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**

[Overview 3](#_Toc183625578)

[Logical Design 3](#_Toc183625579)

[Proposal of Database Build and Data Pipeline 4](#_Toc183625580)

[System Rules and Security 5](#_Toc183625581)

[References 6](#_Toc183625582)

[Appendix 7](#_Toc183625583)

**List of Figures**

[Figure 1 Entity Relationship Diagram for Retail Store 7](#_Toc183435034)

[Figure 2 Explaining Data Types and Reason 8](#_Toc183435035)

[Figure 3 Code to Create Tables in SQL 10](#_Toc183435036)

**Overview**

For our project in Deciphering Big Data, our team selected a retail store as the client organization. This company seeks to migrate its local data to the cloud and implement a loyalty program to better understand customer behavior and improve retention. Retail data is inherently complex, encompassing customer demographics, store locations, sales records, supplier information, and more. Dhanushkodi et al. (2024) highlight that analyzing customer behavior 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 was 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 incentivizes 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 normalized relational model, ensuring consistency and minimizing 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.

**Tables and Relationships**

1. **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.
2. **Order Table:**
   * Records individual customer purchases and links to the Store table via a many-to-one relationship (one store processes many orders). It also links to the OrderItems table via a one-to-many relationship (one order contains multiple order items). Tracks the total amount spent per order.
3. **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).
4. **Product Table:**
   * Lists details of all products available for sale, such as name, category, price, and stock quantity. Links to the Supplier table through a many-to-one relationship, using SupplierID as the foreign key.
5. **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.
6. **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.
7. **Supplier Table:**
   * Identifies vendors supplying products, enabling efficient inventory management and traceability.

**Enhancements to Logical Design**

To improve the database and better meet the requirements, we propose the following enhancements:

* **Loyalty Program History Table:** Tracks loyalty points earned, redeemed, and expired, allowing for detailed program analysis.
* **Customer Feedback Table:** Captures customer feedback on orders or store visits, linking to the Customer and Order tables for better insights.
* **Product Reviews Table:** Stores customer reviews and ratings for products, improving product and inventory decisions.
* **Promotions Table:** Tracks active and expired promotions, linking to the Product and Store tables for better marketing strategies.

**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 its versatility, scalability, and cost-effectiveness (Heck, 2009). Complementing this on-premises database is Snowflake, a cloud-based data warehouse offering elastic scalability and advanced analytics capabilities. This dual-system approach ensures that transactional data and analytical processes are efficiently managed.

**Data Pipeline and Integration**

1. **Data Collection:**
   * Customer data is collected via a front-end system (e.g., point of sale, online forms) and relayed to MySQL.
2. **ETL Process:**
   * An ETL process transfers cleaned and transformed data to Snowflake, ensuring high-quality and actionable data.
3. **Data Deduplication:**
   * Key data points (e.g., email, phone number) are cross-referenced to prevent duplicate accounts.
4. **Data Validation:**
   * Implement input validation and anomaly detection, such as setting age limits for accounts or excluding outlier spending patterns.
5. **Rewards System:**
   * Automated rewards allocation based on purchase milestones or demographics, with notifications sent via email.

**System Rules and Security**

**Scalability:**

* Horizontal scaling via sharding for MySQL and dynamic resource allocation in Snowflake ensure the system grows with the business.

**Security:**

* Measures such as encryption, role-based access control (RBAC), and secure APIs ensure compliance with GDPR guidelines (European Commission, 2016).

**Availability and Recovery:**

* Replication maintains secondary databases for failover.
* Daily full backups and incremental backups ensure data integrity and continuity.

**Conclusion**

This robust database design and pipeline provide a scalable, secure, and efficient solution for managing the complexities of retail data. By leveraging Snowflake’s advanced analytics and MySQL’s relational processing, the proposed system not only meets current requirements but also accommodates future growth and business needs.

**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=1713388284638 [Accessed 24 November 2024].

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

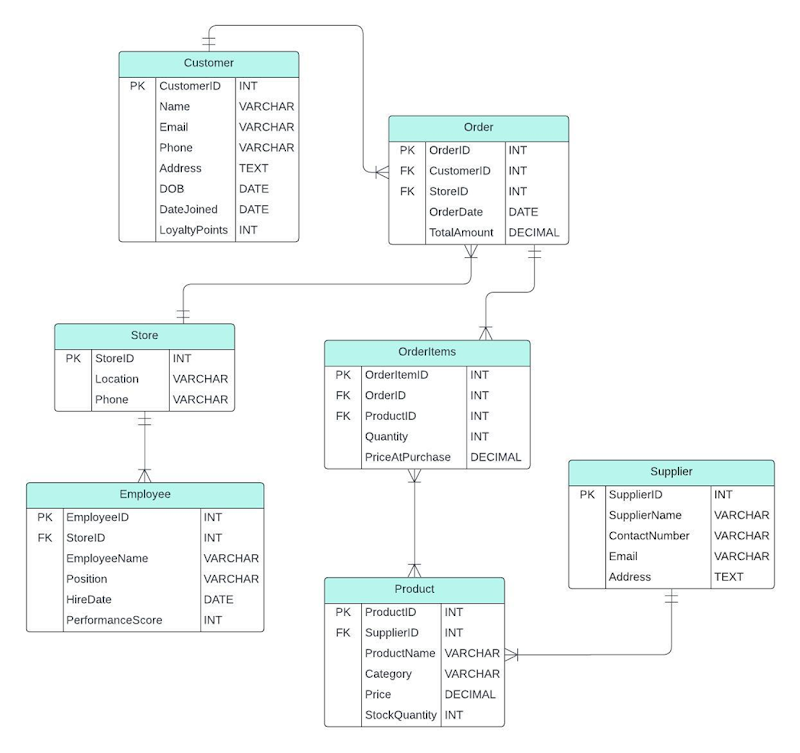
Lacey, R. & Sneath, J.Z. (2006) Customer Loyalty Programs: Are they Fair to Consumers? The Journal of Consumer Marketing 23(7): 458-464. DOI: 10.1108/07363760610713000

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].

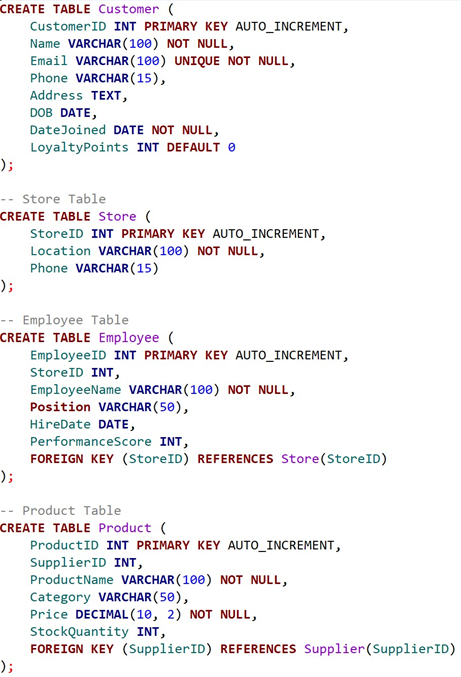
Appendix

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



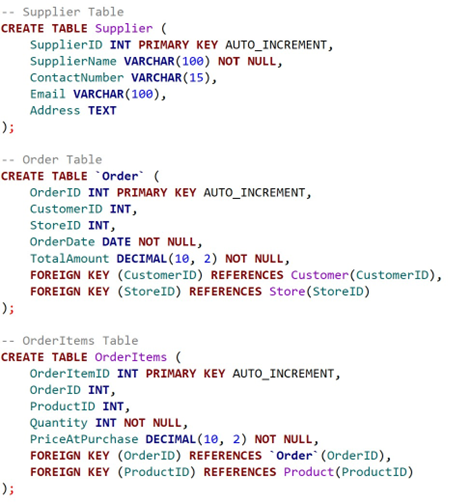


Figure 3 Code to Create Tables in SQL