, invoices, deliveries, and returns.

\*\*Suppliers\*\* provide the essential raw materials or components required for production, ensuring quality and cost-effectiveness. \*\*Products\*\* are the final or intermediate goods that are manufactured or assembled. \*\*Orders\*\* represent the customer requests for these products, which are managed from placement to fulfillment.

\*\*Shipments\*\* involve the transportation of goods from suppliers to manufacturing plants, then to warehouses, and finally to customers. \*\*Warehouses\*\* store these goods, with effective management ensuring the correct products are available as needed. \*\*Inventory\*\* management maintains the optimal levels of stock, balancing supply and demand.

\*\*Customers\*\* are the end-users who purchase the products, and their satisfaction is critical. \*\*Invoices\*\* are generated to bill customers, managing the financial transactions. \*\*Deliveries\*\* ensure that products reach customers timely and in good condition, while the \*\*returns\*\* process handles any products that need to be sent back due to defects or dissatisfaction.

By integrating these components, the SCM system enhances efficiency, reduces costs, improves customer satisfaction, and provides flexibility to adapt to changes in demand and supply. Effective SCM is pivotal for the competitive advantage and operational success of any business.

**Introduction :**

Supply Chain Management (SCM) is the centralized management of the flow of goods and services to and from a company. It encompasses all processes involved in transforming raw materials and components into final products, ensuring their delivery to the ultimate customer.

Key aspects of SCM include:

**-Inventory Management:**

Tracking and managing inventory levels to maintain optimal stock levels.

Reducing stockouts (depleted inventory) and overstock situations (excessive inventory) to prevent financial losses.

**-Order Management:**

Managing orders from creation to fulfillment.

Efficient order processing, tracking, and timely delivery to customers.

**-Supplier Management:**

Managing relationships with suppliers.

Sourcing suppliers, procurement (purchasing goods and services), and tracking supplier performance.

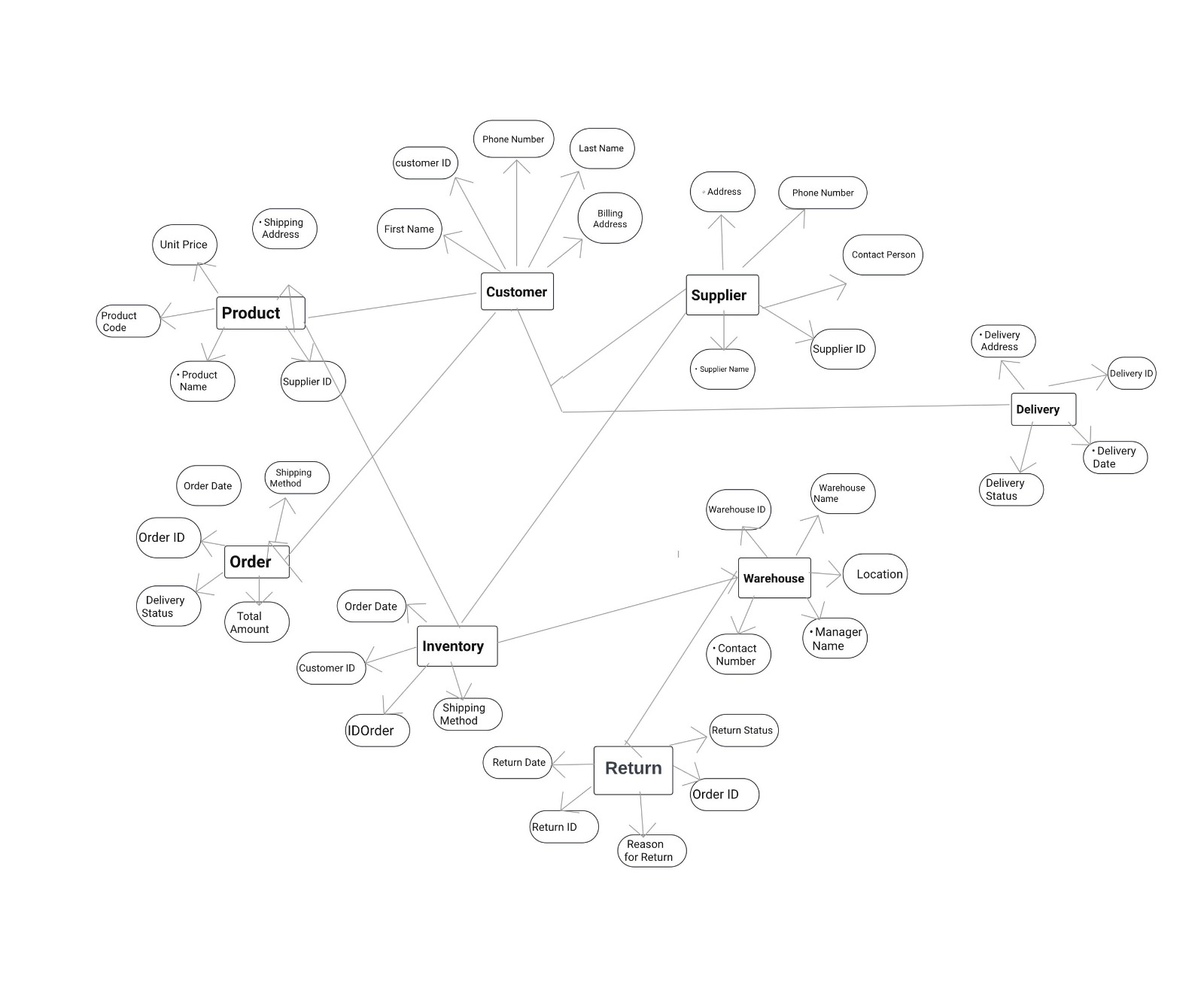
**-Logistics Management:**

Planning, executing, and tracking the movement of goods and materials within the supply chain.

**-Demand Planning:**

Forecasting product demand to optimize inventory levels and production planning.

**ER Diagram :**



FUNCTIONAL REQUIREMENT :

Functional requirements in Unified Modeling Language (UML) refer to the specifications or descriptions of the system's functions or capabilities. UML provides various diagrams to represent these requirements visually. Here are some common ways functional requirements are depicted in UML:

• **Use Case Diagrams**: Use case diagrams in UML represent the functional requirements of a system from the user's perspective. They show the interactions between actors (users or external systems) and the system itself. Each use case represents a specific functionality or action that the system performs in response to an actor's request.

• **Activity Diagrams**: Activity diagrams in UML depict the flow of activities or actions within a system. They can be used to model the workflow of a particular use case or business process, showing the sequence of actions, decision points, and branching logic.

• **Sequence Diagrams**: Sequence diagrams illustrate how objects interact in a particular scenario or use case over time. They show the sequence of messages exchanged between objects and the order of execution of these messages. Sequence diagrams can be used to model the behavior of a system in response to user interactions or external events.

• **State Machine Diagrams**: State machine diagrams represent the behavior of a system or object in response to internal or external events. They show the states that an object can be in and the transitions between these states based on events or conditions. State machine diagrams are particularly useful for modeling the lifecycle of objects or the behavior of complex systems with multiple states.

• **Class Diagrams**: While class diagrams primarily depict the static structure of a system, they can also capture some aspects of functional requirements. By showing the relationships between classes, attributes, and methods, class diagrams can help clarify how different parts of the system collaborate to fulfill specific functionalities.

These are some of the main ways UML can be used to represent functional requirements. Each type of diagram provides a different perspective on the system's behavior and functionality, allowing stakeholders to better understand and communicate the requirements.

|  |
| --- |
| Product |
| Attributes:  - product\_id: int  - name: string  - description: string  - price: float  - supplier\_id: int |
| Operations:  + getProductId(): int  + getName(): string  + getDescription(): string  + getPrice(): float  + getSupplierId(): int  + setProductId(id: int): void  + setName(name: string): void  + setDescription(desc: string): void  + setPrice(price: float): void  + setSupplierId(id: int): void  + toString(): string |

**UML Diagram :**

|  |
| --- |
| Name: Supplier |
| Attribute:  - supplier\_id: int  - name: string  - contact\_info: string |
| Operations :  + getSupplierId(): int  + getName(): string  + getContactInfo(): string  + setSupplierId(id: int): void  + setName(name: string): void  + setContactInfo(info: string): void  + toString(): string |

|  |
| --- |
| Order |
| Attributes:  - order\_id: int  - date: int  - customer\_id: int  - status: string |
| Operations:  + getOrderId(): int  + getDate(): int  + getCustomerId(): in|  + getStatus(): string  + setOrderId(id: int): void  + setDate(date: int): void  + setCustomerId(id: int): void  + setStatus(status: string): void  + toString(): string |

|  |
| --- |
| Shipment |
| Attributes:  - shipment\_id: int  - order\_id: int  - shipment\_date: int  - delivery\_date: int  - status: string |
| Operations:  + getShipmentId(): int  + getOrderId(): int  + getShipmentDate(): int  + getDeliveryDate(): int  + getStatus(): string  + setShipmentId(id: int): void  + setOrderId(id: int): void  + setShipmentDate(date: int): void  + setDeliveryDate(date: int): void  + setStatus(status: string): void  + toString(): string |

|  |
| --- |
| Warehouse |
| Attributes:  - warehouse\_id: int  - location: string  - capacity: int |
| Operations:  + getWarehouseId(): int  + getLocation(): string  + getCapacity(): int  + setWarehouseId(id: int): void  + setLocation(loc: string): void  + setCapacity(cap: int): void  + toString(): string |

|  |
| --- |
| Inventory |
| Attributes:  - inventory\_id: int  - product\_id: int  - warehouse\_id: int  - quantity: int |
| + getInventoryId(): int  + getProductId(): int  + getWarehouseId(): in|  + getQuantity(): int  + setInventoryId(id: int): void  + setProductId(id: int): void  + setWarehouseId(id: int): void  + setQuantity(qty: int): void  + toString(): string |

|  |
| --- |
| Customer |
| Attributes:  - customer\_id: int  - name: string  - contact\_info: string |
| Operations:  + getCustomerId(): int  + getName(): string  + getContactInfo(): string  + setCustomerId(id: int): void  + setName(name: string): void  + setContactInfo(info: string): void  + toString(): string |

|  |
| --- |
| Invoice |
| Attributes:  - invoice\_id: int  - order\_id: int  - amount: float  - date: int |
| Operations:  + getInvoiceId(): int  + getOrderId(): int  + getAmount(): float  + getDate(): int  + setInvoiceId(id: int): void  + setOrderId(id: int): void  + setAmount(amount: float): void  + setDate(date: int): void  + toString(): string |

|  |
| --- |
| Delivery |
| Attributes:  - delivery\_id: int  - shipment\_id: int  - delivery\_date: int  - status: string |
| + getDeliveryId(): int  + getShipmentId(): int  + getDeliveryDate(): int  + getStatus(): string  + setDeliveryId(id: int): void  + setShipmentId(id: int): void  + setDeliveryDate(date: int): void  + setStatus(status: string): void  + toString(): string |

|  |
| --- |
| Return |
| Attributes:  - return\_id: int  - order\_id: int  - product\_id: int  - date: int  - reason: string |
| + getReturnId(): int  + getOrderId(): int  + getProductId(): int  + getDate(): int  + getReason(): string  + setReturnId(id: int): void  + setOrderId(id: int): void  + setProductId(id: int): void  + setDate(date: int): void  + setReason(reason: string): void|  + toString(): string |

**CHALLENGES LIST :**

**Challenges Faced During Project Completion**

Completing a project involves overcoming a variety of challenges at each stage. Initially, embarking on this project without a clear idea or overview was particularly daunting. Here’s how we navigated the hurdles:

1. **MySQL Workbench Learning Curve**:

Initially, we encountered difficulties while using MySQL Workbench. However, through persistence and learning, we managed to figure out the intricacies and successfully set up our database.

1. **Syntax and Rushed Data Entry**:

While inserting and creating records, we found ourselves rushing. Spending time understanding the syntax was crucial to avoid errors.

Beyond data insertion, we also crafted ER diagrams and UML diagrams. These visual representations helped us conceptualize the project better.

1. **Java Code Challenges**:

Writing Java code presented its own set of challenges. Syntax errors and unexpected behavior forced us to dive deeper into debugging and refining our code.

1. **Git Confusion**:

Git, with its myriad commands and workflows, proved confusing. We had to learn the ropes of version control, branching, and merging.

Pulling and pushing data required careful attention to avoid conflicts.

1. **Project Upload Complexity**:

Uploading the project seemed straightforward, but managing data flow through pull and push operations turned out to be quite a task.

In hindsight, we realized the importance of planning ahead—having a clear list of tasks and a project overview would have eased implementation.