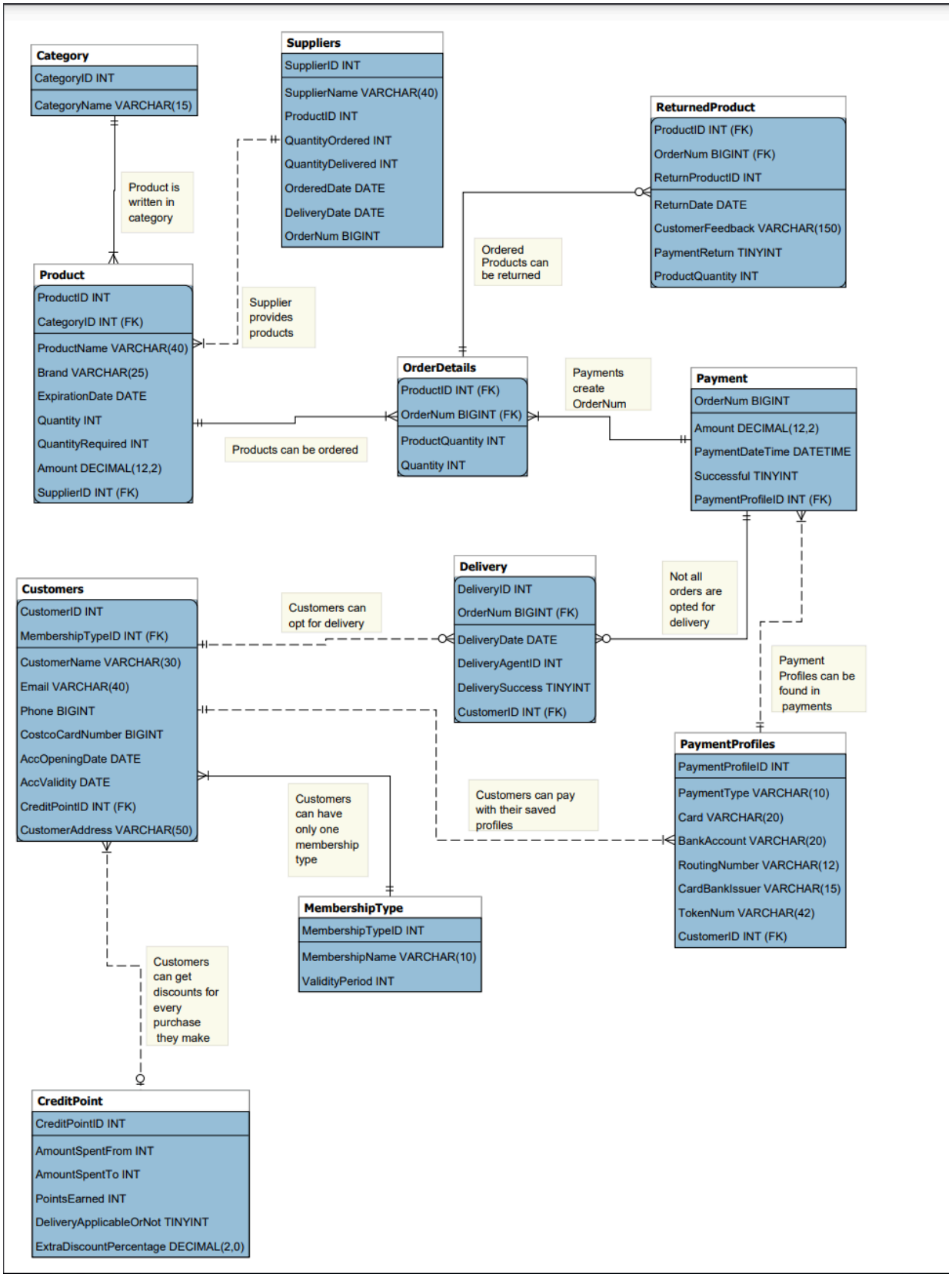
Milestone 2- Database Design
Costco Inventory Management System

Team 3
Lakshit Gupta
Zahra Ahmadi Angali
Mustafa Bhavanagarwala
Nidhi Trivedi

Mission Statement:

Costco is the third-largest retailer in the world. For the proper functioning and success of the corporation in a locality, a well-defined inventory management system forms the backbone of the organization. The large amount of data owned by the organization should be safely stored, well maintained, and rapidly accessible by its database. Thus our project aims to deliver an inventory management system that involves implementing a robust database thereby ensuring seamless management of the vast inventory, ensuring accuracy, and real-time insights which would enhance the overall supply chain, reduce costs, and elevate the customer experience. Our database is characterized by features such as efficient storage, data integrity, and performance optimization. Through our project, we aim to empower Costco to navigate the challenges of inventory management with confidence, resilience, and unparalleled efficiency, making a lasting impact on the success and sustainability of the organization.

Model


Category
CategoryID INT
CategoryName VARCHAR(15)
Product is written in category
Suppliers
SupplierID INT
SupplierName VARCHAR(40)
ProductID INT
QuantityOrdered INT
QuantityDelivered INT
OrderedDate DATE
DeliveryDate DATE
OrderNum BIGINT
Supplier provides products
ReturnedProduct
ProductID INT (FK)
OrderNum BIGINT (FK)
ReturnProductID INT
ReturnDate DATE
CustomerFeedback VARCHAR(150)
PaymentReturn TINYINT
ProductQuantity INT
Ordered Products can be returned
Product
ProductID INT
CategoryID INT (FK)
ProductName VARCHAR(40)
Brand VARCHAR(25)
ExpirationDate DATE
Quantity INT
QuantityRequired INT
Amount DECIMAL(12,2)
SupplierID INT (FK)
Products can be ordered
OrderDetails
ProductID INT (FK)
OrderNum BIGINT (FK)
ProductQuantity INT
Quantity INT
Payments create OrderNum
Payment
OrderNum BIGINT
Amount DECIMAL(12,2)
PaymentDateTime DATETIME
Successful TINYINT
PaymentProfileID INT (FK)
Customers
CustomerID INT
MembershipTypeID INT (FK)
CustomerName VARCHAR(30)
Email VARCHAR(40)
Phone BIGINT
CostcoCardNumber BIGINT
AccOpeningDate DATE
AccValidity DATE
CreditPointID INT (FK)
CustomerAddress VARCHAR(50)
Customers can opt for delivery
Delivery
DeliveryID INT
OrderNum BIGINT (FK)
DeliveryDate DATE
DeliveryAgentID INT
DeliverySuccess TINYINT
CustomerID INT (FK)
Not all orders are opted for delivery
Payment Profiles can be found in payments
PaymentProfiles
PaymentProfileID INT
PaymentType VARCHAR(10)
Card VARCHAR(20)
BankAccount VARCHAR(20)
RoutingNumber VARCHAR(12)
CardBankIssuer VARCHAR(15)
TokenNum VARCHAR(42)
CustomerID INT (FK)
Customers can pay with their saved profiles
Customers can have only one membership type
MembershipType
MembershipTypeID INT
MembershipName VARCHAR(10)
ValidityPeriod INT
Customers can get discounts for every purchase they make
CreditPoint
CreditPointID INT
AmountSpentFrom INT
AmountSpentTo INT
PointsEarned INT
DeliveryApplicableOrNot TINYINT
ExtraDiscountPercentage DECIMAL(2,0)

Queries

use mm_team03_02;

Query 1.

This SQL query retrieves the total quantity of products ordered by each customer by joining information from customer profiles, payments, and order details, providing insights into customer purchasing behavior

```
SELECT c.CustomerID, c.CustomerName, SUM(od.ProductQuantity) AS TotalProductQuantityOrdered
FROM Customers AS c
JOIN PaymentProfiles AS pp ON c.CustomerID = pp.CustomerID
JOIN Payment AS p ON pp.PaymentProfileID = p.PaymentProfileID
JOIN OrderDetails AS od ON p.OrderNum = od.OrderNum
GROUP BY c.CustomerID, c.CustomerName;
```

CustomerID	CustomerName	TotalProductQuantityOrdered
841	Christabella Giddens	6
675	Lilllie Frankham	8
758	Fraser Hatto	9
395	Waverley Nickols	9
227	Nyssa Luther	3
438	Malena Anthoin	7
89	Maddy Daley	10
513	Lilly Bleazard	1
726	Gerti Block	10
252	Kimberlee Beamond	11
674	Caryl Coyne	13
346	Regen Claesens	5
173	Aeriela Coo	12
937	Fanya Grier	9
591	Julissa Knox	13

Query 2.

This SQL query calculates the average amount spent by all customers with different payment types, offering insights into spending habits across payment methods.

```
SELECT pp.PaymentType, AVG(p.Amount) AS AvgAmountSpent
FROM Payment AS p
JOIN PaymentProfiles AS pp ON p.PaymentProfileID = pp.PaymentProfileID
GROUP BY pp.PaymentType;
```

	PaymentType	AvgAmountSpent
▶	Bank	164.108996
	Card	156.619847

Query 3.

This SQL query computes the number of deliveries made by each delivery agent on a specific date, along with the count of successful deliveries and their success rates.

```
SELECT d.DeliveryAgentID, COUNT(*) AS TotalDeliveries, SUM(CASE WHEN d.DeliverySuccess = 1
THEN 1 ELSE 0 END) AS SuccessfulDeliveries,
    AVG(CASE WHEN d.DeliverySuccess = 1 THEN 1 ELSE 0 END) AS SuccessRate
FROM Delivery AS d
WHERE d.DeliveryDate = '2023-08-19'
GROUP BY d.DeliveryAgentID;
```

	DeliveryAgentID	TotalDeliveries	SuccessfulDeliveries	SuccessRate
▶	6	2	1	0.5000
	23	1	1	1.0000
	11	1	1	1.0000
	7	1	1	1.0000

Query 4.

This SQL query identifies the top 10 suppliers based on the maximum quantity of products ordered from each supplier. It assists in supplier performance evaluation and inventory management.

```
SELECT p.SupplierID, s.SupplierName, p.ProductName, MAX(s.QuantityOrdered) AS QuantityOrdered
FROM Product AS p
JOIN Suppliers AS s ON p.SupplierID = s.SupplierID
GROUP BY p.SupplierID, s.SupplierName, p.ProductName
ORDER BY MAX(s.QuantityOrdered) DESC
```

LIMIT 10;

SupplierID	SupplierName	ProductName	QuantityOrdered
1059	Hane-Spencer	Corn - Mini	500
197	Kunze, Leffler and Raynor	Carbonated Water - Cherry	500
607	Hegmann-Lind	Beer - Guiness	500
120	Dickens Group	Pepsi - Diet, 355 Ml	499
542	Hickle-Conroy	Spinach - Packaged	498
846	Walsh LLC	Halibut - Steaks	498
642	Fadel Group	Pastry - Carrot Muffin - Mini	497
962	Flatley-Herzog	Trout - Hot Smkd, Dbl Fillet	496
953	Cassin-Mills	Lamb - Rack	496
1013	Runolfsson Group	Juice - Mango	495

Query 5.

This SQL query calculates the average amount spent per order by customers with specific membership types. It aids in understanding spending behaviors among different membership tiers.

```
SELECT mt.MembershipName, AVG(p.Amount) AS AvgAmountSpent
FROM Payment AS p
JOIN PaymentProfiles AS pp ON p.PaymentProfileID = pp.PaymentProfileID
JOIN Customers AS c ON pp.CustomerID = c.CustomerID
JOIN MembershipType AS mt ON c.MembershipTypeID = mt.MembershipTypeID
GROUP BY mt.MembershipName;
```

MembershipName	AvgAmountSpent
Silver	163.357077
Gold	162.362017
Platinum	154.545189

Query 6.

This SQL query retrieves the total number of products returned by each customer along with their feedback. It provides valuable insights into customer satisfaction and product quality issues, enabling businesses to address concerns.

```
SELECT c.CustomerID, c.CustomerName, COUNT(rp.ReturnProductID) AS TotalProductsReturned,
    GROUP_CONCAT(rp.CustomerFeedback) AS CustomerFeedback
FROM Customers AS c
JOIN PaymentProfiles AS pp ON c.CustomerID = pp.CustomerID
JOIN Payment AS p ON pp.PaymentProfileID = p.PaymentProfileID
```

```
JOIN OrderDetails AS od ON p.OrderNum = od.OrderNum
JOIN ReturnedProduct AS rp ON od.OrderNum = rp.OrderNum and od.ProductID = rp.ProductID
GROUP BY c.CustomerID, c.CustomerName;
```

CustomerID	CustomerName	TotalProductsReturned	CustomerFeedback
6	Laney Boodle	1	Feedback
14	Malorie Lerhinan	1	Feedback
15	Trisha Batting	2	Feedback,Feedback
31	Noah Cordelle	1	Feedback
32	Maximilian Hainey	1	Feedback
38	Frans Cristofolo	1	Feedback
40	Upton Strood	1	Feedback
41	Norine Ringham	2	Feedback,Feedback
42	Millard Schutze	1	Feedback
47	Cyrill Christoforou	1	Feedback
52	Saloma Tunna	1	Feedback
59	Gretna Crookshank	1	Feedback
61	Eleanor Kyllford	1	Feedback
70	Juanita Giovanizio	1	Feedback
76	Pattie Domerq	1	Feedback

Query 7.

This SQL query calculates the total number of products returned by customers who have spent more than a certain amount on their orders. It assists in identifying patterns of product dissatisfaction or quality issues among higher-spending customers.

```
SELECT c.CustomerID, c.CustomerName,
   COUNT(rp.ReturnProductID) AS TotalProductsReturned
FROM Customers AS c
JOIN PaymentProfiles AS pp ON c.CustomerID = pp.CustomerID
JOIN Payment AS p ON pp.PaymentProfileID = p.PaymentProfileID
JOIN OrderDetails AS od ON p.OrderNum = od.OrderNum
JOIN ReturnedProduct AS rp ON od.OrderNum = rp.OrderNum
WHERE p.Amount > 100
GROUP BY c.CustomerID, c.CustomerName;
```

CustomerID	CustomerName	TotalProductsReturned
758	Fraser Hatto	1
937	Fanya Grier	2
38	Frans Cristofolo	1
767	Corena Kloser	2
645	Silvio Okey	1
322	Melba Loftus	2
15	Trisha Batting	4
47	Cyrill Christoforou	3
365	Bertine Gile	1
534	Meir McLoughlin	1
907	Phaidra Jirusek	2
70	Juanita Giovanizio	1
32	Maximilian Hainey	1
549	Rosamond Heas...	2
41	Norine Ringham	3

Query 8.

This SQL query retrieves a list of all customers who have purchased more than 5 products, regardless of whether all their orders are recorded. By joining order details, payments, payment profiles, and customer information, it provides insights into high-volume purchasers.

```
SELECT c.CustomerID, c.CustomerName, COALESCE(COUNT(od.OrderNum), 0) AS TotalOrders
FROM OrderDetails AS od
RIGHT JOIN Payment AS p ON od.OrderNum = p.OrderNum
JOIN PaymentProfiles AS pp ON p.PaymentProfileID = pp.PaymentProfileID
RIGHT JOIN Customers AS c ON pp.CustomerID = c.CustomerID
GROUP BY c.CustomerID, c.CustomerName
HAVING TotalOrders > 5;
```

	CustomerID	CustomerName	TotalOrders
▶	15	Trisha Batting	9
	72	Addie Forkan	6
	105	Agata Venditto	6
	255	Cesar Vardy	7
	389	Crissy Bucklee	6
	537	Tessy Jiggen	6
	591	Julissa Knox	7
	594	Brandyn Gregolotti	6
	597	Rogers Bearward	6
	699	Alberik Kyte	6
	702	Bennett Dellow	6
	824	Calli Norway	6
	979	Bette-ann Morris	6

Query 9.

This SQL query calculates the number of total orders and the number of orders that are being delivered. By utilizing a LEFT OUTER JOIN between the Payment and Delivery tables on the OrderNum field, it ensures that all orders are counted, regardless of whether they have corresponding delivery records. This provides insights into order fulfillment and delivery efficiency.

```
SELECT COUNT(p.OrderNum) AS NumberOfOrders,
    COUNT(d.OrderNum) AS NumberOfDeliveries
FROM Payment AS p
LEFT OUTER JOIN
Delivery AS d ON p.OrderNum = d.OrderNum;
```

	NumberOfOrders	NumberOfDeliveries
▶	1372	1000

Query 10.

This SQL query retrieves the top 10 categorical months with the highest amount of sales, formatted with the financial year. By grouping payment data by month, summing up the sales amounts, and ordering the results in descending order, it provides insights into sales performance over time, facilitating strategic decision-making.

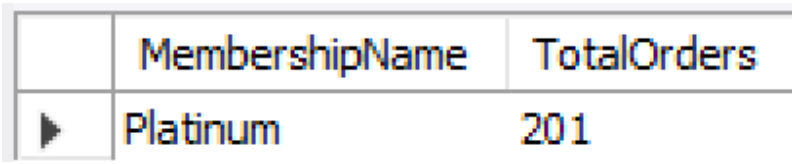
```
SELECT DATE_FORMAT(PaymentDateTime, '%Y-%m') AS PaymentMonth,
    SUM(Amount) AS TotalSales
FROM Payment
GROUP BY DATE_FORMAT(PaymentDateTime, '%Y-%m')
ORDER BY TotalSales DESC
LIMIT 10;
```

	PaymentMonth	TotalSales
▶	2023-06	16218.67
	2023-12	15057.32
	2023-10	14145.83
	2023-04	13209.49
	2023-08	13161.24
	2023-03	12945.27
	2023-05	12566.72
	2023-09	12537.35
	2023-11	12390.08
	2024-01	12362.57

Query 11.

This SQL query calculates the total number of orders placed by customers who have a membership type with a validity period of more than 2 year. It achieves this by filtering orders based on membership types with a validity period greater than 2 year, using nested subqueries to retrieve relevant data from the Customers, PaymentProfiles, Payment, and MembershipType tables.

```
SELECT m.MembershipName, COUNT(*) AS TotalOrders
FROM OrderDetails AS od
JOIN Payment AS p ON od.OrderNum = p.OrderNum
JOIN PaymentProfiles AS pp ON p.PaymentProfileID = pp.PaymentProfileID
JOIN Customers AS c ON pp.CustomerID = c.CustomerID
JOIN MembershipType AS m ON c.MembershipTypeID = m.MembershipTypeID
WHERE c.MembershipTypeID IN (
      SELECT MembershipTypeID
      FROM MembershipType
      WHERE ValidityPeriod > 2
   )
GROUP BY m.MembershipName;
```



	MembershipName	TotalOrders
▶	Platinum	201

Stored Procedures

*Note: For it to run on localhost please change the create procedures like mentioned below:

CREATE DEFINER=`root`@`localhost` PROCEDURE `QuantityByCustNCatID`

Stored Procedure 1.

This procedure calculates the total number of orders placed by a customer for a given category.

CALL QuantityByCustNCatID(888, 1003);

	Quantity	CategoryName	ProductID	CategoryID
▶	15	Dairy	10005527	1003

Stored Procedure 2.

Takes a customer ID as input and returns the following information:
Total number of orders placed by the customer (Total_orders).
Total amount spent by the customer (Total_amount).
Average amount spent per order by the customer (Average_amount).

CALL GetCustomerPurchaseHistory(888, @Total_orders, @Total_amount, @Average_amount);
SELECT @Total_orders, @Total_amount, @Average_amount;

	@Total_orders	@Total_amount	@Average_amount
▶	3	1616.40	538.80