Stationery Store Management System

By

M. Umer Riaz

To

Mr. Irfan Hameed

May 22, 2025

Department of Computer and Information Sciences

PAKISTAN INSTITUTE OF ENGINEERING AND APPLIED SCIENCES

NILORE, ISLAMABAD 45650

TABLE OF CONTENTS

INTRODUCTION

In the dynamic world of retail, data is the lifeblood of successful operations. This project represents a transformative approach to managing a physical stationery store's complex ecosystem through an advanced database solution. By creating an integrated system that captures every nuance of business operations, we empower the store to transcend traditional management limitations.

The database serves as a comprehensive digital nervous system, interconnecting critical business domains such as customer interactions, product management, inventory control, and organizational relationships. It's not just a technical solution, but a strategic tool for business optimization.

PROBLEM STATEMENT

The stationery store's existing operational model was fraught with critical challenges that hindered its potential for growth and efficiency:

- Data fragmentation creating information silos
- Manual tracking leading to frequent human errors
- Inability to generate timely and accurate business insights
- Disconnected systems preventing holistic business understanding
- Lack of real-time visibility into inventory and sales performance

These systemic issues created a significant drag on operational effectiveness, making it challenging to make informed business decisions and respond quickly to market dynamics.

OBJECTIVES

The project's strategic objectives were carefully crafted to address the store's comprehensive technological and operational needs:

- Develop a robust, scalable database architecture that can evolve with business requirements
- Create intricate, meaningful relationships between diverse business entities
- Implement sophisticated data integrity and validation mechanisms
- Establish a dynamic framework for instantaneous data retrieval and advanced reporting
- Transform raw transactional data into actionable business intelligence
- Reduce operational friction through intelligent automation

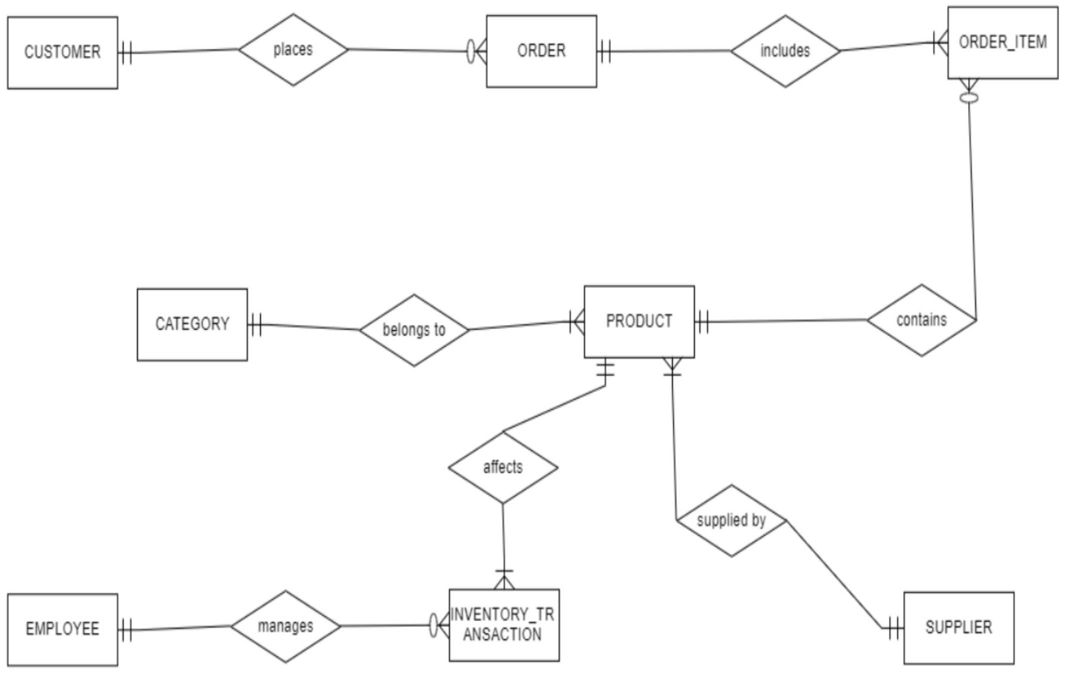
Business Rules

- Each customer must have a unique identification number.
- A customer must be registered to place orders.
- Each order must have a unique identification number.
- An order must include at least one item.
- The quantity of any order item must be a positive value.
- Each product must have a unique identification number.
- Product stock quantity cannot be negative at any time.
- Any product price change must be logged before updating to the new price.
- Sales transactions must reduce the product's stock quantity.
- A sale cannot occur if the product stock is insufficient.
- Each product must have only one supplier.
- A supplier must provide at least one product to remain active.

- Only administrators can update or delete product information.
- Salespersons are only allowed to manage orders and customer records.
- Managers have read-only access to data for reporting purposes.
- Required fields must always contain valid information.
- Referential integrity between tables must be maintained at all times.
- Orders cannot be processed without complete customer details.
- The total quantity in an order must not exceed the available product stock.

SYSTEM DESIGN

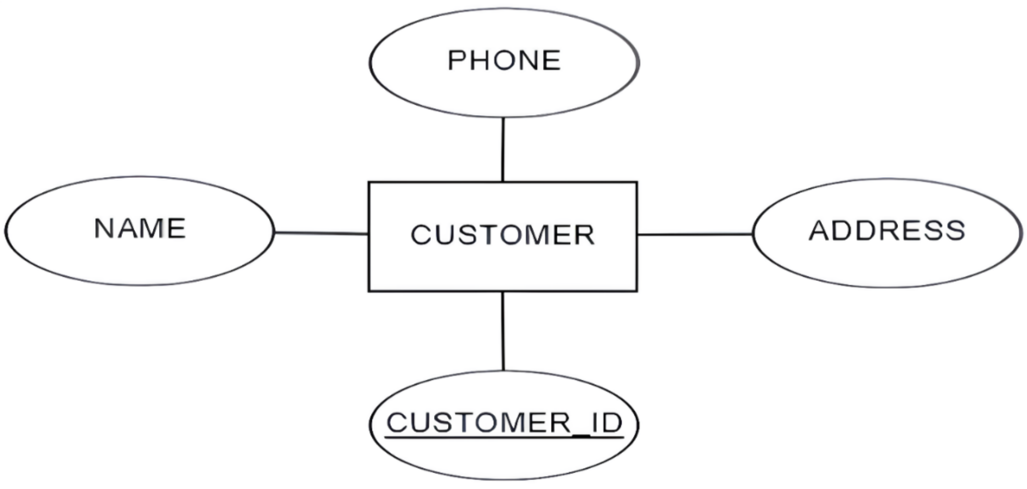
Enterprise Data Model



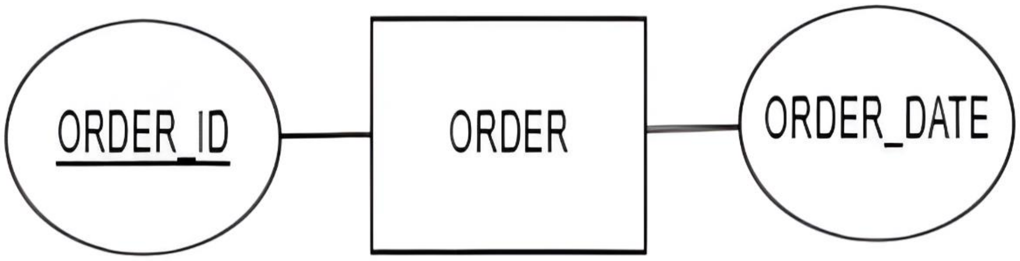
ENTITY RELATIONSHIP DIAGRAMS(ERD)

The database architecture was conceptualized around eight core entities, each representing a critical aspect of the stationery store's operational ecosystem:

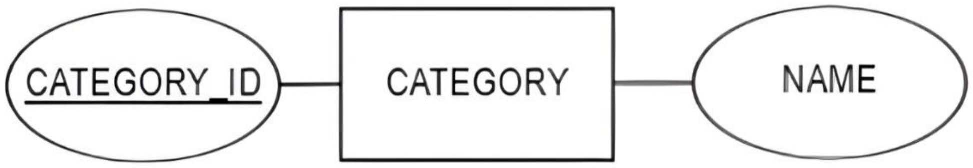
CUSTOMER: The human element, capturing comprehensive interaction histories



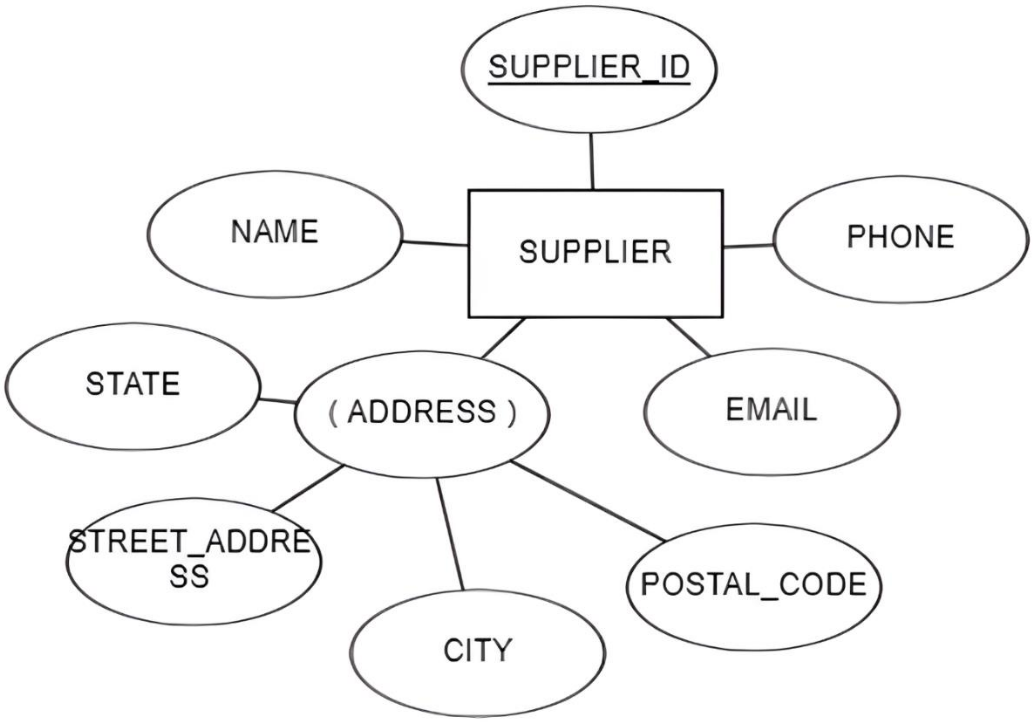
ORDER_T: Transactional records that form the revenue backbone



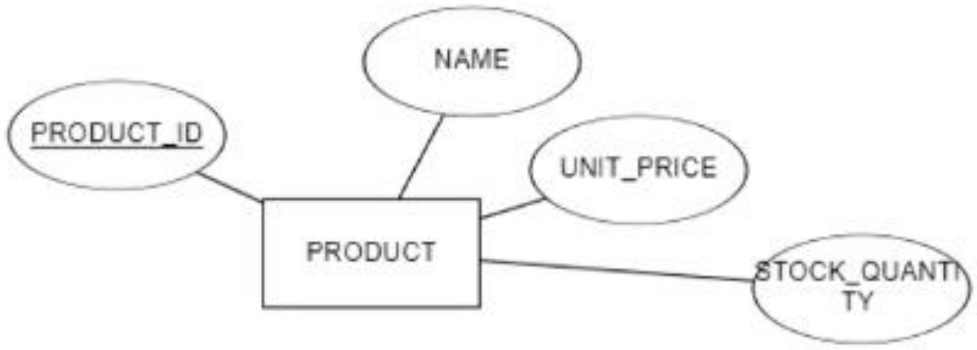
CATEGORY: Intelligent product classification and organizational framework



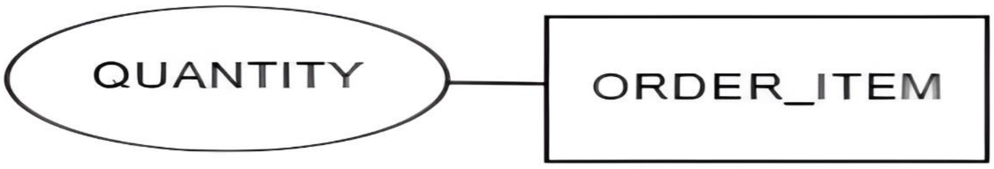
SUPPLIER: External partnership management and supply chain tracking



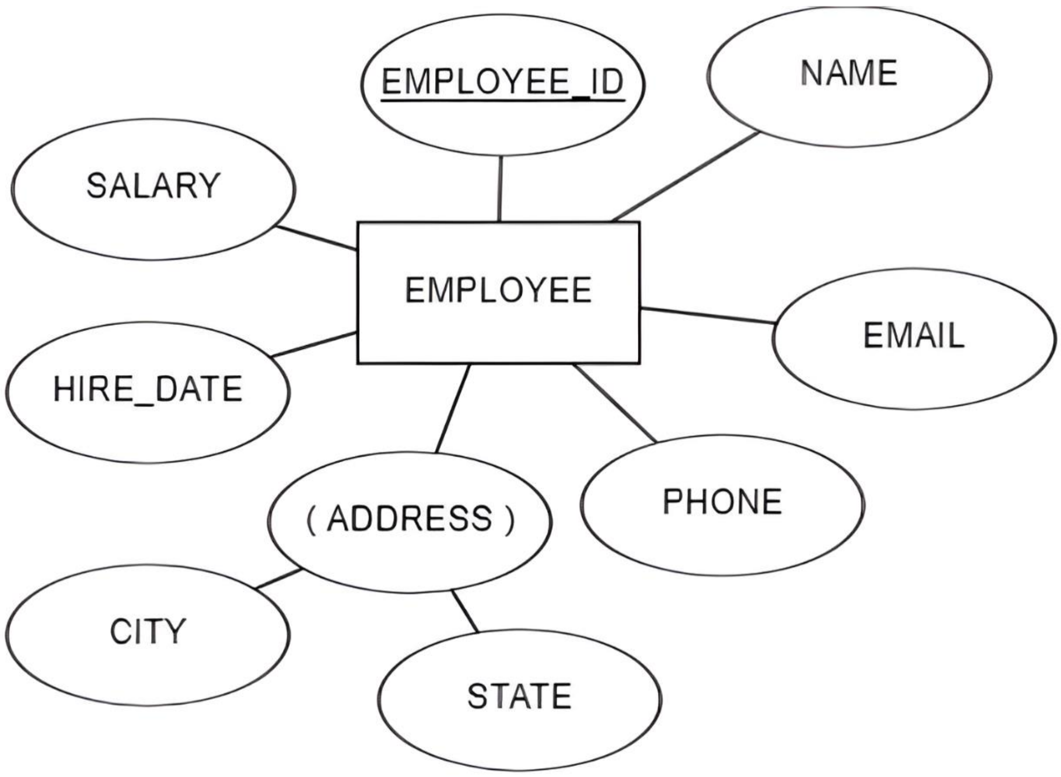
PRODUCT: The core merchandise layer with detailed attributional metadata



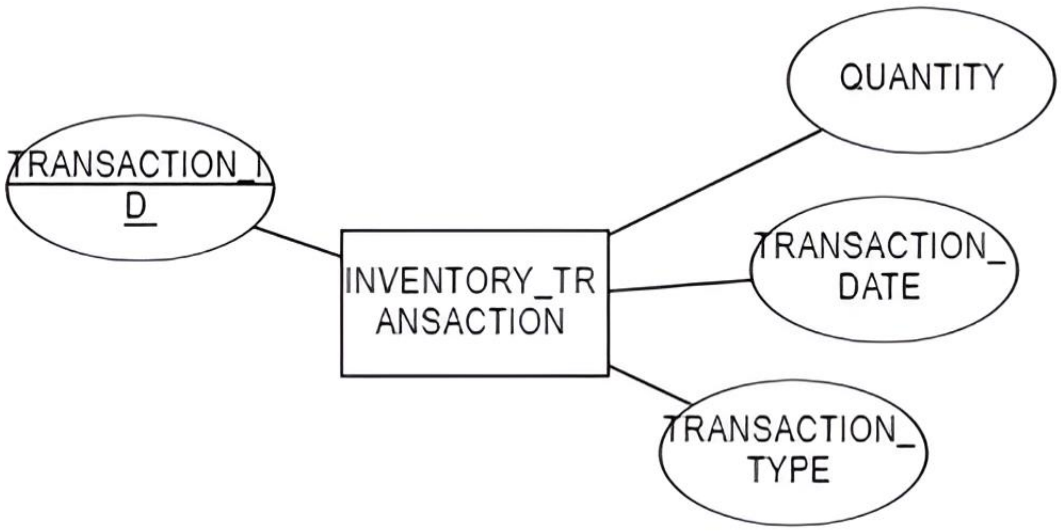
ORDER_ITEM: Granular transactional details capturing precise purchase compositions



EMPLOYEE: Workforce management and operational role tracking



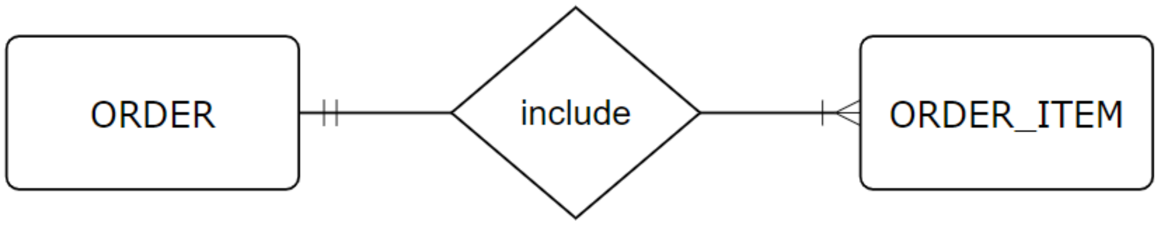
INVENTORY_TRANSACTION: Dynamic stock movement and tracking mechanism



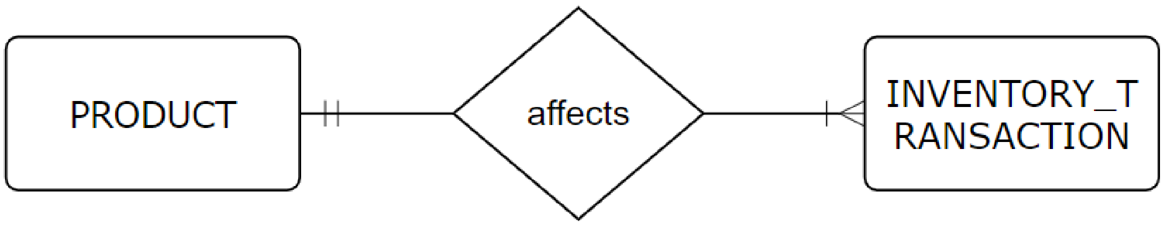
ENTITY RELATIONS

Each relationship represents a sophisticated data interaction:

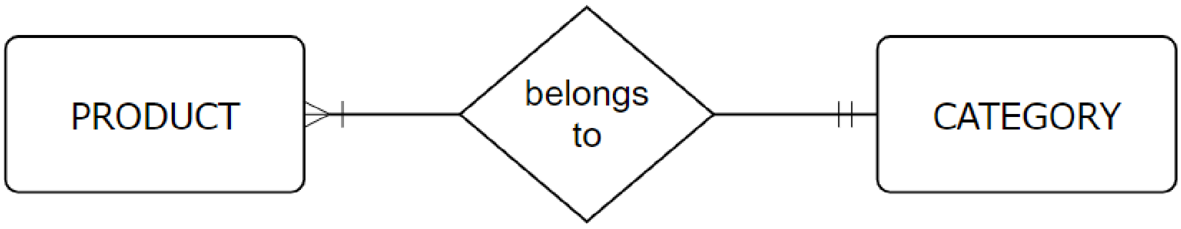
ORDER AND ORDER_ITEM: This relationship links Order and Order_Item, where an order (Order_ID) contains multiple items (Product_ID, Quantity). It ensures accurate tracking of order details, including item quantities and associated prices.



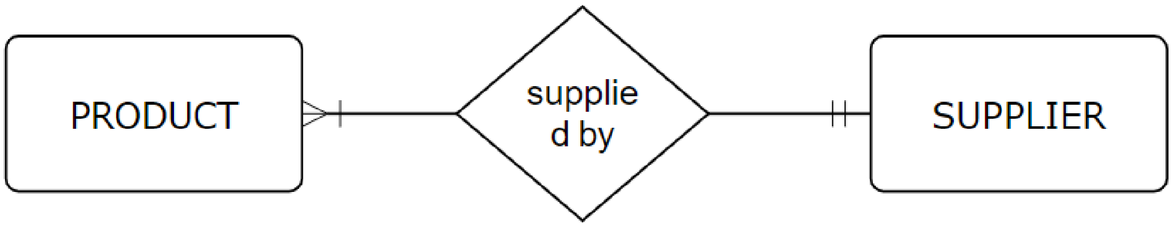
PRODUCT AND INVENTORY_TRANSACTION: Connects Product to Inventory_Transaction, where stock movements like Sale or Restock update Stock_Quantity.



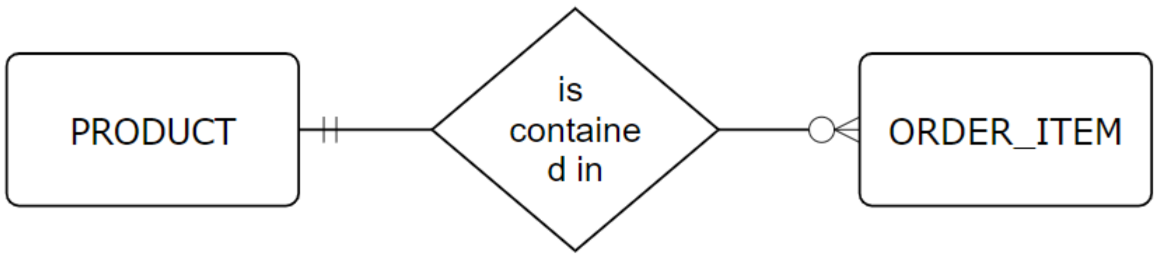
PRODUCT AND CATEGORY: Links Product to Category using Category_ID. Each product belongs to a single category, while a category can have more than one products.



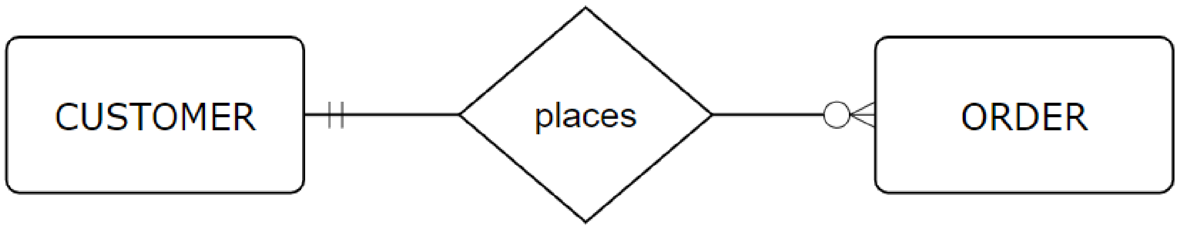
PRODUCT AND SUPPLIER: Associates Product with Supplier through Supplier_ID to streamline sourcing. There is mandatory for a supplier to provide a supply for entry in a table.



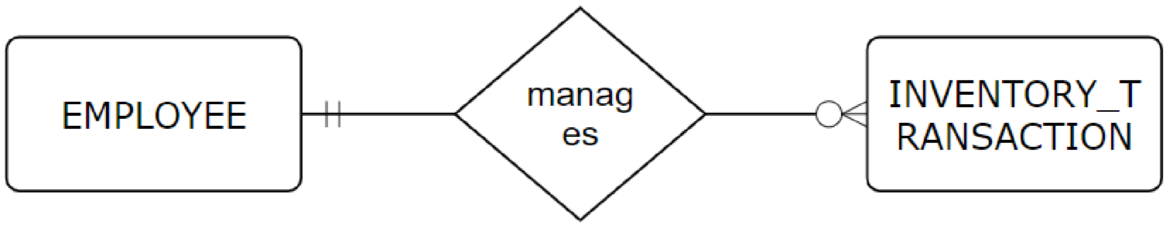
PRODUCT AND ORDER_ITEM: Relates Product to Order_Item, where Product_ID tracks items in orders. It ensures accurate management of quantities (Quantity) and product sales across different orders.



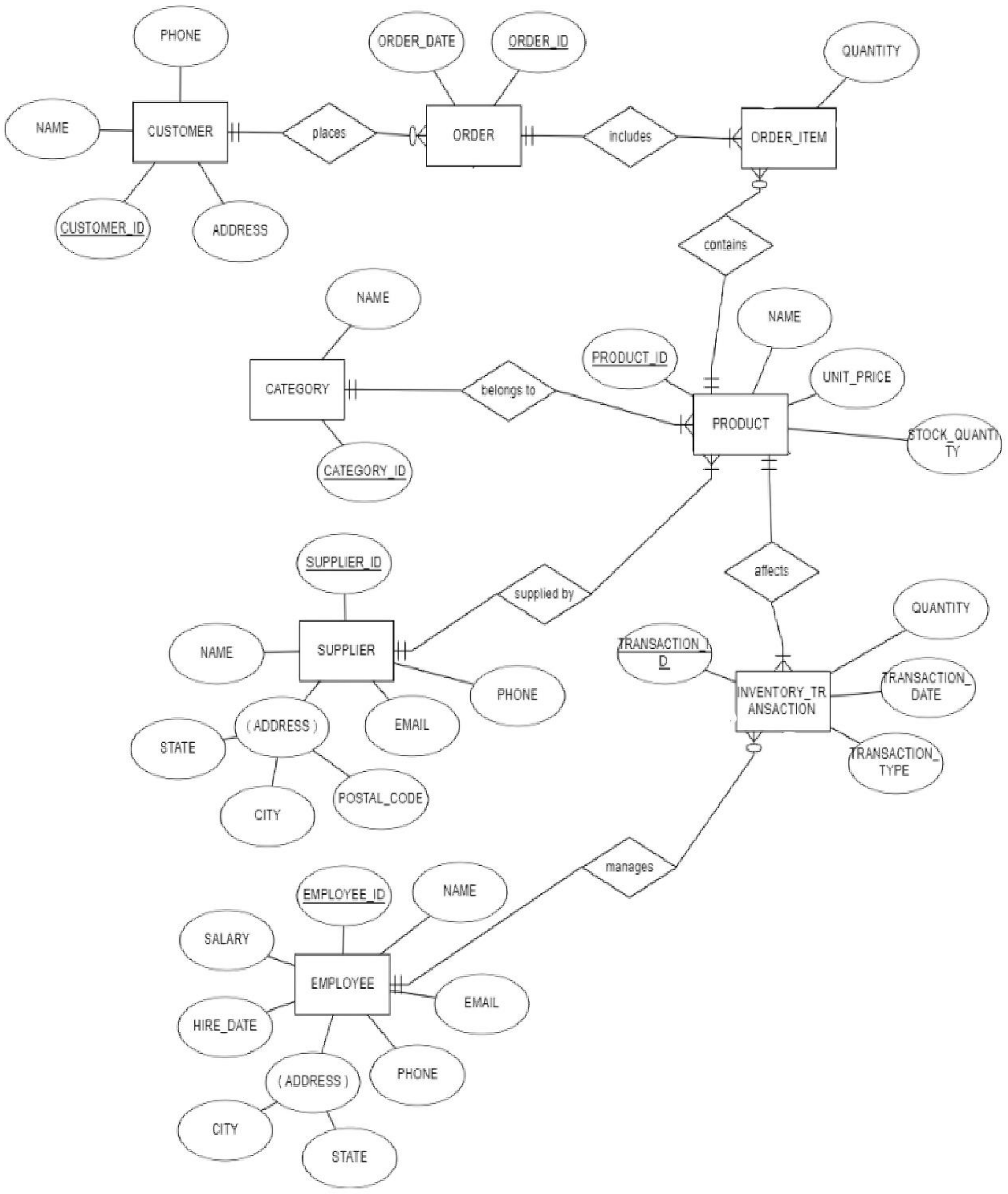
CUSTOMER AND ORDER: Links Customer with Order through Customer_ID, where each customer can place one or many orders.



EMPLOYEE AND INVENTORY_TRANSACTION: Connects Employee to Inventory_Transaction using Employee_ID to track stock movements performed by employees.

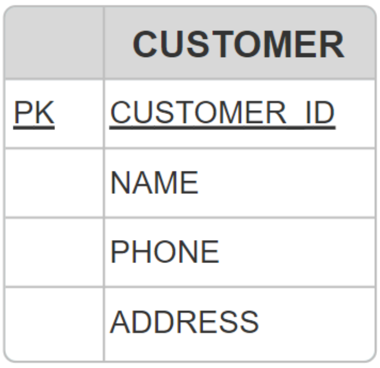


Complete ER DIAGRAM



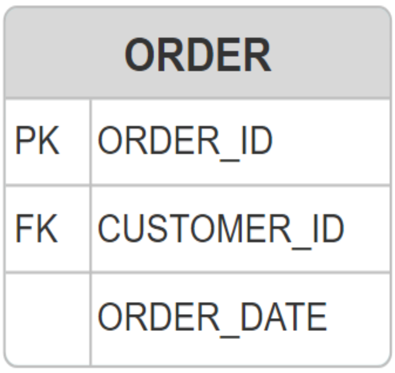
RELATIONAL SCHEMAS

CUSTOMER:



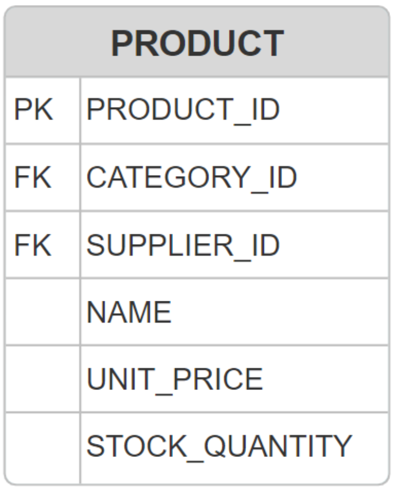
	CUSTOMER
PK	CUSTOMER_ID
	NAME
	PHONE
	ADDRESS

ORDER:



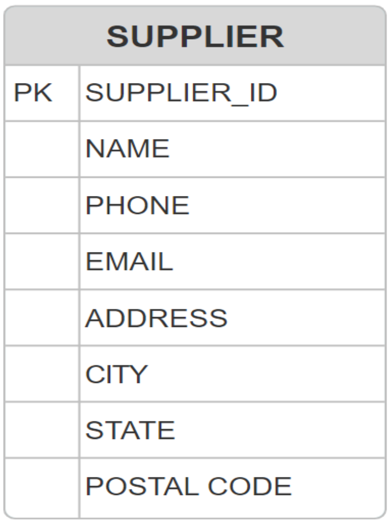
	ORDER
PK	ORDER_ID
FK	CUSTOMER_ID
	ORDER_DATE

PRODUCT:



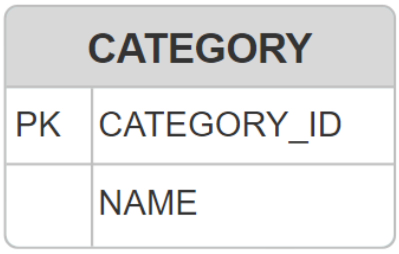
	PRODUCT
PK	PRODUCT_ID
FK	CATEGORY_ID
FK	SUPPLIER_ID
	NAME
	UNIT_PRICE
	STOCK_QUANTITY

SUPPLIER:



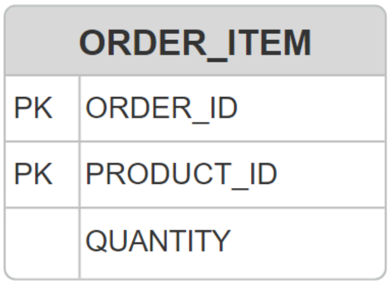
	SUPPLIER
PK	SUPPLIER_ID
	NAME
	PHONE
	EMAIL
	ADDRESS
	CITY
	STATE
	POSTAL CODE

CATEGORY:



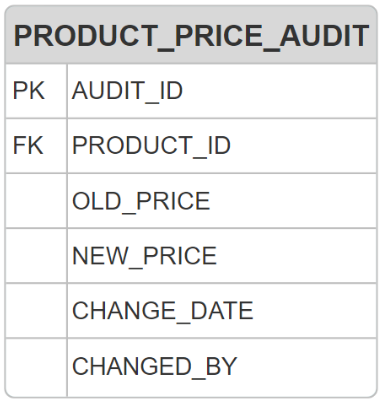
	CATEGORY
PK	CATEGORY_ID
	NAME

ORDER_ITEM:



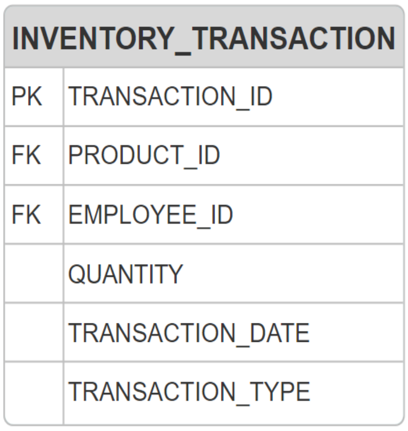
	ORDER_ITEM
PK	ORDER_ID
PK	PRODUCT_ID
	QUANTITY

PRODUCT_PRICE_AUDIT:



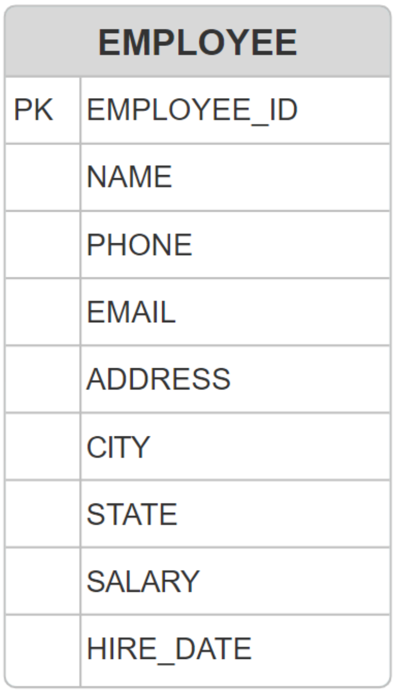
PRODUCT_PRICE_AUDIT	
PK	AUDIT_ID
FK	PRODUCT_ID
	OLD_PRICE
	NEW_PRICE
	CHANGE_DATE
	CHANGED_BY

INVENTORY_TRANSACTION:



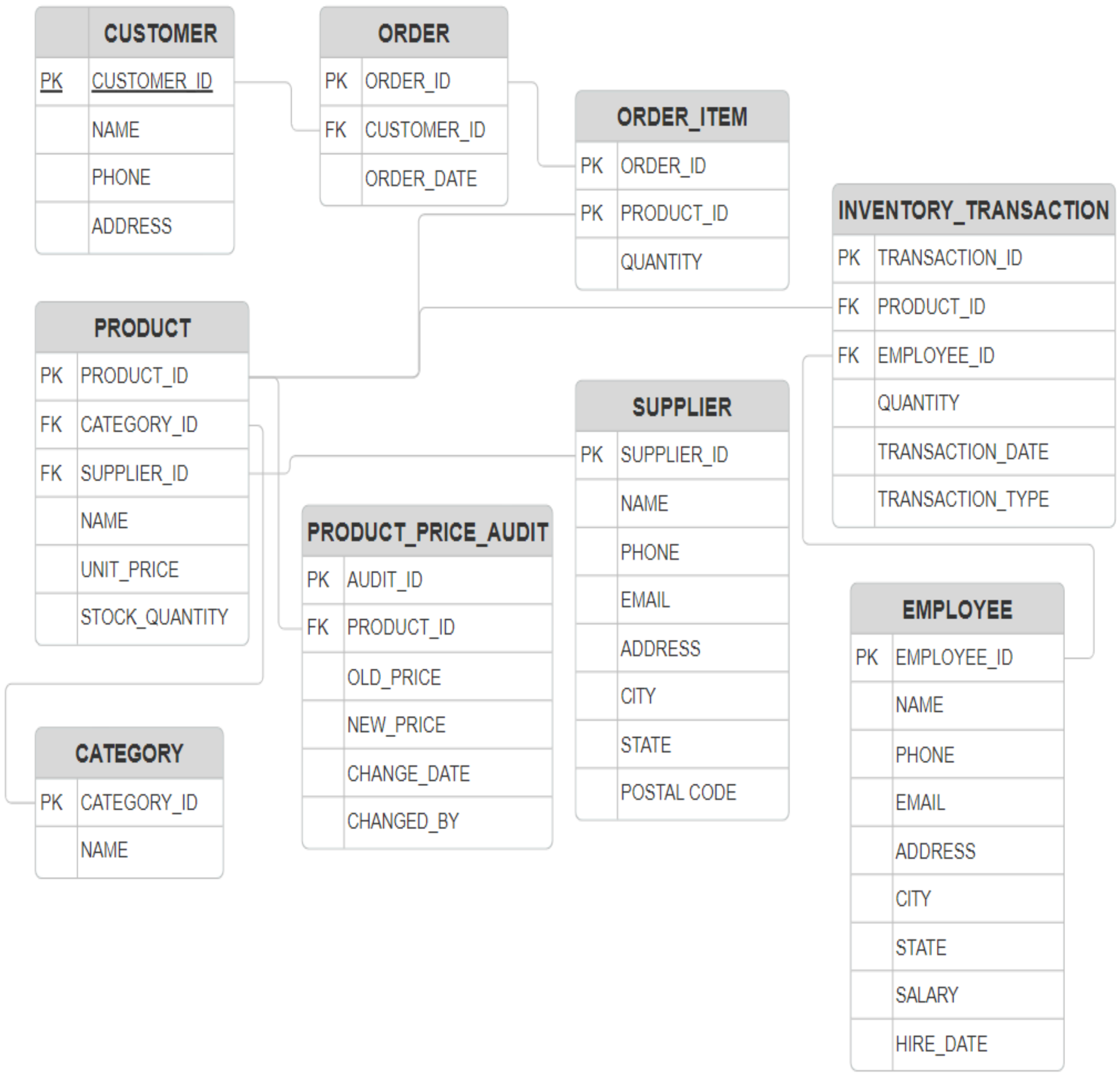
INVENTORY_TRANSACTION	
PK	TRANSACTION_ID
FK	PRODUCT_ID
FK	EMPLOYEE_ID
	QUANTITY
	TRANSACTION_DATE
	TRANSACTION_TYPE

EMPLOYEE:



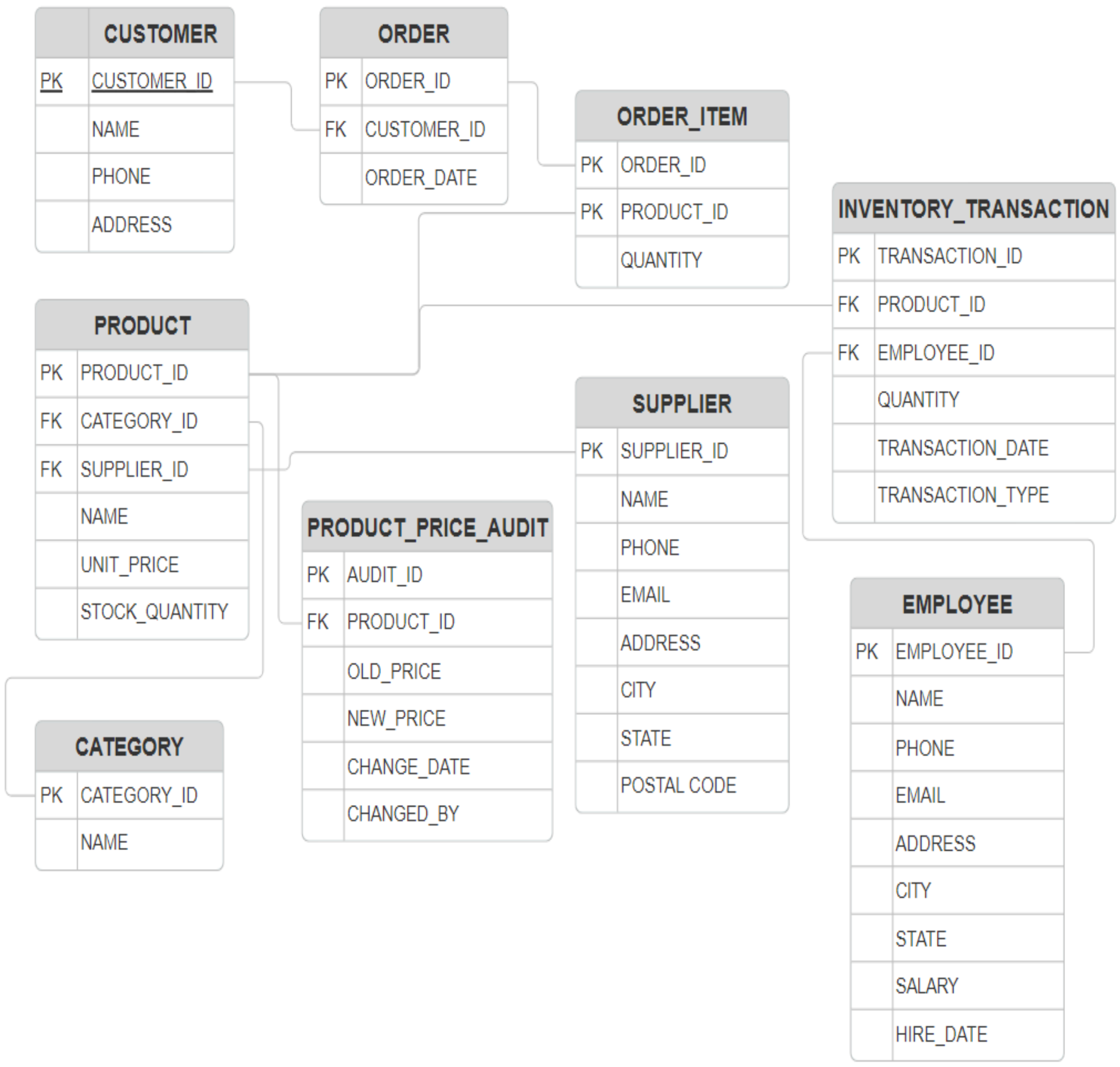
EMPLOYEE	
PK	EMPLOYEE_ID
	NAME
	PHONE
	EMAIL
	ADDRESS
	CITY
	STATE
	SALARY
	HIRE_DATE

RELATIONAL MODEL



NORMALIZED FORM

The database design inherently adhered to third normal form (3NF) principles, ensuring optimal data structure without requiring additional normalization.



IMPLEMENTATION

DATABASE DESIGN AND STRUCTURE

Implemented using SQL, the database architecture provides a comprehensive framework.

Tables

- **CUSTOMER**: Detailed personal and contact information repositories

  Code:

  ```
  CREATE TABLE Customer (
     Customer_ID NUMBER PRIMARY KEY,
     Name VARCHAR2(50) NOT NULL,
     Phone VARCHAR2(15),
     Address VARCHAR2(100) NOT NULL
  );
  ```

- **ORDER_T**: Transactional record-keeping with customer linkages

  Code:

  ```
  CREATE TABLE Order_T (
     Order_ID NUMBER PRIMARY KEY,
     Order_Date DATE NOT NULL,
     Customer_ID NUMBER NOT NULL,
     FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
  );
  ```

- **SUPPLIER**: Comprehensive vendor management system

  Code:

  ```
  CREATE TABLE Supplier (
     Supplier_ID NUMBER PRIMARY KEY,
     Name VARCHAR2(50) NOT NULL,
     Phone VARCHAR2(15) NOT NULL,
     Email VARCHAR2(70),
     Address VARCHAR2(100),
     City VARCHAR2(50),
     State VARCHAR2(50),
     Postal_Code VARCHAR2(10)
  );
  ```

- **CATEGORY**: Intelligent product classification mechanism

  Code:

  ```
  CREATE TABLE Category (
     Category_ID NUMBER PRIMARY KEY,
     Name VARCHAR2(50) NOT NULL
  );
  ```

- **PRODUCT**: Detailed merchandise tracking with dynamic pricing

  Code:

```
CREATE TABLE Product (
   Product_ID NUMBER PRIMARY KEY,
```

```
    Name VARCHAR2(100) NOT NULL,
    Unit_Price NUMBER(10, 2) NOT NULL,
    Stock_Quantity NUMBER NOT NULL,
    Category_ID NUMBER NOT NULL,
    Supplier_ID NUMBER NOT NULL,
    FOREIGN KEY (Category_ID) REFERENCES Category(Category_ID),
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);
```

- **ORDER_ITEM**: Granular transaction item documentation

  Code:

  ```
  CREATE TABLE Order_Item (
      Order_ID NUMBER NOT NULL,
      Product_ID NUMBER NOT NULL,
      Quantity NUMBER NOT NULL,
      PRIMARY KEY (Order_ID, Product_ID),
      FOREIGN KEY (Order_ID) REFERENCES Order_T(Order_ID),
      FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
  );
  ```

- **EMPLOYEE**: Workforce management and role tracking

  Code:

  ```
  CREATE TABLE Employee (
      Employee_ID NUMBER PRIMARY KEY,
  ```

```
    Name VARCHAR2(50) NOT NULL,
    Phone VARCHAR2(15) NOT NULL,
    Email VARCHAR2(50),
    Address VARCHAR2(100),
    City VARCHAR2(50),
    Hire_Date DATE NOT NULL,
    Salary NUMBER(10, 2) NOT NULL
);
```

- **INVENTORY_TRANSACTION**: Real-time stock movement logging

  Code:

```
CREATE TABLE Inventory_Transaction (
    Transaction_ID NUMBER PRIMARY KEY,
    Quantity NUMBER NOT NULL,
    Transaction_Date DATE NOT NULL,
    Transaction_Type VARCHAR2(50) NOT NULL,
    Employee_ID NUMBER NOT NULL,
    Product_ID NUMBER NOT NULL,
    FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID),
    FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);
```

- **PRODUCT_PRICE_AUDIT**: Historical pricing modification tracking

  Code:

```
CREATE TABLE Product_Price_Audit (
  Audit_ID NUMBER PRIMARY KEY,
  Product_ID NUMBER,
  Old_Price NUMBER(10, 2),
  New_Price NUMBER(10, 2),
  Change_Date DATE DEFAULT SYSDATE,
  Changed_By VARCHAR2(50),
  FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
);
```

TRIGGERS

Five sophisticated triggers automate and validate critical business processes:

1. **update_product_stock** Automatically adjusts product inventory upon order placement, ensuring real-time stock accuracy, takes place after INSERT.

```
CREATE OR REPLACE TRIGGER update_product_stock
AFTER INSERT ON Order_Item
FOR EACH ROW
BEGIN
  UPDATE Product
  SET Stock_Quantity = Stock_Quantity - :NEW.Quantity
  WHERE Product_ID = :NEW.Product_ID;
END;/
```

2. **inv_trans_after_order** Generates comprehensive inventory transaction records for each order, capturing detailed movement metadata, takes place after INSERT.

```
CREATE OR REPLACE TRIGGER inv_trans_after_order
AFTER INSERT ON Order_Item
FOR EACH ROW
BEGIN
  -- Sale transaction
  INSERT INTO Inventory_Transaction (Transaction_ID, Quantity, Transaction_Date, Transaction_Type, Employee_ID, Product_ID)
  VALUES (SEQ_Transaction_ID.NEXTVAL, :NEW.Quantity, SYSDATE, 'Sale', 501, :NEW.Product_ID);
END;
/
```

3. **check_product_stock** Implements pre-order validation to prevent overselling and maintain inventory integrity, takes place before INSERT.

```
CREATE OR REPLACE TRIGGER check_product_stock
BEFORE INSERT ON Order_Item
FOR EACH ROW
DECLARE
  available_stock INT;
BEGIN
  -- Get current stock for the product
  SELECT Stock_Quantity INTO available_stock FROM Product WHERE Product_ID = :NEW.Product_ID;
  -- Check if sufficient stock is available
  IF available_stock < :NEW.Quantity THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock for product ID ' || :NEW.Product_ID);
  END IF;
END;
/
```

4. **set_order_date** Standardizes order documentation by automatically assigning transaction timestamps, takes place before INSERT.

```
CREATE OR REPLACE TRIGGER set_order_date
BEFORE INSERT ON Order_T
FOR EACH ROW
BEGIN
  -- Set the order date to current system date if not provided
  IF :NEW.Order_Date IS NULL THEN
    :NEW.Order_Date := SYSDATE;
  END IF;
END;
/
```

5. **audit_product_price_change** Creates a transparent, immutable record of all pricing modifications, takes place after UPDATE of UNIT_PRICE.

```
CREATE OR REPLACE TRIGGER audit_product_price_change
AFTER UPDATE OF Unit_Price ON Product
```

```
FOR EACH ROW

BEGIN

   INSERT INTO Product_Price_Audit
(Audit_ID,Product_ID,Old_Price,New_Price,Change_Date)

   VALUES
(Product_Price_Audit_Seq.NEXTVAL,:OLD.Product_ID,:OLD.Unit_Price,:NE
W.Unit_Price,SYSDATE);

END;

/
```

Significance of Triggers

1. Triggers handle essential operations, such as stock adjustments and transaction logging, without requiring manual input.
2. By enforcing business rules, triggers prevent invalid transactions, such as orders exceeding available stock.
3. Price updates are automatically logged via triggers, creating a reliable record for future analysis and compliance.

Sequences

1. **SEQ_Transaction_ID**

Purpose:
Generates unique transaction identifiers for the Inventory_Transaction table, enabling effective tracking.

SQL Definition:

```
CREATE SEQUENCE SEQ_Transaction_ID

START WITH 611

INCREMENT BY 1

NOCACHE

NOCYCLE;
```

2. **PRODUCT_PRICE_AUDIT_SEQ**

Purpose:
Provides unique audit identifiers for the Product_Price_Audit table, essential for maintaining accurate price change records.

SQL Definition:

```
CREATE SEQUENCE

Product_Price_Audit_Seq

START WITH 1

INCREMENT BY 1;
```

Significance of Sequences

1. Both sequences ensure distinct primary keys, preventing duplication and maintaining database integrity.
2. ID generation is streamlined, reducing manual workload and minimizing human error.
3. The sequences efficiently manage growing data volumes, enabling smooth database operations as the system scales.

Roles

1. **Admin:**
    - Responsible for complete database oversight, including creating, modifying, and deleting database objects.
        - Has full CRUD (Create, Read, Update, Delete) permissions across all data tables.

```
CREATE ROLE Admin;

--Grant full privileges to Admin

GRANT CREATE SESSION TO Admin;

GRANT CREATE TABLE TO Admin;

GRANT CREATE VIEW TO Admin;

GRANT CREATE PROCEDURE TO Admin;

GRANT CREATE TRIGGER TO Admin;

GRANT CREATE SEQUENCE TO Admin;

GRANT ALTER ANY TABLE TO Admin;

GRANT DROP ANY TABLE TO Admin;

GRANT SELECT ANY TABLE TO Admin;
GRANT INSERT ANY TABLE TO Admin;

GRANT UPDATE ANY TABLE TO Admin;

GRANT DELETE ANY TABLE TO Admin;

GRANT EXECUTE ANY PROCEDURE TO Admin;

GRANT ALL ON Product TO Admin;

GRANT ALL ON ORDER_T TO Admin;

GRANT ALL ON Order_Item TO Admin;

GRANT ALL ON Inventory_Transaction TO Admin;

GRANT ALL ON Customer TO Admin;

GRANT ALL ON Category TO Admin;

GRANT ALL ON Supplier TO Admin;

GRANT ALL ON Employee TO Admin;

GRANT ALL ON Product_Price_Audit TO Admin;
```

2. **Salesperson:**
    - Limited access to manage orders and products.
    - Can insert and view data related to sales but are restricted from altering system configurations or sensitive data, such as pricing.

```
CREATE ROLE Salesperson;

--Grant privileges to Salesperson

GRANT CREATE SESSION TO Salesperson;

GRANT SELECT, INSERT, UPDATE ON Product TO Salesperson;

GRANT SELECT, INSERT, UPDATE ON Order_T TO Salesperson;

GRANT SELECT, INSERT, UPDATE ON Order_Item TO Salesperson;
GRANT INSERT ON Customer TO Salesperson;
```

3. **Manager:**
    - Oversees database operations, including inventory management and sales tracking.
    - Authorized to update product details, manage orders, and monitor price changes, ensuring efficient operational management.

```
CREATE ROLE Manager;

--Grant privileges to MANAGER

GRANT CREATE SESSION TO Manager;

GRANT SELECT, INSERT, UPDATE ON Product TO Manager;

GRANT SELECT, INSERT, UPDATE ON ORDER_T TO Manager;

GRANT SELECT, INSERT, UPDATE ON Order_Item TO Manager;

GRANT SELECT, INSERT, UPDATE ON Customer TO Manager;

GRANT SELECT, INSERT, UPDATE ON Inventory_Transaction TO Manager;

GRANT SELECT, INSERT, UPDATE ON Employee TO Manager;
```

Sample Data

-- Insert data into Customer Table

INSERT INTO Customer (Customer_ID, Name, Phone, Address)

VALUES (11, 'Ahmed Khan', '03011234567', '123 Main Road, Lahore');

INSERT INTO Customer (Customer_ID, Name, Phone, Address)

VALUES (12, 'Ayesha Tariq', '03019876543', '456 Canal View, Lahore');

INSERT INTO Customer (Customer_ID, Name, Phone, Address)

VALUES (13, 'Zain Ali', '03214567890', '789 Shahrah-e-Faisal, Lahore');

-- Insert data into Order_T Table

INSERT INTO Order_T (Order_ID, Order_Date, Customer_ID)

VALUES (101, TO_DATE('2024-12-01', 'YYYY-MM-DD'), 11);

INSERT INTO Order_T (Order_ID, Order_Date, Customer_ID)

VALUES (102, TO_DATE('2024-12-02', 'YYYY-MM-DD'), 12);

INSERT INTO Order_T (Order_ID, Order_Date, Customer_ID)

VALUES (103, TO_DATE('2024-12-03', 'YYYY-MM-DD'), 13);

-- Insert data into Supplier Table

INSERT INTO Supplier (Supplier_ID, Name, Phone, Email, Address, City, State, Postal_Code)

VALUES (201, 'Bilal Hassan', '03034445566', 'bilal.hassan@gmail.com', 'Block B, Model Town', 'Lahore', 'Punjab', '54000');

INSERT INTO Supplier (Supplier_ID, Name, Phone, Email, Address, City, State, Postal_Code)

VALUES (202, 'Sana Khan', '03225556677', 'sana.khan@hotmail.com', 'Street 12, F-10', 'Lahore', 'Punjab', '54000');

INSERT INTO Supplier (Supplier_ID, Name, Phone, Email, Address, City, State, Postal_Code)

VALUES (203, 'Farhan Malik', '03114442233', 'farhan.malik@yahoo.com', 'Phase 2, Gulshan-e-Iqbal', 'Lahore', 'Punjab', '54000');

-- Insert data into Category Table

INSERT INTO Category (Category_ID, Name)

VALUES (301, 'Stationery');

```sql
INSERT INTO Category (Category_ID, Name)
VALUES (302, 'Office Supplies');
INSERT INTO Category (Category_ID, Name)
VALUES (303, 'Art Supplies');

-- Insert data into Product Table
INSERT INTO Product (Product_ID, Name, Unit_Price, Stock_Quantity, Category_ID, Supplier_ID)
VALUES (401, 'Ballpoint Pen', 20.00, 200, 306, 201);
INSERT INTO Product (Product_ID, Name, Unit_Price, Stock_Quantity, Category_ID, Supplier_ID)
VALUES (402, 'Notebook', 100.00, 150, 305, 202);
INSERT INTO Product (Product_ID, Name, Unit_Price, Stock_Quantity, Category_ID, Supplier_ID)
VALUES (403, 'Permanent Marker', 50.00, 300, 307, 203);

-- Insert data into Employee Table
INSERT INTO Employee (Employee_ID, Name, Phone, Email, Address, City, Hire_Date, Salary)
VALUES (501, 'Imran Sheikh', '03012345678', 'imran.sheikh@store.com', 'Street 45, DHA', 'Lahore',
TO_DATE('2020-01-15', 'YYYY-MM-DD'), 45000);
INSERT INTO Employee (Employee_ID, Name, Phone, Email, Address, City, Hire_Date, Salary)
VALUES (502, 'Ayesha Siddiq', '03123456789', 'ayesha.siddiq@store.com', 'Block C, Model Town',
'Lahore', TO_DATE('2019-06-12', 'YYYY-MM-DD'), 50000);
INSERT INTO Employee (Employee_ID, Name, Phone, Email, Address, City, Hire_Date, Salary)
VALUES (503, 'Ali Raza', '03214567890', 'ali.raza@store.com', 'Street 10, Gulberg', 'Lahore',
TO_DATE('2021-03-22', 'YYYY-MM-DD'), 47000);

-- Insert data into Order_Item Table
INSERT INTO Order_Item (Order_ID, Product_ID, Quantity)
VALUES (101, 401, 5);
INSERT INTO Order_Item (Order_ID, Product_ID, Quantity)
VALUES (102, 402, 3);
INSERT INTO Order_Item (Order_ID, Product_ID, Quantity)
VALUES (103, 403, 10);
```

-- Insert data into Inventory_Transaction Table

INSERT INTO Inventory_Transaction (Transaction_ID, Quantity, Transaction_Date, Transaction_Type, Employee_ID, Product_ID)

VALUES (601, 50, TO_DATE('2024-11-30', 'YYYY-MM-DD'), 'Restock', 501, 401);

INSERT INTO Inventory_Transaction (Transaction_ID, Quantity, Transaction_Date, Transaction_Type, Employee_ID, Product_ID)

VALUES (602, 30, TO_DATE('2024-12-01', 'YYYY-MM-DD'), 'Restock', 502, 402);

INSERT INTO Inventory_Transaction (Transaction_ID, Quantity, Transaction_Date, Transaction_Type, Employee_ID, Product_ID)

VALUES (603, 40, TO_DATE('2024-12-02', 'YYYY-MM-DD'), 'Sale', 503, 403);

Sample Queries

Testing Sales Transactions in Inventory Transaction Table

SELECT *

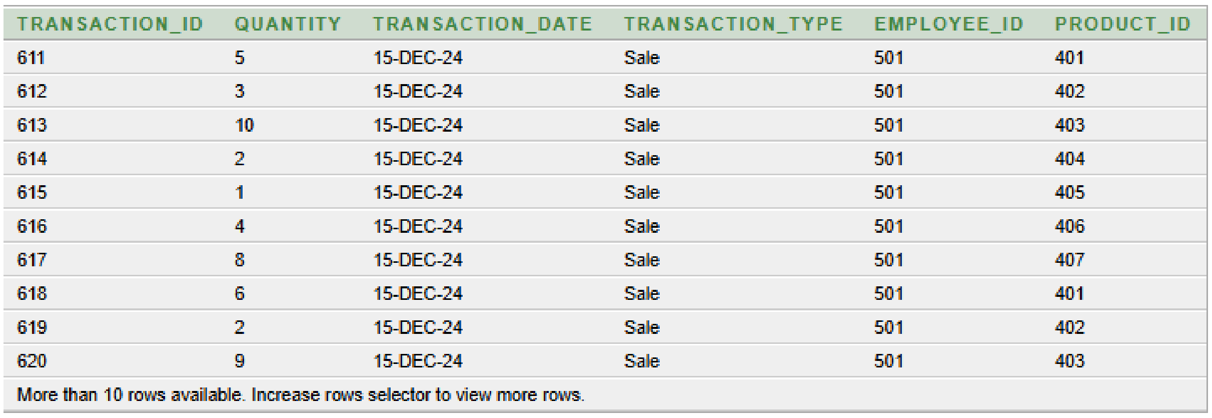
FROM Inventory_Transaction

WHERE Transaction_Type = 'Sale';

Checking a specific Customer:

SELECT *

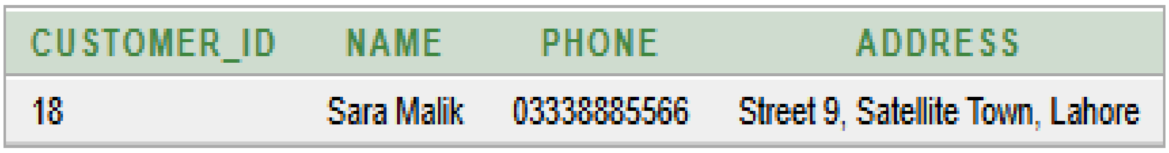
FROM CUSTOMER



TRANSACTION_ID	QUANTITY	TRANSACTION_DATE	TRANSACTION_TYPE	EMPLOYEE_ID	PRODUCT_ID
611	5	15-DEC-24	Sale	501	401
612	3	15-DEC-24	Sale	501	402
613	10	15-DEC-24	Sale	501	403
614	2	15-DEC-24	Sale	501	404
615	1	15-DEC-24	Sale	501	405
616	4	15-DEC-24	Sale	501	406
617	8	15-DEC-24	Sale	501	407
618	6	15-DEC-24	Sale	501	401
619	2	15-DEC-24	Sale	501	402
620	9	15-DEC-24	Sale	501	403

More than 10 rows available. Increase rows selector to view more rows.

WHERE CUSTOMER_ID = 18;



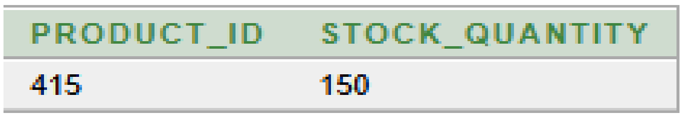
CUSTOMER_ID	NAME	PHONE	ADDRESS
18	Sara Malik	03338885566	Street 9, Satellite Town, Lahore

Checking a Stock before and after adding on Order Item:

SELECT Product_ID, Stock_Quantity

FROM Product

WHERE Product_ID = 415;



PRODUCT_ID	STOCK_QUANTITY
415	150

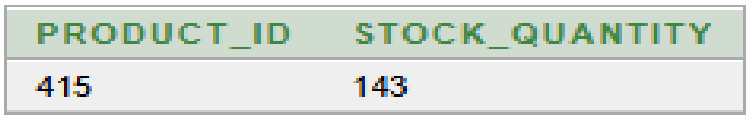
INSERT INTO Order_Item (Order_ID, Product_ID, Quantity)

VALUES (105, 415, 7);

SELECT Product_ID, Stock_Quantity

FROM Product

WHERE Product_ID = 415;



PRODUCT_ID	STOCK_QUANTITY
415	143

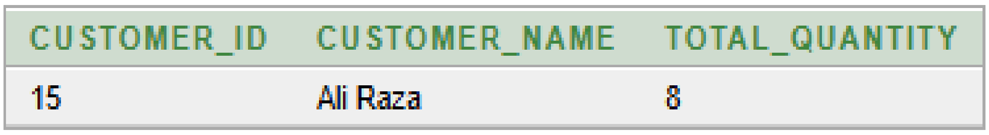
Total Quantity of Products ordered by a specific Customer:

SELECT C.Customer_ID,C.Name AS Customer_Name,SUM(OI.Quantity) AS Total_Quantity

FROM Customer C,Order_T O,Order_Item OI

WHERE C.Customer_ID = O.Customer_ID AND O.Order_ID = OI.Order_ID AND C.Customer_ID = 15

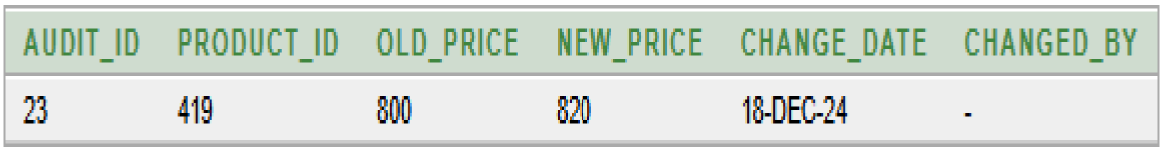
```
GROUP BY C.Customer_ID, C.Name

ORDER BY Total_Quantity DESC;
```



CUSTOMER_ID	CUSTOMER_NAME	TOTAL_QUANTITY
15	Ali Raza	8

Testing Product Price Update and Price Audit Logging:

```
UPDATE Product

SET Unit_Price = 820

WHERE Product_ID = 419;


SELECT *

FROM Product_Price_Audit

WHERE Product_ID = 419;
```



AUDIT_ID	PRODUCT_ID	OLD_PRICE	NEW_PRICE	CHANGE_DATE	CHANGED_BY
23	419	800	820	18-DEC-24	-

Storage Estimation

Table	Bytes Per Record	Initial Records	Growth Rate (Yearly)	Expected Records (5 Years)	Storage in Bytes
Customer	165	500	**100%**	15,500	2,557,500
Order_T	64	2,000	**80%**	34,028	2,177,792
Order_Item	80	10,000	**90%**	148,029	11,842,320
Supplier	315	200	**10%**	322	101,430
Category	54	50	**5%**	64	3,456
Product	184	1,000	**10%**	1,611	296,424
Employee	215	100	**1%**	105	22,575
Inventory_Transaction	141	5,000	**60%**	49,766	7,015,806
Product_Price_Audit	105	3,000	**20%**	7,458	783,090

Conclusion

The database system represents a paradigm shift in stationery store management. By creating an intelligent, interconnected technological ecosystem, we've transformed raw data into a strategic business asset, enabling unprecedented operational efficiency and insights.

More than a technical solution, this database serves as a catalyst for organizational transformation, positioning the stationery store for sustainable growth and competitive advantage.