INVENTORY MANAGEMENT SYSTEM

ABSTRACT

The most important thing about managing inventory is knowing exactly what you have in stock. This helps businesses save money by not buying too much stuff they don't need, and it ensures that customers get their orders on time without running out of things they want to buy. For companies with lots of stores, a good inventory system makes it easy to keep track of everything in one place, so they can make sure each store has just the right amount of stock. This means keeping an eye on what comes in from suppliers and what goes out to customers, so nothing gets lost along the way.

INTRODUCTION

Inventory management is a critical aspect of running a successful business, regardless of its size or industry. It involves overseeing the flow of goods from suppliers to customers while keeping track of stock levels to ensure smooth operations and customer satisfaction. At the heart of effective inventory management lies the need for accurate and up-to-date information about the quantities of products available at any given time.

For businesses with multiple stores or locations, managing inventory becomes even more complex. Ensuring that each store has the right amount of stock to meet customer demand while avoiding excess inventory requires careful planning and coordination. This is where a well-designed inventory management system plays a crucial role.

This paper aims to explore the essential features and functionalities of an inventory management system, with a particular focus on providing real-time information about inventory levels. It will delve into the importance of central control for organizations with multiple stores and how a good inventory management system can help optimize stock levels across all locations.

By examining the core components and benefits of an inventory management system, businesses can gain insights into how to streamline their operations, reduce costs, and enhance customer satisfaction. Throughout the paper, we will emphasize the significance of tracking inventory levels and operations accurately to achieve these objectives effectively.

ENTITY-RELATIONSHIP (ER) DIAGRAM

The Entity-Relationship (ER) diagram for an inventory management system offers a visual representation of the system's architecture, showcasing how various entities interact and the relationships between them. This diagram is instrumental in understanding the underlying structure of the database and its functionality.

Entities represent the core components of the inventory management system, such as products, suppliers, warehouses, and orders. Each entity contains attributes that define its characteristics and properties. For example, a product entity may include attributes like product ID, name, description, price, and quantity.

Relationships between entities illustrate how they are connected and interact within the system. For instance, a supplier entity may have a "supplies" relationship with a product entity, indicating that the supplier provides products to the inventory system. Similarly, an order entity may have relationships with both product and customer entities, representing the items ordered by customers.

Views offer a customized perspective of the data stored in the database, presenting subsets of information tailored to specific user needs. For instance, a manager may have access to a "stock level view" that displays current inventory levels across all warehouses, while a sales representative may have access to a "customer order view" showing pending orders and delivery status.

ENTITIES AND ATTRIBUTES

Product:

- ProductID (Primary Key)

- Name

- Description

- Category

- Price

- QuantityAvailable

Supplier:

- SupplierID (Primary Key)

- Name

- ContactInfo

- Address

Warehouse:

- WarehouseID (Primary Key)
- Location
- Capacity
- Manager
- ContactInfo

Order:

- OrderID (Primary Key)
- CustomerID (Foreign Key referencing Customer)
- OrderDate
- Status
- DeliveryAddress

Transaction:

- TransactionID (Primary Key)
- Date
- Quantity
- Amount
- ProductID (Foreign Key referencing Product)

Customer:

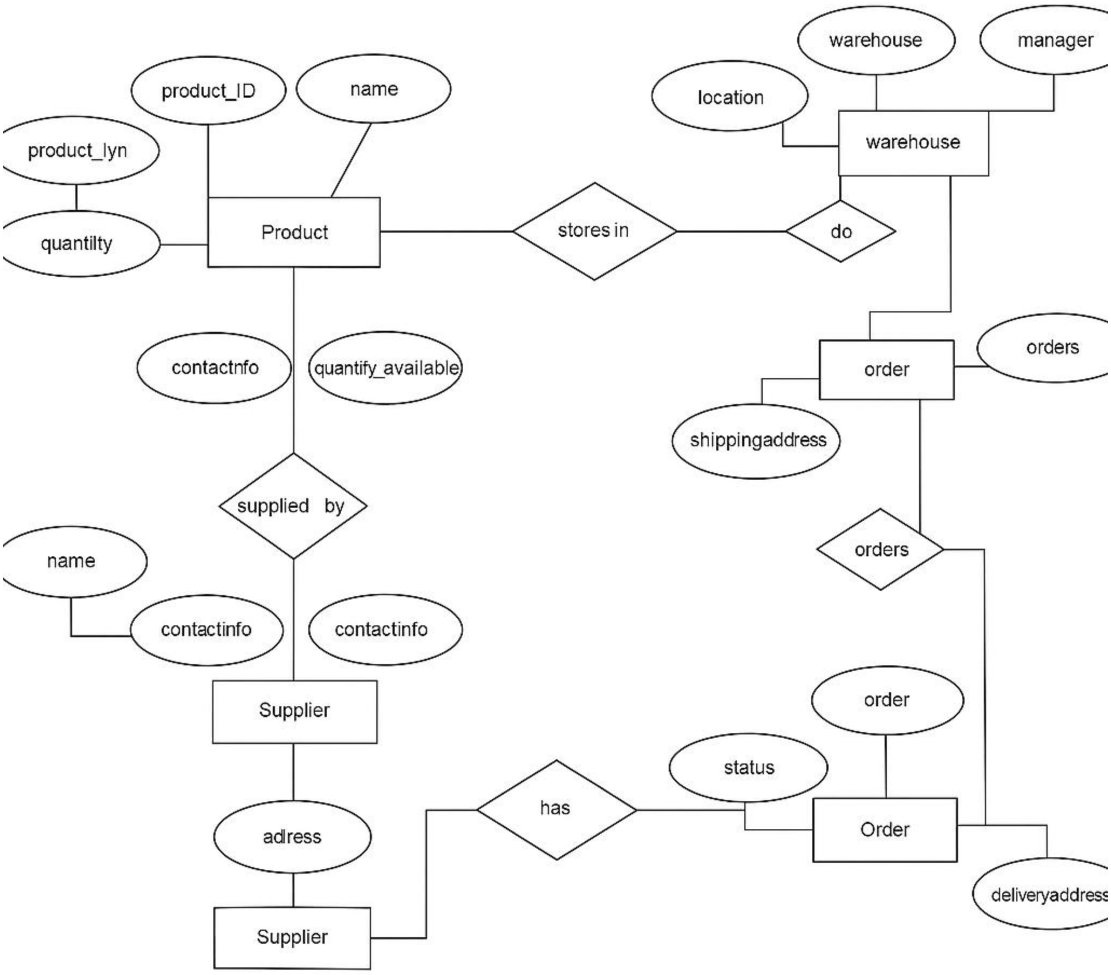
- CustomerID (Primary Key)
- Name
- ContactInfo
- ShippingAddress

Employee:

- EmployeeID (Primary Key)
- Name
- Position

- ContactInfo
- WarehouseID (Foreign Key referencing Warehouse)

ER-DIAGRAM:



IMPLEMENTATION

Creating tables and inserting values:

The CREATE TABLE statement is used to create a new table in a database.

The INSERT INTO statement is used to insert new values or records in a table

```
SQL> CREATE TABLE Product (ProductID INT PRIMARY KEY, Name VARCHAR(15),
Description VARCHAR(30), Category VARCHAR(20), Price DECIMAL(10, 2), QuantityAvailable
INT);

Table created.
```

```
SQL> CREATE TABLE Supplier (SupplierID INT PRIMARY KEY, Name VARCHAR(20),
ContactInfo VARCHAR(30), Address VARCHAR(30));

Table created.
```

```
SQL> CREATE TABLE Warehouse (WarehouseID INT PRIMARY KEY, Location
VARCHAR(30), Capacity INT, Manager VARCHAR(20), ContactInfo VARCHAR(30));

Table created.
```

```
SQL> CREATE TABLE Customer (CustomerID INT PRIMARY KEY, Name VARCHAR(20),
ContactInfo VARCHAR(30), ShippingAddress VARCHAR(30));

Table created.
```

```
CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID INT, OrderDate DATE,
Status VARCHAR(15), DeliveryAddress VARCHAR(30));

Table created.
```

```
SQL> CREATE TABLE Transaction (TransactionID INT PRIMARY KEY, TransactionDate DATE,
Quantity INT, Amount DECIMAL(10, 2),ProductID INT);

Table created.
```

```
SQL> INSERT INTO Product VALUES (1, 'Phoebe', 'Acoustic Guitar', 'Musical Instruments',
299.99, 10);

1 row created.
```

```
SQL> INSERT INTO Product VALUES (2, 'Joey', 'Leather Jacket', 'Apparel', 150.00, 25);

1 row created.


SQL> INSERT INTO Product VALUES (3, 'Monica', 'Chef Knife Set', 'Kitchen', 120.00, 50);

1 row created.


SQL> INSERT INTO Product VALUES (4, 'Rachel', 'Designer Handbag', 'Accessories', 250.00, 30);

1 row created.


SQL> INSERT INTO Product VALUES (5, 'Chandler', 'Comedy Book Collection', 'Books', 75.00,
100);

1 row created.


SQL> INSERT INTO Product VALUES (6, 'Ross', 'Dinosaur Fossil Kit', 'Educational Toys', 40.00,
75);

1 row created.


SQL> select * from Product;


 PRODUCTID NAME           DESCRIPTION                    CATEGORY

---------- --------------- ------------------------------ --------------------

     PRICE QUANTITYAVAILABLE

---------- -----------------

         1 Phoebe         Acoustic Guitar                Musical Instruments

    299.99                10


         2 Joey           Leather Jacket                 Apparel

       150                25


         3 Monica         Chef Knife Set                 Kitchen

       120                50
```

```
         4  Rachel      Designer Handbag          Accessories
       250          30

         5  Chandler    Comedy Book Collection    Books
        75         100

         6  Ross        Dinosaur Fossil Kit       Educational Toys
        40          75
```

```
SQL> INSERT INTO Supplier VALUES (1, 'Phoebe Supplies', 'phoebe@friends.com', 'Apt 14, NY');
1 row created.

SQL> INSERT INTO Supplier VALUES (2, 'Joey Apparel', 'joey@friends.com', 'Apt 19, NY');
1 row created.

SQL> INSERT INTO Supplier VALUES (3, 'Monica Kitchen', 'monica@friends.com', 'Apt 20, NY');
1 row created.

SQL> INSERT INTO Supplier VALUES (4, 'Rachel Access', 'rachel@friends.com', 'Central Perk,
NY');
1 row created.

SQL> INSERT INTO Supplier VALUES (5, 'Chandler Books', 'chandler@friends.com', 'Apt 19,
NY');
1 row created.

SQL> select * from Supplier;

SUPPLIERID NAME                 CONTACTINFO
---------- -------------------- ------------------------------
ADDRESS
------------------------------
```

```
         1   Phoebe Supplies      phoebe@friends.com
Apt 14, NY

         2    Joey Apparel        joey@friends.com
Apt 19, NY

         3     Monica Kitchen       monica@friends.com
Apt 20, NY

         4   Rachel Access        rachel@friends.com
Central Perk, NY

         5     Chandler Books       chandler@friends.com
Apt 19, NY
```

```
SQL> INSERT INTO Warehouse VALUES (1, 'NY Central Whs', 500, 'Heckles',
'heckles@whs.com');
1 row created.

SQL> INSERT INTO Warehouse VALUES (2, 'Brooklyn Store', 300, 'Gunther',
'gunther@whs.com');
1 row created.

SQL> INSERT INTO Warehouse VALUES (3, 'Queens Logist', 700, 'Treeger', 'treeger@whs.com');
1 row created.

SQL> INSERT INTO Warehouse VALUES (4, 'Manhattan Hub', 400, 'Mike', 'mike@whs.com');
1 row created.

SQL> INSERT INTO Warehouse VALUES (5, 'Staten Depot', 200, 'Janice', 'janice@whs.com');
1 row created.
```

```
SQL> select * from Warehouse;

WAREHOUSEID   LOCATION                    CAPACITY MANAGER
----------- ---------------------------- ---------- --------------------
CONTACTINFO
------------------------------
          1     NY Central Whs                    500      Heckles
heckles@whs.com

          2    Brooklyn Store                    300      Gunther
gunther@whs.com

          3    Queens Logist                     700       Treeger
treeger@whs.com

          4     Manhattan Hub                    400      Mike
mike@whs.com

          5     Staten Depot                     200      Janice
janice@whs.com
```

```
SQL> INSERT INTO Customer VALUES (1, 'Phoebe Buffay', '555-1234', 'Apt 14, NY');
1 row created.

SQL> INSERT INTO Customer VALUES (2, 'Joey Tribbiani', '555-5678', 'Apt 19, NY');
1 row created.
SQL> INSERT INTO Customer VALUES (3, 'Monica Geller', '555-8765', 'Apt 20, NY');
1 row created.

SQL> INSERT INTO Customer VALUES (4, 'Rachel Green', '555-4321', 'Central Perk, NY');
1 row created.
```

```
SQL> INSERT INTO Customer VALUES (5, 'Chandler Bing', '555-9876', 'Apt 19, NY');
1 row created.

SQL> INSERT INTO Customer VALUES (6, 'Ross Geller', '555-6543', 'Museum of NH, NY');
1 row created.

SQL> select * from Customer;
CUSTOMERID   NAME                 CONTACTINFO
---------- -------------------- ------------------------------
SHIPPINGADDRESS
------------------------------
         1   Phoebe Buffay        555-1234
Apt 14, NY

         2   Joey Tribbiani       555-5678
Apt 19, NY

         3   Monica Geller        555-8765
Apt 20, NY

         4   Rachel Green         555-4321
Central Perk, NY

          5   Chandler Bing        555-9876
Apt 19, NY

         6   Ross Geller          555-6543
Museum of NH, NY
```

```
SQL> INSERT INTO Orders VALUES (1, 1, TO_DATE('2024-01-15', 'YYYY-MM-DD'), 'Pending',
'Central Perk, NY');
```

```
1 row created.


SQL> INSERT INTO Orders VALUES (2, 2, TO_DATE('2024-02-20', 'YYYY-MM-DD'),
'Processing', 'Apt 19, NY');

1 row created.


SQL> INSERT INTO Orders VALUES (3, 3, TO_DATE('2024-03-05', 'YYYY-MM-DD'), 'Shipped',
'Apt 20, NY');

1 row created.


SQL> INSERT INTO Orders VALUES (4, 4, TO_DATE('2024-04-10', 'YYYY-MM-DD'),
'Delivered', 'Central Perk, NY');

1 row created.


SQL> INSERT INTO Orders VALUES (5, 5, TO_DATE('2024-05-25', 'YYYY-MM-DD'),
'Cancelled', 'Museum of NH, NY');

1 row created.


SQL> INSERT INTO Orders VALUES (6, 6, TO_DATE('2024-06-30', 'YYYY-MM-DD'), 'Pending',
'Apt 14, NY');

1 row created.


SQL> select * from Orders;

   ORDERID CUSTOMERID ORDERDATE STATUS          DELIVERYADDRESS
---------- ---------- --------- --------------- ------------------------------
         1          1 15-JAN-24 Pending         Central Perk, NY
         2          2 20-FEB-24 Processing      Apt 19, NY
         3          3 05-MAR-24 Shipped         Apt 20, NY
         4          4 10-APR-24 Delivered       Central Perk, NY
         5          5 25-MAY-24 Cancelled       Museum of NH, NY
         6          6 30-JUN-24 Pending         Apt 14, NY

6 rows selected.
```

```
SQL> INSERT INTO Transaction VALUES (1, TO_DATE('2024-01-16', 'YYYY-MM-DD'), 2,
599.98, 1);

1 row created.


SQL> INSERT INTO Transaction VALUES (2,TO_DATE('2024-02-21', 'YYYY-MM-DD'), 1,
150.00, 2);

1 row created.


SQL> INSERT INTO Transaction VALUES (3,TO_DATE('2024-03-06', 'YYYY-MM-DD'),5,
600.00, 3);

1 row created.


SQL> INSERT INTO Transaction VALUES (4,TO_DATE('2024-04-11', 'YYYY-MM-DD'),1,
250.00, 4);

1 row created.


SQL> INSERT INTO Transaction VALUES (5,TO_DATE('2024-05-26', 'YYYY-MM-DD'),3,
225.00, 5);

1 row created.


SQL> INSERT INTO Transaction VALUES (6,TO_DATE('2024-06-30', 'YYYY-MM-DD'),4,
160.00, 6);

1 row created.


SQL> select * from Transaction;

TRANSACTIONID TRANSACTI   QUANTITY     AMOUNT  PRODUCTID

------------- --------- ---------- ---------- ----------

            1 16-JAN-24          2     599.98          1

            2 21-FEB-24          1        150          2

            3 06-MAR-24          5        600          3

            4 11-APR-24          1        250          4

            5 26-MAY-24          3        225          5

            6 30-JUN-24          4        160          6

6 rows selected.
```

DDL (Data Definition Language) and DML (Data Manipulation Language) are two types of SQL (Structured Query Language) commands used for different purposes in managing databases:

1. **DDL (Data Definition Language)**:
   - DDL commands are used to define, modify, and manage the structure of database objects such as tables, views, indexes, and constraints.
   - Examples of DDL commands include **CREATE**, **ALTER**, **DROP**, **TRUNCATE**, and **RENAME**.
   - DDL commands do not directly affect the data stored in the database; rather, they define the schema and structure of the database objects.
2. **DML (Data Manipulation Language)**:
   - DML commands are used to manipulate data stored in the database, such as inserting, updating, deleting, and retrieving data.
   - Examples of DML commands include **INSERT**, **UPDATE**, **DELETE**, and **SELECT**.
   - DML commands directly affect the data stored in the database tables, allowing users to perform operations on the data.

```
SQL> UPDATE Product SET Quantity = 20 WHERE ProductID = 1;

1 row updated.

SQL> DELETE FROM Supplier WHERE SupplierID = 4;

1 row deleted.

SQL> ALTER TABLE Product ADD Discount DECIMAL(5, 2);

1 row altered.

SQL> TRUNCATE TABLE Order;

table truncated.

ALTER TABLE Employee TO Employee123;

table altered.

SQL> COMMIT;

Commit complete.

SQL> ROLLBACK;

Rollback complete.
```

```
SQL> SELECT * FROM Product WHERE Category IN ('Electronics', 'Clothing');

no rows selected


SQL> SELECT * FROM Product WHERE Price > ANY (SELECT Price FROM Product WHERE
Category = 'Electronics');

no rows selected


SQL> SELECT * FROM Product WHERE Price > ALL (SELECT Price FROM Product WHERE
Category = 'Electronics');

 PRODUCTID NAME           DESCRIPTION                    CATEGORY
---------- -------------- ------------------------------ --------------------
     PRICE QUANTITYAVAILABLE   DISCOUNT
---------- ----------------- ----------
         6 Ross           Dinosaur Fossil Kit            Educational Toys
        40                75


         5 Chandler       Comedy Book Collection         Books
        75               100


         2 Joey           Leather Jacket                 Apparel
       150                25


         4 Rachel         Designer Handbag               Accessories
       250                30


         1 Phoebe         Acoustic Guitar                Musical Instruments
    299.99                10


         3 Monica         Chef Knife Set                 Kitchen
    349.99                50

6 rows selected.


SQL> SELECT * FROM Product WHERE Price BETWEEN 100 AND 500;
```

```
 PRODUCTID NAME          DESCRIPTION                  CATEGORY
---------- --------------- ----------------------------- --------------------
     PRICE QUANTITYAVAILABLE   DISCOUNT
---------- ----------------- ----------
         1 Phoebe        Acoustic Guitar              Musical Instruments
    299.99              10

         2 Joey          Leather Jacket               Apparel
       150              25

         3 Monica        Chef Knife Set               Kitchen
    349.99              50

         4 Rachel        Designer Handbag             Accessories
       250              30


SQL> SELECT * FROM Product WHERE Name LIKE 'S%';
no rows selected

SQL> SELECT * FROM Customer WHERE EXISTS (SELECT * FROM Orders WHERE
Customer.CustomerID = Orders.CustomerID);
no rows selected

SQL> SELECT Category, AVG(Price) AS AvgPrice FROM Product GROUP BY Category;
CATEGORY               AVGPRICE
-------------------- ----------
Kitchen                  349.99
Books                        75
Accessories                 250
Educational Toys             40
Musical Instruments      299.99
Apparel                     150
```

```
6 rows selected.


SQL> SELECT * FROM Product ORDER BY Price DESC;
 PRODUCTID NAME            DESCRIPTION                   CATEGORY
---------- --------------- ----------------------------- --------------------
     PRICE QUANTITYAVAILABLE   DISCOUNT
---------- ----------------- ----------
         3 Monica          Chef Knife Set                Kitchen
    349.99                50


         1 Phoebe          Acoustic Guitar               Musical Instruments
    299.99                10


         4 Rachel          Designer Handbag              Accessories
       250                30


         2 Joey            Leather Jacket                Apparel
       150                25


         5 Chandler        Comedy Book Collection        Books
        75               100


         6 Ross            Dinosaur Fossil Kit           Educational Toys
        40                75
6 rows selected.


SQL> SELECT Category, AVG(Price) AS AvgPrice FROM Product GROUP BY Category
HAVING AVG(Price) > 500;
no rows selected
```

SQL AGGREGATE FUNCTIONS:

1. SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

It is also used to summarize the data. TYPES OF FUNCTIONS:
Count
sum
average
max
min

COUNT

```
SQL> SELECT COUNT(*) AS TotalProducts FROM Product;

TOTALPRODUCTS
-------------
          6

SQL> SELECT COUNT(*) AS TotalOrders FROM Orders WHERE CustomerID = 1;

TOTALORDERS
-----------
        0
```

SUM

```
SQL> SELECT SUM(QuantityAvailable) AS TotalQuantity FROM Product;

TOTALQUANTITY

-------------

 290
```

AVG

```
SQL> SELECT AVG(Price) AS AveragePrice FROM Product;

AVERAGEPRICE

------------

194.163333
```

MAX

```
SQL> SELECT MAX(Price) AS MaxPrice FROM Product;

 MAXPRICE

----------
```

```
  349.99
```

MIN

```
SQL> SELECT MIN(Price) AS MinPrice FROM Product;

MINPRICE
----------
        40
```

Subqueries:

A subquery is a query within another SQL query within the WHERE clause.

- A subquery can be placed in a number of SQL clause like WHERE clause, FROM clause, HAVING clause.
- Subqueries are on the right side of the comparison operator.

ORDER BY CLAUSE:

The SQL ORDER BY CLAUSE is used to sort the data either in ascending or descending order, based on one or more columns

1. Order Products by Price (Ascending)

```
SQL> SELECT ProductID, Name, Price FROM Product ORDER BY Price ASC;


 PRODUCTID NAME                 PRICE
---------- --------------- ----------
         6 Ross                    40
         5 Chandler                75
         2 Joey                   150
         4 Rachel                 250
         1 Phoebe              299.99
         3 Monica              349.99

6 rows selected.
```

2. Order Products by Price (Descending)

```
SQL> SELECT ProductID, Name, Price FROM Product ORDER BY Price DESC;
```

```
PRODUCTID NAME              PRICE
---------- --------------- ----------
         3 Monica            349.99
         1 Phoebe            299.99
         4 Rachel               250
         2 Joey                 150
         5 Chandler              75
         6 Ross                  40
6 rows selected.
```

3. Order Customers by Name (Ascending)

```
SQL> SELECT CustomerID, Name, ShippingAddress FROM Customer ORDER BY Name ASC;

CUSTOMERID NAME                 SHIPPINGADDRESS
---------- -------------------- ------------------------------
         5 Chandler Bing        Apt 19, NY
         2 Joey Tribbiani       Apt 19, NY
         3 Monica Geller        Apt 20, NY
         1 Phoebe Buffay        Apt 14, NY
         4 Rachel Green         Central Perk, NY
         6 Ross Geller          Museum of NH, NY
6 rows selected.
```

SQL JOINS:

IN SQL, JOIN clause is used to combine the records from two or more tables in a database.

- There are four types of SQL joins
  1. **INNER JOIN**: An inner join returns only the rows where there is a match in both tables based on the join condition.
  2. **LEFT OUTER JOIN:** All the contents of the left table is printed and matching content of right table is printed ,if there is no matching content it brings NULL
  3. **RIGHT OUTER JOIN:** All the contents of the right table is displayed and matching contents of left table is displayed, if the is no matching content it simply shows NULL.

4. **FULL JOIN:** All the contents of RIGHT OUTER JOIN and LEFT OUTER JOIN are combined.

INNER JOIN:

```
SQL> SELECT t.TransactionID, t.TransactionDate, p.Name AS ProductName FROM
Transaction t INNER JOIN Product p ON t.ProductID = p.ProductID;

TRANSACTIONID TRANSACTI PRODUCTNAME
------------- --------- ---------------
            1 16-JAN-24 Phoebe
            2 21-FEB-24 Joey
            3 06-MAR-24 Monica
            4 11-APR-24 Rachel
            5 26-MAY-24 Chandler
            6 30-JUN-24 Ross

6 rows selected.
```

LEFT OUTER JOIN

```
SQL> SELECT t.TransactionID, t.TransactionDate, t.Quantity, t.Amount, p.Name AS ProductName,
p.Price FROM Product p LEFT JOIN Transaction t ON p.ProductID = t.ProductID;

TRANSACTIONID TRANSACTI   QUANTITY     AMOUNT PRODUCTNAME          PRICE
------------ --------- ---------- ---------- --------------- ----------
           1 16-JAN-24          2     599.98 Phoebe              299.99
           2 21-FEB-24          1        150 Joey                   150
           3 06-MAR-24          5        600 Monica              349.99
           4 11-APR-24          1        250 Rachel                 250
           5 26-MAY-24          3        225 Chandler                75
           6 30-JUN-24          4        160 Ross                    40
```

RIGHT OUTER JOIN

```
SQL> SELECT o.OrderID, o.OrderDate, o.Status, c.Name AS CustomerName FROM Orders o
RIGHT JOIN Customer c ON o.CustomerID = c.CustomerID;
```

```
ORDERID ORDERDATE STATUS         CUSTOMERNAME
---------- --------- --------------- --------------------
                                 Phoebe Buffay

                                 Joey Tribbiani

                                 Monica Geller

                                 Rachel Green

                                 Chandler Bing

                                 Ross Geller

6 rows selected.
```

FULL JOIN

SQL> SELECT e.EmployeeID, e.Name AS EmployeeName, e.Position, e.ContactInfo, e.WarehouseID, m.Name AS ManagerName FROM Employee123 e FULL OUTER JOIN Employee123 m ON e.WarehouseID = m.WarehouseID;

```
EMPLOYEEID EMPLOYEENAME         POSITION
---------- -------------------- --------------------
CONTACTINFO                     WAREHOUSEID MANAGERNAME
------------------------------ ----------- --------------------
         6 Janitor              Janitor
555-1111                                 1 Janitor
         7 Technician           Technician
555-2222                                 2 Technician
         8 Supervisor           Supervisor
555-3333                                 3 Supervisor
         9 Manager              Manager
555-4444                                 4 Manager
        10 Clerk                Clerk
555-5555                                 5 Clerk
```

SET OPERATIONS:

The SQL set operations is used to combine the two or more SQL SELECT statements.

- It returns a single result set by eliminating duplicate rows.

- Queries containing set operators are called compound queries.
- Types of set operation
    1. Union
    2. Union All
    3. Intersect
    4. Minus

UNION:

```
SQL> SELECT TransactionID, TransactionDate, Quantity, Amount, ProductID FROM Transaction
WHERE ProductID = 1 UNION SELECT TransactionID, TransactionDate, Quantity, Amount,
ProductID FROM Transaction WHERE ProductID = 2;
```

```
TRANSACTIONID TRANSACTI   QUANTITY     AMOUNT  PRODUCTID

------------- --------- ---------- ---------- ----------

            1 16-JAN-24          2     599.98          1

            2 21-FEB-24          1        150          2
```

UNION ALL:

```
SQL> SELECT CustomerID, Name, ContactInfo, ShippingAddress FROM Customer WHERE
CustomerID = 1 UNION ALL SELECT CustomerID, Name, ContactInfo, ShippingAddress FROM
Customer WHERE CustomerID = 2;
```

```
CUSTOMERID NAME                 CONTACTINFO

---------- -------------------- ------------------------------

SHIPPINGADDRESS

------------------------------

         1 Phoebe Buffay        555-1234

Apt 14, NY

         2 Joey Tribbiani       555-5678

Apt 19, NY
```

INTERSECT:

```
SQL> SELECT OrderID, OrderDate, Status, CustomerID FROM Orders WHERE CustomerID = 1
INTERSECT SELECT OrderID, OrderDate, Status, CustomerID FROM Orders WHERE
CustomerID = 2;
```

```
no rows selected
```

MINUS:

```
SQL> SELECT CustomerID, Name, ContactInfo, ShippingAddress FROM Customer WHERE CustomerID = 1 MINUS SELECT CustomerID, Name, ContactInfo, ShippingAddress FROM Customer WHERE CustomerID = 2;


CUSTOMERID NAME                 CONTACTINFO
---------- -------------------- ------------------------------
SHIPPINGADDRESS
------------------------------
         1 Phoebe Buffay        555-1234
Apt 14, NY
```

CONCLUSION:

The inventory management system efficiently organizes and tracks inventory-related data including products, suppliers, warehouses, customers, orders, transactions, and employees. Leveraging a relational database and SQL queries, it ensures accurate inventory management, streamlined operations, and data-driven decision-making. With robust features such as product and supplier management, order tracking, and employee assignment, the system promotes efficiency, transparency, and customer satisfaction. Ongoing monitoring and optimization ensure adaptability to evolving business needs, driving organizational growth and success.