Development Team Project: Project Report
Retail Store – Database Design

Group No: 2

Names: Kieron Hamilton, Naveed Jamal, Valentina Mercieca

Module: Deciphering Big Data

Date: 2nd December 2024

Table of Contents

List of Figures

Overview

For our project in Deciphering Big Data, our team chose a retail store as the client organisation. This company seeks to migrate its local data to the cloud and implement a loyalty program to better understand customer behaviour and improve customer retention.

Retail data is inherently complex, encompassing customer demographics, store locations, sales records, supplier information, and more. Dhanushkodi et al. (2024) highlight that analysing customer behaviour plays a vital role in shaping effective business strategies and driving profitability in the ever-changing retail industry.

To address these needs, we propose designing a cloud-based database using Snowflake to handle large-scale data storage and analytics, while MySQL Workbench serves as the relational database management system for efficient backend processing. Snowflake is chosen for its flexibility, scalability, and ability to facilitate efficient data storage, processing, and analysis (Altexsoft, 2024).

A key feature of this design is the loyalty program, which incentivises customers by offering vouchers and discounts tailored to their spending patterns. Lacey and Sneath (2006) note that such programs are effective in encouraging customers to share their personal information, driven by the promise of exclusive benefits.

Logical Design

Our database employs a structured and normalised relational model, ensuring consistency and minimising redundancy by adhering to third normal form (3NF). The design includes several interconnected tables that capture the complexities of the retail business. The database, named Retail_Store_Database, consists of the following key tables: Customer, Order, Store, OrderItems, Employee, Product, and Supplier (Figure 1).

- Customer Table:
  The Customer table stores customer data and links to the Order table via CustomerID (primary key in Customer, foreign key in Order) in a one-to-many relationship.

- Order Table:
  The Order table records individual customer purchases and links to the Store table via a many-to-one relationship (one store processes many orders) and to the OrderItems table via a one-to-many relationship (one order contains multiple order items). It also tracks the total amount spent per order.

- OrderItems Table:
  The OrderItems table tracks the individual products within each order, connecting to the Order table through a many-to-one relationship (one order can have multiple items) and to the Product table through a many-to-many relationship (multiple products can appear in multiple orders). This allows the store to identify which products were sold and in what quantities.

- Product Table:
  The Product table lists details of all products available for sale, such as name, category, price, and stock quantity. It links to the Supplier table through a many-to-one relationship, using SupplierID as the foreign key.

- Store Table:
  The Store table tracks physical store locations and connects to the Employee table via a one-to-many relationship, with each store employing multiple staff members.

- Employee Table:
  The Employee table stores information about store staff, including EmployeeID as the primary key and StoreID as a foreign key to link each employee to a specific store.

- Supplier Table:
  The Supplier table identifies vendors supplying products, enabling efficient inventory management and traceability.

Careful selection of data types ensures efficiency and functionality. For instance, VARCHAR is used for variable-length text fields like Name and Email, minimising space wastage, while DECIMAL is utilised for monetary fields like TotalAmount to maintain precision. Further details and the SQL code for creating the tables are included in the appendix (Figures 2 and 3).

Proposal of Database Build and Data Pipeline

The physical deployment of the database involves implementing the logical design into an actual relational database management system (RDBMS). MySQL Workbench was selected for this purpose due to its versatility, scalability, and cost-effectiveness (Heck, 2009). As an open-source tool, it allows customisation to meet the specific needs of the retail store while facilitating integration with other systems. MySQL is also well-suited to scaling, ensuring that the database can grow alongside the business.

Complementing this on-premises database is Snowflake, a cloud-based data warehouse that offers elastic scalability and advanced analytics capabilities. This dual-system approach ensures that transactional data and analytical processes are efficiently managed.

A key component is the data pipeline, which ensures the seamless flow of information between MySQL and Snowflake. The loyalty program relies on customer details as its primary data source. Each participant submits information, such as full name, email, phone number, address, and date of birth. This is submitted either via a front-end system by an employee within a store at the point of sale, or by customers who can sign up online through the retailer's website or app. The collected data is relayed via an API to a MySQL database and stored as a customer record under the loyalty program, with each customer assigned a unique ID number. Then, an Extract, Transform, Load (ETL) process transfers this data to Snowflake for further cleaning, transformation, and indexing, ensuring high-quality, actionable data.

To prevent duplicate accounts or customers creating multiple accounts, key data points are cross-referenced with existing accounts. If a duplicate is identified, the system blocks the creation of the new account. Additionally, any anomalies are addressed, such as setting a maximum age limit for account holders or excluding outlier spending in data analysis. As customers make purchases, rewards are generated based on milestones, special occasions, or demographics, and notifications are sent via email.

Centralising the data in this way offers significant advantages. As Sargo (N.D.) notes, centralised databases enhance security through easier access monitoring, reduce time spent on data preparation, and improve analytics accuracy by minimising errors, incomplete records, and duplicates.

System Rules and Security

Designing a database for a retail store necessitates addressing key constraints. Scalability is essential as the customer base and product inventory grow. This is addressed by implementing horizontal scaling through sharding and vertical scaling via hardware upgrades (Slingerland, 2024). Snowflake further supports scalability through dynamic resource allocation.

Security is another priority, particularly for sensitive customer data. Measures such as encryption, role-based access control (RBAC), and secure APIs ensure compliance with GDPR guidelines. For example, employees access only job-relevant data, while customers use password-protected portals to view their own data. Accounts with prolonged inactivity are flagged for deletion in accordance with GDPR rules (European Commission, 2016).

Downtime, whether due to system updates or unexpected failures, poses a risk to operations. This is mitigated by replication, maintaining secondary databases that take over during primary database outages to ensure high availability. Data consistency between MySQL and Snowflake is managed through ETL pipelines that validate and synchronise data during transfers.

A robust backup and recovery strategy ensures data integrity and business continuity. Full backups are performed daily, with incremental backups capturing changes since the last full backup. These backups are securely stored in cloud platforms, providing off-site redundancy and protection against physical system damage.

References

Altexsoft. (2024) The Good and the Bad of Snowflake Data Warehouse. Available from: https://www.altexsoft.com/blog/snowflake-data-warehouse-pros-cons/ [Accessed 24 November 2024].

Dhanushkodi, K., Bala, A., Kodipyaka, N. & Shreyas, V. (2024) Customer Behaviour Analysis and Predictive Modelling in Supermarket Retail: A Comprehensive Data Mining Approach. *IEEE Access*. DOI: 10.1109/ACCESS.2024.3407151

European Commission (2016) GDPR Regulation of the European Parliament and Council. Available at: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679&qid=171 3388284638 [Accessed 24 November 2024].

Heck, W. (2009) Using MySQL in Your Organisation. Available from: https://library.iated.org/view/HECK2009USI [Accessed 24 November 2024].

Lacey, R. & Sneath, J.Z. (2006) Customer Loyalty Programs: Are they Fair to Consumers? *The Journal of Consumer Marketing* 23(7): 458-464. DOI: 10.1108/07363760610713000

Sargo, M. (N.D.) Top 5 Benefits to Centralized Data. Available from: https://www.dataideology.com/top-5-benefits-to-centralized-data/ [Accessed 24 November 2024].

Slingerland, C. (2024) Horizontal vs Vertical Scaling: Which Should You Choose? Available from: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/#:~:text=While%20horizontal%20scaling%20refers%20to,%2C%20storage%2C%20or%20network%20speed [Accessed 25 November 2024].

Appendix

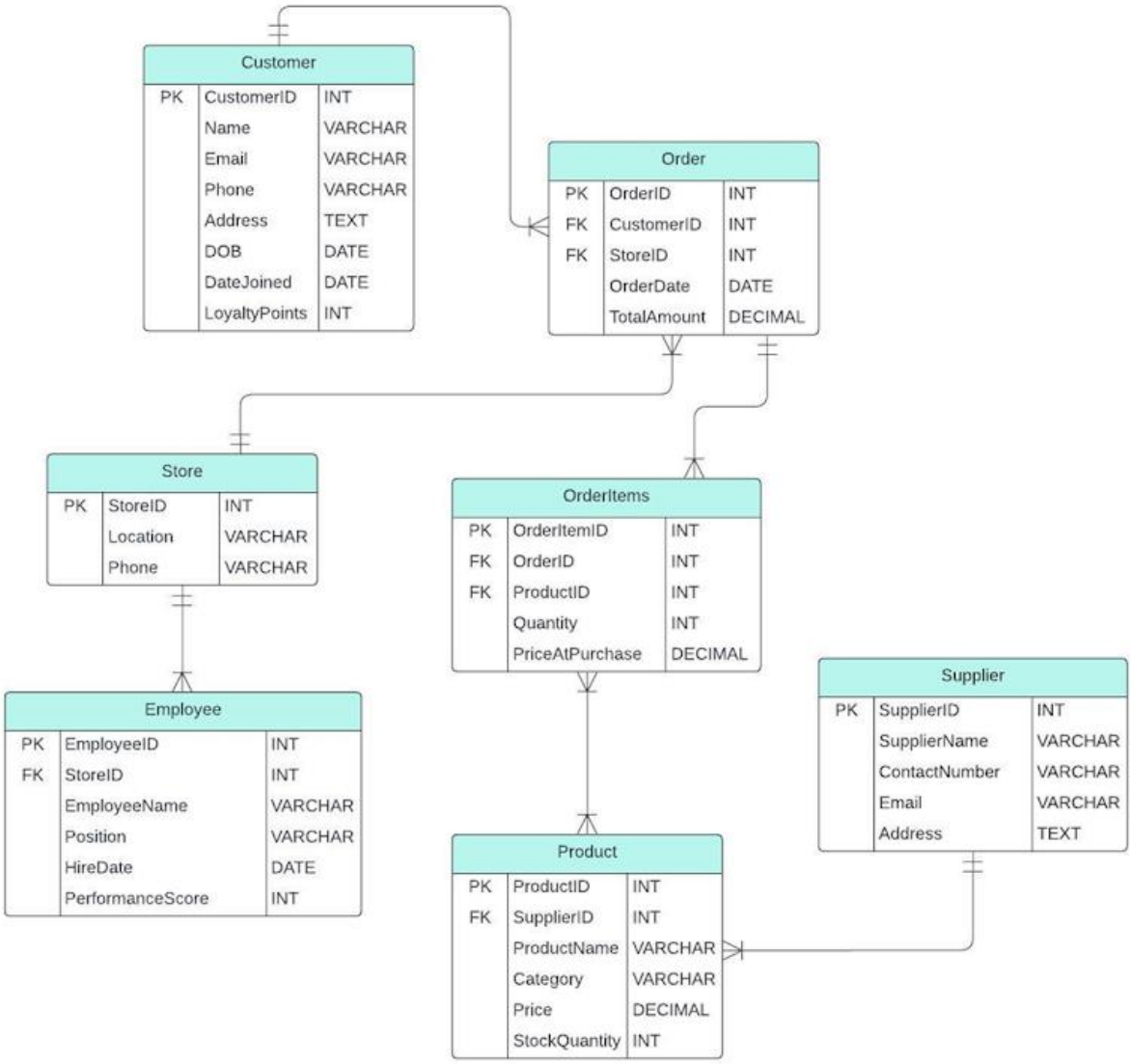

Figure 1 Entity Relationship Diagram for Retail Store

Table	Attribute	Data Type	Reason
Customer	CustomerID	INT	A unique integer identifier for each customer, efficient for indexing and queries.
	Name	VARCHAR	Flexible text format to store customer names of varying lengths (e.g., "John Doe").
	Email	VARCHAR	Emails are alphanumeric and variable-length; VARCHAR ensures storage efficiency.
	Phone	VARCHAR	Phone numbers often include formatting characters (e.g., +1, -), making VARCHAR suitable.
	Address	TEXT	Addresses can be lengthy and vary significantly, so TEXT provides sufficient flexibility.
	DOB	DATE	A specific date format is needed to store the customer's birth date.
	DateJoined	DATE	Captures the date the customer joined the loyalty program for tracking membership history.
	LoyaltyPoints	INT	An integer value to store points earned by customers in the loyalty program.
Order	OrderID	INT	Unique identifier for each order, optimised for indexing and relational links.
	CustomerID	INT	Foreign key linking to the Customer table. Matches the data type of CustomerID.
	StoreID	INT	Foreign key linking to the Store table. Matches the data type of StoreID.
	OrderDate	DATE	Tracks the specific date the order was placed.
	TotalAmount	DECIMAL	Stores the total cost of the order, including decimals for accuracy in monetary values.
Store	StoreID	INT	Unique identifier for each store, suitable for indexing.
	Location	VARCHAR	Text format to store store locations (e.g., city names or addresses).
	Phone	VARCHAR	Allows storage of phone numbers with varying formats.
OrderItems	OrderItemID	INT	Unique identifier for each order item, optimised for indexing.
	OrderID	INT	Foreign key linking to the Order table. Matches the data type of OrderID.
	ProductID	INT	Foreign key linking to the Product table. Matches the data type of ProductID.
	Quantity	INT	Integer to store the number of items purchased.
	PriceAtPurchase	DECIMAL	Captures the price of the product at the time of purchase, with decimal precision.
Employee	EmployeeID	INT	Unique identifier for each employee, efficient for indexing and queries.
	StoreID	INT	Foreign key linking to the Store table. Matches the data type of StoreID.
	EmployeeName	VARCHAR	Stores employee names of varying lengths.
	Position	VARCHAR	Tracks employee job titles (e.g., "Manager").
	HireDate	DATE	Tracks when the employee was hired.
	PerformanceScore	INT	Integer used for performance tracking or evaluations.
Product	ProductID	INT	Unique identifier for each product, optimised for indexing.
	SupplierID	INT	Foreign key linking to the Supplier table. Matches the data type of SupplierID.
	ProductName	VARCHAR	Stores product names, allowing flexibility for different lengths.
	Category	VARCHAR	Tracks product categories (e.g., "Electronics").
	Price	DECIMAL	Stores product prices with decimal precision for accuracy.
	StockQuantity	INT	Tracks the quantity of the product in stock.
Supplier	SupplierID	INT	Unique identifier for each supplier.
	SupplierName	VARCHAR	Stores supplier names with varying lengths.
	ContactNumber	VARCHAR	Allows flexibility for phone numbers with varying formats.
	Email	VARCHAR	Suitable for storing email addresses.
	Address	TEXT	Provides flexibility for lengthy addresses.

Figure 2 Explaining Data Types and Reason

```sql
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Phone VARCHAR(15),
    Address TEXT,
    DOB DATE,
    DateJoined DATE NOT NULL,
    LoyaltyPoints INT DEFAULT 0
);

-- Store Table
CREATE TABLE Store (
    StoreID INT PRIMARY KEY AUTO_INCREMENT,
    Location VARCHAR(100) NOT NULL,
    Phone VARCHAR(15)
);

-- Employee Table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    StoreID INT,
    EmployeeName VARCHAR(100) NOT NULL,
    Position VARCHAR(50),
    HireDate DATE,
    PerformanceScore INT,
    FOREIGN KEY (StoreID) REFERENCES Store(StoreID)
);

-- Product Table
CREATE TABLE Product (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    SupplierID INT,
    ProductName VARCHAR(100) NOT NULL,
    Category VARCHAR(50),
    Price DECIMAL(10, 2) NOT NULL,
    StockQuantity INT,
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)
);
```

```sql
-- Supplier Table
CREATE TABLE Supplier (
    SupplierID INT PRIMARY KEY AUTO_INCREMENT,
    SupplierName VARCHAR(100) NOT NULL,
    ContactNumber VARCHAR(15),
    Email VARCHAR(100),
    Address TEXT
);

-- Order Table
CREATE TABLE `Order` (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    StoreID INT,
    OrderDate DATE NOT NULL,
    TotalAmount DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (StoreID) REFERENCES Store(StoreID)
);

-- OrderItems Table
CREATE TABLE OrderItems (
    OrderItemID INT PRIMARY KEY AUTO_INCREMENT,
    OrderID INT,
    ProductID INT,
    Quantity INT NOT NULL,
    PriceAtPurchase DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

Figure 3 Code to Create Tables in SQL