**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | B.Tech Artificial Intelligence & Data Science | |
| Semester | 4th | |
| Name of the Project: | Inventory Ordering and Management System | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| 2023-2027 | A005 | Aditya Madhav Mantri |
| 2023-2027 | A017 | Mahi Dubey |
| 2023-2027 | A018 | Adeeb Qureshi |
| Date of Submission: | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| A005 | Aditya Madhav Mantri |  |
| A017 | Mahi Dubey |  |
| A018 | Adeeb Qureshi |  |

**Project Report**

**Inventory Ordering and Management System**

**by**

**Aditya Mantri, Roll number: A005**

**Mahi Dubey, Roll number: A017**

**Adeeb Qureshi, Roll number: A018**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

|  |  |
| --- | --- |
| **Sr no.** | **Topic** |
| **1** | Storyline |
| **2** | Components of Database Design |
| **3** | Entity Relationship Diagram |
| **4** | Relational Model |
| **5** | Normalization |
| **6** | SQL Queries |
| **7** | Learning from the Project |
| **8** | Project Demonstration |
| **9** | Self-learning beyond classroom |
| **10** | Learning from the project |
| **11** | Challenges faced |
| **12** | Conclusion |

**TechNova Electronics**

**Company Name: TechNova Electronics  
Industry: Consumer Electronics & Smart Home Solutions  
Business Model: Retail & Distribution**

**About TechNova Electronics**

TechNova Electronics is a leading retailer and distributor of high-quality consumer electronics, smart home devices, and personal gadgets. We specialize in sourcing, stocking, and selling the latest technology driven products to enhance everyday life. Whether it’s cutting-edge home appliances, personal gadgets, or computing devices, TechNova ensures customers have access to the best products on the market.

**Business Operations**

TechNova does not manufacture its own products but works with trusted suppliers to bring toptier electronics to customers. We manage inventory across multiple warehouses, track product availability, and ensure seamless order fulfilment through an optimized supply chain.

**Product Categories**

TechNova offers a diverse range of products, including:

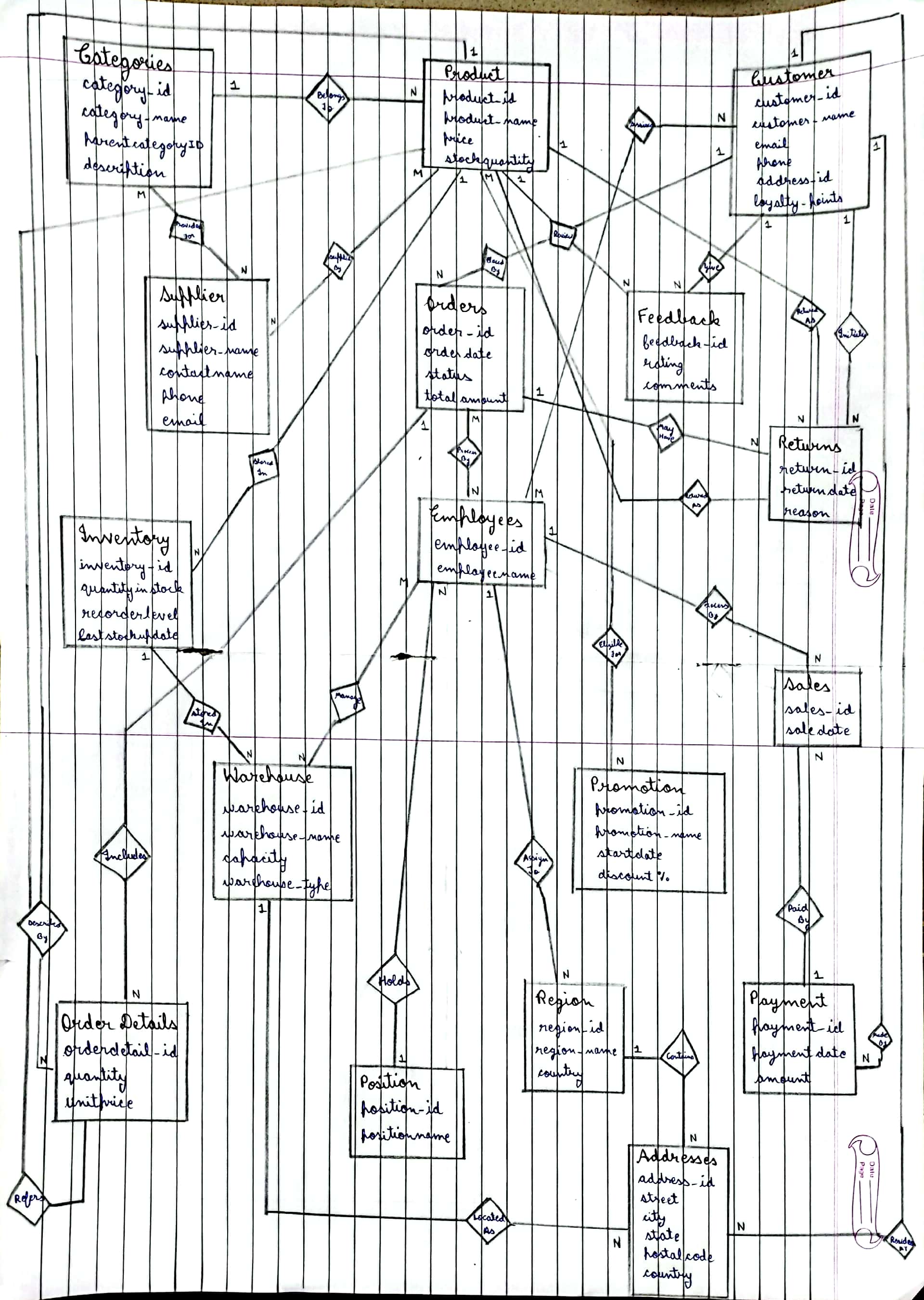
* Home Appliances – Smart refrigerators, air conditioners, microwaves
* Consumer Electronics – Smart TVs, home theatres, digital cameras
* Personal Gadgets – Smartwatches, wireless earbuds, VR headsets
* Computing Devices – Laptops, tablets, external SSDs
* Smart Home Devices – Smart security cameras, door locks, LED bulbs
* Kitchen Appliances – Induction cooktops, air fryers, coffee makers

And many other products….

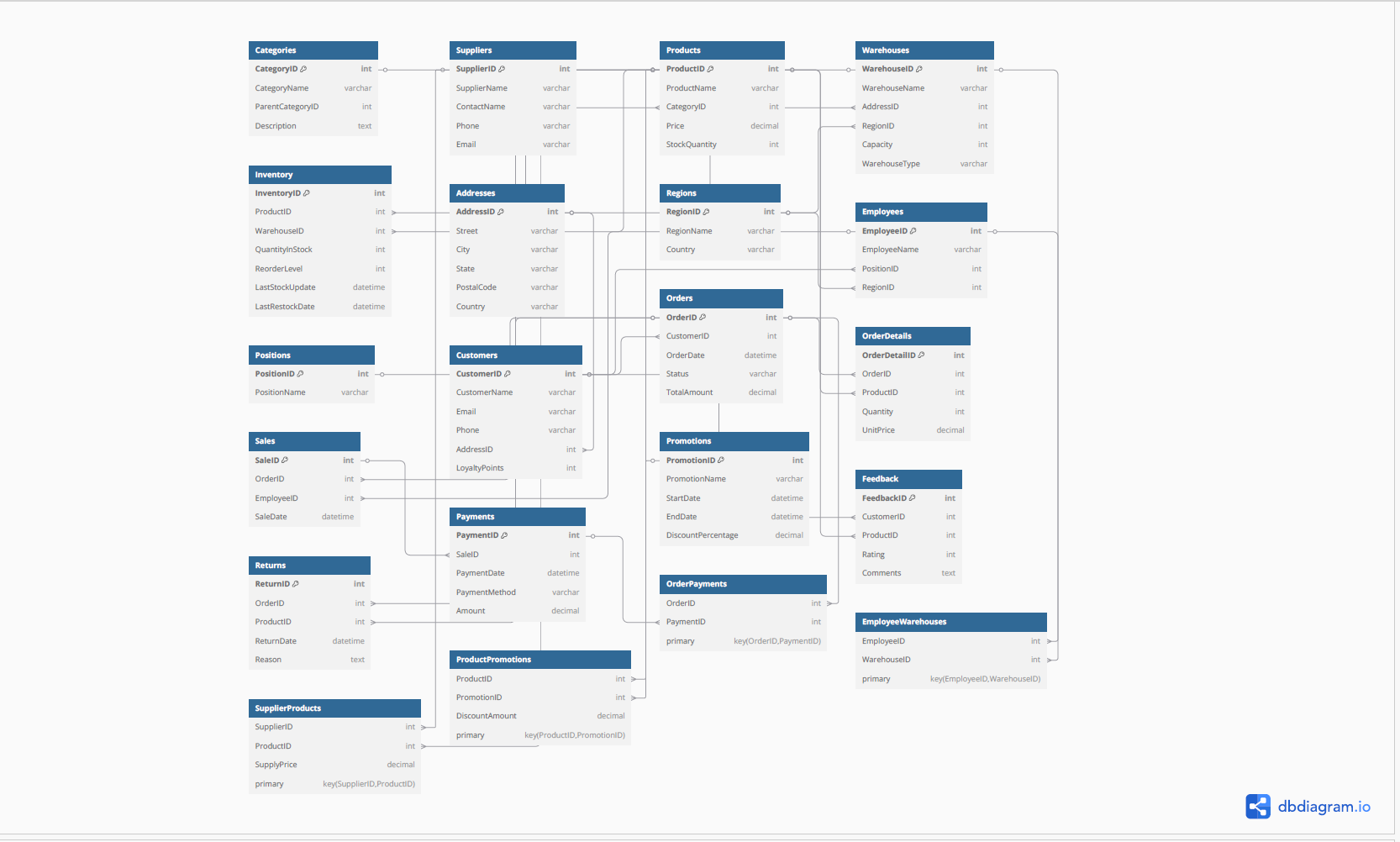
**Key Features of TechNova’s Operations**

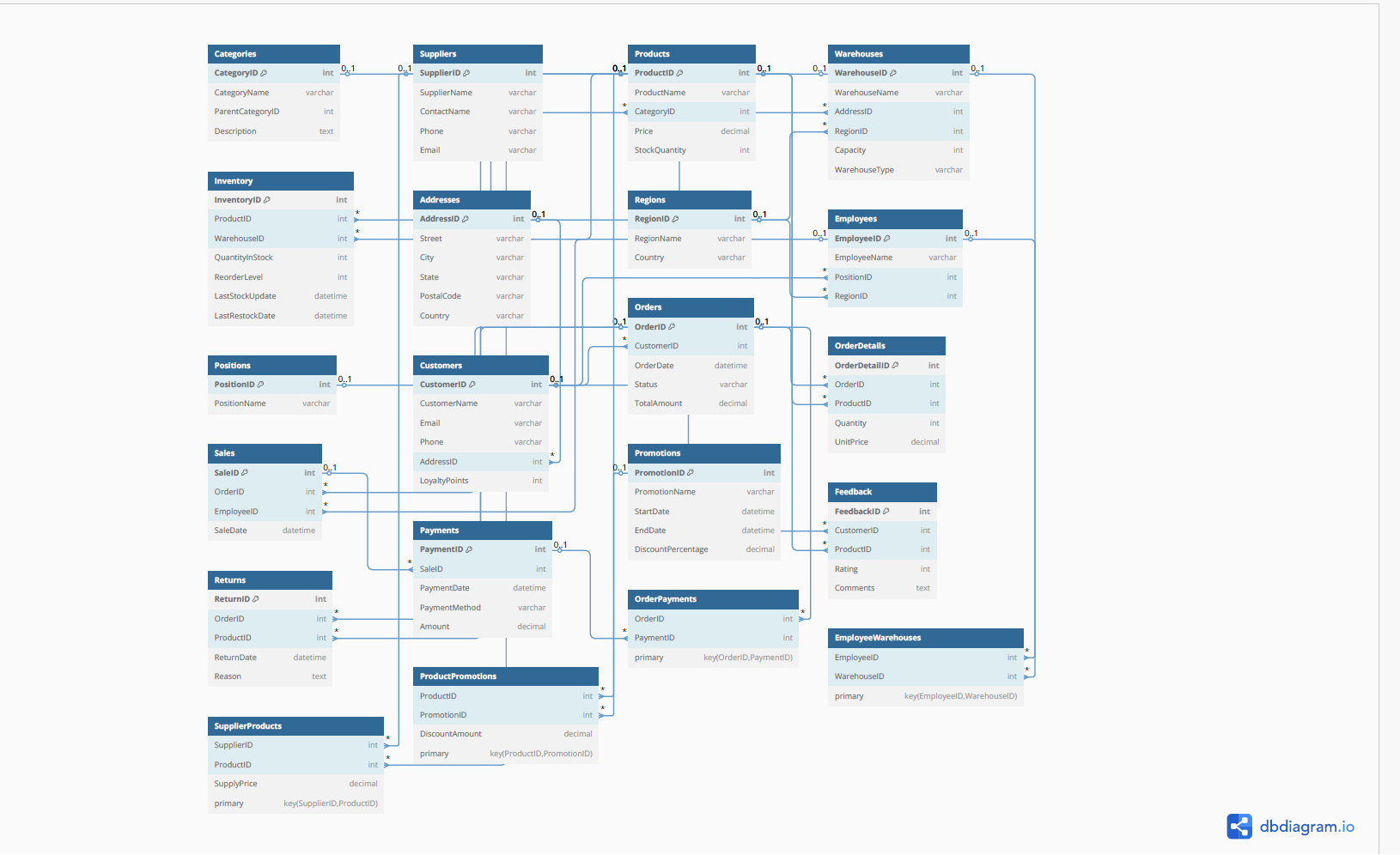
* Efficient Inventory Management – Realtime tracking of stock levels and warehouse distribution
* Seamless Order Processing – Customers can place and track orders easily
* Reliable Supplier Network – Partnering with trusted brands and manufacturers
* Customer Centric Approach – Loyalty programs, returns, and feedback integration
* Promotions & Discounts – Seasonal and special promotions to maximize customer savings

**ER Diagram**

****

**Relational Diagram**

****

****

**Entities**

**Entities & Attributes**

1. **Categories**

* CategoryID (int)
* CategoryName (varchar)
* ParentCategoryID (int)
* Description (text)

2. **Suppliers**

* SupplierID (int)
* SupplierName (varchar)
* ContactName (varchar)
* Phone (varchar)
* Email (varchar)

3. **Products**

* ProductID (int)
* ProductName (varchar)
* Price (decimal)
* StockQuantity (int)

4. **Warehouses**

* WarehouseID (int)
* WarehouseName (varchar)
* Capacity (int)
* WarehouseType (varchar)

5. **Inventory**

* InventoryID (int)
* QuantityInStock (int)
* ReorderLevel (int)
* LastStockUpdate (datetime)

6. **Addresses**

* AddressID (int)
* Street (varchar)
* City (varchar)
* State (varchar)
* PostalCode (varchar)
* Country (varchar)

7. **Regions**

* RegionID (int)
* RegionName (varchar)
* Country (varchar)

8. **Customers**

* CustomerID (int)
* CustomerName (varchar)
* Email (varchar)
* Phone (varchar)
* AddressID *(FK, excluded)*
* LoyaltyPoints (int)

9. **Employees**

* EmployeeID (int)
* EmployeeName (varchar)

10. **Positions**

* PositionID (int)
* PositionName (varchar)

11. **Sales**

* SaleID (int)
* SaleDate (datetime)

12. **Orders**

* OrderID (int)
* OrderDate (datetime)
* Status (varchar)
* TotalAmount (decimal)

13. **OrderDetails**

* OrderDetailID (int)
* Quantity (int)
* UnitPrice (decimal)

14. **Payments**

* PaymentID (int)
* PaymentDate (datetime)
* Amount (decimal)

15. **Returns**

* ReturnID (int)
* ReturnDate (datetime)
* Reason (text)

16. **Feedback**

* FeedbackID (int)
* Rating (int)
* Comments (text)

17. **Promotions**

* PromotionID (int)
* PromotionName (varchar)
* StartDate (datetime)
* DiscountPercentage (decimal)

**Relations:**

* Categories <> Categories (SelfReferencing) (ParentChild Relationship) (1:N)
* Categories <> Products (Belongs To) (1:N)
* Suppliers <> Categories (Provides) (N:M)
* Suppliers <> Products (Supplies) (N:M)
* Products <> Inventory (Tracks Stock) (1:N)
* Products <> OrderDetails (Included In) (1:N)
* Products <> Returns (May Be Returned) (1:N)
* Products <> Feedback (Reviewed By) (1:N)
* Products <> Promotions (Discounted In) (N:M)
* Warehouses <> Inventory (Stores) (1:N)
* Warehouses <> Employees (Managed By) (N:M)
* Addresses <> Customers (Resides At) (1:N)
* Addresses <> Warehouses (Located At) (1:N)
* Regions <> Addresses (Contains) (1:N)
* Regions <> Employees (Assigned To) (1:N)
* Customers <> Orders (Places) (1:N)
* Customers <> Feedback (Provides) (1:N)
* Customers <> Returns (Requests) (1:N)
* Customers <> Payments (Makes) (1:N)
* Employees <> Sales (Handles) (1:N)
* Employees <> Orders (Processes) (N:M)
* Employees <> Customers (Assists) (N:M)
* Positions <> Employees (Holds) (1:N)
* Sales <> Payments (Receives) (1:N)
* Orders <> OrderDetails (Contains) (1:N)
* Orders <> Returns (May Have) (1:N)
* Orders <> Payments (Paid By) (1:N)
* Returns <> Products (Of) (N:1)

**Normalization**

**Current Highest normal form of each table**

1. **Categories**

PK: CategoryID

FDs: CategoryID → CategoryName, ParentCategoryID, Description

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

2. **Suppliers**

PK: SupplierID

FDs: SupplierID → SupplierName, ContactName, Phone, Email

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**3. Supplier\_Categories (Bridge Table)**

PK: (SupplierID, CategoryID)

FDs: (SupplierID, CategoryID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**4. Supplier\_Products (Bridge Table)**

PK: (SupplierID, ProductID)

FDs: (SupplierID, ProductID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**5. Products**

PK: ProductID

FDs: ProductID → ProductName, CategoryID, Price, StockQuantity

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**6. Inventory**

PK: InventoryID

FDs:

InventoryID → ProductID, WarehouseID, QuantityInStock, ReorderLevel, LastStockUpdate

(ProductID, WarehouseID) → QuantityInStock, ReorderLevel, LastStockUpdate (assuming)

Analysis: If InventoryID is a primary key, it's in BCNF. If composite primary key (ProductID, WarehouseID) were PK, still BCNF

Highest Normal Form: BCNF

**7. Warehouses**

PK: WarehouseID

FDs: WarehouseID → WarehouseName, Capacity, WarehouseType, AddressID

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**8. Warehouse\_Employees (Bridge Table)**

PK: (WarehouseID, EmployeeID)

FDs: (WarehouseID, EmployeeID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**9. Addresses**

PK: AddressID

FDs: AddressID → Street, City, State, PostalCode, Country, RegionID

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

Or

**Addresses**

**NF:** 2NF

**Partial Dependency:** No

**Transitive Dependency:** **Possible**

If City → State → Country, there’s a **transitive dependency**.

But this depends on whether the schema treats them independently.

**10. Regions**

PK: RegionID

FDs: RegionID → RegionName, Country

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**11. Customers**

PK: CustomerID

FDs: CustomerID → CustomerName, Email, Phone, AddressID, LoyaltyPoints

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**12. Employees**

PK: EmployeeID

FDs: EmployeeID → EmployeeName, RegionID, PositionID

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**13. Employee\_Customers (Bridge Table)**

PK: (EmployeeID, CustomerID)

FDs: (EmployeeID, CustomerID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**14. Positions**

PK: PositionID

FDs: PositionID → PositionName

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**15. Sales**

PK: SaleID

FDs: SaleID → EmployeeID, SaleDate

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**16. Orders**

PK: OrderID

FDs: OrderID → CustomerID, OrderDate, Status, TotalAmount

Analysis: TotalAmount is likely derived from OrderDetails (redundant), which would be a denormalization for performance

Highest Normal Form: 3NF (if TotalAmount is derived, violates BCNF)

**17. Employee\_Orders (Bridge Table)**

PK: (EmployeeID, OrderID)

FDs: (EmployeeID, OrderID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**18. OrderDetails**

PK: OrderDetailID

FDs:

OrderDetailID → OrderID, ProductID, Quantity, UnitPrice

(OrderID, ProductID) → Quantity, UnitPrice (assuming)

Analysis: If OrderDetailID is a surrogate key, BCNF. If natural key (OrderID, ProductID) were PK, still BCNF

Highest Normal Form: BCNF

**19. Payments**

PK: PaymentID

FDs: PaymentID → OrderID, CustomerID, SaleID, PaymentDate, Amount

Analysis: CustomerID is functionally dependent on OrderID (redundant), which would be a transitive dependency if OrderID → CustomerID

Highest Normal Form: 2NF (if considering CustomerID redundant)

**20. Returns**

PK: ReturnID

FDs: ReturnID → OrderID, ProductID, CustomerID, ReturnDate, Reason

Analysis: CustomerID is functionally dependent on OrderID (redundant)

Highest Normal Form: 2NF (if considering CustomerID redundant)

**21. Feedback**

PK: FeedbackID

FDs: FeedbackID → CustomerID, ProductID, Rating, Comments

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**22. Promotions**

PK: PromotionID

FDs: PromotionID → PromotionName, StartDate, DiscountPercentage

Analysis: No partial dependencies or transitive dependencies

Highest Normal Form: BCNF

**23. Product\_Promotions (Bridge Table)**

PK: (ProductID, PromotionID)

FDs: (ProductID, PromotionID) → (no nonkey attributes)

Analysis: All attributes are part of the key

Highest Normal Form: BCNF

**Summary:**

Most tables are in BCNF. The exceptions are:

Orders (3NF due to derived TotalAmount)

Payments (2NF due to redundant CustomerID)

Returns (2NF due to redundant CustomerID)

These exceptions appear to be intentional denormalizations for practical reasons (performance or convenience) rather than normalization failures.

**Final Summary of Changes to Achieve BCNF**

|  |  |  |
| --- | --- | --- |
| **Table** | **Issue** | **Solution** |
| **Orders** | TotalAmount is derived (not BCNF) | Remove TotalAmount (calculate from OrderDetails). |
| **Payments** | CustomerID is redundant (transitive dependency via OrderID). | Remove CustomerID (fetch via Orders). |
| **Returns** | CustomerID is redundant (transitive dependency via OrderID). | Remove CustomerID (fetch via Orders). |

**Queries**

1. **Orders handled by each employee**

SELECT e.EmployeeID, e.EmployeeName, COUNT(eo.OrderID) AS TotalOrders

FROM Employee\_Orders eo

JOIN Employees e ON eo.EmployeeID = e.EmployeeID

GROUP BY e.EmployeeID, e.EmployeeName

ORDER BY TotalOrders DESC;

1. **Total number of orders per customer**

SELECT CustomerID, COUNT(\*) AS OrderCount

FROM Orders

GROUP BY CustomerID;

1. **Average order value per customer**

Select CustomerID, SUM(TotalAmount) AS Revenue

FROM Orders

GROUP BY CustomerID;

1. **Orders above average value**

select \* from Orders

WHERE TotalAmount > (SELECT AVG(TotalAmount) FROM Orders);

1. **Adding 50 loyalty points to customers who have placed highvalue orders (orders over 500 in value).**

update Customers

set LoyaltyPoints = LoyaltyPoints + 50

where CustomerID in (select CustomerID from Orders where TotalAmount > 500);

1. **identifying Region with most revenue**

SELECT r.RegionName, SUM(o.TotalAmount) AS Revenue

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Addresses a ON c.AddressID = a.AddressID

JOIN Regions r ON a.RegionID = r.RegionID

GROUP BY r.RegionName

ORDER BY Revenue DESC;

1. **Count of customers per country**

select a.Country, COUNT(\*) as CustomerCount

from Customers c

JOIN Addresses a ON c.AddressID = a.AddressID

GROUP BY a.Country;

1. **Count Customers with Zero Orders**

SELECT COUNT(\*) AS ZeroOder

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderID =0;

1. **Employee Count Region Wise**

SELECT r.RegionName, COUNT(e.EmployeeID) AS EmployeeCount

FROM Employees e

JOIN Regions r ON e.RegionID = r.RegionID

GROUP BY r.RegionName;

1. **Total revenue per country**

SELECT a.Country, SUM(o.TotalAmount) AS Revenue

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Addresses a ON c.AddressID = a.AddressID

GROUP BY a.Country;

1. **Most Active Employee**

SELECT EmployeeID

FROM Employee\_Orders

GROUP BY EmployeeID

ORDER BY COUNT(OrderID) DESC;

1. **Number of orders handled by each employee**

CREATE VIEW EmployeePerformance AS

SELECT e.EmployeeID, e.EmployeeName, COUNT(eo.OrderID) AS OrdersHandled

FROM Employees e

JOIN Employee\_Orders eo ON e.EmployeeID = eo.EmployeeID

GROUP BY e.EmployeeID, e.EmployeeName;

select \* from EmployeePerformance;

1. **High spending customers**

SELECT CustomerID

FROM Orders

GROUP BY CustomerID

HAVING SUM(TotalAmount) > 650;

1. **Monthly order count**

SELECT DATE\_FORMAT(OrderDate, '%Y%m') AS OrderMonth, COUNT(\*) AS OrderCount

FROM Orders

GROUP BY OrderMonth;

1. **Customers Who Have Given 5 Star Feedback**

SELECT DISTINCT c.CustomerName, f.Rating, f.Comments

FROM Feedback f

JOIN Customers c ON f.CustomerID = c.CustomerID

WHERE f.Rating = 5;

1. **Most Returned Product**

SELECT p.ProductName, COUNT(r.ReturnID) AS ReturnCount

FROM Returns r

JOIN Products p ON r.ProductID = p.ProductID

GROUP BY p.ProductName

ORDER BY ReturnCount DESC

LIMIT 1;

1. **Suppliers and the Categories They Provide**

SELECT s.SupplierName, c.CategoryName

FROM Supplier\_Categories sc

JOIN Suppliers s ON sc.SupplierID = s.SupplierID

JOIN Categories c ON sc.CategoryID = c.CategoryID

ORDER BY s.SupplierName;

1. **Products Below Reorder Level in Any Warehouse**

SELECT p.ProductName, w.WarehouseName, i.QuantityInStock, i.ReorderLevel

FROM Inventory i

JOIN Products p ON i.ProductID = p.ProductID

JOIN Warehouses w ON i.WarehouseID = w.WarehouseID

WHERE i.QuantityInStock < i.ReorderLevel;

1. **Products Supplied by Multiple Supplier**

SELECT p.ProductName, COUNT(sp.SupplierID) AS SupplierCount

FROM Supplier\_Products sp

JOIN Products p ON sp.ProductID = p.ProductID

GROUP BY p.ProductName

HAVING SupplierCount > 1;

1. **Employees Not Assigned to Any Warehouse**

SELECT e.EmployeeID, e.EmployeeName

FROM Employees e

LEFT JOIN Warehouse\_Employees we ON e.EmployeeID = we.EmployeeID

WHERE we.EmployeeID IS NULL;

1. **Average Product Rating**

SELECT p.ProductName, ROUND(AVG(f.Rating), 2) AS AvgRating

FROM Feedback f

JOIN Products p ON f.ProductID = p.ProductID

GROUP BY p.ProductName

ORDER BY AvgRating DESC;

1. **Number of Products per Category**

SELECT c.CategoryName, COUNT(p.ProductID) AS ProductCount

FROM Categories c

LEFT JOIN Products p ON c.CategoryID = p.CategoryID

GROUP BY c.CategoryName;

1. **Employee Count per Position**

SELECT pos.PositionName, COUNT(e.EmployeeID) AS EmployeeCount

FROM Employees e

JOIN Positions pos ON e.PositionID = pos.PositionID

GROUP BY pos.PositionName;

1. **Last Stock Update per Warehouse**

SELECT w.WarehouseName, MAX(i.LastStockUpdate) AS LastUpdated

FROM Inventory i

JOIN Warehouses w ON i.WarehouseID = w.WarehouseID

GROUP BY w.WarehouseName;

1. **Customers with Highest Loyalty Points**

SELECT CustomerName, LoyaltyPoints

FROM Customers

ORDER BY LoyaltyPoints DESC

LIMIT 5;

1. **Customers Who Ordered the Same Product More Than Once**

SELECT o.CustomerID, p.ProductName, COUNT(\*) AS TimesOrdered

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY o.CustomerID, od.ProductID

HAVING TimesOrdered > 1;

1. **Orders Placed on a Weekend**

SELECT OrderID, OrderDate, DAYNAME(OrderDate) AS DayOfWeek

FROM Orders

WHERE DAYOFWEEK(OrderDate) IN (1, 7);

1. **Count of orders by status**

SELECT Status, COUNT(\*) AS Count

FROM Orders

GROUP BY Status;

1. **View of top 10 customers**

CREATE VIEW Top\_Customers AS

SELECT c.CustomerID, c.CustomerName, SUM(o.TotalAmount) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.CustomerName

ORDER BY TotalSpent DESC

LIMIT 10;

Select \* from top\_customers;

1. **Count of products under each promotion**

SELECT promo.PromotionName, COUNT(pp.ProductID) AS ProductCount

FROM Product\_Promotions pp

JOIN Promotions promo ON pp.PromotionID = promo.PromotionID

GROUP BY promo.PromotionName;

1. **Products in promotions with discount > 10%**  
   SELECT p.ProductName, promo.PromotionName, promo.DiscountPercentage

FROM Product\_Promotions pp

JOIN Products p ON pp.ProductID = p.ProductID

JOIN Promotions promo ON pp.PromotionID = promo.PromotionID

WHERE promo.DiscountPercentage > 10.00;

1. **Top 5 promotions with the most products**

SELECT promo.PromotionName, COUNT(pp.ProductID) AS ProductCount

FROM Product\_Promotions pp

JOIN Promotions promo ON pp.PromotionID = promo.PromotionID

GROUP BY promo.PromotionName

ORDER BY ProductCount DESC

LIMIT 5;

1. **Create a view of best performing employees by sales amount**

CREATE VIEW TopEmployees AS

SELECT s.EmployeeID, e.EmployeeName, SUM(p.Amount) AS TotalSales

FROM Sales s

JOIN Payments p ON s.SaleID = p.SaleID

JOIN Employees e ON s.EmployeeID = e.EmployeeID

GROUP BY s.EmployeeID, e.EmployeeName;

select \* from topemployees

order by totalSales DESC;

1. **List of orders with multiple payments**

SELECT OrderID, COUNT(\*) AS PaymentCount

FROM Payments

GROUP BY OrderID

HAVING COUNT(\*) > 1;

1. **Daily sales average**

SELECT DATE(PaymentDate) AS Day, AVG(Amount) AS AvgSale

FROM Payments

GROUP BY Day;

1. **Update promotion discount to 35% where more than 3 products are attached**

UPDATE Promotions

SET DiscountPercentage = 35.00

WHERE PromotionID IN (

SELECT PromotionID

FROM Product\_Promotions

GROUP BY PromotionID

HAVING COUNT(ProductID) > 3

);

select \* from promotions;

1. **Products Without Promotions**

SELECT ProductName

FROM Products

WHERE ProductID NOT IN (SELECT ProductID FROM Product\_Promotions);

1. **Products Underperforming in Sales**

SELECT p.ProductName, COUNT(pay.OrderID) AS TimesSold

FROM Payments pay

JOIN Products p ON pay.OrderID = p.ProductID

GROUP BY p.ProductName

HAVING TimesSold < 2;

1. **Top reasons for returns**

SELECT Reason, COUNT(\*) AS Frequency

FROM Returns

GROUP BY Reason

ORDER BY Frequency DESC;

1. **Inventory Summary by Region**

SELECT r.RegionName, SUM(i.QuantityInStock) AS TotalStock

FROM Inventory i

JOIN Warehouses w ON i.WarehouseID = w.WarehouseID

JOIN Addresses a ON w.AddressID = a.AddressID

JOIN Regions r ON a.RegionID = r.RegionID

GROUP BY r.RegionName;

1. **Regions wise count of warehouses:**

SELECT r.RegionName, COUNT(w.WarehouseID) AS WarehouseCount

FROM Warehouses w

JOIN Addresses a ON w.AddressID = a.AddressID

JOIN Regions r ON a.RegionID = r.RegionID

GROUP BY r.RegionName

ORDER BY WarehouseCount DESC;

1. **Top 5 Best-Selling Products by Revenue**

SELECT p.ProductName, SUM(od.Quantity \* od.UnitPrice) AS TotalRevenue

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY p.ProductName

ORDER BY TotalRevenue DESC

LIMIT 5;

1. **Warehouses That Store More Than 3 Unique Products**

SELECT w.WarehouseName, COUNT(DISTINCT i.ProductID) AS ProductTypes

FROM Inventory i

JOIN Warehouses w ON i.WarehouseID = w.WarehouseID

GROUP BY w.WarehouseName

HAVING COUNT(DISTINCT i.ProductID) > 2;

1. **Categories with Average Product Price Above 100**

SELECT c.CategoryName, ROUND(AVG(p.Price), 2) AS AvgPrice

FROM Categories c

JOIN Products p ON c.CategoryID = p.CategoryID

GROUP BY c.CategoryName

HAVING AVG(p.Price) > 100;

1. **Employees Who Handled Orders for Customers from 'India'**

SELECT DISTINCT e.EmployeeID, e.EmployeeName

FROM Employees e

JOIN Employee\_Orders eo ON e.EmployeeID = eo.EmployeeID

JOIN Orders o ON eo.OrderID = o.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

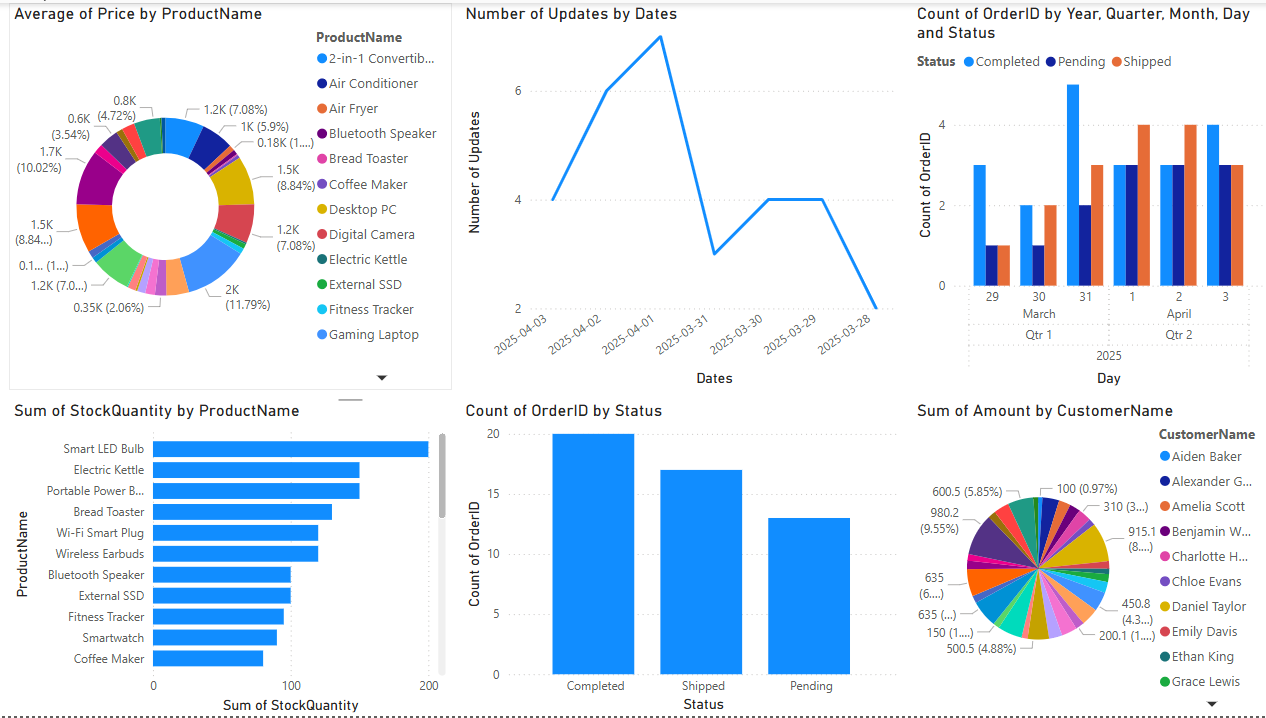
JOIN Addresses a ON c.AddressID = a.AddressID

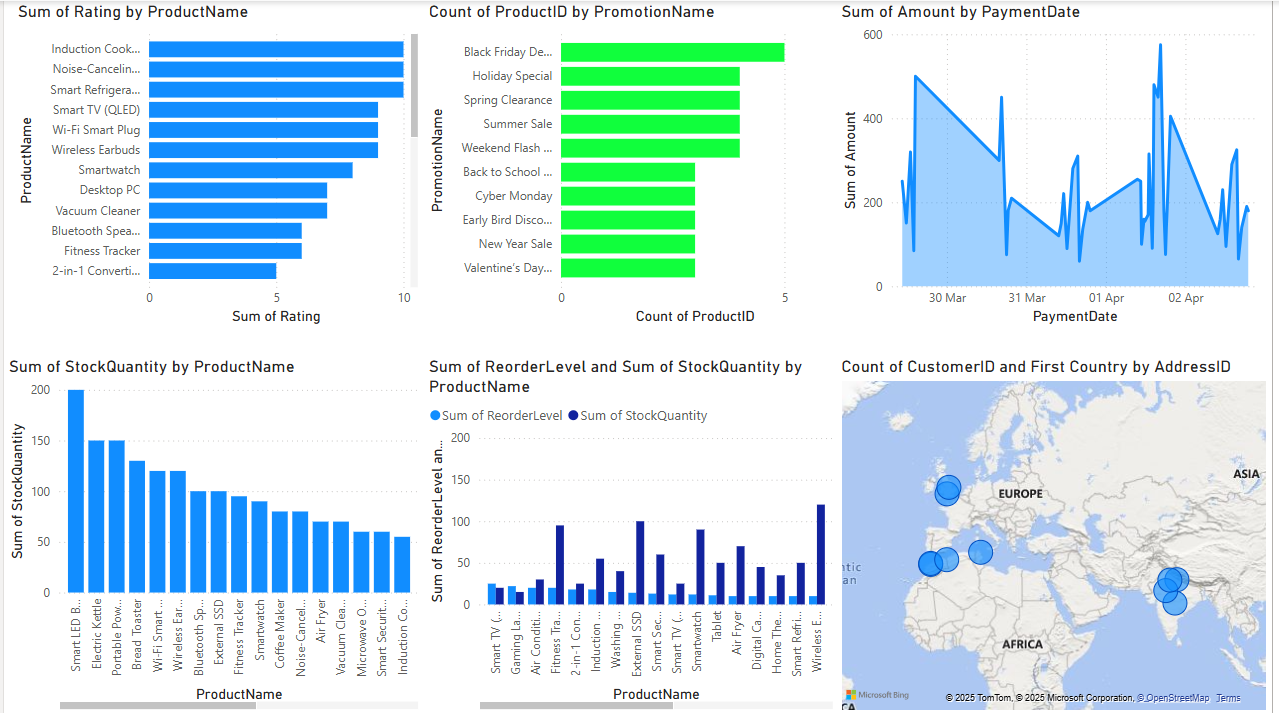
WHERE a.Country = 'India';

**Project demonstration**

**Tools used:**

* MySQL workbench
* Power BI





**Self-Learning Beyond Classroom**

Beyond the curriculum, I acquired a number of new knowledge and tools. These knowledge and tools consist of:

• Designing a SQL database from scratch - learning to identify entities and relationships, and to design a database schema.

• Normalization techniques - understanding how to normalize up to BCNF for large-scale databases to ensure data consistency and avoid data redundancy.

• Advanced SQL querying - able to write complex queries using joins, grouping, filtering, and aggregates to extract meaningful data.

• Hands-on experience using Power BI - development of engaging visualizations that enable reporting insights clearly.

• Mapping customer distribution - using geolocation data within BI tools like Power BI to visualize region trends.

• Integrating MySQL and BI tools - connecting a live MySQL database to Power BI to learn how to produce live reporting.

• Building an interactive dashboard - to design and construct a dashboard that supports a business decision-making with interactive and dynamic visual elements.

**Learning from the Project**

* **Project Planning & Execution**: I learned how to plan the development of a full-stack data system, including the order of designing tables, connecting them, and setting up data flow.
* **Problem Solving**: During implementation, several issues arose—such as resolving foreign key constraints and optimizing slow queries—which improved my debugging and analytical thinking.
* **System Thinking**: Seeing how customer data, inventory, orders, and employee interactions connect gave me a stronger grasp on how businesses manage and monitor their operations.
* **Data Validation**: I gained an appreciation for maintaining data integrity through constraints and normalization, ensuring consistent and reliable reporting.
* **Practical Use of BI Tools**: Building a dashboard was not just about design, but understanding which visualizations best communicate which metrics, depending on stakeholder needs.
* **Collaboration Readiness**: Preparing the database and BI reports in a way that could be understood and used by others taught me to document better and build user-friendly outputs.

**Challenges Faced**

* **Database Design Complexity**: Designing a normalized schema with multiple related entities was initially difficult. It took time to ensure all relationships were correctly defined without creating redundancy or anomalies.
* **Understanding Normalization**: Applying normalization up to BCNF required a deeper understanding of functional dependencies and identifying partial and transitive relationships across tables.
* **BI Integration Issues**: Connecting MySQL with Power BI was not straightforward at first. Configuring the ODBC connector and ensuring real-time syncing posed technical difficulties.
* **Data Visualization Mapping**: Translating raw SQL data into meaningful charts was sometimes challenging—especially with geographical maps, where matching addresses and regions correctly was essential for accurate visuals.
* **Learning Curve with Power BI**: While powerful, Power BI has a steep learning curve when it comes to formatting visuals, applying filters, customizing charts, and using DAX expressions.

**Conclusion**

This project served as a comprehensive learning experience, integrating database design, SQL querying, and data visualization into a practical, real-world application. The key takeaways from this project include:

* **End-to-End System Design**: Gained experience in designing a fully normalized relational database from scratch, understanding how entities interact in a business environment like inventory and order management.
* **Practical SQL Mastery**: Developed proficiency in writing advanced SQL queries involving joins, aggregations, and nested logic to derive actionable insights from data.
* **Visualization Skills**: Learned to transform raw data into meaningful and interactive dashboards using Power BI, enhancing the decision-making process.
* **Technical Integration**: Understood how to bridge backend (MySQL) with frontend visualization tools (Power BI), and handle real-time data connectivity challenges.
* **Critical Thinking**: Strengthened analytical thinking by resolving data redundancy, handling large-scale data structures, and addressing data inconsistencies.

Overall, the project bridged the gap between academic concepts and their application in industry scenarios, making me more confident and equipped for future data-driven roles.

**Git Hub Link->**

[GITHUB](https://github.com/AdityaMMantri/DBMS)