University for the Creative Arts

BERLIN SCHOOL OF BUSINESS & INNOVATION

Essay / Assignment Title: Online Marketplace Data Warehouse Management & Business Analytics

Programme title: Enterprise Data Warehouses and Database Management Systems

Name: Darshan Thevar Mahalingam

Year: 2025

CONTENTS

LIST OF FIGURES

Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).

Dharshan.T.M

Name and Surname (Capital letters):

DARSHAN THEVAR MAHALINGAM

Date: 2025/02/03

ABSTRACT

The online shopping platform is growing fast and rapidly, with the access to internet this way of shopping is very popular among customers due to its ease of access. The rapid growth demands robust and scalable database solutions to manage huge amounts of data efficiently. This project focuses on developing a database for an Online Furniture Marketplace, connecting vendors and customers and provide analytical decision-making and operational management. The database is implemented using MySQL and with additional features such as customer insights, real-time shipping updates, and a fraud management system, the platform delivers data which can be used to gather business insights.

Keywords: Database, Fraud Management, MySQL.

CHAPTER 1 - INTRODUCTION

In today's digital era, the convenience of online shopping has made people to purchase products easier, including furniture. The Online Furniture Marketplace is a bridge between customers and vendors. Here, customers can easily explore furniture options and varieties on their comfort and place orders. Also, they can share feedback on purchased items, which helps in credibility of the product. Vendors can manage their stocks and inventory also for them necessity of a physical store and expenses like rent and employee expenses can be reduced which helps vendors to quote a competitive price on their products and increase their sales.

Using the data we can gain valuable insights which ultimately benefits customers with personalized engagement and vendors to identify customer needs and improve their sales. We have used MYSQL to design our database.

In the '**Customers**' table, it contains some basic information about customers and their membership status. We also have another table '**SiteVisits**' table which has the customer behaviour data, like time spent on website. By these information we can gather valuable insights to understand the customers needs and provide a good user experience. Also, the '**Rewards**' table which provides a system that attracts customers in repeat purchases. Also it helps customers to track their reward points. This helps in customer loyalty and repeated purchasing.

For vendors, our platform simplifies operations and helps increase in sales. '**Vendors**' table, linked with '**ProductVendors**', ensures that each vendor has a clear information on their product listings and their sales numbers. We also have Product Promotions which advertises the products to customers. This is managed using the '**Promotions**' table linked with '**ProductPromotions**' tables, which enables vendors to provide offers and flash sales, and helps in more sales while offering customers competitive and best deals. These features helps vendors to grow their businesses effectively in the digital marketplace.

The '**Orders**' table, which is linked to '**Payments**', ensures each and every payment transaction with their status is logged. Helps to track the payment. And once the payment is success and order is confirmed. Our '**Shipment**' table will update the customers regarding the delivery status. These tables here provide a vital data for customers because once the order is confirmed customers need the visibility and statuses of their ordered products which enhances the customer experience.

Also, we have a dedicated '**FraudAndCosts'** table helps identify if their is any issue in orders placed or any disputes. This helps to resolve disputes, minimizing losses and maintaining the platform's trust.

1.1 OBJECTIVES

Evaluating Promotions Effectiveness: By Analyzing the effectiveness of promotions that will increase revenue and sales volume. It is very important to understand promotion effectiveness to optimize the budget.

Identifying High-Value Customers: Identifying the top customers contributing the most revenue in our marketplace, will help in getting to know their purchase frequency. These customers help in increasing the revenue. By identifying them, the company can prioritize loyalty programs and personalized offers.

Evaluate Product Performance: Understanding the highest selling products are and checking whether customer reviews affect with sales. It also helps to optimize the inventory, their prizing to improve profits.

In next chapter we have discussed in detail on database design journey, starting from a conceptual Entity-Relationship (ER) diagram and progressing to a normalized relational schema in MySQL. We also have discussed Advanced SQL queries to show how the database can deliver actionable insights, such as identifying best-selling products or analyzing vendor performance etc. The implementation also includes theoretical knowledge like ACID properties and CAP theorem, which ensures data integrity and efficient query execution.

CHAPTER 2 - ENTITY RELATIONSHIP DIAGRAM

In this chapter we will be discussing the entities and attributes used in our database for online furniture marketplace. Also, the relationships between each attributes is explained in detail.

ENTITIES

Customers table is used store the information about each individuals who use our marketplace.They may be Guest or a member, here we have **‘CustomerID**’ which is a unique identifier for each customer. **‘Name’** will have the full name of the customer, which helps in personalizing their experience. **‘Email’** is used both for communication with them and as a unique login credential. **‘Phone’** stores the customer’s contact number for support or to get order updates. **‘Address’** will hold the customer’s delivery address for orders. **‘CustomerType’** will help in differentiating between Guest and member customers, allowing to offer different features and user experience. **‘JoinDate’** captures the date when the customer is registered on the platform.

Orders table is used to track every purchase made by customers. **‘OrderID’** is the unique for each order, **‘OrderDate’** is the date when the order was placed, **‘TotalAmount’** stores the total cost of the order placed. **‘Status’** shows the live status of order such as Pending, Shipped, Delivered, or Cancelled.

OrderDetails table has the detailed information about each product within an order. Here, **‘OrderDetailID’** is a unique identifier for each record in the table. **‘Quantity’** tracks how many number of products were ordered, **‘PriceAtPurchase’** records the price of the product at the time of the transaction.

Products table is used to store the items available for purchase on the platform. **‘ProductID’** is unique number used to identify each product. **‘Name’** stores the product name like “Wooden Dining Table” or “Leather Sofa”, **‘Category’** classifies product types **‘BasePrice’** will have the original price of the product. **‘Description’** provides additional details about the product.

SiteVisits table is used to track customer interactions with the platform. **'VisitID'** will be the unique identifier for each visit. **'TimeSpent'** records the total time that a customer spends on the platform during a single visit. **'VisitDate'** stores the date of the visit.

Reviews table is used to store feedback from customers about products they have purchased. **'ReviewID'** is unique ID used to identify each review. **'Rating'** is the score given from 1 to 5 stars. **'ReviewText'** contains the customer's written feedback about the product, sharing their experience or issues. **'ReviewDate'** has the date when review was posted.

Payments table is used to store and track payment transactions made for orders. **'PaymentID'** is a unique id for each payment. **'PaymentDate'** stores the date of payment was processed. **'PaymentMethod'** is used to store the payment type used, such as Credit Card, PayPal, or Bank Transfer. **'Amount'** is the total amount paid by the customer. **'Status'** shows the payment status like (e.g., Completed, Pending, or Failed).

Vendors table is used to stores information about the vendors who are supplying products on the platform. **'VendorID'** is a unique identifier for each vendor. **'Name'** is the vendor's business name. **'Email'** is used to contact the vendor, and **'Phone'** is the phone number of vendor. **'Address'** stores the details of vendor's business location. **'RegDate'** stores the date of registration of vendor on the platform.

FraudAndCosts table stores the data related to fraud, disputes. **'CostID'** is a unique attribute used to identify each record. **'IssueType'** describes problem (e.g., fraud, payment dispute). **'Description'** will provide additional details about the issue. **'Cost'** stores the amount if there is any financial losses resulting from the issue. **'ReportedDate'** is the date when the issue was raised. **'ResolutionStatus'** shows the status of issue, **'ResolvedDate'** is the date when the issue was closed.

Shipment table tracks the delivery data of orders. **'ShipmentID'** is unique for each shipment. **'Courier'** stores the delivery partner handling the shipment (e.g., FedEx, DHL). **'ShipmentDate'** is the date when order was shipped for delivery, and its status is updated by **'Status'.**

Promotions table stores details about promotional offers on the products. **'PromotionID'** is a unique ID for each promotion. **'PromotionName'** is used to store the name of the promotional offer (e.g., "Winter Sale"). **'StartDate'** and **'EndDate'** defines the duration of the promotion. And **‘DiscountType’** indicates the kind of promotion whether it is based percentage or fixed amount, while **‘DiscountValue’** stores the exact value of the discount.

Rewards table is used to track customer loyalty and reward points. **‘RewardID’** is a unique ID for each record. **‘RewardPoints’** is used to store the total points gathered by a customer, **‘RewardTier’** specifies the customer’s membership status (e.g., Bronze, Silver, Gold). **‘LastUpdated’** holds the recent date when the reward account was updated.

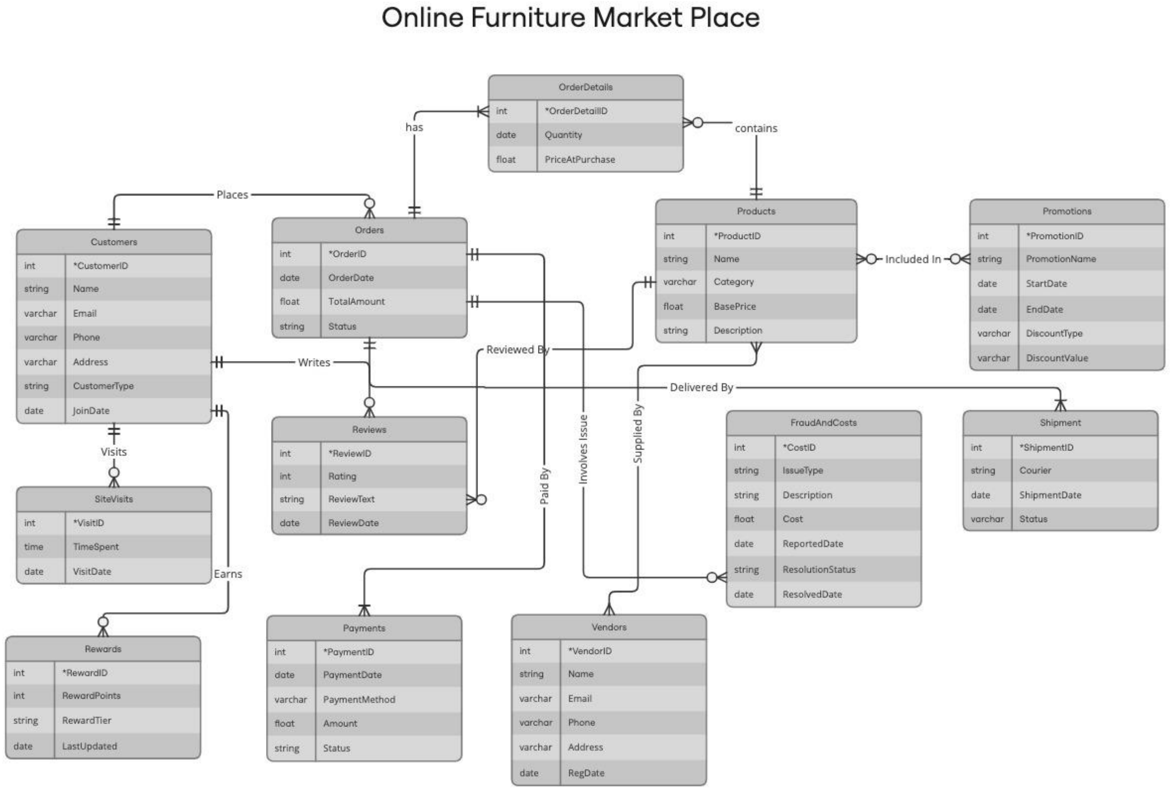


Figure 1: Entity Relationship Diagram

In Figure 1 the ER diagram developed for the Online furniture marketplace is represented, and the relationships between each entities is shown. A Customer has a one-to-many (1:N) relationship with Orders entity, because each customer can place multiple orders, but each order belongs to a customer. Customers and Reviews entity has a one-to-many (1:N) relationship here, each customer can write many reviews, but each review can be written by only one customer. Customers entity has a one-to-many (1:N) relationship with SiteVisits, because a customer can visit the site multiple times, but a site visit can be of only one customer. Customers and Rewards entity has (1:N) relationship, each customer can have may rewards account but a reward account can be of only one customer.

An Order entity has a one-to-many (1:N) relationship with OrderDetails entity, because here one order can contain multiple products, but each OrderDetails can belong to only one order. Orders has a one-to-many (1:N) relationship with Shipments, as each order can have one or more shipment depending upon their quantity but a shipment can be of only one order. Additionally, Payments entity have a many-to-one (N:1) relationship with Orders, as each order can have multiple payments, and each payment corresponds to a single order. Orders have an (1:N) relationship with FraudAndCosts because each order can have multiple issue related to frauds but a Fraud must be one order.

The OrderDetails entity has a many-to-one (N:1) relationship with Products entity, here each product can appear in multiple order details, but each record in OrderDetails is of a single product.

The Products entity has a many-to-many (M:N) relationship with Promotions entity, because a single product can be part of multiple promotions, and a single promotion can apply to many products. Products and Reviews has a one-to-many (1:N) relationship. Here, a product can have many reviews, but each review can be of a single product. Vendors and Products entities has a many-to-many (M:N) relationship, as each vendor can supply or sell multiple products, but each product is supplied by many vendors.

The FraudAndCosts entity captures issues related to shipments, forming a one-to-many (1:N) relationship with Shipments, as a single shipment can have multiple costs, but each cost relates to one shipment.

This is all the relationships between entities for our online furniture marketplace. In next Chapter we will be discussing the Schema creation, MYSQL implementation and data generation for our database.

CHAPTER 3 - DATABASE SCHEMA

In this Chapter, we will discuss on the database schema and the database implementation for our Online Furniture Marketplace. Here, we are MySQL to implement our database because for our use-case we need a relational database because each entity is linked/related with another. Using MySQL ensures data integrity through primary keys, foreign keys and provides scalability.

3.1 SCHEMA

Customers (PK(CustomerID), Name, Email, Phone, Address, CustomerType, JoinDate);

Customers table stores details about each customer, CustomerID is the primary key. This table has attributes such as Name, Email, Phone, Address, CustomerType (e.g., Guest or Member), and JoinDate.

Orders (PK(OrderID), OrderDate, TotalAmount, Status, FK(CustomerID));

Orders table records purchases made by customers. With a primary key OrderID. Also, there are attributes like OrderDate, TotalAmount, and Status (e.g., Pending, Completed). The foreign key CustomerID references the Customers table, linking each order to a specific customer.

OrderDetails(PK(OrderDetailID),Quantity,PriceAtPurchase,FK(OrderID),FK(ProductID));

OrderDetails table is used for tracking of individual products in a order, including the quantity and price at the time of purchase. Each has a primary key OrderDetailID. We have attributes such as Quantity, PriceAtPurchase, and foreign keys OrderID and ProductID, which reference the Orders and Products tables, respectively.

Products (PK(ProductID), Name, Category, BasePrice, Description);

Products table is used to store catalog of all items available in the marketplace, with each record identified by the primary key ProductID. Attributes include Name, Category, BasePrice, and Description.

SiteVisits (PK(VisitID), TimeSpent, VisitDate, FK(CustomerID));

SiteVisits table is used to track customer interactions with the website. Each site visit is tracked by the primary key VisitID. It has attributes such as TimeSpent and VisitDate, with the foreign key CustomerID referencing the Customers table.

Reviews(PK(ReviewID), Rating, ReviewText, BasePrice, ReviewDate, FK(CustomerID), FK(ProductID));

Reviews table used to store customer feedback for products, with each review having primary key ReviewID. It has attributes like Rating, ReviewText, BasePrice, and ReviewDate. The foreign keys CustomerID and ProductID link reviews to the Customers and Products tables.

Payments (PK(PaymentID), PaymentDate, PaymentMethod, Amount, Status, RegDate, FK(OrderID));

Payments table records payment transactions. Each payment is has a primary key PaymentID and includes attributes such as PaymentDate, PaymentMethod, Amount, Status (e.g., Successful, Failed), and RegDate. The foreign key OrderID references the Orders table.

Vendors (PK(VendorID), Name, Email, Phone, Address, RegDate);

Vendors table stores information about suppliers, with a primary key VendorID. Attributes include Name, Email, Phone, Address, and RegDate.

FraudAndCosts(PK(CostID),IssueType,Description,Cost,ReportedDate,ResolutionStatus, ResolvedDate, FK(OrderID));

FraudAndCosts table is used to store data related to incidents, fraud or operational costs, unique by the primary key CostID. Including attributes IssueType, Description, Cost, ReportedDate, ResolutionStatus, and ResolvedDate. The foreign key OrderID references the Orders table.

Shipments (PK(ShipmentID), Courier, ShipmentDate, Status, FK(OrderID));

Shipment table is used to store delivery information for orders, with each shipment uniquely identified by the primary key ShipmentID. It includes attributes such as Courier, ShipmentDate, and Status. The foreign key OrderID references the Orders table.

Promotions(PK(PromotionID), PromotionName, StartDate, EndDate, DiscountType, DiscountValue);

Promotions table is used to manage discounts and promotional campaigns for products, with the primary key PromotionID. Attributes include PromotionName, StartDate, EndDate, DiscountType, and DiscountValue.

Rewards (PK(RewardID), RewardPoints, RewardTier, LastUpdated, FK(CustomerID));

Rewards table is used to store customer loyalty points, with unique the primary key RewardID. Attributes include RewardPoints, RewardTier (e.g., Bronze, Silver, Gold), and LastUpdated. The foreign key CustomerID references the Customers table.

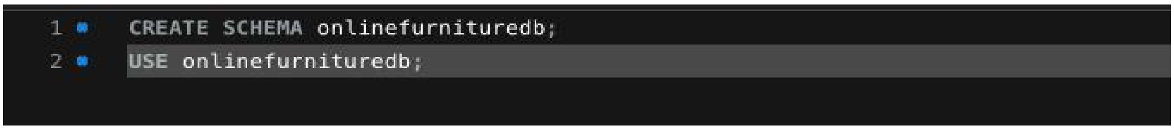
ProductPromotions (FK(ProductID) FK(PromotionID));

ProductPromotions table handles the many-to-many relationship between products and promotions.

ProductVendors (FK(ProductID), FK(VendorID), Commission rate, Commission amount);

The ProductVendors table tracks the relationship between products and vendors.

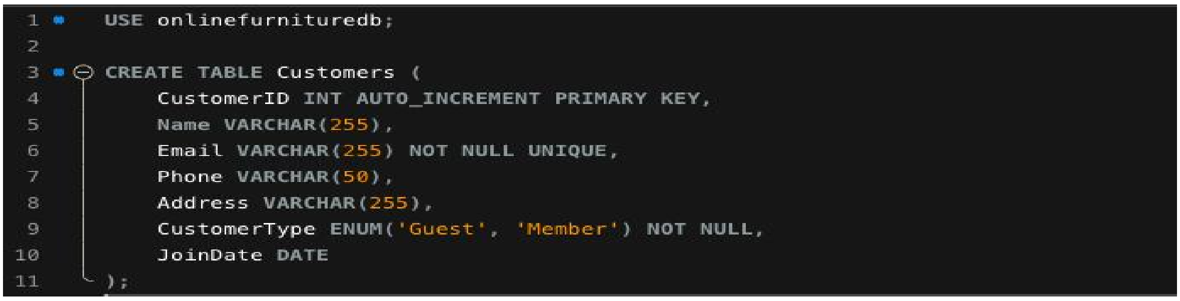
3.2 MySQL IMPLEMENTATION



```sql
CREATE SCHEMA onlinefurnituredb;
USE onlinefurnituredb;
```

Figure 2: Create Schema

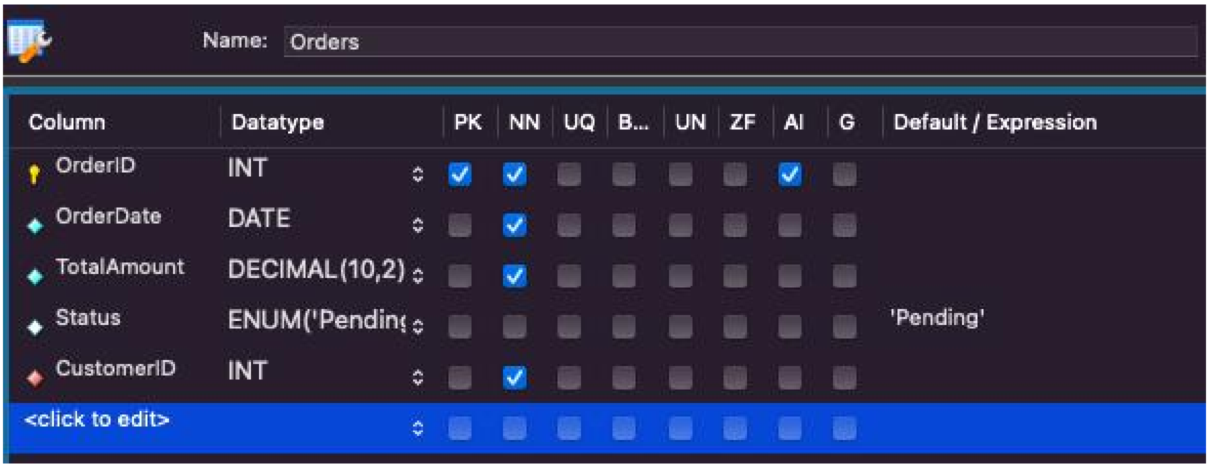
In Figure 2, we are creating Schema for our Online Furniture Marketplace in MySQL using MySQL workbench.



```sql
USE onlinefurnituredb;

CREATE TABLE Customers (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255) NOT NULL UNIQUE,
    Phone VARCHAR(50),
    Address VARCHAR(255),
    CustomerType ENUM('Guest', 'Member') NOT NULL,
    JoinDate DATE
);
```

Figure 3: Customers Table

In Figure 3, the Customers table is created using MySQL with the attributes like CustomerID which is a Primary Key and have AUTO_INCREMENT so that for every record the ID will be auto incremented. Email is Unique because each customer should have a mail address. Customer Type is a ENUM (Guest or Member).

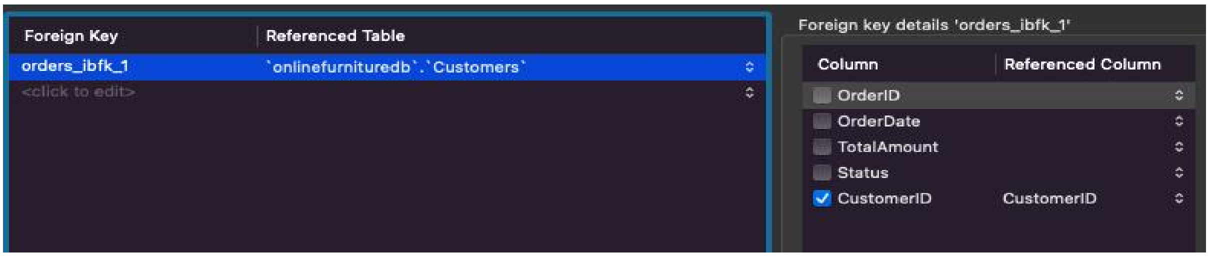


Name: Orders

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
OrderID	INT	☑	☑	□	□	□	□	☑	□	
OrderDate	DATE	□	☑	□	□	□	□	□	□	
TotalAmount	DECIMAL(10,2)	□	☑	□	□	□	□	□	□	
Status	ENUM('Pendin(	□	□	□	□	□	□	□	□	'Pending'
CustomerID	INT	□	☑	□	□	□	□	□	□	
<click to edit>		□	□	□	□	□	□	□	□	

Figure 4: Orders Table

In Figure 4 the table Orders is created with all the columns, the status is a ENUM with values ‘Pending’, ‘Shipped’, ‘Delivered’, ‘Returned’) and ‘Pending’ is Default Value



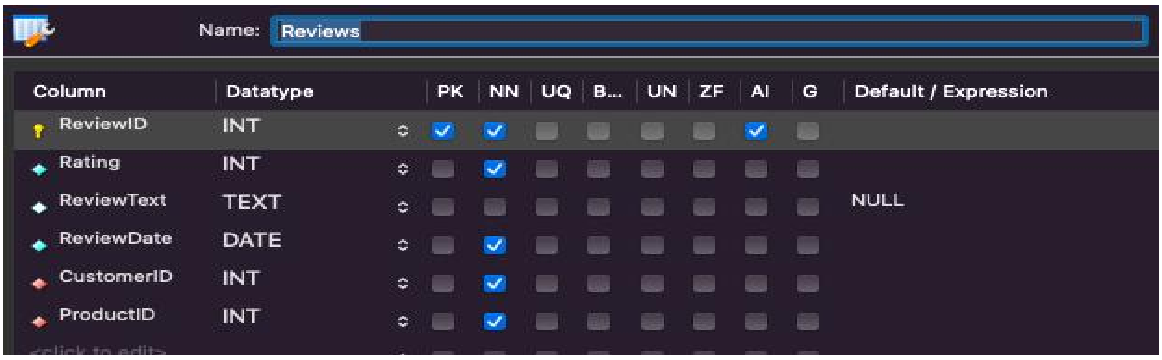
Foreign Key	Referenced Table
orders_ibfk_1	\`onlinefurnituredb\`.\`Customers\`
<click to edit>	

Foreign key details 'orders_ibfk_1'

Column	Referenced Column
□ OrderID	
□ OrderDate	
□ TotalAmount	
□ Status	
☑ CustomerID	CustomerID

Figure 5: Orders Table Foreign Key

Figure 5 represents the foreign key CustomerID in Orders table which is referenced with CustomerID from customer Table.

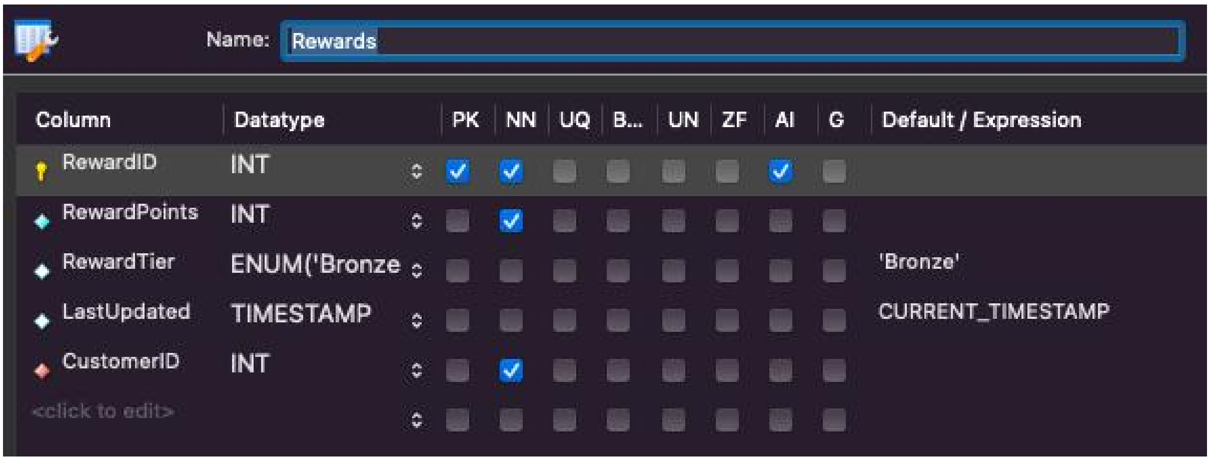


Name: Reviews

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ReviewID	INT	☑	☑	□	□	□	□	☑	□	
Rating	INT	□	☑	□	□	□	□	□	□	
ReviewText	TEXT	□	□	□	□	□	□	□	□	NULL
ReviewDate	DATE	□	☑	□	□	□	□	□	□	
CustomerID	INT	□	☑	□	□	□	□	□	□	
ProductID	INT	□	☑	□	□	□	□	□	□	
<click to edit>										

Figure 6: Review Table

Figure 6 shows the Review Table, here we have 2 Foreign keys CustomerID is referenced with CustomerID from customer table and ProductID from Products Table.

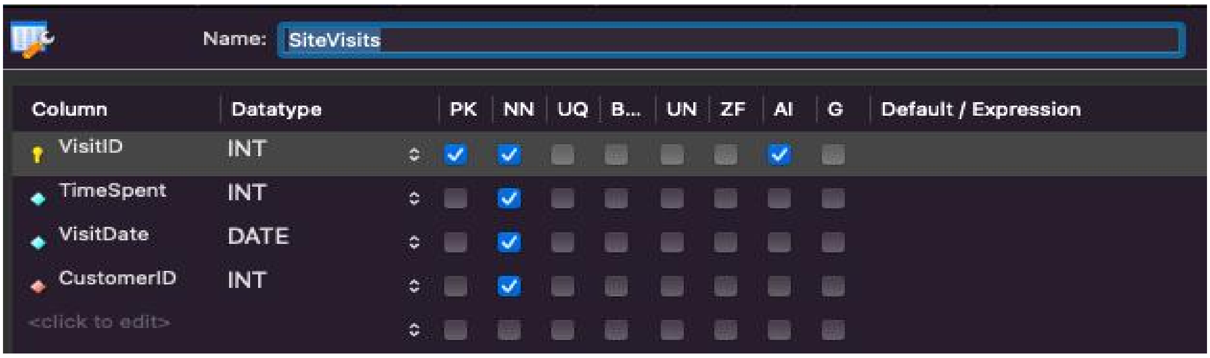


Name: Rewards

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
RewardID	INT	☑	☑	□	□	□	□	☑	□	
RewardPoints	INT	□	☑	□	□	□	□	□	□	
RewardTier	ENUM('Bronze	□	□	□	□	□	□	□	□	'Bronze'
LastUpdated	TIMESTAMP	□	□	□	□	□	□	□	□	CURRENT_TIMESTAMP
CustomerID	INT	□	☑	□	□	□	□	□	□	
<click to edit>		□	□	□	□	□	□	□	□	

Figure 7: Rewards Table

Figure 7 shows the Rewards Table, here we have 1 Foreign key CustomerID is referenced with CustomerID from customer table.

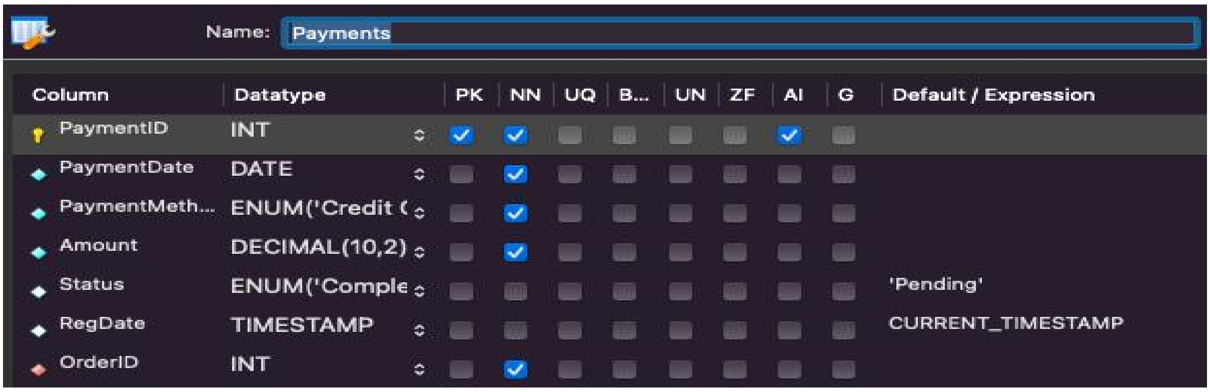


Name: SiteVisits

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
VisitID	INT	☑	☑	□	□	□	□	☑	□	
TimeSpent	INT	□	☑	□	□	□	□	□	□	
VisitDate	DATE	□	☑	□	□	□	□	□	□	
CustomerID	INT	□	☑	□	□	□	□	□	□	
<click to edit>		□	□	□	□	□	□	□	□	

Figure 8: SiteVisits Table

Figure 8 represents the SiteVisits Table, here we have 1 Foreign key CustomerID is referenced with CustomerID from customer table.

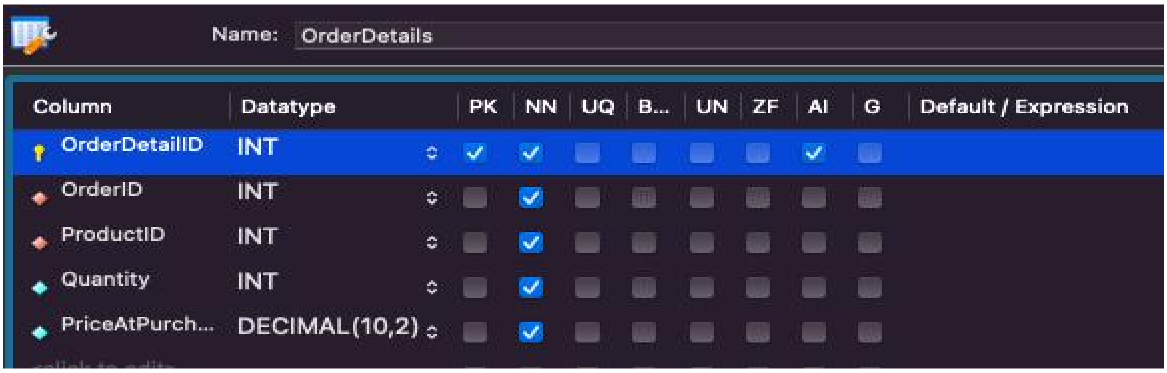


Name: Payments

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
PaymentID	INT	☑	☑	□	□	□	□	☑	□	
PaymentDate	DATE	□	☑	□	□	□	□	□	□	
PaymentMeth...	ENUM('Credit (	□	☑	□	□	□	□	□	□	
Amount	DECIMAL(10,2)	□	☑	□	□	□	□	□	□	
Status	ENUM('Comple	□	□	□	□	□	□	□	□	'Pending'
RegDate	TIMESTAMP	□	□	□	□	□	□	□	□	CURRENT_TIMESTAMP
OrderID	INT	□	☑	□	□	□	□	□	□	

Figure 9: Payments Table

Figure 9 shows the Payments Table, here we have 1 Foreign key OrderID is referenced with OrderID from Orders table.

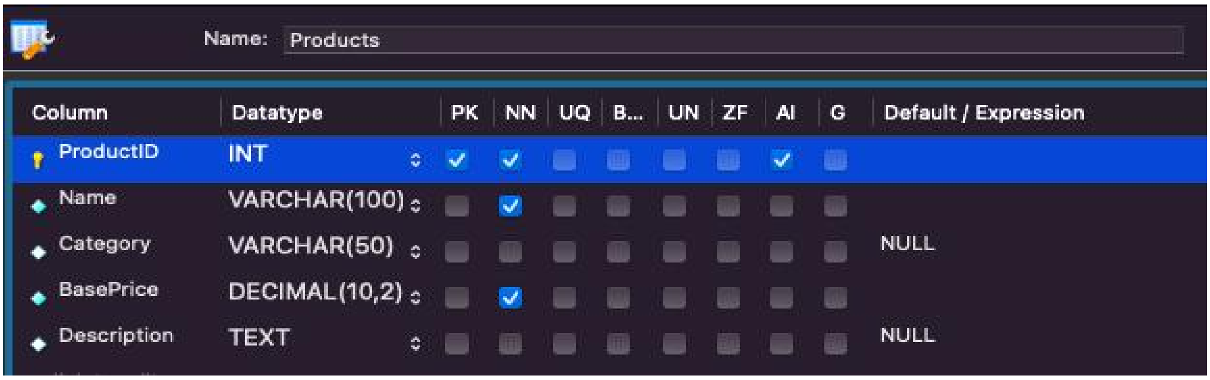


Name: OrderDetails

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
OrderDetailID	INT	☑	☑	□	□	□	□	☑	□	
OrderID	INT	□	☑	□	□	□	□	□	□	
ProductID	INT	□	☑	□	□	□	□	□	□	
Quantity	INT	□	☑	□	□	□	□	□	□	
PriceAtPurch...	DECIMAL(10,2)	□	☑	□	□	□	□	□	□	

Figure 10: Order Details Table

In Figure 10 the Order Details Table, has we 2 Foreign keys OrderID is referenced with OrderID from Orders table and ProductID is referenced with ProductID from Products Table.

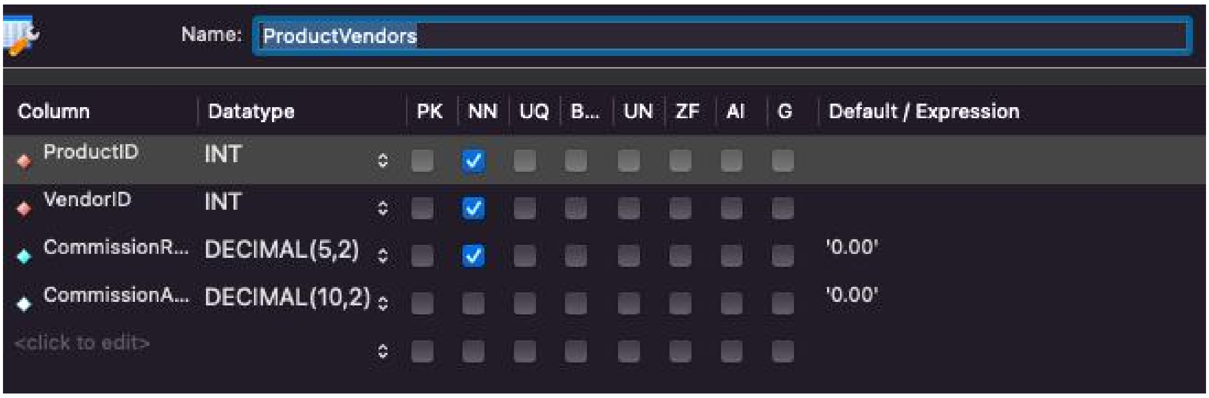


Name: Products

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ProductID	INT	☑	☑	□	□	□	□	☑	□	
Name	VARCHAR(100)	□	☑	□	□	□	□	□	□	
Category	VARCHAR(50)	□	□	□	□	□	□	□	□	NULL
BasePrice	DECIMAL(10,2)	□	☑	□	□	□	□	□	□	
Description	TEXT	□	□	□	□	□	□	□	□	NULL

Figure 11: Products Table

In Figure 11 the Products Table, has all required columns with ProductID as Primary Key.

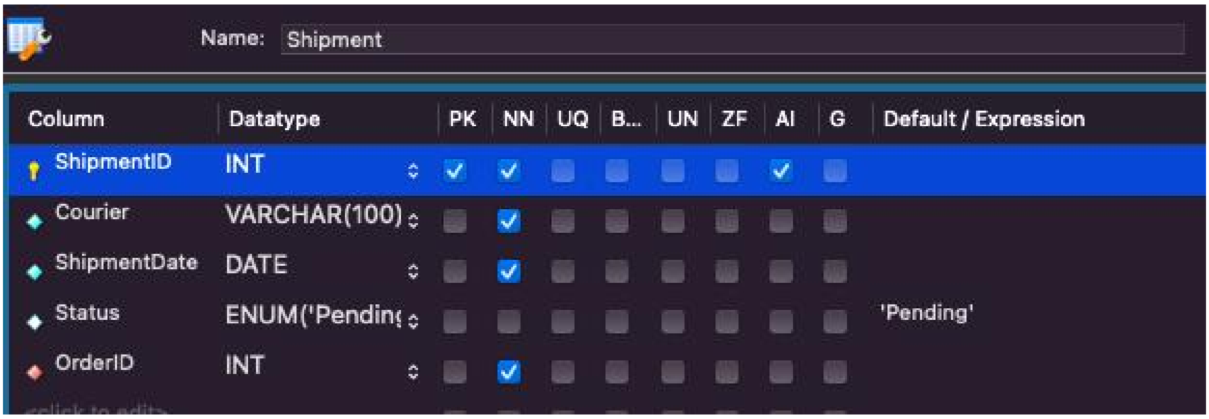


Name: ProductVendors

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ProductID	INT	□	☑	□	□	□	□	□	□	
VendorID	INT	□	☑	□	□	□	□	□	□	
CommissionR...	DECIMAL(5,2)	□	☑	□	□	□	□	□	□	'0.00'
CommissionA...	DECIMAL(10,2)	□	□	□	□	□	□	□	□	'0.00'
<click to edit>		□	□	□	□	□	□	□	□	

Figure 12: Promotions Table

In Figure 12 the Promotions Table, has all required columns with PromotionID as Primary Key.

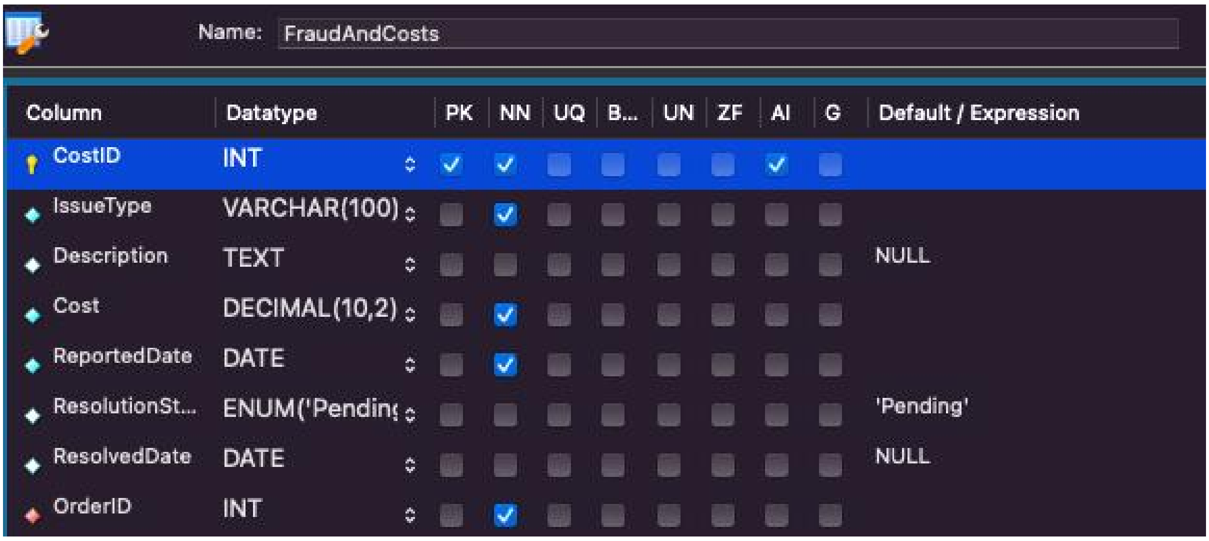


Name: Shipment

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ShipmentID	INT	☑	☑	□	□	□	□	☑	□	
Courier	VARCHAR(100)	□	☑	□	□	□	□	□	□	
ShipmentDate	DATE	□	☑	□	□	□	□	□	□	
Status	ENUM('Pendin(	□	□	□	□	□	□	□	□	'Pending'
OrderID	INT	□	☑	□	□	□	□	□	□	

Figure 13: Shipment Table

Figure 13 the Shipment Table, has 1 Foreign keys OrderID is referenced with OrderID from Orders table.

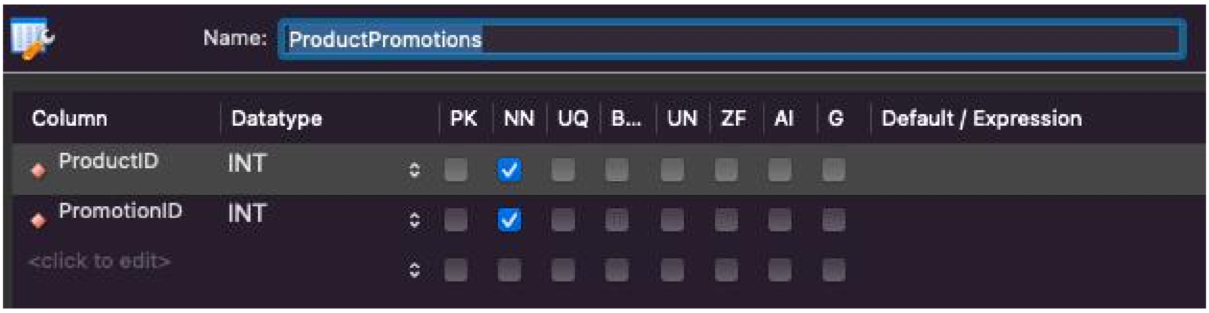


Name: FraudAndCosts

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
CostID	INT	☑	☑	□	□	□	□	☑	□	
IssueType	VARCHAR(100)	□	☑	□	□	□	□	□	□	
Description	TEXT	□	□	□	□	□	□	□	□	NULL
Cost	DECIMAL(10,2)	□	☑	□	□	□	□	□	□	
ReportedDate	DATE	□	☑	□	□	□	□	□	□	
ResolutionSt...	ENUM('Pendin(	□	□	□	□	□	□	□	□	'Pending'
ResolvedDate	DATE	□	□	□	□	□	□	□	□	NULL
OrderID	INT	□	☑	□	□	□	□	□	□	

Figure 14: FraudAndCosts Table

In Figure 14 the FraudAndCosts Table, has we one Foreign key OrderID is referenced with OrderID from Orders table.

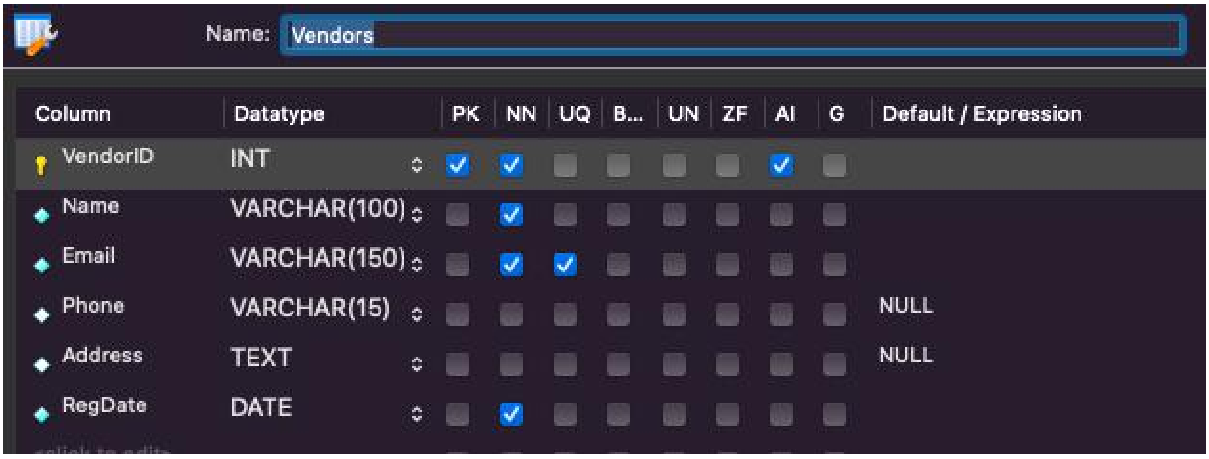


Name: ProductPromotions

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ProductID	INT	□	☑	□	□	□	□	□	□	
PromotionID	INT	□	☑	□	□	□	□	□	□	
<click to edit>		□	□	□	□	□	□	□	□	

Figure 15: ProductPromotions Table

Figure 15 ProductPromotions Table, is to manage the many to many relation between Product and Promotion Table.

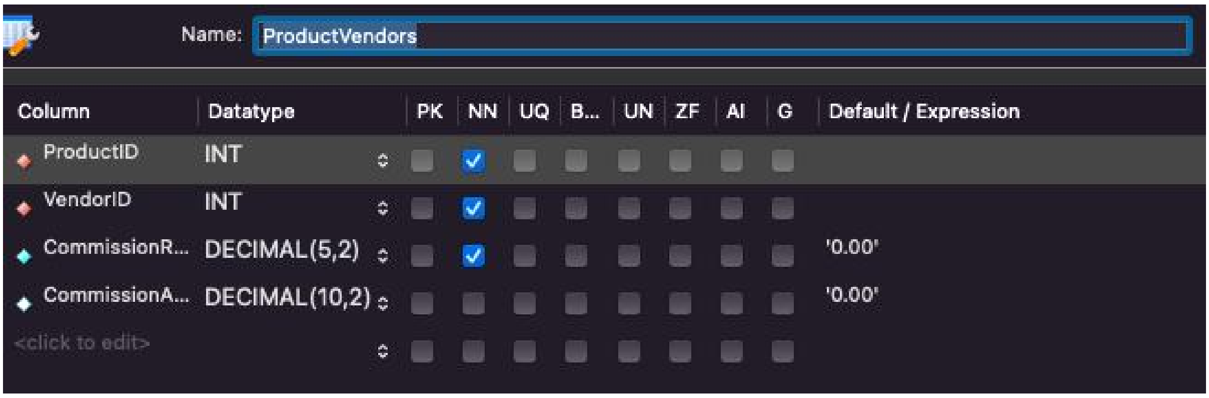


Name: Vendors

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
VendorID	INT	☑	☑	☐	☐	☐	☐	☑	☐	
Name	VARCHAR(100)	☐	☑	☐	☐	☐	☐	☐	☐	
Email	VARCHAR(150)	☐	☑	☑	☐	☐	☐	☐	☐	
Phone	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Address	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
RegDate	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
<click to edit>										

Figure 16: Vendors Table

In Figure 16 the Vendors Table, it has all required columns with VendorID is Primary Key.



Name: ProductVendors

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
ProductID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
VendorID	INT	☐	☑	☐	☐	☐	☐	☐	☐	
CommissionR...	DECIMAL(5,2)	☐	☑	☐	☐	☐	☐	☐	☐	'0.00'
CommissionA...	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	'0.00'
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 17: ProductVendors Table

Figure 17 ProductVendors Table, is to manage the many to many relation between Product and Vendors Table with attributes like commission rate and commission amount.

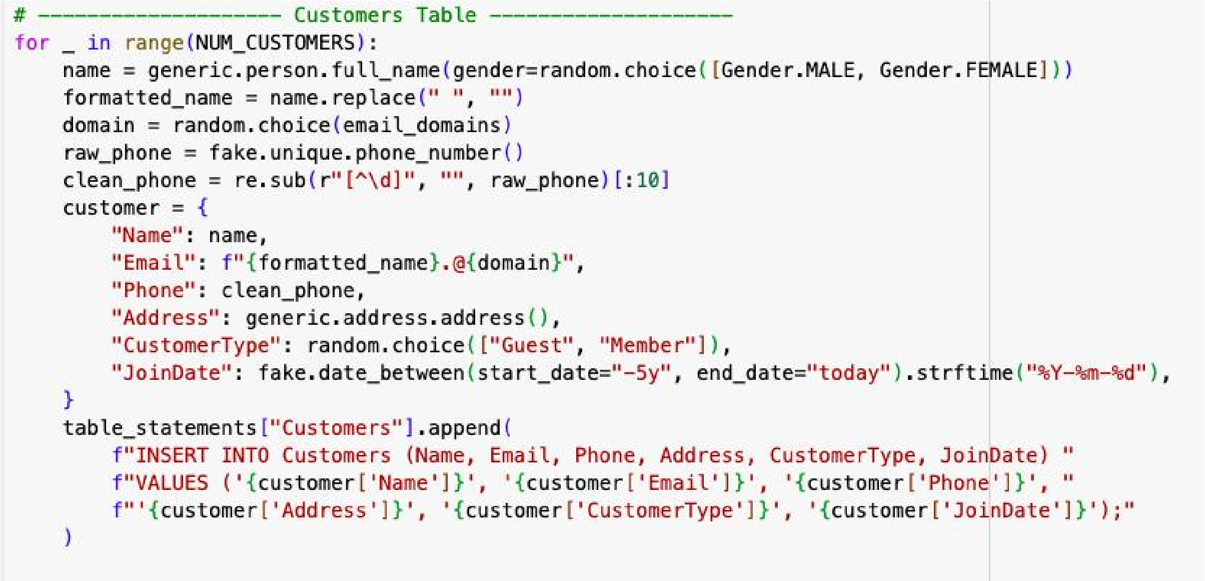
3.3 DATA GENERATION

For Data Generation, we have used python script with **faker()** and **mimesis()** library these are very popular Python library for generating fake data for testing or populating databases.Using this we generated dummy data for the online furniture marketplace.

Code link: Data generation Python code

Here, is the summary of data generated for our tables. The marketplace has 1200 Customers, with 1250 Products. Total Orders is 12000, and Orderdetails is 14000. The Site visits is 10000, Payments is 2500. The total vendors available are 100. Totally, there are 500 issue in

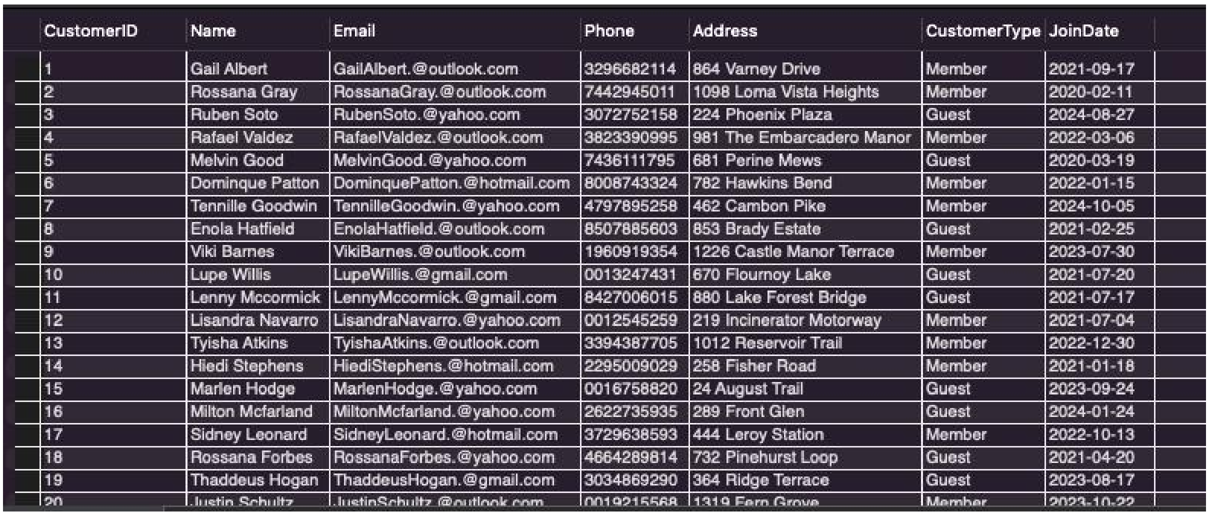
Fraud and cost related. We have 3000 shipments shipped including delivered, pending and cancelled. Almost around 100 Promotions Campaign has been launched. The member customer has 500 reward account, and 2000 reviews.



```python
# -------------------- Customers Table --------------------
for _ in range(NUM_CUSTOMERS):
    name = generic.person.full_name(gender=random.choice([Gender.MALE, Gender.FEMALE]))
    formatted_name = name.replace(" ", "")
    domain = random.choice(email_domains)
    raw_phone = fake.unique.phone_number()
    clean_phone = re.sub(r"[^\d]", "", raw_phone)[:10]
    customer = {
        "Name": name,
        "Email": f"{formatted_name}.@{domain}",
        "Phone": clean_phone,
        "Address": generic.address.address(),
        "CustomerType": random.choice(["Guest", "Member"]),
        "JoinDate": fake.date_between(start_date="-5y", end_date="today").strftime("%Y-%m-%d"),
    }
    table_statements["Customers"].append(
        f"INSERT INTO Customers (Name, Email, Phone, Address, CustomerType, JoinDate) "
        f"VALUES ('{customer['Name']}', '{customer['Email']}', '{customer['Phone']}', "
        f"'{customer['Address']}', '{customer['CustomerType']}', '{customer['JoinDate']}');"
    )
```

Figure 18: Data Generation code Snippet

Figure 18, it represents the Data generation code for Customers Table using the Python script. We have used faker() and mimesis() to generate the data.



CustomerID	Name	Email	Phone	Address	CustomerType	JoinDate
1	Gail Albert	GailAlbert.@outlook.com	3296682114	864 Varney Drive	Member	2021-09-17
2	Rossana Gray	RossanaGray.@outlook.com	7442945011	1098 Loma Vista Heights	Member	2020-02-11
3	Ruben Soto	RubenSoto.@yahoo.com	3072752158	224 Phoenix Plaza	Guest	2024-08-27
4	Rafael Valdez	RafaelValdez.@outlook.com	3823390995	981 The Embarcadero Manor	Member	2022-03-06
5	Melvin Good	MelvinGood.@yahoo.com	7436111795	681 Perine Mews	Guest	2020-03-19
6	Dominque Patton	DominquePatton.@hotmail.com	8008743324	782 Hawkins Bend	Member	2022-01-15
7	Tennille Goodwin	TennilleGoodwin.@yahoo.com	4797895258	462 Cambon Pike	Member	2024-10-05
8	Enola Hatfield	EnolaHatfield.@outlook.com	8507885603	853 Brady Estate	Guest	2021-02-25
9	Viki Barnes	VikiBarnes.@outlook.com	1960919354	1226 Castle Manor Terrace	Member	2023-07-30
10	Lupe Willis	LupeWillis.@gmail.com	0013247431	670 Flournoy Lake	Guest	2021-07-20
11	Lenny Mccormick	LennyMccormick.@gmail.com	8427006015	880 Lake Forest Bridge	Guest	2021-07-17
12	Lisandra Navarro	LisandraNavarro.@yahoo.com	0012545259	219 Incinerator Motorway	Member	2021-07-04
13	Tyisha Atkins	TyishaAtkins.@outlook.com	3394387705	1012 Reservoir Trail	Member	2022-12-30
14	Hiedi Stephens	HiediStephens.@hotmail.com	2295009029	258 Fisher Road	Member	2021-01-18
15	Marlen Hodge	MarlenHodge.@yahoo.com	0016758820	24 August Trail	Guest	2023-09-24
16	Milton Mcfarland	MiltonMcfarland.@yahoo.com	2622735935	289 Front Glen	Guest	2024-01-24
17	Sidney Leonard	SidneyLeonard.@hotmail.com	3729638593	444 Leroy Station	Member	2022-10-13
18	Rossana Forbes	RossanaForbes.@yahoo.com	4664289814	732 Pinehurst Loop	Guest	2021-04-20
19	Thaddeus Hogan	ThaddeusHogan.@gmail.com	3034869290	364 Ridge Terrace	Guest	2023-08-17
20	Justin Schultz	JustinSchultz.@outlook.com	0019215568	1319 Fern Grove	Member	2023-10-22

Figure 19: Customers Data

In Figure 19, the data generated for Customers is shown, with CustomerID, name, unique email ID, Phone number, Address, Their membership status and Joining Date.

3.4 Re-GENERATED ER-DIAGRAM

Using the MySQL Workbench the Entity relationship diagram is generated for our implemented database **onlinefurnituredb.**

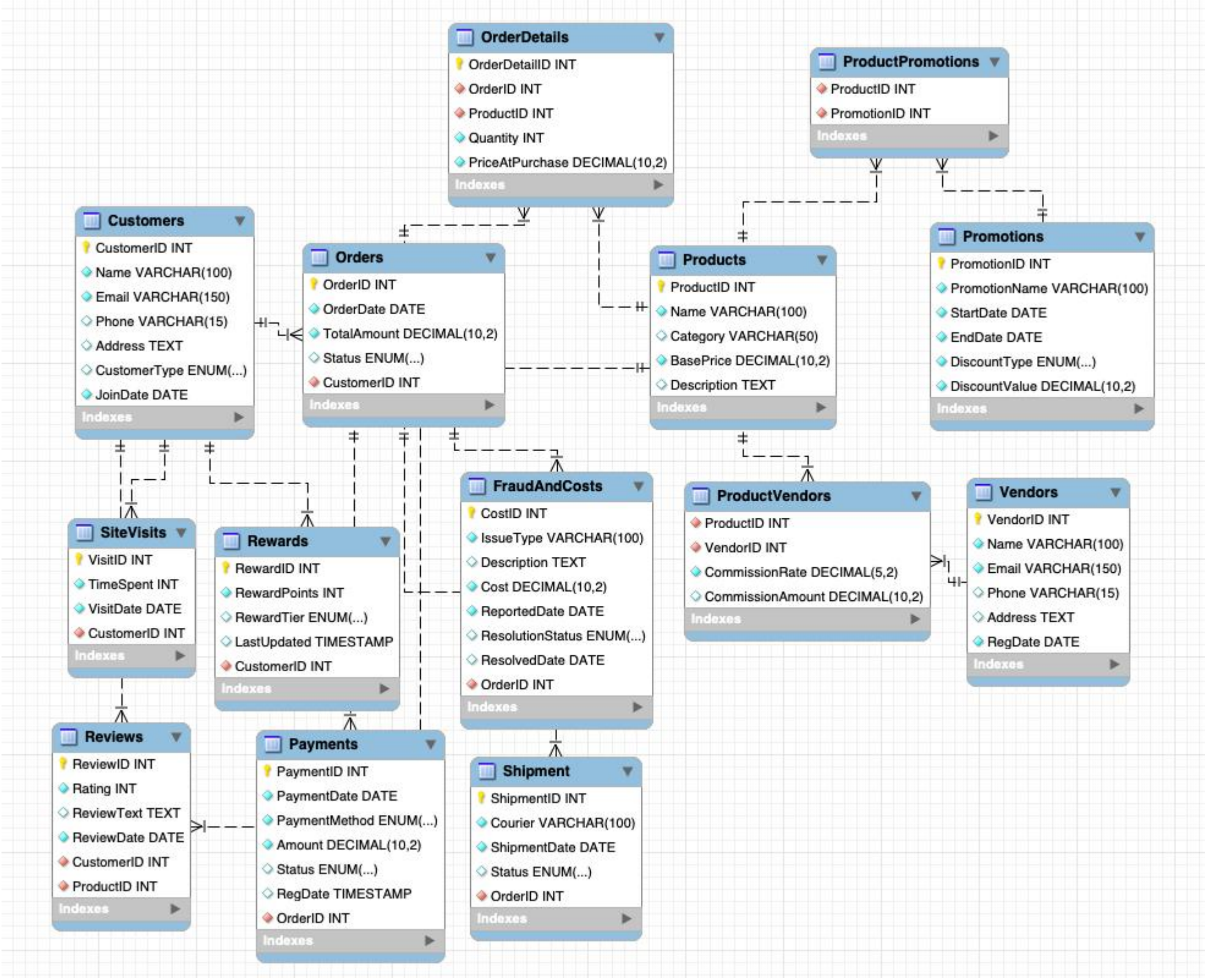


Figure 20: Regenerated ER Diagram

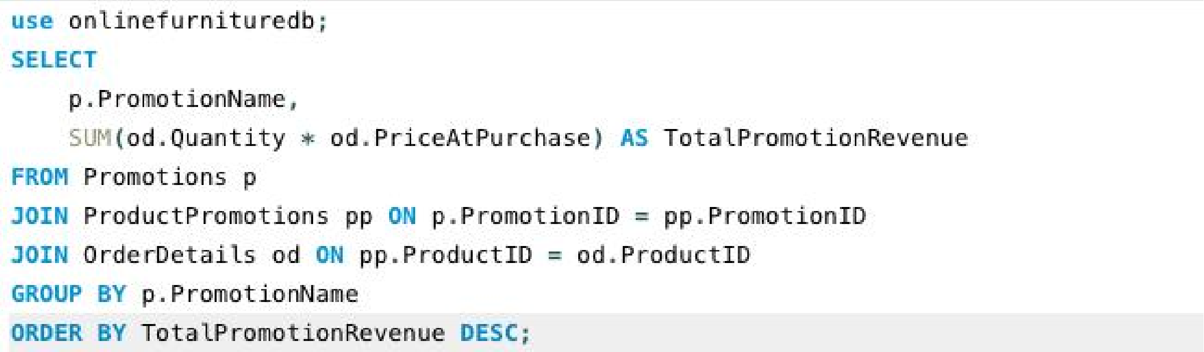
Figure 20, represents the ER diagram generated using MySQL workbench for our online furniture database. Customers, Orders, Products and Vendors, these are main entities while each of them have specific attributes and relationships. Customers can place orders, which can have multiple order details, and it is linked to products. Promotions and products table are combined to get discounts. Vendors supply products via the ProductVendors table. Shipments table used to track couriers and order delivery statuses. Payments table is used to store processed payments details for every order, and rewards are issued to customers based on their Shopping activity. Customer usage are tracked by SiteVisits table and Reviews table stores the reviews on products by customer. Also, we have FraudAndCosts table to monitor issues like fraud or additional costs.

CHAPTER 4 - SQL QUERIES

In this chapter we will be perform SQL queries on our database and discuss on the Objectives outlined in Chapter 1 using JOIN, Filtering, Aggregation, Window Function.

4.1 Evaluating Promotions Effectiveness

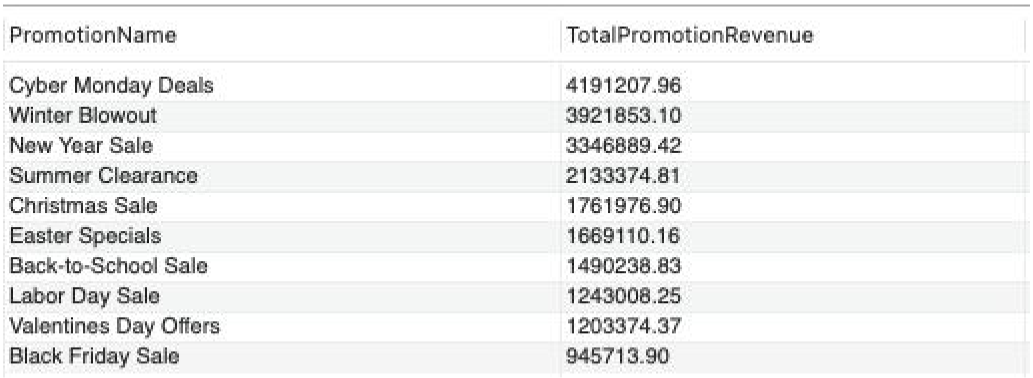
Analyzing the effectiveness of each promotion campaign is very essential for a marketplace. This analysis will help in identifying the most effective campaign and customers purchasing trends.



```
use onlinefurnituredb;
SELECT
    p.PromotionName,
    SUM(od.Quantity * od.PriceAtPurchase) AS TotalPromotionRevenue
FROM Promotions p
JOIN ProductPromotions pp ON p.PromotionID = pp.PromotionID
JOIN OrderDetails od ON pp.ProductID = od.ProductID
GROUP BY p.PromotionName
ORDER BY TotalPromotionRevenue DESC;
```

Figure 21: Query 1 - Calculate and Rank Total Revenue Generated by Promotions

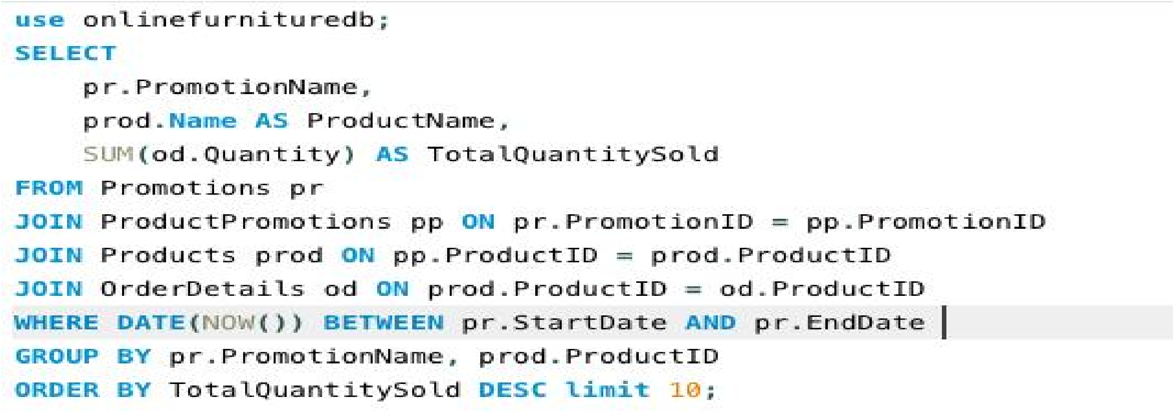
This above Query in Figure 21 used to calculate the total revenue generated by each promotion campaign in our marketplace and ranks them in descending order. By getting the the total products sold and their price at the time of purchase. Then we perform inner JOIN on 3 tables: Promotions, ProductPromotions and OrderDetails. And group the data by promotion name.



PromotionName	TotalPromotionRevenue
Cyber Monday Deals	4191207.96
Winter Blowout	3921853.10
New Year Sale	3346889.42
Summer Clearance	2133374.81
Christmas Sale	1761976.90
Easter Specials	1669110.16
Back-to-School Sale	1490238.83
Labor Day Sale	1243008.25
Valentines Day Offers	1203374.37
Black Friday Sale	945713.90

Figure 22: Query 1 - Output

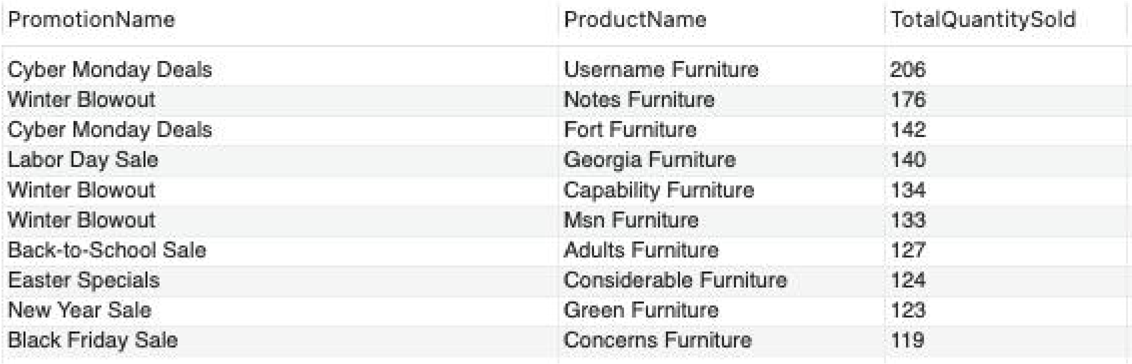
The output of query 1 is shown in Figure 22, it is observed that “Cyber Monday Deals” promotion has led more revenue to market place. And followed by “Winter Blowout” and “New Year sale”. This indicated most of the sale is on end of year. We can suggest to have more such offers in Winter to enhance further sales.



```
use onlinefurnituredb;
SELECT
    pr.PromotionName,
    prod.Name AS ProductName,
    SUM(od.Quantity) AS TotalQuantitySold
FROM Promotions pr
JOIN ProductPromotions pp ON pr.PromotionID = pp.PromotionID
JOIN Products prod ON pp.ProductID = prod.ProductID
JOIN OrderDetails od ON prod.ProductID = od.ProductID
WHERE DATE(NOW()) BETWEEN pr.StartDate AND pr.EndDate
GROUP BY pr.PromotionName, prod.ProductID
ORDER BY TotalQuantitySold DESC limit 10;
```

Figure 23: Query 2 - Calculate and Rank Total Quantity Sold for Products under Active Promotions

Figure 23, shows the query used to identify and rank the total quantity sold for each product under active promotions. This analysis would help in identifying the products which are in high demand.



PromotionName	ProductName	TotalQuantitySold
Cyber Monday Deals	Username Furniture	206
Winter Blowout	Notes Furniture	176
Cyber Monday Deals	Fort Furniture	142
Labor Day Sale	Georgia Furniture	140
Winter Blowout	Capability Furniture	134
Winter Blowout	Msn Furniture	133
Back-to-School Sale	Adults Furniture	127
Easter Specials	Considerable Furniture	124
New Year Sale	Green Furniture	123
Black Friday Sale	Concerns Furniture	119

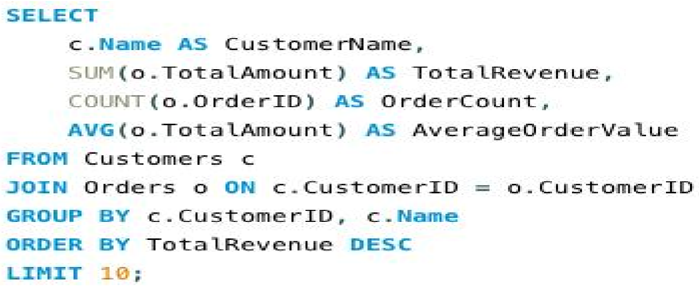
Figure 24: Query 2 - Output

In Figure 24, the Output shows the Products that are most in demand and sold in high numbers on a promotion campaign. By this data we can suggest to vendors to have more stocks in their inventory.

In this analysis, it is found that my identifying the trends in which promotion campaign the highest sales happen and the products that are demand. We can suggest to have more such offers in End of year and with the knowledge of products that are in demand, it will be helpful for vendors also for marketplace to think on giving more offers on least demand products to Customers to improve their sales and customer experience.

4.2 Identifying High-Value Customers

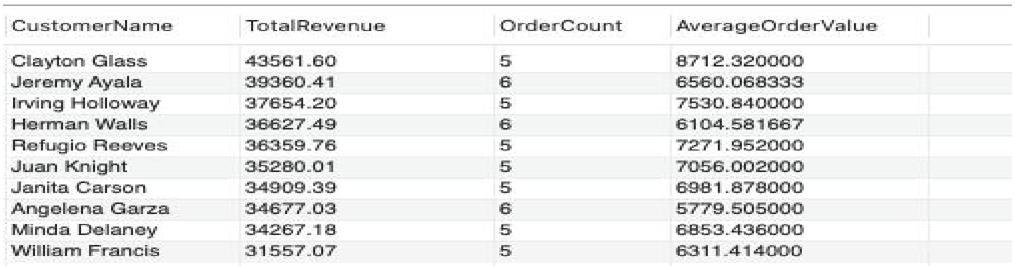
Identifying the high value customers and analyzing their purchase trends, frequency. These customers helps the marketplace with more increase in revenue. By identifying this data, stake holders can prioritize loyalty programs and personalized offers.



```sql
SELECT
    c.Name AS CustomerName,
    SUM(o.TotalAmount) AS TotalRevenue,
    COUNT(o.OrderID) AS OrderCount,
    AVG(o.TotalAmount) AS AverageOrderValue
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.Name
ORDER BY TotalRevenue DESC
LIMIT 10;
```

Figure 25:   Query 3 - Top 10 Customers by Total Revenue

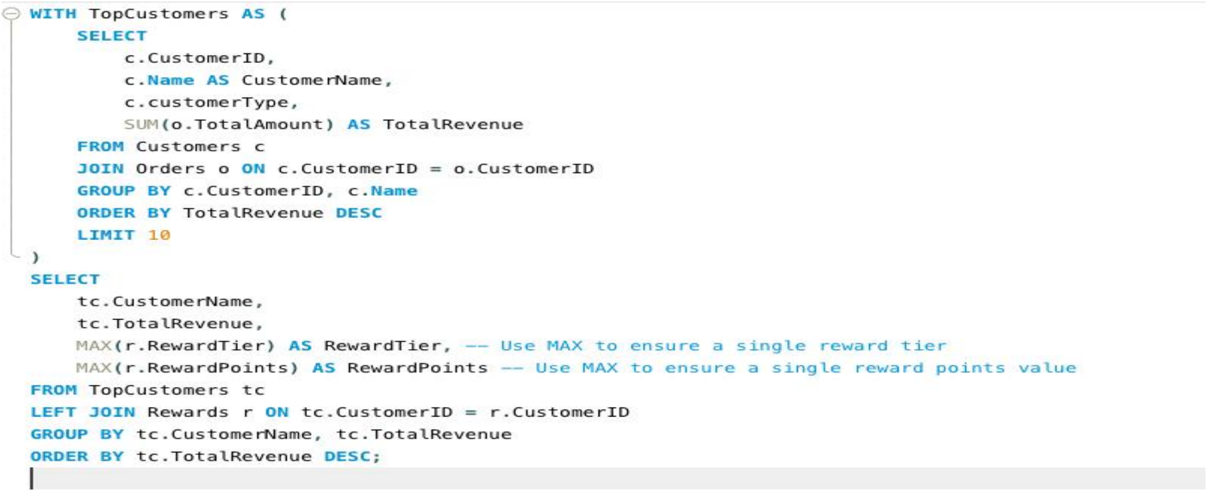
In Figure 25, the query used to identify the top 10 customers who have generated the highest total revenue in the marketplace. The query joins the Customers table with the Order table based on the 'CustomerID'.



CustomerName	TotalRevenue	OrderCount	AverageOrderValue
Clayton Glass	43561.60	5	8712.320000
Jeremy Ayala	39360.41	6	6560.068333
Irving Holloway	37654.20	5	7530.840000
Herman Walls	36627.49	6	6104.581667
Refugio Reeves	36359.76	5	7271.952000
Juan Knight	35280.01	5	7056.002000
Janita Carson	34909.39	5	6981.878000
Angelena Garza	34677.03	6	5779.505000
Minda Delaney	34267.18	5	6853.436000
William Francis	31557.07	5	6311.414000

Figure 26:   Query 3 - Output

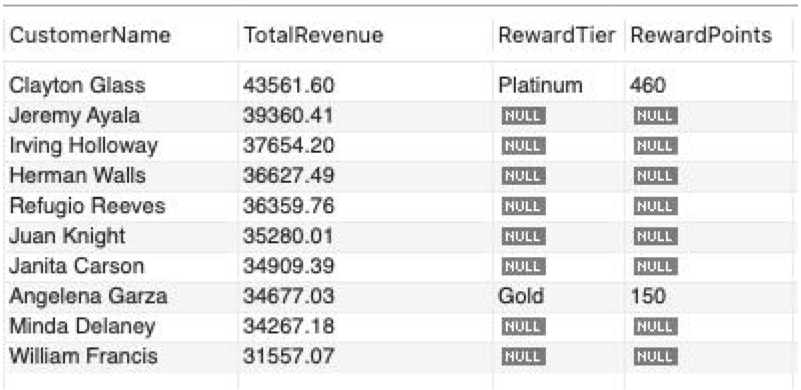
Figure 26, it shows the top 10 customers generating most revenue in our marketplace. This analysis helps us to understand the average order value and the total revenue generated by them.



```
WITH TopCustomers AS (
    SELECT
        c.CustomerID,
        c.Name AS CustomerName,
        c.customerType,
        SUM(o.TotalAmount) AS TotalRevenue
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    GROUP BY c.CustomerID, c.Name
    ORDER BY TotalRevenue DESC
    LIMIT 10
)
SELECT
    tc.CustomerName,
    tc.TotalRevenue,
    MAX(r.RewardTier) AS RewardTier, -- Use MAX to ensure a single reward tier
    MAX(r.RewardPoints) AS RewardPoints -- Use MAX to ensure a single reward points value
FROM TopCustomers tc
LEFT JOIN Rewards r ON tc.CustomerID = r.CustomerID
GROUP BY tc.CustomerName, tc.TotalRevenue
ORDER BY tc.TotalRevenue DESC;
```

Figure 27: Query 4 - Identify Loyalty Rewards Status for Top 10 Customers by Revenue

The Query in Figure 27, help us to get the loyalty membership status and their reward points accrued in marketplace.



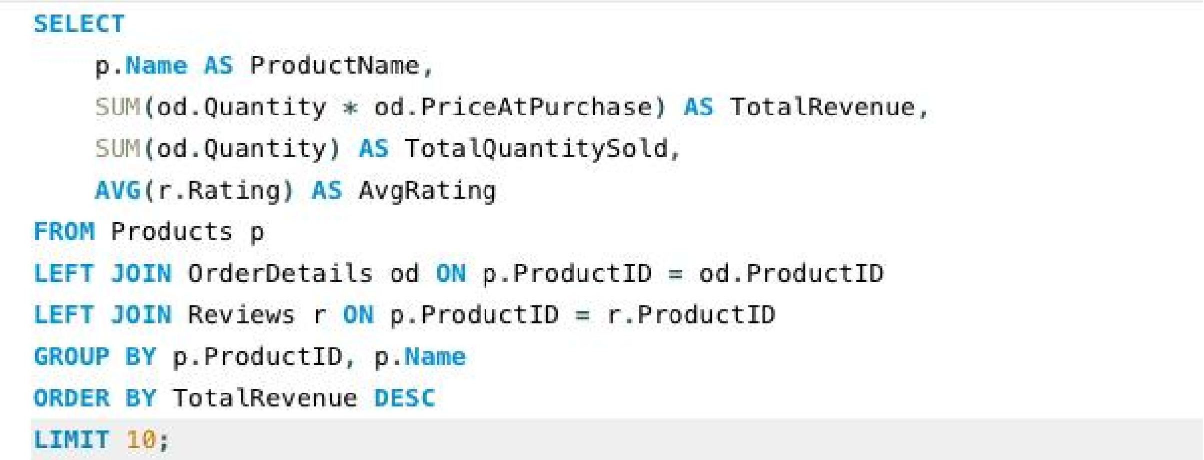
CustomerName	TotalRevenue	RewardTier	RewardPoints
Clayton Glass	43561.60	Platinum	460
Jeremy Ayala	39360.41	NULL	NULL
Irving Holloway	37654.20	NULL	NULL
Herman Walls	36627.49	NULL	NULL
Refugio Reeves	36359.76	NULL	NULL
Juan Knight	35280.01	NULL	NULL
Janita Carson	34909.39	NULL	NULL
Angelena Garza	34677.03	Gold	150
Minda Delaney	34267.18	NULL	NULL
William Francis	31557.07	NULL	NULL

Figure 28: Query 4 - Output

The Output in Figure 28, shows that the top 10 customers who have generated more revenue for our marketplace are not a loyalty member. This should be addressed, there might be issue in marketing the loyalty program or the customers are not interested in loyalty rewards. The stakeholders must analyze this and if the reward program is not beneficial for customers than removing it with an alternate solution is advisable.

4.3 Evaluate Product Performance

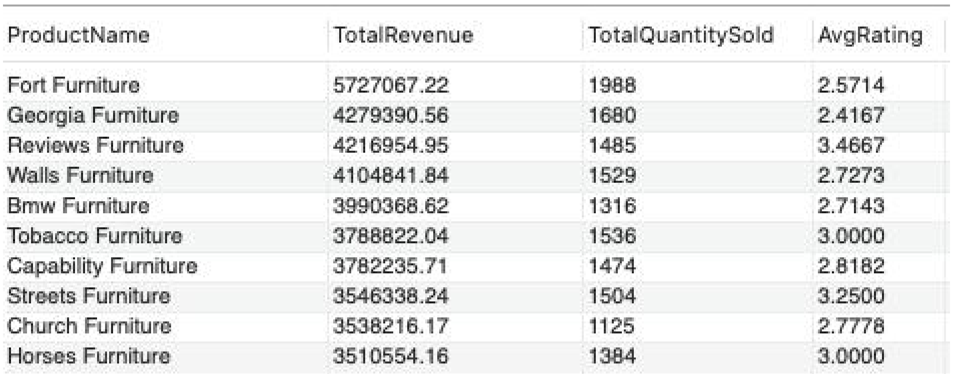
The product prizing plays a important role in over smarting the competitors, the customers want the best offers on the products they purchase. They tend to check on all other marketplaces and choose the best ones.



```
SELECT
    p.Name AS ProductName,
    SUM(od.Quantity * od.PriceAtPurchase) AS TotalRevenue,
    SUM(od.Quantity) AS TotalQuantitySold,
    AVG(r.Rating) AS AvgRating
FROM Products p
LEFT JOIN OrderDetails od ON p.ProductID = od.ProductID
LEFT JOIN Reviews r ON p.ProductID = r.ProductID
GROUP BY p.ProductID, p.Name
ORDER BY TotalRevenue DESC
LIMIT 10;
```

Figure 29: Query 5 - Top 10 Products by Revenue with Sales and Ratings Analysis

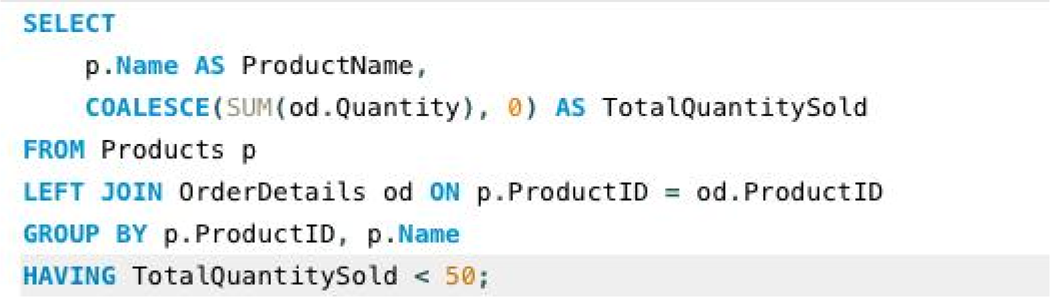
In Figure 29, the query is used to retrieve the top 10 products by total revenue and provides additional insights into their sales performance and customer satisfaction. A LEFT JOIN is used so that products without sales or reviews are still included in the results. Then grouped by product ID and name.



ProductName	TotalRevenue	TotalQuantitySold	AvgRating
Fort Furniture	5727067.22	1988	2.5714
Georgia Furniture	4279390.56	1680	2.4167
Reviews Furniture	4216954.95	1485	3.4667
Walls Furniture	4104841.84	1529	2.7273
Bmw Furniture	3990368.62	1316	2.7143
Tobacco Furniture	3788822.04	1536	3.0000
Capability Furniture	3782235.71	1474	2.8182
Streets Furniture	3546338.24	1504	3.2500
Church Furniture	3538216.17	1125	2.7778
Horses Furniture	3510554.16	1384	3.0000

Figure 30: Query 5 - Output

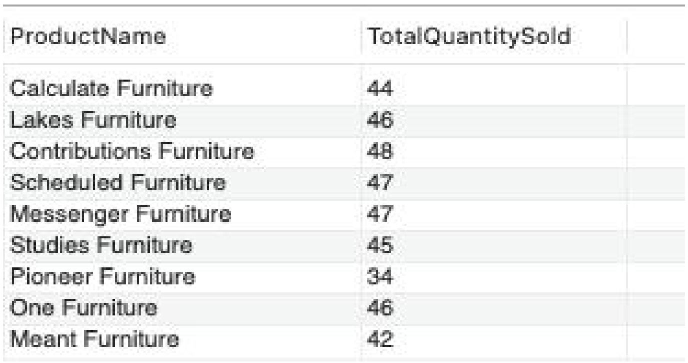
The output in figure 30, represents the analysis of the top-performing products that in generating more revenue and also the insights into their quantity sold and customer satisfaction. This enables in data driven decision and marketing, with optimized pricing.



```
SELECT
    p.Name AS ProductName,
    COALESCE(SUM(od.Quantity), 0) AS TotalQuantitySold
FROM Products p
LEFT JOIN OrderDetails od ON p.ProductID = od.ProductID
GROUP BY p.ProductID, p.Name
HAVING TotalQuantitySold < 50;
```

Figure 31: Query 6 - Least selling products

The query in figure 31, shows to retrieve the on least selling products only less that 50 quantities are purchased by customers. By combing OrderDetails and Products tables.

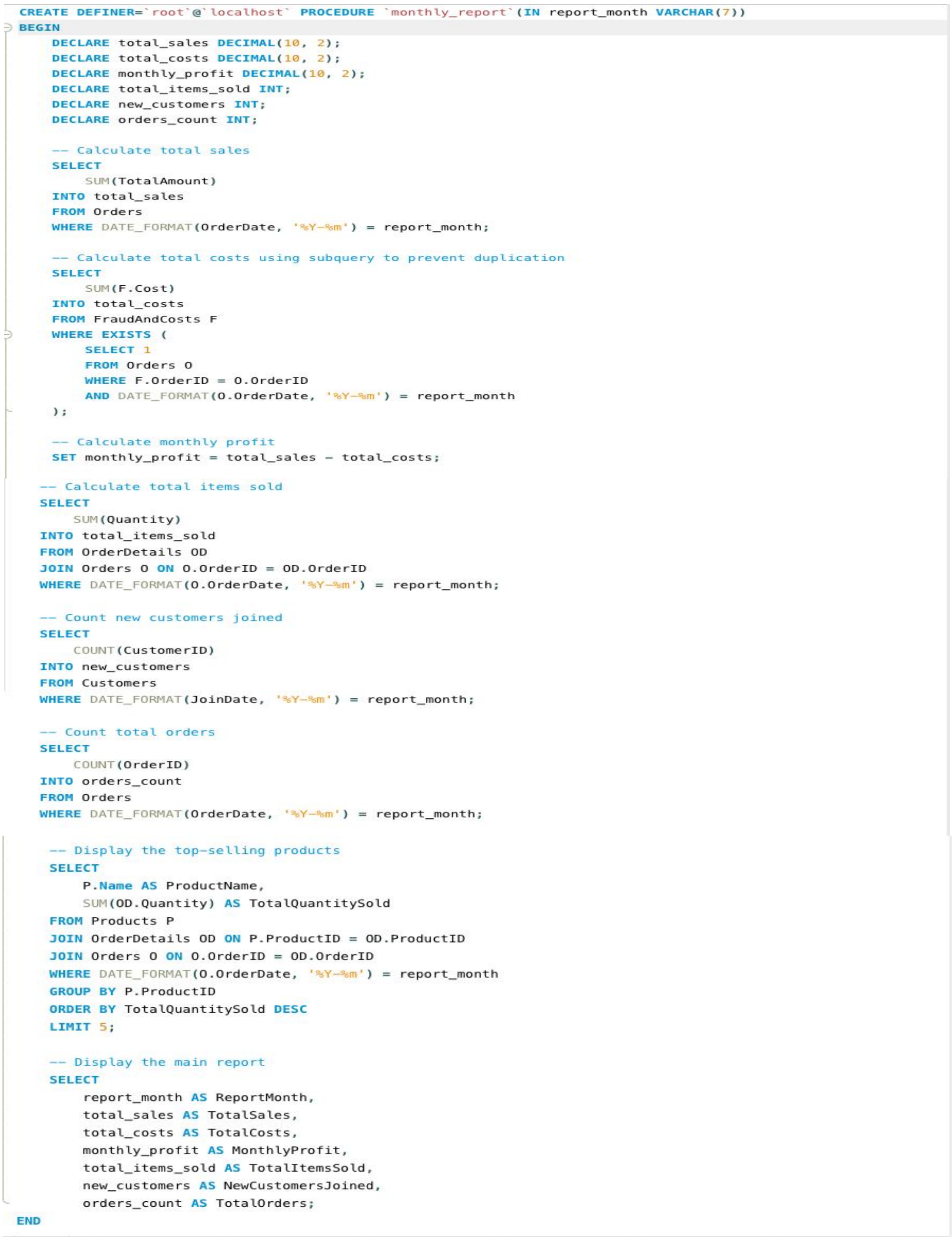


ProductName	TotalQuantitySold
Calculate Furniture	44
Lakes Furniture	46
Contributions Furniture	48
Scheduled Furniture	47
Messenger Furniture	47
Studies Furniture	45
Pioneer Furniture	34
One Furniture	46
Meant Furniture	42

Figure 32: Query 6 - Output

The output in figure 32, shows the least selling products in our marketplace. This proves that these items are less in demand and vendors can have less stocks on their inventory. This data helps in knowing the best- and worst-performing products helps optimize inventory for vendors, and helps in competitive pricing, and promotions.

4.4 Stored Procedure

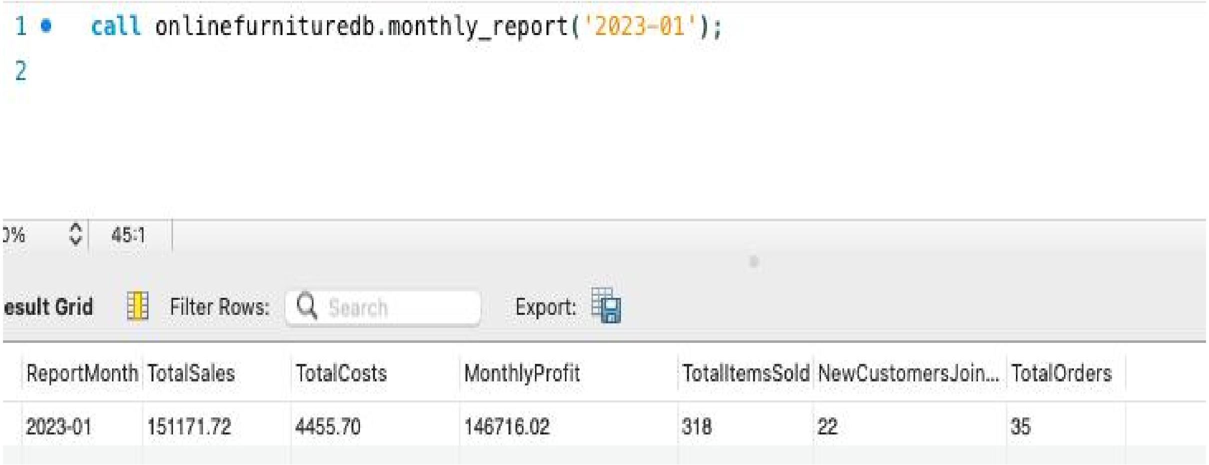


```
CREATE DEFINER=`root`@`localhost` PROCEDURE `monthly_report`(IN report_month VARCHAR(7))
BEGIN
    DECLARE total_sales DECIMAL(10, 2);
    DECLARE total_costs DECIMAL(10, 2);
    DECLARE monthly_profit DECIMAL(10, 2);
    DECLARE total_items_sold INT;
    DECLARE new_customers INT;
    DECLARE orders_count INT;

    -- Calculate total sales
    SELECT
        SUM(TotalAmount)
    INTO total_sales
    FROM Orders
    WHERE DATE_FORMAT(OrderDate, '%Y-%m') = report_month;

    -- Calculate total costs using subquery to prevent duplication
    SELECT
        SUM(F.Cost)
    INTO total_costs
    FROM FraudAndCosts F
    WHERE EXISTS (
        SELECT 1
        FROM Orders O
        WHERE F.OrderID = O.OrderID
        AND DATE_FORMAT(O.OrderDate, '%Y-%m') = report_month
    );

    -- Calculate monthly profit
    SET monthly_profit = total_sales - total_costs;

    -- Calculate total items sold
    SELECT
        SUM(Quantity)
    INTO total_items_sold
    FROM OrderDetails OD
    JOIN Orders O ON O.OrderID = OD.OrderID
    WHERE DATE_FORMAT(O.OrderDate, '%Y-%m') = report_month;

    -- Count new customers joined
    SELECT
        COUNT(CustomerID)
    INTO new_customers
    FROM Customers
    WHERE DATE_FORMAT(JoinDate, '%Y-%m') = report_month;

    -- Count total orders
    SELECT
        COUNT(OrderID)
    INTO orders_count
    FROM Orders
    WHERE DATE_FORMAT(OrderDate, '%Y-%m') = report_month;

    -- Display the top-selling products
    SELECT
        P.Name AS ProductName,
        SUM(OD.Quantity) AS TotalQuantitySold
    FROM Products P
    JOIN OrderDetails OD ON P.ProductID = OD.ProductID
    JOIN Orders O ON O.OrderID = OD.OrderID
    WHERE DATE_FORMAT(O.OrderDate, '%Y-%m') = report_month
    GROUP BY P.ProductID
    ORDER BY TotalQuantitySold DESC
    LIMIT 5;

    -- Display the main report
    SELECT
        report_month AS ReportMonth,
        total_sales AS TotalSales,
        total_costs AS TotalCosts,
        monthly_profit AS MonthlyProfit,
        total_items_sold AS TotalItemsSold,
        new_customers AS NewCustomersJoined,
        orders_count AS TotalOrders;
END
```

Figure 33: Stored Procedure - monthly report

Figure 33 shows the code of stored procedure “monthly_report” which expects a month and year value as a parameter. The output of this stored procedure includes Total sales revenue generated each month, expenses related to frauds, Nett Profit, Total items Sold, Count of new

Customers joined. This helps in getting a detailed data for every month based on the necessity of the user.



```
call onlinefurnituredb.monthly_report('2023-01');
```

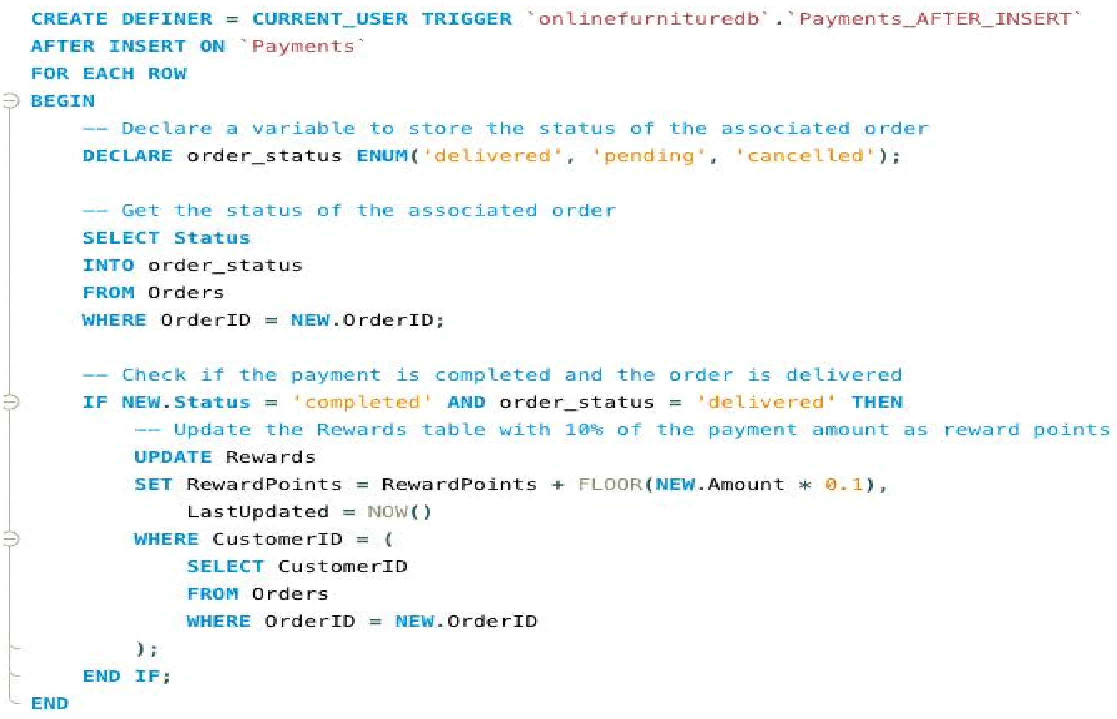
ReportMonth	TotalSales	TotalCosts	MonthlyProfit	TotalItemsSold	NewCustomersJoin...	TotalOrders
2023-01	151171.72	4455.70	146716.02	318	22	35

Figure 34: Stored Procedure - Output

The above output in figure 34, shows the output of the stored procedure. The input for calling the stored procedure monthly_report() is Year-month as a string. The output includes the Report Month, Total Sales, Total Costs, Monthly Profit, Total Items Sold, New Customers Joined, and Total orders placed on that month. These data can be used to understand the sales trends and Customer behavior. Helps in getting a detailed data by using this stored procedure. The monthly report provides a overview of the online furniture marketplace's performance in areas like sales, costs, customer acquisition, and product demand. This analysis helps stakeholders to make informed decisions, or identify areas for improvement, and strategize for future growth of the business. The report also gives and actionable insights for optimizing operations, to increase profit, and improve customer satisfaction.

4.5 Trigger to Update Rewards

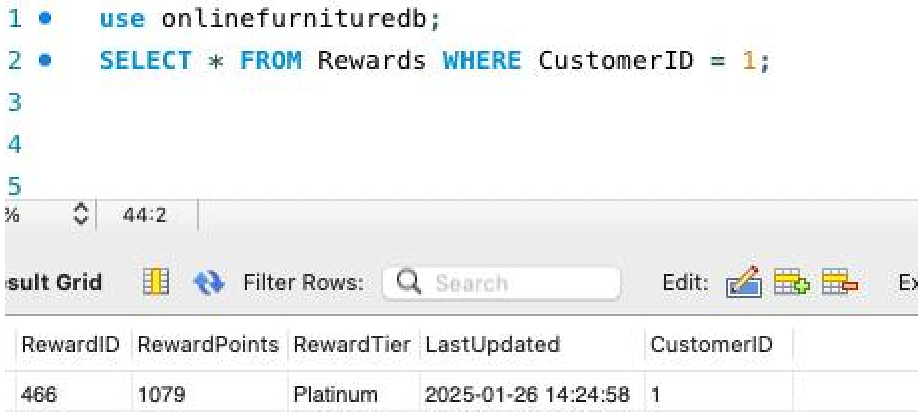
A Customer after successfully placing an order and payment. Once the order is delivered to customer without any issue, then the 10% of the Total amount paid by customer is given back as a reward points.



```
CREATE DEFINER = CURRENT_USER TRIGGER `onlinefurnituredb`.`Payments_AFTER_INSERT`
AFTER INSERT ON `Payments`
FOR EACH ROW
BEGIN
    -- Declare a variable to store the status of the associated order
    DECLARE order_status ENUM('delivered', 'pending', 'cancelled');

    -- Get the status of the associated order
    SELECT Status
    INTO order_status
    FROM Orders
    WHERE OrderID = NEW.OrderID;

    -- Check if the payment is completed and the order is delivered
    IF NEW.Status = 'completed' AND order_status = 'delivered' THEN
        -- Update the Rewards table with 10% of the payment amount as reward points
        UPDATE Rewards
        SET RewardPoints = RewardPoints + FLOOR(NEW.Amount * 0.1),
            LastUpdated = NOW()
        WHERE CustomerID = (
            SELECT CustomerID
            FROM Orders
            WHERE OrderID = NEW.OrderID
        );
    END IF;
END
```

Figure 35: Trigger - Update Reward Points Based on Completed Payments and Delivered Orders

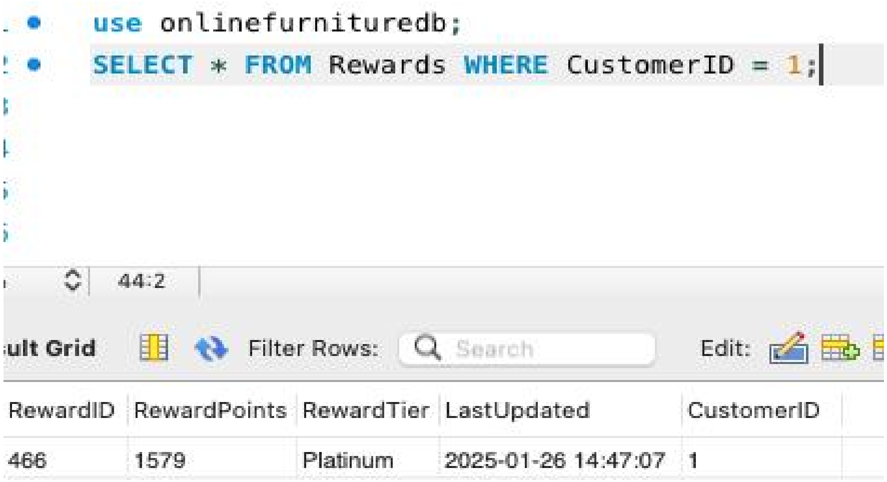
In Figure 35, the code for trigger **Payments_AFTER_INSERT** is shown. It is designed to automatically update and add reward points to customers. The trigger is activated after a new payment record is inserted into the Payments table. The trigger first checks if the payment status is completed and if that specific Order's order status should be in delivered. If both conditions are met, then it calculates reward points as 10% of the payment amount and the points are then added to the customer's existing reward points in the Rewards table, and the LastUpdated timestamp is updated. This process automates and ensures a seamless and accurate allocation of rewards.



```
use onlinefurnituredb;
SELECT * FROM Rewards WHERE CustomerID = 1;
```

RewardID	RewardPoints	RewardTier	LastUpdated	CustomerID
466	1079	Platinum	2025-01-26 14:24:58	1

Figure 36: Before Trigger Execution



```
use onlinefurnituredb;
SELECT * FROM Rewards WHERE CustomerID = 1;
```

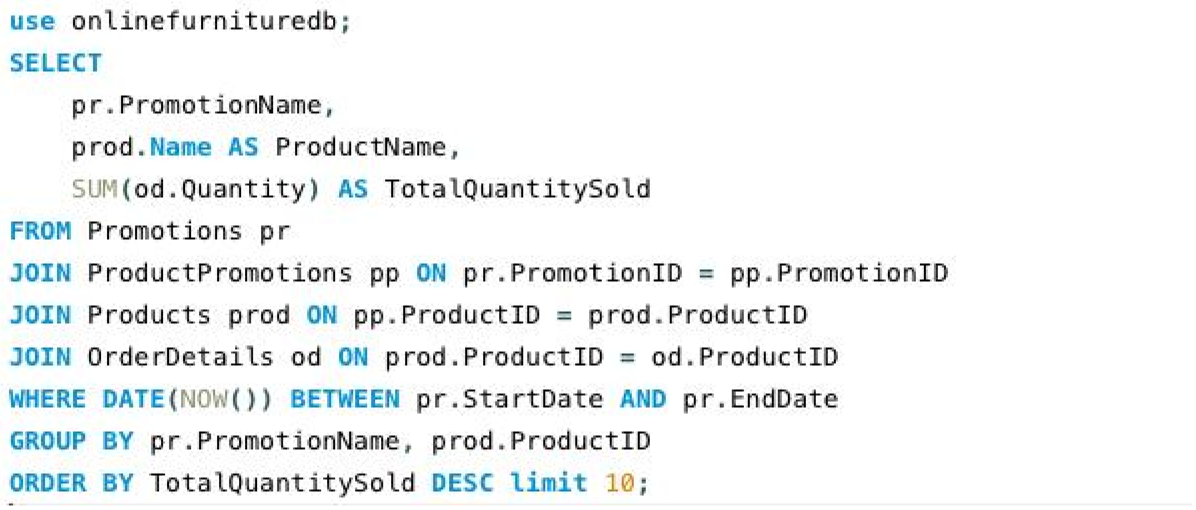
RewardID	RewardPoints	RewardTier	LastUpdated	CustomerID
466	1579	Platinum	2025-01-26 14:47:07	1

Figure 37: After Trigger Execution

Figure 36 and 37 both shows the reward table data before and after execution of the trigger **Payments_AFTER_INSERT**. It is Observed that before the Reward Points was 1079 and after that the customer placed a order worth of 5000 and then payment was also successful so once the order was delivered the Rewards table was updated with 10% of the order value. That 500 was added in Reward account of that customer. Now, the updated balance in Reward account is 1579 and the LastUpdated column is also updated with the latest time.

CHAPTER 5 - DATABASE MODEL EVALUATION

In this Chapter, we evaluate our database designed for online furniture marketplace. And also its support for analytical queries, data integrity and a discussion on how this design complies with the ACID properties and CAP theorem. Finally, we present a real-world case study of an analytic data warehouse used in eBay an e-commerce company.



```sql
use onlinefurnituredb;
SELECT
    pr.PromotionName,
    prod.Name AS ProductName,
    SUM(od.Quantity) AS TotalQuantitySold
FROM Promotions pr
JOIN ProductPromotions pp ON pr.PromotionID = pp.PromotionID
JOIN Products prod ON pp.ProductID = prod.ProductID
JOIN OrderDetails od ON prod.ProductID = od.ProductID
WHERE DATE(NOW()) BETWEEN pr.StartDate AND pr.EndDate
GROUP BY pr.PromotionName, prod.ProductID
ORDER BY TotalQuantitySold DESC limit 10;
```

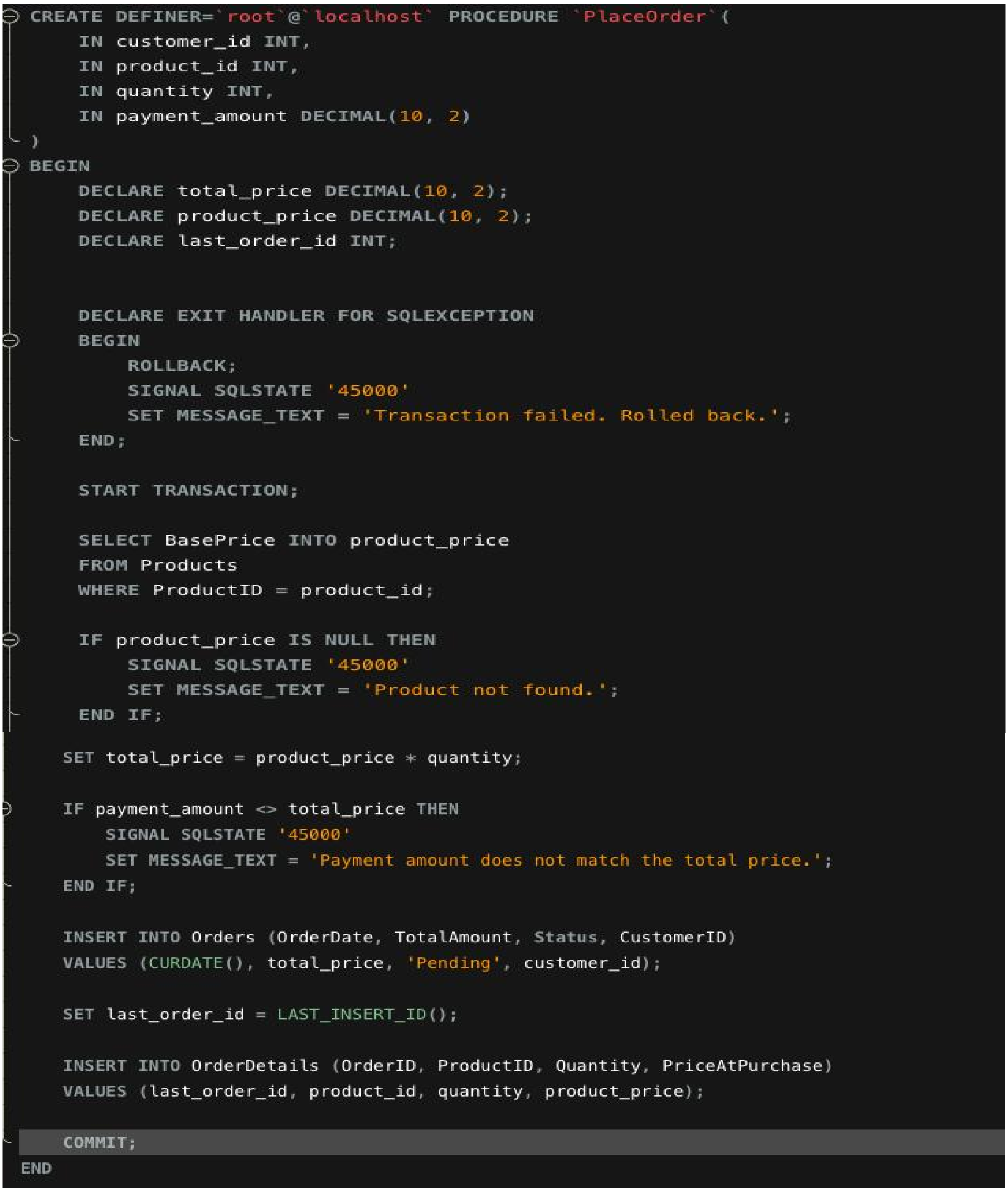
Figure 38: Advanced analytical SQL query

In Figure 38, this query is used to calculate the top 10 products which are sold and their total quantity sold on a specific promotion campaign. Here, we are joining four tables Promotions, ProductPromotions Products, and OrderDetails to get all the necessary data. Then it is filtered out to include only active promotions based on the current date and then sorted by total quantity sold in descending order. The performance of this query will degrade if the data is more because it involves multiple joins.

Currently, the execution time is 0.054 sec for existing size which has around 2000 Orders and 4000 Order Details. When the Data is scaled to 10x it is expected to take more execution time. To further Optimize we can perform indexing the key unique columns in tables like ProductID, PromotionID, OrderID and perform partitioning the large tables like OrderDetails, could help optimize the query.

5.1 ACID Properties

ACID states Atomicity, Consistency, Isolation and Durability. We apply ACID properties on a database where data integrity and its consistency matters a lot. It is a set of properties that ensures the reliability of database transactions these properties were initially developed with traditional, business-oriented applications (e.g., banking) in mind (Gray and Reuter, 2010).



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `PlaceOrder`(
    IN customer_id INT,
    IN product_id INT,
    IN quantity INT,
    IN payment_amount DECIMAL(10, 2)
)
BEGIN
    DECLARE total_price DECIMAL(10, 2);
    DECLARE product_price DECIMAL(10, 2);
    DECLARE last_order_id INT;


    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Transaction failed. Rolled back.';
    END;

    START TRANSACTION;

    SELECT BasePrice INTO product_price
    FROM Products
    WHERE ProductID = product_id;

    IF product_price IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Product not found.';
    END IF;
    SET total_price = product_price * quantity;

    IF payment_amount <> total_price THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Payment amount does not match the total price.';
    END IF;

    INSERT INTO Orders (OrderDate, TotalAmount, Status, CustomerID)
    VALUES (CURDATE(), total_price, 'Pending', customer_id);

    SET last_order_id = LAST_INSERT_ID();

    INSERT INTO OrderDetails (OrderID, ProductID, Quantity, PriceAtPurchase)
    VALUES (last_order_id, product_id, quantity, product_price);

    COMMIT;
END
```

Figure 39: Stored Procedure - Place Order

Figure 39, it shows the stored procedure used for placing an order in our online furniture marketplace. This stored procedure complies with ACID properties ensuring our database is reliable.

Atomicity: The PlaceOrder procedure complies atomicity by considering the entire order process as a single atom or transaction. The procedure begins with a 'START TRANSACTION' statement, and we have handlers to catch SQL exceptions. If an error occurs, like missing product or a mismatch in payment amount, the transaction is rolled back using the 'ROLLBACK' command. For example, if the product is invalid or incorrect payment, the transaction fails and aborts all changes, leaving the database in its original state.

Consistency: Consistency is very important in any database, this helps in efficient data retrieval. Here, in our PlaceOrder procedure, first it checks whether the product exists and retrieves its price. If the product does not exist, there will be error. This ensures that only valid data can be in transaction. Also, it calculates the total price by 'product_price * quantity' and then checks whether the payment amount matches. If not, the transaction is aborted, this prevents inconsistent data to be inserted into the database.

Isolation: Using MySql database ensures isolation, it ensures that concurrent transactions do not interfere with each other. In PlaceOrder procedure it retrieves the product price from Product table and then inserts order details, this entire operation is within scope of transaction. So, this ensures that no parallel transactions affect the current process.

Durability: Here, durability is ensured by committing the transaction at the end of the procedure using the 'COMMIT'. Once the commit is executed all changes including the new order and order details, are stored in the database. Even when there is a OS crash or power failure the committed data remains safe and secure.

5.2 CAP Theorem

According to Brewer in CAP theorem it states that any database system can have achieve any two from these three properties **Consistency**, **Availability** and **Partition Tolerant (Scalable)** (Brewer E, 2012).

In context, of our online furniture marketplace, we mainly focus on Consistency, Availability and keeping Scalability as optional for now. Applying the CAP theorem, on our database Here, Consistency means that data updates, such as updating rewards table when there is a successful order placed and payments processed instantly and accurately across all nodes. But when there are more customers this could result in errors or delays during network partitions. Availability, ensures that customers can browse products and place orders even during partitions.

Consistency is ensured in our online furniture marketplace, when a order is placed by customer, first the requested product is checked for availability and then checks for payment amount. If all conditions are met then the Rewards table is updated with reward points. With consistency guaranteed, this updated reward balance is immediately reflected in all subsequent queries. This ensures that loyalty reward account is being kept consistent. However, inconsistency might occur due to external system failure like if there is a delay payments vendor API because of this the loyalty points might not get updated in time.

Availability: In online furniture marketplace, the availability is very important, especially during seasonal sales or promotional events. For a customer to view the products and place order the availability of the system is a must. Using MySQL as a database, it ensures that if a server experiences a high load or failure, other replica servers can handle the queries, which ensured the availability to customers. But, if a primary or main server crashes and changeover to a replica server takes more, there might be an temporary unavailability.

Partition tolerance: This online furniture marketplace is designed to with hold some scalability upto 10x load but this was more focusing on Availability and Consistency design.

By focusing on availability and consistency, the MySQL-based online furniture marketplace supports real time analytics and maintain high available time. Transactions ensures data integrity, this design meets the needs of the furniture marketplace in its current phase. But when the the database size increases we might need to consider include scalability to accommodate more data. For now, our system ensures that this system has accurate and up-to-date data for decision-making, aligning with the core objectives of the business.

5.3 Case Study: eBay's Analytic Data Warehouse Transformation

eBay is one of the largest global e-commerce platform connecting the buyers and sellers, providing variety of products and services (eBay, n.d.).

What technology is used in the data warehouse?

At the initial stages, eBay predominately relied on a vendor-based data warehouse system powered by **Teradata**. This was a very structured traditional database. This system became inefficient when the data started increasing. Then they started transitioning towards an open-source, in-house analytics platform using big data technologies such as Hadoop, Apache Hive, and Presto.

Why was that technology chosen, and alternatives were considered?

On transition using the open-source technologies there was some key considerations. For them Scalability was first priority, as data growth was exponential. Using Hadoop and other distributed frameworks they started to expand storage and computational capacity linearly. The second priority was flexibility. Using open-source systems allowed them for customization. And Cost efficiency, when compared to open source while using Teradata it had a significant operational expenses.

What is one use case that is particularly important to the company?

The most critical use case for eBay's analytic data warehouse was customer behavior analysis. The platform has to process massive amounts like user interaction data,search queries, purchase history. Analyzing this data, helps eBay to deliver personalized recommendations, improves search results and the end goal was to improve user experience.

What access management considerations must be taken into account in managing this company's databases?

Access management is a important aspect of eBay's data warehouse operations. Role-based access control (RBAC) ensures that all internal stakeholders, have access only to the data necessary for their specific roles. They also implemented data privacy measures to comply with global regulations such as GDPR, to safeguard sensitive customer information.

This transformation supports data-driven decision-making across the organization and ensures a seamless experience for both users and internal stakeholders.

REFERENCES

Brewer, E.A., 2000. Towards robust distributed systems.

Gray, J. and Reuter, A., 2010. Transaction processing: Concepts and techniques. Purdue University.

eBay, n.d., From vendor to in-house: How eBay re imagined its analytics landscape, eBay Innovation Blog. Available at: https://innovation.ebayinc.com/tech/engineering/from-vendor-to-in-house-how-ebay-reimagined-its-analytics-landscape