BVRIT HYDERABAD

College of Engineering for Women

Department of CSE(AIML)

CASE STUDY I

INVENTORY MANAGEMENT SYSTEM

TEAM 1

Sadiya - 22WH1A6605

R.Ishwarya - 22WH1A6609

I.Sivani - 22WH1A6613

V.Keavalya - 22WH1A6618

K.Laxmi Prasanna - 22WH1A6620

L.Lasya - 22WH1A6634

P.Jahnavi- 22WH1A6639

G.Anusha-22WH1A6641

B.Soumya - 22WH1A6646

M.Siri Chandana-22WH1A6661

Under the guidance of

Ms V. Asha(Asst.Professor)

INDEX

INTRODUCTION

An Inventory Management System (IMS) is a software solution to oversee and manage the ordering,storage and use of materials or product a company will sell.It involves monitoring the flow of products from manufacturers to warehouses and from these facilities to point of sale.It helps businesses to maintain optimal stock levels, reduce costs and improve efficiency by automating tasks using products.

Benefits:

1.Increased Accuracy:
Reduces human errors associated with manual tracking, ensuring accurate inventory records.

2.Improved Productivity:
Streamlines inventory-related tasks, freeing up staff to focus on other critical business activities.

3.Better Inventory Control:
Prevents overstocking and stockouts by maintaining optimal inventory levels.

4.Enhanced Customer Service:
Ensures product availability, leading to higher customer satisfaction and loyalty.

5.Cost Reduction:
Minimizes storage costs and capital tied up in excess inventory, improving cash flow.

REQUIRMENT ANALYSIS

Requirement analysis for a DBMS-based Inventory Management System involves identifying the needs and constraints of the system, understanding the workflows, and defining the data requirements. Here's a structured approach to this analysis:

1. *Stakeholder Identification*

Identify all stakeholders involved in the inventory management process. This typically includes:

- Inventory Managers
- Warehouse Staff
- Sales Team
- Procurement Team
- Finance Team
- IT Support Team
- End Customers (if relevant)

2. *Functional Requirements*

Define what the system should be able to do. Common functional requirements for an inventory management system include:

- *Inventory Tracking:* Track the quantity, location, and status of each item in inventory.
- *Order Management:* Manage purchase orders, sales orders, and order fulfillment.
- *Supplier Management:* Maintain supplier information and manage relationships.
- *Stock Management:* Handle stock levels, replenishments, stock transfers, and adjustments.
- *Reporting:* Generate various reports such as inventory levels, turnover rates, and stock valuation.
- *Alerts and Notifications:* Provide alerts for low stock, overstock, and expiring items.
- *User Management:* Manage user roles and permissions.

3. *Non-Functional Requirements*

Identify the system's performance criteria and constraints, such as:

- *Scalability:* Ability to handle increasing amounts of data and transactions.
- *Reliability:* System uptime and error handling capabilities.
- *Security:* Data protection, user authentication, and authorization.
- *Usability:* User interface and experience considerations.
- *Integration:* Ability to integrate with other systems such as ERP, CRM, and accounting software.

4. *Data Requirements*
Define the data entities and their relationships. Common entities include:

- *Products:* Product ID, name, description, category, price, etc.
- *Inventory:* Inventory ID, product ID, quantity, location, status, etc.
- *Suppliers:* Supplier ID, name, contact information, products supplied, etc.
- *Orders:* Order ID, customer ID, product ID, quantity, order date, delivery date, status, etc.
- *Customers:* Customer ID, name, contact information, order history, etc.
- *Transactions:* Transaction ID, product ID, quantity, transaction type (in/out), date, etc.

5. *Process Flows*
Document the workflows for key processes:

- *Receiving Stock:* From suppliers, including quality checks and inventory updates.
- *Storing Stock:* Assigning locations within the warehouse and updating the inventory database.
- *Picking and Packing:* Processing orders, picking items from inventory, packing, and preparing for shipment.
- *Shipping:* Updating inventory levels and handling logistics.
- *Inventory Audits:* Regular checks to reconcile physical inventory with the database records.

6. *User Interface Requirements*
Define the interfaces for different types of users, such as:

- *Dashboard:* For inventory managers to view key metrics and alerts.
- *Order Entry Screen:* For sales and procurement teams to enter orders.
- *Inventory Lookup:* For warehouse staff to search and locate items.
- *Reporting Interface:* For generating and viewing reports.

7. *System Integration*
Consider how the inventory management system will interact with other systems:

- *ERP Integration:* Sync inventory data with enterprise resource planning systems.
- *Accounting Integration:* Share financial data with accounting systems.
- *CRM Integration:* Connect with customer relationship management systems to track orders and customer interactions.
- *API:* Define APIs for data exchange with external systems.

8. *Constraints and Assumptions*
Identify any constraints and assumptions for the project, such as:

- *Budget Constraints:* Financial limits for the project.
- *Technology Stack:* Preferred technologies or existing infrastructure.
- *Timeline:* Project deadlines and milestones.
- *Regulatory Compliance:* Any industry-specific regulations that must be adhered to.

9. *Future Enhancements*
Consider potential future enhancements that may be needed:

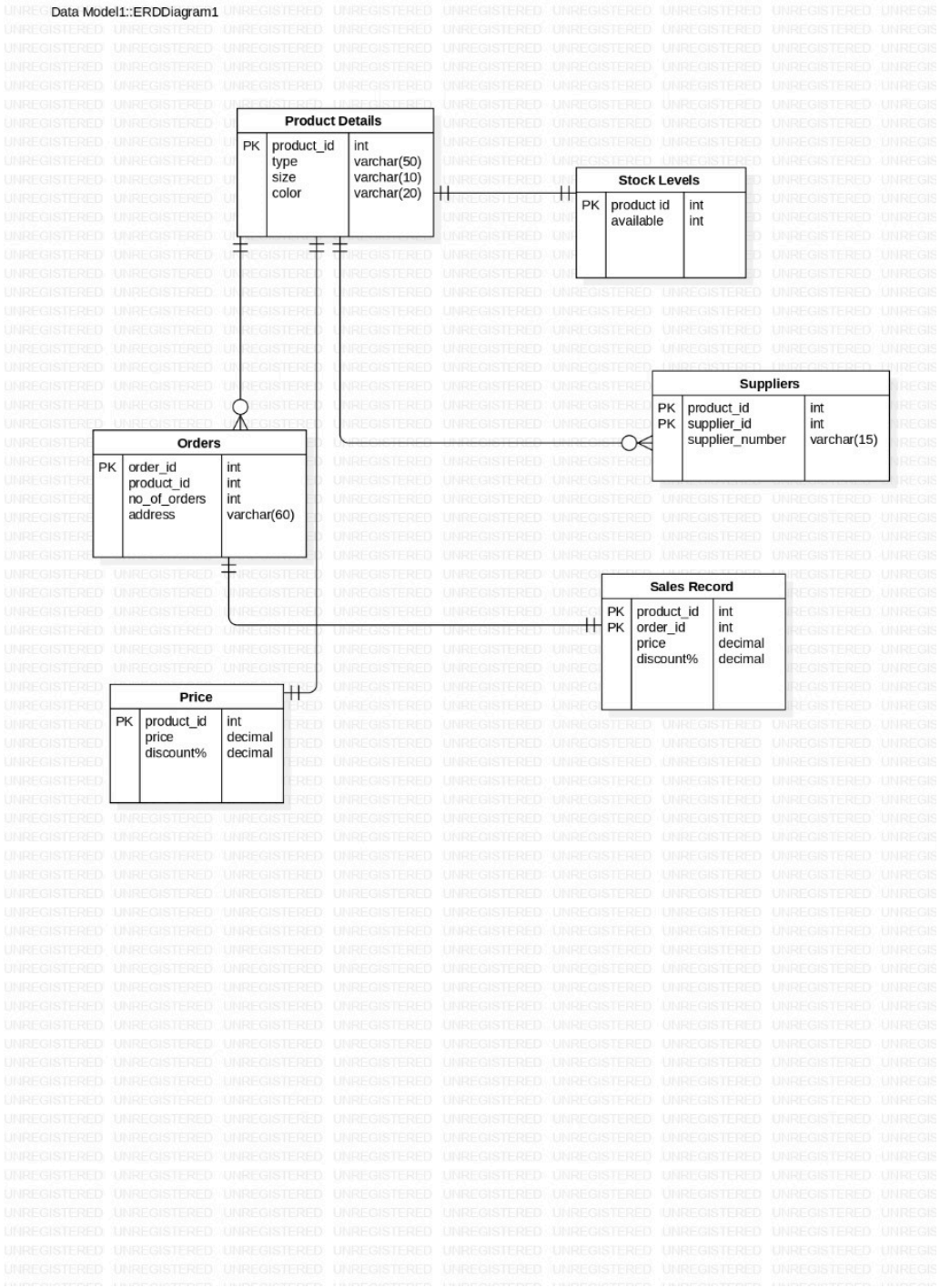
- *Mobile Access:* Support for mobile devices.
- *Advanced Analytics:* Implementing predictive analytics for inventory management.
- *AI Integration:* Using artificial intelligence for demand forecasting and stock optimization.

10. *Documentation and Training*
Plan for comprehensive documentation and training for users and administrators:

- *User Manuals:* Detailed guides for using the system.
- *Training Programs:* Sessions and materials to train users on the new system.
- *Support Resources:* Ongoing support and resources for troubleshooting and optimization.

Conclusion
By thoroughly analyzing these requirements, you can design a robust and efficient DBMS-based Inventory Management System that meets the needs of all stakeholders and supports the organization's inventory management goals.

DATABASE DESIGN



DATABASE IMPLEMENTATION

Creation of Product Details table

1.Creation of Product Details table

```
CREATE TABLE ProductDetails (
  ProductID INT PRIMARY KEY,
  ProductType VARCHAR(50),
  Size VARCHAR(10),
  Color VARCHAR(20),
  Brand VARCHAR(50));

INSERT INTO ProductDetails (ProductID, ProductType, Size,
Color, Brand) VALUES
);
(1, 'Dress', 'M', 'Red', 'Zara'),
(2, 'Shoe', '42', 'Black', 'Nike'),
(3, 'Jewellery', 'N/A', 'Gold', 'Tanishq'),
(4, 'Dress', 'L', 'Blue', 'H&M'),
(5, 'Shoe', '40', 'White', 'Adidas'),
(6, 'Jewellery', 'N/A', 'Silver', 'Kalyan Jewellers'),
(7, 'Dress', 'S', 'Green', 'Forever 21'),
(8, 'Shoe', '41', 'Brown', 'Puma'),
(9, 'Jewellery', 'N/A', 'Platinum', 'Malabar Gold and
Diamonds'),
(10, 'Dress', 'XL', 'Yellow', 'Uniqlo');
```



```
SELECT * FROM ProductDetails;
```

ProductID	ProductType	Size	Color	Brand
1	Dress	M	Red	Zara
2	Shoe	42	Black	Nike
3	Jewellery	N/A	Gold	Tanishq
4	Dress	L	Blue	H&M
5	Shoe	40	White	Adidas
6	Jewellery	N/A	Silver	Kalyan Jewellers
7	Dress	S	Green	Forever 21
8	Shoe	41	Brown	Puma
9	Jewellery	N/A	Platinum	Malabar Gold and Diamonds
10	Dress	XL	Yellow	Uniqlo

Creation of Stock Levels table

```
create table StockLevels (
ProductID INT PRIMARY KEY,
Available INT,
FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID));
INSERT INTO StockLevels (ProductID, Available) VALUES
(1, 50),
(2, 20),
(3, 15),
(4, 40),
(5, 30),
(6, 10),
(7, 60),
(8, 25),
(9, 5),
(10, 35);
```

);

SELECT * FROM StockLevels;

ProductID	Available
1	50
2	20
3	15
4	40
5	30
6	10
7	60
8	25
9	5
10	35

Creation of Orders table

```
CREATE TABLE Orders(
OrderID INT PRIMARY KEY,
  ProductID INT,
NumberOfOrders INT,
  Address VARCHAR(100),
  FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID));
INSERT INTO Orders (OrderID, ProductID, NumberOfOrders, Address) VALUES
(101, 1, 2, '123 Street, Hyderabad'),
(102, 2, 1, '456 Avenue Pune'),
(103, 3, 4, '789 Boulevard, Chennai'),
(104, 4, 3, '101 Road, Mumbai'),
(105, 5, 1, '242 Highway, Banglore'),
(106, 6, 2, '373 Lane, Kochi'),
(107, 7, 1, '454 Path, Hyderabad'),
(108, 8, 3, '545 Way, Chennai'),
(109, 9, 2, '66 Trail, Hyderabad'),
(110, 10, 1, '797 Drive, Kurnool');
SELECT * FROM Orders;
```

OrderID	ProductID	NumberOfOrders	Address
101	1	2	123 Street, Hyderabad
102	2	1	456 Avenue Pune
103	3	4	789 Boulevard, Chennai
104	4	3	101 Road, Mumbai
105	5	1	242 Highway, Banglore
106	6	2	373 Lane, Kochi
107	7	1	454 Path, Hyderabad
108	8	3	545 Way, Chennai
109	9	2	66 Trail, Hyderabad
110	10	1	797 Drive, Kurnool

Creation of Suppliers table

```
CREATE TABLE Suppliers (

SupplierID INT PRIMARY KEY,

ProductID INT,

SupplierNumber VARCHAR(20),

FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID));

INSERT INTO Suppliers (SupplierID, ProductID, SupplierNumber) VALUES

(101, 1, '9876543210'),

(102, 2, '9123456789'),

(103, 3, '9988776655'),

(104, 4, '9876501234'),

(105, 5, '9867543210'),

(106, 6, '9345678901'),

(107, 7, '9456781230'),

(108, 8, '9567812345'),

(109, 9, '9678901234'),

(110, 10, '9789012345');

SELECT * FROM Suppliers;
```

SupplierID	ProductID	SupplierNumber
101	1	9876543210
102	2	9123456789
103	3	9988776655
104	4	9876501234
105	5	9867543210
106	6	9345678901
107	7	9456781230
108	8	9567812345
109	9	9678901234
110	10	9789012345

Creation of Pricing table

```
CREATE TABLE Pricing (
ProductID INT PRIMARY KEY,
  Price DECIMAL(10, 2),
  DiscountPercent DECIMAL(5, 2),
  FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID));
INSERT INTO Pricing (ProductID, Price, DiscountPercent) VALUES
(1, 1000.00, 10.00),
(2, 1500.00, 5.00),
(3, 2000.00, 15.00),
(4, 1200.00, 20.00),
(5, 1300.00, 0.00),
(6, 1800.00, 25.00),
(7, 1100.00, 5.00),
(8, 1600.00, 10.00),
(9, 2100.00, 30.00),
(10, 1400.00, 20.00);
SELECT * FROM Pricing;
```

ProductID	Price	DiscountPercent
1	1000.00	10.00
2	1500.00	5.00
3	2000.00	15.00
4	1200.00	20.00
5	1300.00	0.00
6	1800.00	25.00
7	1100.00	5.00
8	1600.00	10.00
9	2100.00	30.00
10	1400.00	20.00

Creation of Sales Record table

```
CREATE TABLE SalesRecord (
  SalesID INT PRIMARY KEY,
  ProductID INT,
  OrderID INT,
  Price DECIMAL(10, 2),
  DiscountPercent DECIMAL(5, 2),
  FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID),
  FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);


INSERT INTO SalesRecord (SalesID, ProductID, OrderID, Price,
DiscountPercent) VALUES
(1, 1, 101, 1000.00, 10.00),
(2, 2, 102, 1500.00, 5.00),
```

(3, 3, 103, 2000.00, 15.00),

(4, 4, 104, 1200.00, 20.00),

(5, 5, 105, 1300.00, 0.00),

(6, 6, 106, 1800.00, 25.00),

(7, 7, 107, 1100.00, 5.00),

(8, 8, 108, 1600.00, 10.00),

(9, 9, 109, 2100.00, 30.00),

(10, 10, 110, 1400.00, 20.00);

SELECT * FROM SalesRecord;

SalesID	ProductID	OrderID	Price	DiscountPercent
1	1	101	1000.00	10.00
2	2	102	1500.00	5.00
3	3	103	2000.00	15.00
4	4	104	1200.00	20.00
5	5	105	1300.00	0.00
6	6	106	1800.00	25.00
7	7	107	1100.00	5.00
8	8	108	1600.00	10.00
9	9	109	2100.00	30.00
10	10	110	1400.00	20.00

TESTING AND RESULTS

1. Count the number of dresses available

```
SELECT COUNT(*) AS DressCount

FROM ProductDetails

WHERE ProductType = 'Dress';
```

OUTPUT:

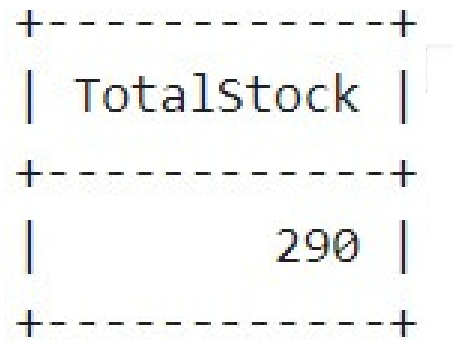
DressCount
4

2.Find the total stock available for all products:

```
SELECT SUM(Available) AS TotalStock

FROM StockLevels;
```

OUTPUT:



TotalStock
290

3.Get the total number of orders placed for each product type:

```
SELECT PD.ProductType, SUM(O.NumberOfOrders) AS TotalOrders

FROM Orders O

JOIN ProductDetails PD ON O.ProductID = PD.ProductID

GROUP BY PD.ProductType;
```

OUTPUT:

```
+-------------+-------------+
| ProductType | TotalOrders |
+-------------+-------------+
| Dress       |           7 |
| Shoe        |           5 |
| Jewellery   |           8 |
+-------------+-------------+
```

4.Retrieve the average price of shoes:

```
SELECT AVG(Price) AS AverageShoePrice

FROM Pricing P

JOIN ProductDetails PD ON P.ProductID =

PD.ProductID

WHERE PD.ProductType = 'Shoe';
```

OUTPUT:

```
| AverageShoePrice |
+------------------+
|      1466.666667 |
```

5.List the products that have a discount of 20% or more:

```
SELECT PD.ProductID, PD.ProductType, PD.Brand,

P.DiscountPercent

FROM Pricing P

JOIN ProductDetails PD ON P.ProductID = PD.ProductID

WHERE P.DiscountPercent >= 20;
```

OUTPUT:

ProductID	ProductType	Brand	DiscountPercent
4	Dress	H&M	20.00
6	Jewellery	Kalyan Jewellers	25.00
9	Jewellery	Malabar Gold and Diamonds	30.00
10	Dress	Uniqlo	20.00

6.Get the total revenue before and after discount for all sales:

```
SELECT SUM(Price) AS TotalRevenueBeforeDiscount,

    SUM(Price * (1 - DiscountPercent / 100)) AS

TotalRevenueAfterDiscount

FROM SalesRecord;
```

Find the supplier with the most products supplied:

OUTPUT:

TotalRevenueBeforeDiscount	TotalRevenueAfterDiscount
15000.00	12710.00000000

7.Retrieve all products supplied by a specific supplier (e.g., supplier with ID 101):

```
SELECT PD.ProductID, PD.ProductType, PD.Brand

FROM Suppliers S

JOIN ProductDetails PD ON S.ProductID = PD.ProductID

WHERE S.SupplierID = 101;
```

Calculate the total number of each type of product in stock:

OUTPUT:

ProductID	ProductType	Brand
1	Dress	Zara

8.Find the supplier with the most products supplied:

```
SELECT SupplierID, COUNT(ProductID) AS ProductsSupplied

FROM Suppliers

GROUP BY SupplierID

ORDER BY ProductsSupplied DESC

LIMIT 1;
```

ProductID	ProductType	Brand
1	Dress	Zara

9.Calculate the total number of each type of product in stock:

```
SELECT PD.ProductType, SUM(SL.Available) AS TotalStock

FROM StockLevels SL

JOIN ProductDetails PD ON SL.ProductID = PD.ProductID

GROUP BY PD.ProductType;
```

OUTPUT:

ProductType	TotalStock
Dress	185
Shoe	75
Jewellery	30

10.List all products that are out of stock:

```
SELECT PD.ProductID, PD.ProductType, PD.Brand

FROM StockLevels SL

JOIN ProductDetails PD ON SL.ProductID = PD.ProductID

WHERE SL.Available = 0;
```

11.Retrieve all orders with the total price after discount

```
SELECT O.OrderID, O.ProductID, O.NumberOfOrders,
SR.Price * O.NumberOfOrders * (1 - SR.DiscountPercent / 100) AS
TotalPriceAfterDiscount
FROM Orders O
JOIN SalesRecord SR ON O.OrderID = SR.OrderID;
```

OUTPUT:

OrderID	ProductID	NumberOfOrders	TotalPriceAfterDiscount
101	1	2	1800.00000000
102	2	1	1425.00000000
103	3	4	6800.00000000
104	4	3	2880.00000000
105	5	1	1300.00000000
106	6	2	2700.00000000
107	7	1	1045.00000000
108	8	3	4320.00000000
109	9	2	2940.00000000
110	10	1	1120.00000000

12. Find the most expensive product and its details:

```
SELECT PD.ProductID, PD.ProductType, PD.Brand, P.Price
FROM Pricing P
JOIN ProductDetails PD ON P.ProductID = PD.ProductID
ORDER BY P.Price DESC
LIMIT 1;
```

OUTPUT:

ProductID	ProductType	Brand	Price
9	Jewellery	Malabar Gold and Diamonds	2100.00

13.Calculate the total discount given for all sales:

SELECT SUM((Price * DiscountPercent / 100)) AS

TotalDiscountGiven

FROM SalesRecord;

OUTPUT:

TotalDiscountGiven
2290.00000000

14.Get the number of products supplied by each brand:

SELECT SUM((Price * DiscountPercent / 100)) AS TotalDiscountGiven

FROM SalesRecord;

SELECT PD.Brand, COUNT(PD.ProductID) AS NumberOfProducts

FROM ProductDetails PD

GROUP BY PD.Brand;

OUTPUT:

Brand	NumberOfProducts
Zara	1
Nike	1
Tanishq	1
H&M	1
Adidas	1
Kalyan Jewellers	1
Forever 21	1
Puma	1
Malabar Gold and Diamonds	1
Uniqlo	1

15. List all orders and their respective product types and brands:

SELECT O.OrderID, PD.ProductType, PD.Brand, O.NumberOfOrders,

O.Address

FROM Orders O

JOIN ProductDetails PD ON O.ProductID = PD.ProductID;

OUTPUT:

OrderID	ProductType	Brand	NumberOfOrders	Address
101	Dress	Zara	2	123 Street, Hyderabad
102	Shoe	Nike	1	456 Avenue Pune
103	Jewellery	Tanishq	4	789 Boulevard, Chennai
104	Dress	H&M	3	101 Road, Mumbai
105	Shoe	Adidas	1	242 Highway, Banglore
106	Jewellery	Kalyan Jewellers	2	373 Lane, Kochi
107	Dress	Forever 21	1	454 Path, Hyderabad
108	Shoe	Puma	3	545 Way, Chennai
109	Jewellery	Malabar Gold and Diamonds	2	66 Trail, Hyderabad
110	Dress	Uniqlo	1	797 Drive, Kurnool