DEPARTMENT OF UNIVERSITY

INSTITUTE OF COMPUTING

CHANDIGARH UNIVERSITY

FINAL PROJECT REPORT

Title: MANAGING A COMPANY'S INVENTORY

Submitted by:
Name: Devansh Sood
UID: 24BCA10066
Section: 24BCA-1(B)

Submitted to:
Name: Mr. Jagwinder Singh
Subject: Database management system
Subject Code: 24CAP-204

ACKNOWLEDGMENT

I am deeply grateful to **Chandigarh University** for providing me with the opportunity and resources to successfully complete this project. The institution's supportive academic environment and emphasis on practical learning have played a vital role in enhancing my technical and analytical skills.

I would like to express my sincere gratitude to **Mr. Jagwinder Singh**, my mentor and guide, for his valuable support, expert guidance, and continuous encouragement throughout the development of this project. His insights and constructive feedback helped me overcome challenges and gain a deeper understanding of the subject.

Lastly, I extend my appreciation to all faculty members, classmates, and everyone who directly or indirectly contributed to the successful completion of this project.

TABLE OF CONTENTS

1 Introduction

Introduction to the project
Problem Definition & Solution Overview

2 Literature Review / Background Study

Review Summary
Goals and Objectives

3 Setup and Design

Methodologies, Tools, and Technologies
Hardware and Software Specifications

4 Implementation and Result

5 Future Scope

6 Conclusion

Introduction

Design an ER diagram for managing a company's inventory. Products are stored in warehouses. Each product has details like name, quantity, and supplier. The system must track stock levels, purchase orders, and supplier details.

1) Introduction to the Project

Inventory management is one of the most critical operations in any company that deals with products, suppliers, and warehouses. The efficiency of this system determines how well an organization maintains stock levels, avoids shortages, and prevents overstocking.
This project focuses on designing and implementing an Inventory Management System that maintains information about products, suppliers, warehouses, and purchase orders. The system provides a structured way to monitor product availability, manage supplier relationships, and track stock movement in real-time, ensuring smooth business operations.

2) Problem Definition & Solution Overview

Problem:
Many organizations still rely on manual or semi-automated systems to manage inventory. This often leads to issues such as data redundancy, stock mismanagement, delayed order processing, and lack of visibility across multiple warehouses.

Solution Overview:
The proposed solution is a centralized database-driven system that:

- Tracks products, quantities, and suppliers in real-time.
- Manages multiple warehouses and their stock levels.
- Handles purchase orders efficiently.

- Provides accurate and up-to-date information to decision-makers.

  The system uses an Entity-Relationship (ER) model to structure data relationships among products, suppliers, warehouses, and purchase orders, forming the foundation for future database.

Literature Review / Background Study

1) Review Summary

Previous studies on inventory management systems emphasize automation, database integration, and supply chain optimization. Traditional approaches relied heavily on manual record-keeping, which increased human error and inefficiency.

Modern systems use database management tools like **MySQL, Oracle, or PostgreSQL** and languages such as **C++ or Python** to automate and streamline stock management. The literature shows that digital inventory systems significantly reduce operational costs, enhance accuracy, and improve responsiveness in procurement and distribution.

2) Goals and Objectives

- To design an efficient and normalized ER model for inventory management.
- To integrate supplier, warehouse, and product data under one framework.
- To maintain accurate and real-time product stock levels.
- To facilitate easy tracking of purchase orders and supplier performance.
- To minimize manual errors and data duplication.

Setup and Design

Methodologies, Tools, and Technologies

- **Methodology**: Database System Design using Entity-Relationship Modelling.
- **Approach**: Top-down design, starting from identifying entities, attributes, and relationships, followed by normalization.
- **Tools Used**:
  - *ER Diagram Design:* Draw.io / Lucid chart / Figma
  - *Database:* MySQL / SQLite
  - *Programming Language (optional for implementation):* C++ or Python
  - *Version Control:* GitHub for version management and collaboration

Hardware and Software Specifications

Hardware Requirements:

- Processor: Intel i3 or higher
- RAM: Minimum 4 GB
- Storage: 500 GB HDD or higher

Software Requirements:

- Operating System: Windows / Linux
- Database Software: MySQL / SQLite
- IDE: Visual Studio Code / MySQL Workbench
- Web Browser: Chrome / Firefox

Implementation & Results

ER Diagram & Relational Schema

The Entity–Relationship (ER) Diagram represents the logical structure of the database used in the Inventory Management System.
It defines entities (tables), their attributes (fields), and the relationships among them. The ER model ensures data consistency, eliminates redundancy, and provides a clear blueprint for database implementation.

- **Entities and Attributes:**

1. Supplier
    - Supplier_ID (PK) — Unique identifier for each supplier.
    - Supplier_Name — Name of the supplier or vendor.
    - Contact_Number — Supplier's contact information.
    - Email — Supplier's email address.
    - Address — Physical address of the supplier.

2. Product
    - Product_ID (PK) — Unique ID for each product.
    - Product_Name — Name or description of the product.
    - Category — Type or category of the product.
    - Unit_Price — Cost per unit of the product.
    - Supplier_ID (FK) — References the supplier providing the product.

3. Warehouse
    - Warehouse_ID (PK) — Unique ID for each warehouse.
    - Warehouse_Name — Name or location label of the warehouse.
    - Location — Geographical location or address.
    - Capacity — Maximum number of items or units that can be stored.

4. Stock
    - Product_ID (FK) — References the product.
    - Warehouse_ID (FK) — References the warehouse.
    - Quantity — Number of units of a product stored in that warehouse.
    - Reorder_Level — Minimum quantity before reordering is needed.
      (Composite Key: Product_ID + Warehouse_ID)

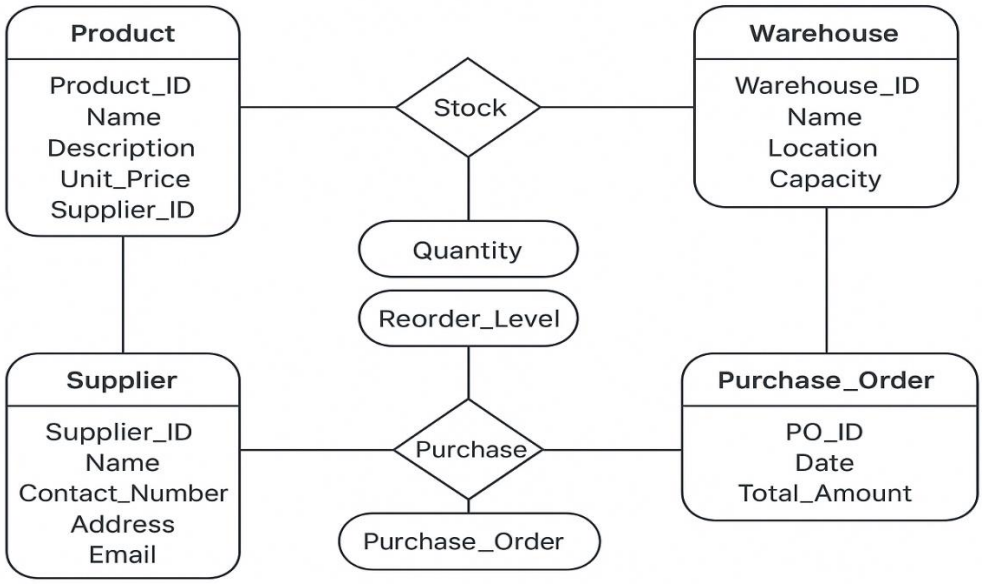
5. Purchase_Order
    - PO_ID (PK) — Unique purchase order number.
    - PO_Date — Date on which the purchase order was created.
    - Supplier_ID (FK) — References the supplier who will fulfill the order.
    - Total_Amount — Total monetary value of the purchase order.

6. Purchase_Order_Detail
    - PO_ID (FK) — References the purchase order.
    - Product_ID (FK) — References the product in that order.
    - Quantity — Number of units ordered.
    - Unit_Cost — Cost per unit for that order.
      (Composite Key: PO_ID + Product_ID)

- **Relationships in ER Diagram**

Relationship	Type	Description
Supplier – Product	1 : M	One supplier can supply many products, but each product is supplied by only one supplier.
Product – Warehouse	M : M	A product can be stored in multiple warehouses, and a warehouse can store multiple products. Managed through the **Stock** entity.
Supplier – Purchase_Order	1 : M	One supplier can have multiple purchase orders.
Purchase_Order – Product	M : M	Each order can contain multiple products, and the same product can appear in different orders. Managed through **Purchase_Order_Detail**.

- **ER DIAGRAM:**



S

The ER diagram visually shows the connection between suppliers, products, warehouses, and purchase orders. It forms the conceptual foundation for converting the design into a relational database schema.

- **Relational Schema:**

The Relational Schema translates the ER model into actual database tables, defining primary keys (PK), foreign keys (FK), and relationships. Each entity becomes a table, and relationships are represented by foreign key constraints.

1. Supplier Table

```
CREATE TABLE Supplier (
   Supplier_ID INT PRIMARY KEY,
   Supplier_Name VARCHAR(100),
   Contact_Number VARCHAR(15),
   Email VARCHAR(100),
```

```
Address VARCHAR(255)
);

2. Product Table
CREATE TABLE Product (
   Product_ID INT PRIMARY KEY,
   Product_Name VARCHAR(100),
   Category VARCHAR(50),
   Unit_Price DECIMAL(10,2),
   Supplier_ID INT,
   FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);

3. Warehouse Table
CREATE TABLE Warehouse (
   Warehouse_ID INT PRIMARY KEY,
   Warehouse_Name VARCHAR(100),
   Location VARCHAR(100),
   Capacity INT
);

4. Stock Table (Associative Entity)
CREATE TABLE Stock (
   Product_ID INT,
   Warehouse_ID INT,
   Quantity INT,
   Reorder_Level INT,
   PRIMARY KEY (Product_ID, Warehouse_ID),
   FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
   FOREIGN KEY (Warehouse_ID) REFERENCES Warehouse(Warehouse_ID)
);

5. Purchase_Order Table
CREATE TABLE Purchase_Order (
   PO_ID INT PRIMARY KEY,
```

```
    PO_Date DATE,
    Supplier_ID INT,
    Total_Amount DECIMAL(12,2),
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);

6. Purchase_Order_Detail Table
CREATE TABLE Purchase_Order_Detail (
    PO_ID INT,
    Product_ID INT,
    Quantity INT,
    Unit_Cost DECIMAL(10,2),
    PRIMARY KEY (PO_ID, Product_ID),
    FOREIGN KEY (PO_ID) REFERENCES Purchase_Order(PO_ID),
    FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);
```

- **Query Outputs:**

After creating and populating the database with sample data, the following SQL queries were executed to test and verify the functionality of the inventory management system.

- **Query 1: Display total stock of each product across all warehouses**

```
SELECT p.Product_ID, p.Product_Name, SUM(s.Quantity) AS Total_Stock
FROM Product p
JOIN Stock s ON p.Product_ID = s.Product_ID
GROUP BY p.Product_ID, p.Product_Name;
```

Output:

Product_ID	Product_Name	Total_Stock
101	Laptop	55
102	Keyboard	120
103	Mouse	200
104	Monitor	75

- **Query 2: List all products that have reached or fallen below the reorder level**

```
SELECT p.Product_Name, w.Warehouse_Name, s.Quantity,
s.Reorder_Level
FROM Stock s
JOIN Product p ON s.Product_ID = p.Product_ID
JOIN Warehouse w ON s.Warehouse_ID = w.Warehouse_ID
WHERE s.Quantity <= s.Reorder_Level;
```

Output:

Product_Name	Warehouse_Name	Quantity	Reorder_Level
Keyboard	Central Depot	15	20
Mouse	West Wing	10	25

- **Query 3: Display purchase order details with supplier name**

```
SELECT po.PO_ID, po.PO_Date, s.Supplier_Name, pod.Product_ID,
pod.Quantity, pod.Unit_Cost
FROM Purchase_Order po
JOIN Supplier s ON po.Supplier_ID = s.Supplier_ID
JOIN Purchase_Order_Detail pod ON po.PO_ID = pod.PO_ID;
```

Output:

PO_ID	PO_Date	Supplier_Name	Product_ID	Quantity	Unit_Cost
5001	2025-10-05	ABC Traders	101	20	45000.00
5001	2025-10-05	ABC Traders	102	50	800.00
5002	2025-10-06	TechSupplies	103	100	500.00

- **Query 4: Display supplier-wise total purchase amount**

```
SELECT s.Supplier_Name, SUM(po.Total_Amount) AS
Total_Purchase_Value
FROM Purchase_Order po
JOIN Supplier s ON po.Supplier_ID = s.Supplier_ID
GROUP BY s.Supplier_Name;
```

Output:

Supplier_Name	Total_Purchase_Value
ABC Traders	960000.00
TechSupplies	50000.00

Normalization

Normalization is applied to organize data efficiently, reduce redundancy, and improve integrity.The database design for the inventory management system follows **up to Third Normal Form (3NF)**.

1. First Normal Form (1NF)
Rule: Each cell contains atomic (indivisible) values; no repeating groups or arrays.

Before Normalization (example):

Product_ID	Product_Name	Supplier	Supplier_Contact	Warehouse1_Qty	Warehouse2_Qty

Problem: Data repetition and multiple warehouse quantities in one row.

After 1NF:

- Separate data into distinct entities: **Product**, **Supplier**, **Warehouse**, and **Stock**.
- Each field now stores atomic values only.

2. Second Normal Form (2NF)
Rule: Remove partial dependency; all non-key attributes must depend on the entire primary key.

Example:
In **Stock(Product_ID, Warehouse_ID, Quantity, Reorder_Level)**
→ The composite key is *(Product_ID, Warehouse_ID)*.
All other fields depend on both, not on part of it — hence it satisfies 2NF.

3. Third Normal Form (3NF)

Rule: Remove transitive dependency; non-key attributes must depend only on the primary key.

Example:

In **Product(Product_ID, Product_Name, Supplier_ID, Supplier_Name)**
→ Supplier_Name depends on Supplier_ID, not directly on Product_ID.
To achieve 3NF, move supplier data to a separate **Supplier** table, linked through Supplier_ID.
After this step, every table in the database holds data relevant only to its key, ensuring no redundancy or data anomalies.

Normalization Summary

Normal Form	Achieved By	Result
1NF	Eliminating repeating fields and ensuring atomic data	Separate entities created
2NF	Removing partial dependencies	Composite keys handled properly
3NF	Removing transitive dependencies	Supplier, Product, Warehouse fully normalized

Final Result

After normalization:

- **No redundant data** exists.
- **All relationships** are defined using foreign keys.
- **Data updates and deletions** do not cause anomalies.
- The database is efficient and scalable.

FUTURE SCOPE

- Integration with a **web or mobile interface** for real-time access.
- Implementation of **AI-driven stock prediction** and automated reordering.
- Use of **cloud-based databases** for multi-location synchronization.
- Addition of **analytics dashboards** for performance monitoring.

CONCLUSION

The Inventory Management System successfully establishes a robust foundation for tracking and managing product stocks, supplier relationships, and purchase orders.
The ER model ensures data integrity, efficient relationships, and scalability for future expansion.
This system can evolve into a full-fledged business inventory platform, integrating front-end applications, cloud connectivity, and AI-driven insights, thereby improving operational efficiency and decision-making accuracy.