MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
Database Systems Lab (CSE 2262)

Inventory Management System: Streamlining Customer Orders and Warehouse Logistics

SUBMITTED
BY

Dhruv Bajaj CSE-A 38, 210905202
Lakshay Saxena CSE-A 59, 210905384

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.
April 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
12/05/2023

CERTIFICATE

This is to certify that the project titled **"Inventory Management System: Streamlining Customer Orders and Warehouse Logistics"** is a record of the bonafide work done by **Dhruv Bajaj (Reg. No. 210905202)** and **Lakshay Saxena (Reg. No. 210905384)** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Dr. Anup Bhat, CSE Dept.**

TABLE OF CONTENTS

ABSTRACT

This project seeks to create a database for an online electronics store with inventory management. The database will be designed using Structured Query Language (SQL) and will store information about the store's inventory, customers, orders, and more. The database will provide the store with a way to better track, manage, and analyze its inventory, allowing them to maximize efficiency and profits. Additionally, the database will be designed to be easily expandable and flexible, allowing the store to easily make changes to its inventory and customer information. The project will benefit the store by providing them with a reliable, secure, and efficient system to manage their inventory.

Introduction

This project aims to create a comprehensive and well-organized database system that meets the needs of an online electronics store. The store's success heavily relies on its ability to manage inventory efficiently and provide customers with high-quality services. With the help of this database system, the store will have a robust, secure, and scalable platform to manage their inventory, customers, and orders. The project will utilize Structured Query Language (SQL) to design a database that will be able to store and manage all the essential data, including product information, customer details, order history, and transaction records. The system will be designed with flexibility and expandability in mind to allow the store to grow and adapt to changing customer needs. The proposed database system will provide the store with several benefits, including accurate inventory tracking, real-time updates on stock levels, and effective order management. The system will enable the store to optimize its inventory management, ensuring that they have the right products available at the right time. With this database system in place, the store will be able to analyze customer data and generate insights to improve their marketing strategies and customer retention rates. In summary, this project will create a high-performance, secure, and scalable database system that will enable an online electronics store to manage their inventory and customers effectively. The system will provide the store with accurate inventory tracking, effective order management, and customer data analysis to enhance their operations and maximize their profits.

Problem Statement & Objectives

Online electronics stores have become increasingly popular in recent years, offering a wide range of products and services to customers. However, as these stores grow and expand their offerings, they often struggle with effectively managing their inventory and tracking transactions. This can lead to inefficient use of resources, lost sales opportunities, and dissatisfied customers.

Inventory management is a critical aspect of any online electronics store, as it directly affects the availability of products and the ability to meet customer demand. Without an efficient system for managing inventory, stores may face issues such as overstocking or stockouts, which can result in lost sales and decreased profits. In addition, managing inventory manually can be a time-consuming process that requires significant resources.

Customer management is another crucial aspect of online electronics stores. It is important to maintain accurate and up-to-date customer information to provide a personalized experience and maintain customer loyalty. Without an effective system for customer management, stores may struggle to keep track of customer orders, preferences, and purchase history, leading to missed sales opportunities and a decrease in customer satisfaction.

Transaction tracking is also a critical aspect of an online electronics store, as it allows for the accurate tracking of sales, returns, and refunds. Without an effective system for tracking transactions, stores may face issues such as inaccurate financial reporting, lost revenue, and legal compliance issues.

To address these challenges, our project will create a comprehensive database using Structured Query Language (SQL) to manage inventory, customers, and transactions. This database will provide a reliable, secure, and efficient system to manage the store's inventory, track customer orders and preferences, and accurately track transactions. By implementing this Solution, online electronics stores will be able to maximize efficiency, increase profits, and improve customer satisfaction. This project will involve the following features:

- Addition/Removal of Inventory
- Addition removal of new Customers
- Keeping track of previous users of our marketplace
- Viewing transactional history
- User Login Data

Methodology

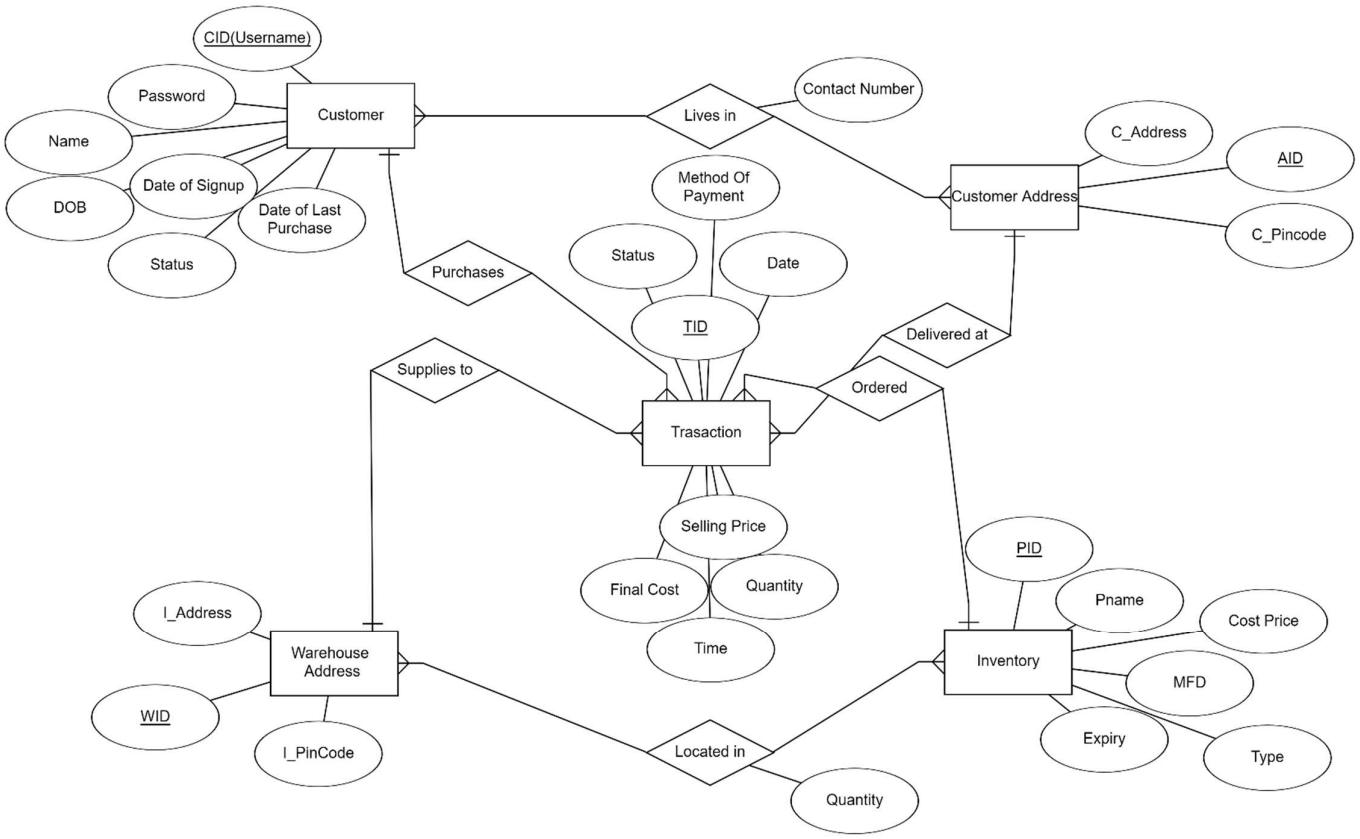
A comprehensive all-encompassing methodology was followed in order to create optimal schema and processes. A process of code, test then deploy was followed. Based on the gathered data, a detailed database schema was created to represent the various entities and their relationships. The team analyzed the requirements and identified the key entities, including customers, addresses, suppliers, orders, and payments. These entities were organized into separate tables, ensuring the appropriate attributes, constraints and relationships were defined. The schema consisted of primary keys, foreign keys, and the necessary constraints to maintain data integrity. At the same time attention was given to important real-life implications such as one individual having multiple addresses or a product being stored at multiple locations. In order to ensure data integrity, eliminate redundancy, and improve database performance, the tables in the inventory management system were normalized. Normalization is organizing data into efficient structures by minimizing data duplication and establishing relationships between entities. The normalization process involved analyzing the data requirements and applying the principles of normalization, specifically up to the Boyce-Codd normal form (BCNF). This involved breaking down tables into smaller, more focused entities and arranging them in a way that reduced redundancy. Overall, the methodology encompassed data gathering, schema design, and data modelling, all aimed at developing a robust inventory management system. By combining the insights gathered from stakeholders, a well-structured database schema was created, enabling efficient data management. Simultaneously, the data design process focused on delivering a performance focused experience that met the needs of modern professionals. The rigorous process of database design laid the foundation for an effective system that streamlines inventory management, sales tracking, and customer relationship management in the company.

In order to provide a better insight into the database system design process, the following diagrams and processes were used.

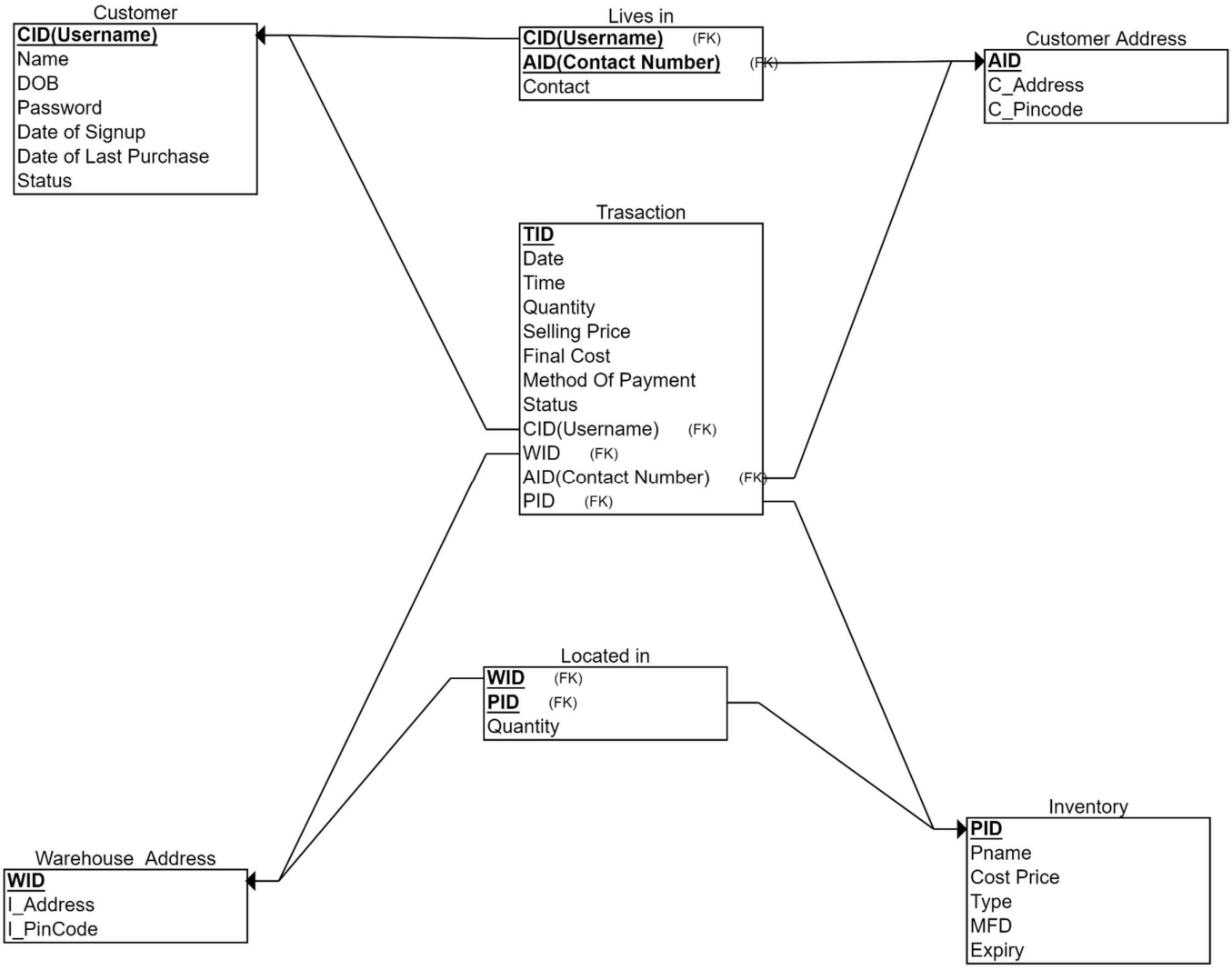
1. ER Diagram
2. Schema Diagram
3. List of Functional Dependencies
4. Normalization Process.

In the upcoming pages we shall take a deeper look into each one of these.

ER Diagram:



Schema Diagram:



List of Functional Dependencies:

- Customer:
  - Customer_ID -> Customer_Name
  - Customer_ID -> Date_of_Birth
  - Customer_ID -> Customer_Password
  - Customer_ID -> Date_of_Signup
  - Customer_ID -> Date_of_Last_Purchase
  - Customer_ID -> Customer_Status

- Inventory:
  - Product_ID -> Product_Name
  - Product_ID -> Cost_Price
  - Product_ID -> Prod_Type
  - Product_ID -> Manufacture_Date
  - Product_ID -> Expiry_Date

- Customer_Address:
  - Address_ID -> Customer_Address
  - Address_ID -> Customer_Pincode

- Warehouse_Address:
  - Warehouse_ID -> Warehouse_Address
  - Warehouse_ID -> Warehouse_PinCode

- Lives_in:
  - (Customer_ID, Address_ID) -> Contact

- Located_in:
  - (Warehouse_ID, Product_ID) -> Quantity

- Orders:
  - Transaction_ID -> Transaction_Date
  - Transaction_ID -> Transaction_Time
  - Transaction_ID -> Quantity_of_Sale
  - Transaction_ID -> Selling_Price
  - Transaction_ID -> Final_Cost
  - Transaction_ID -> Method_Of_Payment
  - Transaction_ID -> Order_Staus
  - Transaction_ID -> Customer_ID
  - Transaction_ID -> Warehouse_ID
  - Transaction_ID -> Address_ID
  - Transaction_ID -> Product_ID

Normalization Process:

1NF

All the tables are in First Normal Form (1NF), which means that each column has only atomic values, there are no repeating groups, the foreign keys, if present, reference primary keys of other tables and there are primary keys to identify each record uniquely. Since all the tables satisfy the requirements, all are in 1NF.

2NF

A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF). . Since all the tables satisfy the requirements, all are in 2NF.

3NF

To verify if a table is in 3NF, we need to check if it satisfies the following three conditions, firstly, the table is in second normal form (2NF). Secondly, there are no transitive dependencies. Finally, all non-key attributes are dependent on the primary key. Let's analyze each table to check if it meets these conditions.

Table Customer:

- The table has a primary key (Customer_ID).
- All the attributes are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Inventory:

- The table has a primary key (Product_ID).
- All the attributes are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Customer_Address:

- The table has a primary key (Address_ID).
- All the attributes are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Warehouse_Address:

- The table has a primary key (Warehouse_ID).
- All the attributes are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Lives_in:

- The table has a composite primary key (Customer_ID, Address_ID).
- All the non-key attributes (Contact) are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Located_in:

- The table has a composite primary key (Warehouse_ID, Product_ID).
- All the non-key attributes (Quantity) are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Table Orders:

- The table has a primary key (Transaction_ID).
- All non-key attributes are dependent on the primary key.
- There are no transitive dependencies.
- Therefore, the table satisfies the 3NF conditions.

Hence, all the provided tables satisfy the 3NF conditions.

BCNF

All the tables are in Boyce-Codd Normal Form (BCNF), which means they satisfy the requirements of BCNF. In BCNF, a relation is considered to be in BCNF if and only if every determinant is a candidate key. Let's go through each table and check if it satisfies the requirements of BCNF:

1. Customer table:

- The primary key is Customer_ID.
- There are no transitive dependencies since all the attributes depend on the primary key.
- Therefore, this table is in BCNF.

2. Inventory table:

- The primary key is Product_ID.
- There are no transitive dependencies since all the attributes depend on the primary key.
- Therefore, this table is in BCNF.

3. Customer_Address table:

- The primary key is Address_ID.
- There are no transitive dependencies since all the attributes depend on the primary key.
- Therefore, this table is in BCNF.

4. Warehouse_Address table:

- The primary key is Warehouse_ID.
- There are no transitive dependencies since all the attributes depend on the primary key.
- Therefore, this table is in BCNF.

5. Lives_in table:

- The primary key is (Customer_ID, Address_ID).
- Customer_ID and Address_ID are candidate keys, and there are no other determinants.
- Therefore, this table is in BCNF.

6. Located_in table:

- The primary key is (Warehouse_ID, Product_ID).
- Warehouse_ID and Product_ID are candidate keys, and there are no other determinants.
- Therefore, this table is in BCNF.

7. Orders table:

- The primary key is Transaction_ID.
- Customer_ID, Warehouse_ID, Address_ID, and Product_ID are candidate keys, and there are no other determinants.
- Therefore, this table is in BCNF.

In conclusion, all the tables are in BCNF.

Codes:

DDL Commands:

```
drop table Lives_in;
drop table Located_in;
drop table Orders;
Drop table Customer;
drop table Inventory;
drop table Customer_Address;
drop table Warehouse_Address;
drop view Customer_Lives_in;
drop view Inventory_Warehouse_Address;


CREATE TABLE Customer
(
  Customer_ID VARCHAR(30) NOT NULL,--Username is Customer_ID
  Customer_Name VARCHAR(30) NOT NULL,
  Date_of_Birth DATE NOT NULL,
  Customer_Password VARCHAR(30) NOT NULL,
  Date_of_Signup DATE NOT NULL,
  Date_of_Last_Purchase DATE NOT NULL,
  Customer_Status VARCHAR(30) NOT NULL,
  check (Customer_Status in ('Active','Inactive')),
  PRIMARY KEY (Customer_ID)
);

CREATE TABLE Inventory
(
  Product_ID NUMBER(5) NOT NULL,
  Product_Name VARCHAR(30) NOT NULL,--Dove
  Cost_Price NUMERIC(6,2) NOT NULL,
  Prod_Type VARCHAR(50) NOT NULL,--Soap
  Manufacture_Date DATE NOT NULL,
  Expiry_Date DATE NOT NULL,
  PRIMARY KEY (Product_ID)
);

CREATE TABLE Customer_Address
(
  Address_ID Number(5) NOT NULL,
  Customer_Address VARCHAR(50) NOT NULL,
  Customer_Pincode NUMBER(6) NOT NULL,
  PRIMARY KEY (Address_ID)
);

CREATE TABLE Warehouse_Address
(
  Warehouse_ID NUMBER(5) NOT NULL,
  Warehouse_Address VARCHAR(50) NOT NULL,
  Warehouse_PinCode NUMBER NOT NULL,
  PRIMARY KEY (Warehouse_ID)
);

CREATE TABLE Lives_in
```

```
(
  Contact NUMBER(10) NOT NULL,
  Customer_ID VARCHAR(30) NOT NULL,
  Address_ID Number(5) NOT NULL,
  PRIMARY KEY (Customer_ID, Address_ID),
  FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE,
  FOREIGN KEY (Address_ID) REFERENCES Customer_Address(Address_ID) ON DELETE
CASCADE
);

CREATE TABLE Located_in
(
  Quantity NUMBER(10) NOT NULL,
  Warehouse_ID NUMBER(5) NOT NULL,
  Product_ID NUMBER(5) NOT NULL,
  PRIMARY KEY (Warehouse_ID, Product_ID),
  FOREIGN KEY (Warehouse_ID) REFERENCES Warehouse_Address(Warehouse_ID) ON DELETE
CASCADE,
  FOREIGN KEY (Product_ID) REFERENCES Inventory(Product_ID) ON DELETE CASCADE
);

CREATE TABLE Orders
(
  Transaction_ID NUMBER(10) NOT NULL,
  Transaction_Date DATE NOT NULL,
  Transaction_Time DATE NOT NULL,
  Quantity_of_Sale NUMBER(5) NOT NULL,
  Selling_Price NUMERIC(6,2) NOT NULL,
  Final_Cost NUMERIC(6,2) NOT NULL,
  Method_Of_Payment VARCHAR(25) NOT NULL,
  Order_Staus VARCHAR(25) NOT NULL,
  Customer_ID VARCHAR(30) NOT NULL,
  Warehouse_ID NUMBER NOT NULL,
  Address_ID Number(5) NOT NULL,
  Product_ID NUMBER NOT NULL,
  PRIMARY KEY (Transaction_ID),
  FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE,
  FOREIGN KEY (Warehouse_ID) REFERENCES Warehouse_Address(Warehouse_ID) ON DELETE
CASCADE,
  FOREIGN KEY (Address_ID) REFERENCES Customer_Address(Address_ID) ON DELETE
CASCADE,
  FOREIGN KEY (Product_ID) REFERENCES Inventory(Product_ID) ON DELETE CASCADE,
  check(Method_Of_Payment in('UPI','COD','Debit Card','Credit Card','Net Banking') and Order_Staus
in('Fulfilled','Ordered','Shipped','Cancelled'))
);
CREATE VIEW Inventory_Warehouse_Address AS
SELECT i.Product_ID, i.Product_Name, i.Cost_Price, i.Prod_Type, i.Manufacture_Date, i.Expiry_Date,
      l.Quantity, w.Warehouse_ID, w.Warehouse_Address, w.Warehouse_PinCode
FROM Inventory i
JOIN Located_in l ON i.Product_ID = l.Product_ID
JOIN Warehouse_Address w ON l.Warehouse_ID = w.Warehouse_ID;

CREATE VIEW Customer_Lives_in AS
SELECT c.Customer_ID AS customer_id, c.Customer_Name AS customer_name, c.Date_of_Birth AS
date_of_birth, c.Customer_Password AS customer_password, c.Date_of_Signup AS date_of_signup,
c.Date_of_Last_Purchase AS date_of_last_purchase, c.Customer_Status AS customer_status,
      la.Address_ID AS address_id, la.Customer_Address AS customer_address, la.Customer_Pincode AS
customer_pincode,
```

```
    li.Contact AS contact
FROM Customer c
JOIN Lives_in li ON c.Customer_ID = li.Customer_ID
JOIN Customer_Address la ON li.Address_ID = la.Address_ID;
```

DML Commands:

```
--Customer
INSERT INTO Customer(Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
VALUES ('C001', 'John Doe', TO_DATE('15-JUN-90', 'DD-MON-YY'), 'johndoe123', TO_DATE('01-
JAN-21', 'DD-MON-YY'), TO_DATE('01-APR-23', 'DD-MON-YY'), 'Active');

INSERT INTO Customer(Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
VALUES ('C002', 'Jane Smith', TO_DATE('22-SEP-95', 'DD-MON-YY'), 'janesmith456', TO_DATE('15-
FEB-21', 'DD-MON-YY'), TO_DATE('01-MAY-23', 'DD-MON-YY'), 'Active');

INSERT INTO Customer(Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
VALUES ('C003', 'Tom Wilson', TO_DATE('08-APR-87', 'DD-MON-YY'), 'tomwilson789',
TO_DATE('31-DEC-20', 'DD-MON-YY'), TO_DATE('15-MAR-23', 'DD-MON-YY'), 'Inactive');

INSERT INTO Customer(Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
VALUES ('C004', 'Emily Davis', TO_DATE('17-NOV-98', 'DD-MON-YY'), 'emilydavis101',
TO_DATE('22-MAR-21', 'DD-MON-YY'), TO_DATE('30-APR-23', 'DD-MON-YY'), 'Active');

INSERT INTO Customer(Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
VALUES ('C005', 'David Lee', TO_DATE('10-AUG-92', 'DD-MON-YY'), 'davidlee222', TO_DATE('10-
JAN-21', 'DD-MON-YY'), TO_DATE('05-MAY-23', 'DD-MON-YY'), 'Inactive');

--Inventory
INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
VALUES (1, 'Dove Soap', 50.00, 'Soap', TO_DATE('2022-01-01', 'YYYY-MM-DD'), TO_DATE('2023-
12-31', 'YYYY-MM-DD'));

INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
VALUES (2, 'Colgate Toothpaste', 25.00, 'Toothpaste', TO_DATE('2022-01-01', 'YYYY-MM-DD'),
TO_DATE('2023-12-31', 'YYYY-MM-DD'));

INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
VALUES (3, 'Maggi Noodles', 20.00, 'Noodles', TO_DATE('2022-01-01', 'YYYY-MM-DD'),
TO_DATE('2023-12-31', 'YYYY-MM-DD'));

INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
VALUES (4, 'Tata Tea', 10.00, 'Tea', TO_DATE('2022-01-01', 'YYYY-MM-DD'), TO_DATE('2023-12-
31', 'YYYY-MM-DD'));

INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
VALUES (5, 'Lays Chips', 15.00, 'Chips', TO_DATE('2022-01-01', 'YYYY-MM-DD'), TO_DATE('2022-
12-31', 'YYYY-MM-DD'));
```

```
--Customer_Address
INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode)
VALUES (1, '123 Main St', 560001);

INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode)
VALUES (2, '456 Park Ave', 560002);

INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode)
VALUES (3, '789 Broadway', 560003);

INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode)
VALUES (4, '111 State St', 560004);

INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode)
VALUES (5, '222 Market St', 560005);

--Warehouse_Address
INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
VALUES (1, '10th Cross Road', 560001);

INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
VALUES (2, '5th Main Road', 560002);

INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
VALUES (3, '3rd Block', 560003);

INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
VALUES (4, '1st Avenue', 560004);

INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
VALUES (5, '7th Street', 560005);

--Lives_in
INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (1234567890, 'C001', 1);

INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (2345678901, 'C002', 2);

INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (3456789012, 'C003', 3);

INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (4567890123, 'C004', 4);

INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (5678901234, 'C005', 5);

--Located_in
INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID) VALUES (100, 1, 1);

INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID) VALUES (200, 2, 2);

INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID) VALUES (150, 3, 3);

INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID) VALUES (50, 4, 4);

INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID) VALUES (75, 5, 5);

--Orders
```

```
INSERT INTO Orders (Transaction_ID, Transaction_Date, Transaction_Time, Quantity_of_Sale,
Selling_Price, Final_Cost, Method_Of_Payment, Order_Staus, Customer_ID, Warehouse_ID,
Address_ID, Product_ID)
VALUES (1, TO_DATE('2023-05-10', 'YYYY-MM-DD'), TO_DATE('12:00:00', 'HH24:MI:SS'), 3,
50.00, 150.00, 'Debit Card', 'Ordered', 'C003', 1, 1, 1);

INSERT INTO Orders (Transaction_ID, Transaction_Date, Transaction_Time, Quantity_of_Sale,
Selling_Price, Final_Cost, Method_Of_Payment, Order_Staus, Customer_ID, Warehouse_ID,
Address_ID, Product_ID)
VALUES (2, TO_DATE('2023-05-09', 'YYYY-MM-DD'), TO_DATE('11:00:00', 'HH24:MI:SS'), 1,
25.00, 25.00, 'COD', 'Fulfilled', 'C004', 2, 3, 2);

INSERT INTO Orders (Transaction_ID, Transaction_Date, Transaction_Time, Quantity_of_Sale,
Selling_Price, Final_Cost, Method_Of_Payment, Order_Staus, Customer_ID, Warehouse_ID,
Address_ID, Product_ID)
VALUES (3, TO_DATE('2023-05-08', 'YYYY-MM-DD'), TO_DATE('10:00:00', 'HH24:MI:SS'), 5,
30.00, 150.00, 'Credit Card', 'Shipped', 'C003', 3, 5, 3);

INSERT INTO Orders (Transaction_ID, Transaction_Date, Transaction_Time, Quantity_of_Sale,
Selling_Price, Final_Cost, Method_Of_Payment, Order_Staus, Customer_ID, Warehouse_ID,
Address_ID, Product_ID)
VALUES (4, TO_DATE('2023-05-07', 'YYYY-MM-DD'), TO_DATE('15:00:00', 'HH24:MI:SS'), 2,
20.00, 40.00, 'Net Banking', 'Fulfilled', 'C001', 4, 4, 4);

INSERT INTO Orders (Transaction_ID, Transaction_Date, Transaction_Time, Quantity_of_Sale,
Selling_Price, Final_Cost, Method_Of_Payment, Order_Staus, Customer_ID, Warehouse_ID,
Address_ID, Product_ID)
VALUES (5, TO_DATE('2023-05-06', 'YYYY-MM-DD'), TO_DATE('14:00:00', 'HH24:MI:SS'), 1,
15.00, 15.00, 'COD', 'Cancelled', 'C002', 5, 2, 5);
--1
CREATE OR REPLACE TRIGGER Customer_Lives_in_trigger
INSTEAD OF INSERT ON Customer_Lives_in
FOR EACH ROW
DECLARE
  v_Customer_ID VARCHAR(30);
  v_Quantity NUMBER(10);
  v_Address_ID Number(5);
  v_wh_exist Number(10);
BEGIN
  SELECT count(Customer_ID) INTO v_wh_exist FROM Customer WHERE Customer_ID =
:new.Customer_ID;
  IF v_wh_exist = 0 THEN
    INSERT INTO Customer (Customer_ID, Customer_Name, Date_of_Birth, Customer_Password,
Date_of_Signup, Date_of_Last_Purchase, Customer_Status)
    VALUES (:new.Customer_ID, :new.Customer_Name, :new.Date_of_Birth, :new.Customer_Password,
:new.Date_of_Signup, :new.Date_of_Last_Purchase, :new.Customer_Status);
  END IF;
--   SELECT Quantity, Warehouse_ID INTO v_Quantity, v_Warehouse_ID FROM Located_in WHERE
Product_ID = :new.Product_ID;
  SELECT count(Address_ID) INTO v_wh_exist FROM Customer_Address WHERE Address_ID =
:new.Address_ID;
  IF v_wh_exist = 0 THEN
    INSERT INTO Customer_Address (Address_ID, Customer_Address, Customer_Pincode) VALUES
    (:new.address_id, :new.Customer_Address, :new.Customer_Pincode);
    v_Quantity := 0;
  END IF;
  select count(Contact) into v_wh_exist FROM Lives_in where Address_ID = :new.Address_ID and
Customer_ID = :new.Customer_ID;
```

```
 IF v_wh_exist = 0 THEN
   INSERT INTO Lives_in (Contact, Customer_ID, Address_ID) VALUES (:new.Contact,
:new.Customer_ID,:new.Address_id);
 ELSE
   UPDATE Lives_in SET Contact = :new.Contact where Address_ID = :new.Address_ID and
Customer_ID = :new.Customer_ID;
 END IF;
END;
/
--2
CREATE OR REPLACE TRIGGER update_customer_status
Before INSERT ON Orders
FOR EACH ROW
DECLARE
 last_order_date DATE;
 three_months_ago DATE := add_months(sysdate, -3);
 CURSOR customer_cur IS SELECT * FROM Customer for update;
 customer_rec Customer%ROWTYPE;
BEGIN
 -- Check if the customer exists in the Customer table
 SELECT MAX(Transaction_Date) INTO last_order_date
 FROM Orders
 WHERE Customer_ID = :NEW.Customer_ID;
   -- Update the customer's date of last purchase
   UPDATE Customer
   SET Date_of_Last_Purchase = sysdate
   WHERE Customer_ID = :NEW.Customer_ID;
  OPEN customer_cur;
 LOOP
   FETCH customer_cur INTO customer_rec;
   EXIT WHEN customer_cur%NOTFOUND;
   IF customer_rec.Date_of_Last_Purchase >= three_months_ago THEN
     UPDATE Customer SET Customer_Status = 'Active'
     WHERE CURRENT OF customer_cur;
   ELSE
     UPDATE Customer SET Customer_Status = 'Inactive'
     WHERE CURRENT OF customer_cur;
   END IF;
 END LOOP;
 CLOSE customer_cur;
END;
/
--3
CREATE OR REPLACE TRIGGER Inventory_Warehouse_Address_trigger
INSTEAD OF INSERT ON Inventory_Warehouse_Address
FOR EACH ROW
DECLARE
v_Cust_ID NUMBER(5);
 v_Quantity NUMBER(10);
 v_Warehouse_ID NUMBER(5);
 v_wh_exist Number(10);
BEGIN
 SELECT count(Product_ID) INTO v_wh_exist FROM Inventory WHERE Product_ID =
:new.Product_ID;
 IF v_wh_exist = 0 THEN
   INSERT INTO Inventory (Product_ID, Product_Name, Cost_Price, Prod_Type, Manufacture_Date,
Expiry_Date)
```

```
    VALUES (:new.Product_ID, :new.Product_Name, :new.Cost_Price, :new.Prod_Type,
:new.Manufacture_Date, :new.Expiry_Date);
  END IF;
--   SELECT Quantity, Warehouse_ID INTO v_Quantity, v_Warehouse_ID FROM Located_in WHERE
Product_ID = :new.Product_ID;
  SELECT count(Warehouse_ID) INTO v_wh_exist FROM warehouse_address WHERE warehouse_id =
:new.warehouse_id;
  IF v_wh_exist = 0 THEN
    INSERT INTO Warehouse_Address (Warehouse_ID, Warehouse_Address, Warehouse_PinCode)
    VALUES (:new.Warehouse_ID, :new.Warehouse_Address, :new.Warehouse_PinCode);
    v_Quantity := 0;
  ELSE
    SELECT Quantity INTO v_Quantity FROM Located_in WHERE Product_ID = :new.Product_ID;
  END IF;

  IF v_Quantity = 0 THEN
    INSERT INTO Located_in (Quantity, Warehouse_ID, Product_ID)
    VALUES (:new.Quantity, :new.Warehouse_ID, :new.Product_ID);
  ELSE
    UPDATE Located_in SET Quantity = Quantity + :new.Quantity WHERE Product_ID =
:new.Product_ID;
  END IF;
END;
/
--4
CREATE OR REPLACE TRIGGER remove_expired_inventory
BEFORE INSERT ON Orders
FOR EACH ROW
BEGIN
  FOR i IN (SELECT * FROM Inventory)
  LOOP
    IF i.Expiry_Date < TRUNC(SYSDATE) THEN
       DELETE FROM Located_in WHERE Product_ID = i.Product_ID;
      DELETE FROM Inventory WHERE Product_ID = i.Product_ID;

    END IF;
  END LOOP;
END;
/
--5
CREATE OR REPLACE TRIGGER cancel_order_trigger
BEFORE UPDATE OF Order_Staus ON Orders
FOR EACH ROW
BEGIN
  IF :OLD.Order_Staus = 'Fulfilled' AND :NEW.Order_Staus = 'Cancelled' THEN
  RAISE_APPLICATION_ERROR(-20001, 'Cannot cancel the order as its already been delivered.');
  END IF;
END;
/
--6
CREATE OR REPLACE TRIGGER update_warehouse_qty
Before INSERT ON orders
FOR EACH ROW
BEGIN
  UPDATE located_in
  SET quantity = quantity - :new.quantity_of_sale
  WHERE warehouse_id = :new.warehouse_id
  AND product_id = :new.product_id;
```

```
END;
/
--Write a query to fetch the details of all the customers along with their transaction history.
SELECT C.Customer_ID, C.Customer_Name, O.Transaction_ID, O.Transaction_Date,
O.Transaction_Time, O.Quantity_of_Sale, O.Selling_Price, O.Final_Cost, O.Method_Of_Payment,
O.Order_Staus
FROM Customer C
JOIN Orders O ON C.Customer_ID = O.Customer_ID;
--Write a query to fetch the details of all the customers who have not made any purchases yet.
SELECT C.Customer_ID, C.Customer_Name
FROM Customer C
WHERE NOT EXISTS (
   SELECT 1 FROM Orders O WHERE C.Customer_ID = O.Customer_ID
);
--Write a query to fetch the details of all the customers who made a purchase on or after 1st January
2023.
SELECT C.Customer_ID, C.Customer_Name, O.Transaction_Date
FROM Customer C
JOIN Orders O ON C.Customer_ID = O.Customer_ID
WHERE O.Transaction_Date >= TO_DATE('01-JAN-23', 'DD-MON-YY');
--Write a query to fetch the details of all the customers who made a purchase from a warehouse located in
pincode 123456.
SELECT C.Customer_ID, C.Customer_Name, O.Transaction_Date
FROM Customer C
JOIN Orders O ON C.Customer_ID = O.Customer_ID
JOIN Warehouse_Address W ON O.Warehouse_ID = W.Warehouse_ID
WHERE W.Warehouse_PinCode = 123456;
--Write a query to fetch the details of all the products that have not been sold yet.
SELECT I.Product_ID, I.Product_Name, I.Cost_Price, I.Prod_Type, I.Manufacture_Date, I.Expiry_Date
FROM Inventory I
WHERE NOT EXISTS (
   SELECT 1 FROM Orders O WHERE I.Product_ID = O.Product_ID
);
--Write a query to fetch the details of all the products that have been sold at least once.
SELECT DISTINCT I.Product_ID, I.Product_Name, I.Cost_Price, I.Prod_Type, I.Manufacture_Date,
I.Expiry_Date
FROM Inventory I
JOIN Orders O ON I.Product_ID = O.Product_ID;
--Write a query to fetch the details of all the products that are currently in stock in a warehouse located in
pincode 123456.
SELECT I.Product_ID, I.Product_Name, I.Cost_Price, I.Prod_Type, I.Manufacture_Date, I.Expiry_Date,
L.Quantity
FROM Inventory I
JOIN Located_in L ON I.Product_ID = L.Product_ID
JOIN Warehouse_Address W ON L.Warehouse_ID = W.Warehouse_ID
WHERE W.Warehouse_PinCode = 123456;
--Write a query to fetch the details of all the customers who live in the same address.
SELECT C1.Customer_ID, C1.Customer_Name, C2.Customer_ID, C2.Customer_Name,
CA.Customer_Address, CA.Customer_Pincode
FROM Customer C1
JOIN Lives_in L1 ON C1.Customer_ID = L1.Customer_ID
JOIN Customer_Address CA ON L1.Address_ID = CA.Address_ID
JOIN Lives_in L2 ON CA.Address_ID = L2.Address_ID
JOIN Customer C2 ON L2.Customer_ID = C2.Customer_ID
WHERE C1.Customer_ID <> C2.Customer_ID;
--Query to find the details of all the customers who have made purchases:
SELECT *
FROM Customer
```

```
WHERE Customer_ID IN (SELECT DISTINCT Customer_ID FROM Orders);
--Query to find the total revenue generated from each method of payment:
SELECT Method_Of_Payment, SUM(Final_Cost) AS Revenue
FROM Orders
GROUP BY Method_Of_Payment;
--Query to find the total quantity of each product sold:
SELECT Product_Name, SUM(Quantity_of_Sale) AS Total_Quantity
FROM Inventory
JOIN Orders ON Inventory.Product_ID = Orders.Product_ID
GROUP BY Product_Name
--Query to find the address of the warehouse with the largest inventory:
SELECT Warehouse_Address, Warehouse_PinCode
FROM Warehouse_Address
WHERE Warehouse_ID = (
  SELECT Warehouse_ID
  FROM Located_in
  GROUP BY Warehouse_ID
  ORDER BY SUM(Quantity) DESC
  FETCH FIRST ROW ONLY
);
--Query to find the customer who has made the most purchases:
SELECT Customer.Customer_Name, COUNT(*) AS Num_Purchases
FROM Orders
JOIN Customer ON Orders.Customer_ID = Customer.Customer_ID
GROUP BY Customer.Customer_Name
ORDER BY Num_Purchases DESC
FETCH FIRST ROW ONLY;
```

Conclusion:

In summary, the proposed SQL project aims to establish an efficient inventory management system that would greatly benefit companies having all types of inventories and storage. The system allows easy access to critical data such as customer information, supplier details and inventory status, streamlining daily operations and enhancing productivity. The improved efficiency can lead to increased customer satisfaction by ensuring that necessary products are always available. The system's potential benefits highlight the importance of implementing an inventory management system in companies. By investing in this technology, businesses can improve their services and overall performance, allowing them to compete effectively in today's competitive retail environment. The system's automation of manual processes reduces errors, improves accuracy, and speeds up transactions, allowing employees to concentrate on more critical tasks such as customer service.

In addition to enhancing productivity, the system can also help businesses identify trends and patterns in sales, inventory, and customer behavior. By gaining insights into consumer preferences, businesses can optimize their product offerings and promotional campaigns to better meet customer needs. This proactive approach to customer engagement can foster long-term customer loyalty, which can positively impact the bottom line. In conclusion, implementing the proposed inventory management system can provide companies and businesses with several benefits, including increased productivity, improved customer satisfaction, and enhanced insights into sales and consumer behavior. The adoption of this technology can help businesses stay competitive in today's retail environment and achieve long-term growth and success.

Limitations:

Implementing an inventory management system can significantly benefit a company by improving efficiency, reducing costs, and increasing accuracy. However, there are limitations and challenges that companies may face when implementing such a system. These include cost, integration with existing systems, data quality, employee training, maintenance, scalability, security, system downtime, human error, and compatibility. Small businesses may find it expensive to invest in the necessary hardware, software, and infrastructure for inventory management systems. Integrating inventory management systems with existing software and systems can be challenging, especially for companies using outdated or incompatible systems. The accuracy and reliability of data are crucial for inventory management

systems, and poor data quality, incomplete or inaccurate data, or data entry errors can lead to incorrect inventory levels and delays in order fulfillment. Proper employee training is required to use the inventory management system effectively. Maintenance, updates, and troubleshooting are required for inventory management systems, and the system may need to be updated or replaced as a company grows. It is essential to ensure the system is secure to protect against data breaches and other security threats. Technical issues or system failures can result in downtime, which can be costly and disruptive to the business. Even with an inventory management system, human error can still occur, such as miscounts or incorrect data entry, which can lead to incorrect inventory levels and other issues. Lastly, some inventory management systems may not be compatible with specific types of products or industries, which can limit their usefulness for some businesses.

Future Work:

In the future, to enhance the functionality of this SQL project, there are various options available. The first suggestion is to integrate data visualization tools, such as Tableau or Power BI, to allow users to gain insights from data more easily through interactive visualizations. Another way to improve the project is to use cloud storage options, such as Google Cloud Storage, to store and manage data more efficiently. The project can also be updated with advanced data analytics techniques, like machine learning algorithms, to provide predictive analytics and recommendations for customers. Better encryption techniques, such as AES or RSA, can be used to secure the project and prevent data breaches. Real-time updates can also be integrated to provide more accurate and up-to-date information. Additionally, integration with mobile apps and chatbots can improve customer service, while integration with social media platforms can increase brand awareness. Overall, these updates can enhance the project's value to customers and businesses alike.

References:

Though this code truly gave us a lot of insight into how SQL works, it would be inappropriate to say that we were responsible for 100% percent of the code. Credit where credit is due, the following resources were priceless during the software development process:

1. Oracle Documentation: Oracle provides extensive documentation on SQL and its various features, as well as tutorials and guides on how to use Oracle databases.

2. SQL for Data Analysis – Tutorial for Beginners: This tutorial on the Mode Analytics website provides a comprehensive introduction to SQL for data analysis, including examples and exercises.
3. SQL Tutorial: This tutorial on W3Schools provides a basic introduction to SQL, including syntax and basic concepts.
4. Oracle SQL Developer: Oracle SQL Developer is a powerful tool for working with Oracle databases, and its website provides a range of resources, including documentation, tutorials, and a community forum.
5. Oracle-Base: This website provides a wealth of resources for Oracle developers, including articles, tutorials, and sample code.
6. Oracle Community: The Oracle Community is a forum for Oracle developers and users, where they can exchange ideas, ask questions, and find resources on a wide range of Oracle-related topics, including SQL.