Individual Project: Executive Summary
Database Design for a Retail Store

Name: Valentina Mercieca

Student ID: 12696474

Module: Deciphering Big Data

Due: 20th January 2025

Table of Contents

List of Figures

Introduction

This executive summary outlines the work carried out by the team to design and build a logical database for a retail store transitioning to a cloud-based system. The project aimed to boost customer retention by implementing a loyalty program while addressing the challenges of managing complex retail data such as customer demographics, sales records, and supplier information. This initiative is expected to strengthen customer satisfaction and provide the business with a competitive advantage in a data-driven market.

Summary of Work Completed

The project involved designing a logical database model, implementing it using a relational database management system (RDBMS), and integrating it with a cloud-based data warehouse. The Entity-Relationship Diagram (ERD) in Figure 1 is a visual representation of the database's structure that defines the relationships between different entities (Belcic & Stryker, 2024). The ERD adheres to the principles of third normal form (3NF), ensuring minimal data redundancy and high consistency, while providing a clear blueprint for implementing the database effectively through interrelated tables for customers, orders, order items, products, stores, employees, and suppliers.

Execution was carried out using MySQL Workbench, chosen for its capabilities in managing structured data and ability to handle complex queries. Figure 2 shows the Structured Query Language (SQL) code used for the schema setup and creation of the tables within the database. To address analytical needs and scalability, Snowflake was employed to manage large-scale data storage and enable advanced analytics. A critical component of the project was establishing a seamless flow of data between MySQL and Snowflake using an Extract, Transform, Load (ETL) process. This pipeline efficiently processed customer data collected through loyalty program signups, transferring it to Snowflake for in-depth analysis. Security measures, such as encryption, role-based access control, and GDPR-compliant practices, ensured data protection and legal compliance.

The loyalty program was a central feature of the database, designed to encourage repeat business through a dynamic, point-based reward system. Figures 3-6 illustrate this using dummy data. For example, when Alice Smith made two purchases totalling £2,999.95, the system calculated her points at a rate of one point per £10 spent, awarding her 299 points. This updated her LoyaltyPoints in the Customer table from 50 to 349. Similarly, David Lee earned 49 points for his £499.99 purchase, and Eva Green earned 39 points for her £399.99 order.

Beyond transaction-based rewards, the system is designed to accommodate other milestones that incentivise customer loyalty. For instance, customers can earn bonus points on their birthdays (e.g., an extra 10 points), for reaching one year of membership, or for hitting specific spending thresholds. These customisable features allow the system to adapt to the store's needs while showcasing its ability to calculate

and manage incentives dynamically. By automating updates, the database ensures accuracy, reduces manual errors, and supports real-time reward tracking.

The success of loyalty programs in enhancing customer retention has been demonstrated in various retail case studies. For instance, Sephora's Beauty Insider program has over 25 million members who benefit from a tiered reward system. This structure incentivises spending to unlock exclusive perks, fostering a sense of achievement (Woolnough-Rai, 2023). Personalised offers based on individual preferences and shopping behaviours further enhance customer satisfaction and deepen emotional connections, driving both loyalty and sales.

Another example of an effective loyalty program is Starbucks' use of technology to create seamless and engaging customer experiences. Its Mobile App integrates rewards and payment systems, allowing customers to earn and redeem points while enjoying personalised promotions and custom order options. This approach encourages frequent visits and higher spending. In fact, Starbucks (2022) reported 26.4 million active reward members in a 90-day period, accounting for 53% of U.S. store spending (Akcam, 2023).

Review of Database Modelling Concepts

Database modelling forms the foundation of the retail store system by structuring data for efficiency and reliability. At its core is data abstraction, encompassing conceptual, logical, and physical levels. The conceptual level, often represented by an ER diagram, captures key entities like Customer, Product, and Order, along with their relationships. This abstraction transitions into the logical level, refining the ER diagram into a relational schema aligned with DBMS rules, and finally to the physical level, which optimises the schema for storage and retrieval via indexing and partitioning.

Entities and attributes are vital in database modelling, representing elements and their properties. For example, the Customer entity includes attributes like Name, Phone, and DOB, while the Product entity features ProductName, Category, and Price. Relationships between entities reflect business workflows, including one-to-many (a customer placing multiple orders), many-to-one (products from a single supplier), and many-to-many (products in multiple orders). These relationships ensure relational integrity and accurate interaction mapping.

Primary and foreign keys are needed for the database's structure, where the former uniquely identify records and the latter link related tables – boosting consistency and efficient navigation. Normalisation organises data to reduce redundancy and improve integrity, adhering to 1NF, 2NF, and 3NF principles. For instance, separating customer and order data minimises duplication of information.

Despite its strengths, the relational model has limitations. Its rigid structure can hinder adaptability to changing needs or unstructured data. Strict schema definitions require time-consuming changes for scaling, such as distributing databases to accommodate new data types or relationships (Coșofreț & Ciuciu, 2018). For example, storing customer feedback or multimedia content demands extensive customisation or

auxiliary solutions, which may not always be practical. Singhal and Nambiar (2016) note that as data grows, query performance may degrade, necessitating optimisation through indexing and query tuning. Another challenge is the difficulty in efficiently handling hierarchical or network data structures, which are better suited to alternative data models. Finally, the reliance on normalisation to maintain integrity can increase query complexity, potentially slowing down operations for real-time analytics or frequent multi-table joins.

Nevertheless, the advantages of relational modelling outweigh these drawbacks for this project at this stage. It ensures consistency, integrity, and precision in queries, crucial for transactional and operational tasks. By integrating Snowflake, scalability and flexibility issues are mitigated, enabling effective handling of large-scale analytics. This hybrid approach combines a robust transactional database with advanced analytics and adaptability.

DBMS Analysis (SQL and NoSQL)

The decision to use SQL as the core interaction language in the chosen RDBMS was driven by the structured nature of retail data. MySQL's adherence to ACID properties ensures reliable and consistent transactions, critical for operations like order processing and inventory management. SQL's capabilities for handling complex queries, such as joins, aggregations, and nested queries, are essential for deriving insights from interconnected datasets. Its extensive tool ecosystem, community support, and seamless integration with third-party applications streamline development, reduce costs, and enable workflow automation.

Figure 7 demonstrates a MySQL code example for three queries showcasing the database's analytical capabilities: identifying best-selling products, assessing customer engagement, and detecting low-performing stores. These queries use joins, grouping, and filtering to extract meaningful information.

The first query, results shown in Figure 8, identifies best-selling products by aggregating the total quantity sold per product. It joins the Product and OrderItems tables, groups results by ProductID and ProductName, and calculates the total quantity sold, sorting the top 10 products. The analysis revealed the Smartphone as the top-selling item (three units sold), followed by the Laptop (two units). This helps the business focus on stocking high-demand items and optimising inventory.

The second query calculates the Customer Retention Rate, using a subquery to identify customers with multiple orders and dividing this by the total unique customers. Figure 9 reveals 20% of customers made repeat purchases, with Alice Smith as the sole example. A finding like this would underscore an opportunity for the business to work on its loyalty program's effectiveness by using targeted promotions or personalised offers.

The third query identifies low-performing stores by calculating quarterly revenue and filtering results below £700. It joins the Store and Order tables, groups data by StoreID and Location, and uses a HAVING clause to isolate underperformers. Figure 10

highlights the Uptown store, with quarterly revenue of £699.99, prompting management to address potential issues like local market conditions or staffing.

SQL is well-suited for managing the retail store database and loyalty program, offering precision and reliability for structured and interconnected datasets. However, as business demands grow and the loyalty program becomes more complex, Non-Structured Query Language (NoSQL) databases may be better suited for certain elements of the system.

With an expanding customer base and increasing interactions, the data collected will likely include semi-structured or unstructured types, such as behavioural patterns, clickstream data, real-time customer interactions, and multimedia content. NoSQL databases offer the flexibility to handle these dynamic datasets without the constraints of a fixed schema. For example, as customer profiles become richer with additional metadata, such as preferences, location histories, and personalised engagement data, document-oriented NoSQL databases like MongoDB would allow for seamless expansion. Unlike traditional relational databases, these systems enable the addition of new fields without requiring schema migrations, thus efficiently accommodating evolving business requirements.

NoSQL systems excel in horizontal scalability, making them ideal for managing high-volume data streams from transactional activities, app interactions, and promotions. This distributed architecture ensures reliability and performance, even during peak usage. Additionally, NoSQL databases excel in real-time data processing (Kekevi & Aydin, 2022) which is crucial for delivering instant personalised rewards based on live interactions.

The schema-less nature of NoSQL also supports the dynamic nature of loyalty program rules. Frequent updates to reward structures, customer segmentation, and promotional strategies can be implemented without disruption, ensuring the system remains agile and responsive to market demands.

Adapted from Phiri and Kunda (2017), Figure 11 compares SQL and NoSQL, highlighting their strengths and use cases. A dual approach would allow the organisation to maximise data value, enhance customer engagement, and maintain operational excellence.

Compliance and Legal Standards

The database design adhered to stringent compliance and security standards, including GDPR. Data security measures included encryption of sensitive data during both storage and transmission, secure APIs for communication, and role-based access control (RBAC) to ensure that users could only access data relevant to their roles. These features collectively reduced risks of data breaches, whether accidental or malicious.

Compliance with GDPR (Regulation (EU) 2016/679) was ensured through consent-based data collection, aligning with Article 6 of the directive, which mandates explicit

consent for data processing. The system also implemented measures for data minimisation as required by Article 5. This included identifying and managing inactive accounts, thereby reducing the retention of unnecessary data. Backup strategies were central to ensuring data integrity and resilience in line with Article 32. Daily full backups and incremental backups were conducted, stored securely in both on-premises systems and in the cloud, ensuring redundancy for recovery in case of system failures.

Beyond GDPR, the database also adhered to industry standards such as ISO/IEC 27001 for information security management. ISO/IEC 27001 compliance was achieved through the establishment of a robust information security management system (ISMS). This system included comprehensive risk assessments to identify and mitigate vulnerabilities, strict access control policies to limit data exposure to authorised personnel, and incident management protocols to promptly handle potential breaches (ISO/IEC, 2022).

Scalability considerations were integrated into the compliance framework to future-proof the system. Horizontal scaling via sharding and vertical scaling with hardware enhancements were incorporated to effectively handle growing workloads (Slingerland, 2024). Additionally, Snowflake's dynamic resource allocation enabled the system to seamlessly adapt to fluctuating business demands. Comprehensive logging mechanisms ensured compliance monitoring and facilitated the detection and response to potential breaches, further bolstering the system's security and compliance posture.

Conclusions and Recommendations

The completed database design provides a robust framework for managing retail operations, balancing transactional efficiency with analytical capabilities. To enhance the system further, several recommendations have been prioritised:

Firstly, optimising the ETL processes is essential for improving the efficiency of data synchronisation between MySQL and Snowflake. This can be achieved by automating data extraction tasks, optimising transformation workflows to reduce processing times, and implementing incremental loading to minimise redundant data transfer. Monitoring tools should also be used to identify and resolve bottlenecks in real time.

Secondly, future iterations of the database should evaluate NoSQL options for specific use cases, such as handling unstructured data or supporting dynamic data needs. Incorporating MongoDB or similar systems for targeted applications could enhance the database's versatility and position the business to adapt to emerging trends in data management. As the company grows, a hybrid approach that includes NoSQL elements could provide a competitive edge in addressing new challenges.

Thirdly, regular audits should be conducted to ensure ongoing compliance with GDPR and other legal standards. These audits should focus on verifying data protection measures, such as encryption protocols, access controls, and record-keeping practices, to ensure they align with evolving regulations. Audits also provide an

opportunity to identify vulnerabilities and recommend enhancements to maintain a high standard of data security.

Lastly, the loyalty program should incorporate the return period for purchased items before awarding points to ensure fairness and accuracy. Points accumulation should trigger predefined rewards, such as discounts or vouchers, once customers reach specific thresholds – providing clear incentives for continued engagement. Delaying rewards until the return period has ended will prevent discrepancies and reduce fraudulent behaviours, such as returning items after earning points. This adjustment will also enhance the accuracy of loyalty program analytics, ensuring reliable insights and a fair customer experience.

References

Akcam, B.K. (2023) ‘The Impact of Technology on the Starbucks Experience’, *Journal of Information Technology Teaching Cases*, 13(1), pp. 104-110. Available at: https://doi.org/10.1177/20438869221099305

Belcic, I. and Stryker, C. (2024) *What is an Entity Relationship Diagram?* Available at: https://www.ibm.com/think/topics/entity-relationship-diagram (Accessed: 29 December 2024).

Coșofreț, G. and Ciuciu, I. (2018) ‘Superstore Sales Reporting: A Comparative Analysis of Relational and Non-relational Databases’, *On the Move to Meaningful Internet Systems: OTM 2017 Workshops*. Rhodes, Greece, 23-27 October. Switzerland: Springer International Publishing. 98-102. Available at: https://doi.org/10.1007/978-3-319-73805-5_10

ISO/IEC. (2022) *ISO/IEC 27001:2022 – Information Security, Cybersecurity and Privacy Protection – Information Security Management Systems – Requirements*. Available at: https://www.iso.org/standard/27001 (Accessed: 2 January 2025).

Kekevi, U. and Aydin, A. A. (2022) ‘Real-Time Big Data Processing and Analytics: Concepts, Technologies, and Domains’, *Computer Science*, 7(2), pp. 111-123. Available at: https://doi.org/10.53070/bbd.1204112

Phiri, H. and Kunda, D. (2017) ‘Comparative Study of NoSQL and Relational Database’, *Zambia ICT Journal*, 1(1), pp. 1-4. Available at: https://doi.org/10.33260/zictjournal.v1i1.8

Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) (2016) *Official Journal* L 119, 27 April, pp. 1-88. Available at: http://data.europa.eu/eli/reg/2016/679/oj (Accessed: 31 December 2024).

Singhal, R. and Nambiar, M. (2016) ‘Predicting SQL Query Execution Time for Large Data Volume’, *IDEAS '16: Proceedings of the 20th International Database Engineering & Applications Symposium*. Montreal, QC, Canada, 11-13 July. New York, NY, United States: Association for Computing Machinery. 378-385. Available at: https://doi.org/10.1145/2938503.2938552

Slingerland, C. (2024) *Horizontal vs Vertical Scaling: Which Should You Choose?* Available at: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/#:~:text=While%20horizontal%20scaling%20refers%20to,%2C%20storage%2C%20or%20network%20speed (Accessed: 3 January 2025).

Starbucks. (2022) *Starbucks Q1 Fiscal Year 2022 Earnings Conference Call*. Available at: https://investor.starbucks.com/events-and-presentations/events/event-details/2022/Starbucks-Q1-Fiscal-Year-2022-Earnings-Conference-Call/default.aspx (Accessed: 2 January 2025).

Woolnough-Rai, M. (2023) *Scale Success Story: Sephora’s Beauty Insider*. Available at: https://loyaltylion.com/blog/scale-success-story-sephoras-beauty-insider (Accessed: 2 January 2025).

Appendix

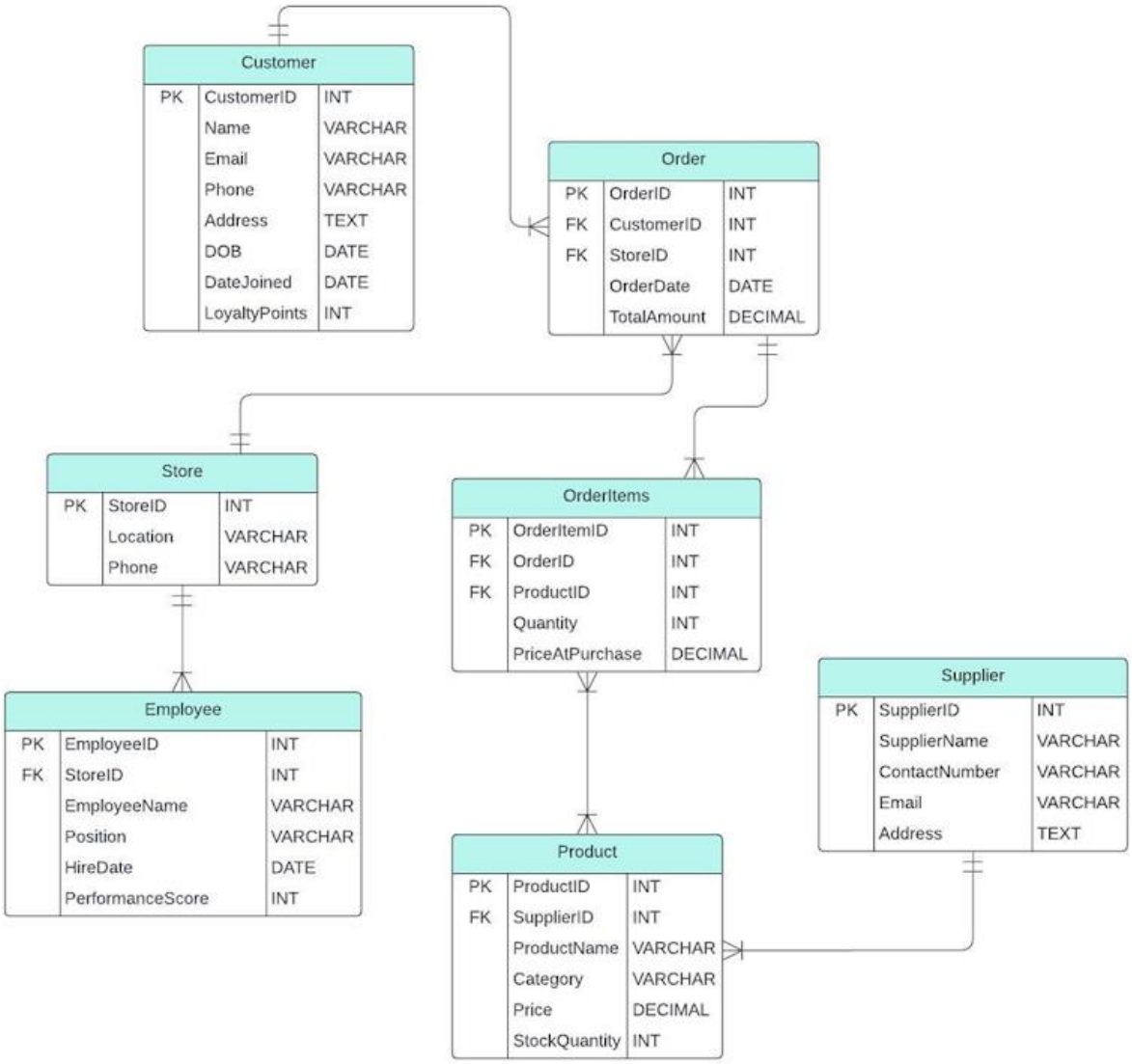

Figure 1 Entity Relationship Diagram (ERD) for Retail Store

```sql
/* Creating a new database */
CREATE DATABASE Retail_Store_Database;
USE Retail_Store_Database;

-- ----------------------------------------------------

/* Creating the tables */
-- Customer Table
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)
);

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

```sql
-- 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)
);

-- 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 2 SQL Schema Setup and Table Creation

```sql
/* Inserting Dummy Data */
-- Customer Table
INSERT INTO Customer (Name, Email, Phone, Address, DOB, DateJoined, LoyaltyPoints)
VALUES
('Alice Smith', 'alice@example.com', '123-456-7890', '123 Main St', '1985-06-15', '2023-01-10', 50),
('Bob Johnson', 'bob@example.com', '123-456-7891', '456 Elm St', '1990-11-22', '2023-01-15', 0),
('Cathy Brown', 'cathy@example.com', '123-456-7892', '789 Oak St', '1995-02-10', '2023-02-01', 120),
('David Lee', 'david@example.com', '123-456-7893', '222 Maple St', '1988-07-20', '2023-03-01', 30),
('Eva Green', 'eva@example.com', '123-456-7894', '444 Pine St', '1992-04-12', '2023-05-15', 70);

-- Store Table
INSERT INTO Store (Location, Phone)
VALUES
('Downtown', '555-1234'),
('Uptown', '555-5678'),
('Suburb', '555-4321');

INSERT INTO Supplier (SupplierName, ContactNumber, Email, Address)
VALUES
('ElectroSupply', '555-1122', 'contact@electrosupply.com', '100 Tech Ave'),
('HomeGoods Co.', '555-3344', 'sales@homegoods.com', '200 Home St');

-- Product Table
INSERT INTO Product (SupplierID, ProductName, Category, Price, StockQuantity)
VALUES
(1, 'Smartphone', 'Electronics', 699.99, 50),
(1, 'Laptop', 'Electronics', 999.99, 20),
(2, 'Sofa', 'Furniture', 499.99, 10),
(2, 'Dining Table', 'Furniture', 299.99, 15),
(2, 'Chair', 'Furniture', 99.99, 100),
(1, 'Tablet', 'Electronics', 399.99, 30);

-- Order Table
INSERT INTO `Order` (CustomerID, StoreID, OrderDate, TotalAmount)
VALUES
(1, 1, '2024-10-15', 1699.97), -- Alice's large order
(2, 2, '2024-11-01', 699.99),  -- Bob's medium purchase
(3, 1, '2024-12-10', 399.99),  -- Cathy's smaller order
(4, 3, '2024-12-15', 499.99),  -- David's medium purchase
(5, 3, '2024-12-25', 399.99),  -- Eva's small purchase
(1, 1, '2024-12-30', 1299.98); -- Alice's second large order

-- OrderItems Table
INSERT INTO OrderItems (OrderID, ProductID, Quantity, PriceAtPurchase)
VALUES
(1, 1, 1, 699.99), -- Alice buys a Smartphone
(1, 2, 1, 999.99), -- Alice buys a Laptop
(2, 1, 1, 699.99), -- Bob buys a Smartphone
(3, 6, 1, 399.99), -- Cathy buys a Tablet
(4, 3, 1, 499.99), -- David buys a Sofa
(5, 4, 1, 399.99), -- Eva buys a Dining Table
(6, 1, 1, 699.99), -- Alice buys another Smartphone
(6, 2, 1, 599.99); -- Alice buys another Laptop

-- ------------------------------------------------------
```

Figure 3 Inserting Dummy Data for Demonstration Purposes

CustomerID	Name	Email	Phone	Address	DOB	DateJoined	LoyaltyPoints
1	Alice Smith	alice@example.com	123-456-7890	123 Main St	1985-06-15	2023-01-10	50
2	Bob Johnson	bob@example.com	123-456-7891	456 Elm St	1990-11-22	2023-01-15	0
3	Cathy Brown	cathy@example.com	123-456-7892	789 Oak St	1995-02-10	2023-02-01	120
4	David Lee	david@example.com	123-456-7893	222 Maple St	1988-07-20	2023-03-01	30
5	Eva Green	eva@example.com	123-456-7894	444 Pine St	1992-04-12	2023-05-15	70
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4 View of Customer Table before Updating Loyalty Points

```
138     /* Updating loyalty points based on TotalAmount */
139 •   CREATE TEMPORARY TABLE TempLoyaltyPoints AS
140     SELECT
141         CustomerID,
142         FLOOR(COALESCE(SUM(o.TotalAmount), 0) / 10) AS PointsToAdd
143     FROM
144         `Order` o
145     GROUP BY
146         CustomerID;
147
148 •   UPDATE Customer c
149     JOIN TempLoyaltyPoints t ON c.CustomerID = t.CustomerID
150     SET c.LoyaltyPoints = c.LoyaltyPoints + t.PointsToAdd;
151
152 •   DROP TEMPORARY TABLE TempLoyaltyPoints;
153
154     -- ------------------------------------------------------
```

Figure 5 SQL Code for Updating Loyalty Points

CustomerID	Name	Email	Phone	Address	DOB	DateJoined	LoyaltyPoints
1	Alice Smith	alice@example.com	123-456-7890	123 Main St	1985-06-15	2023-01-10	349
2	Bob Johnson	bob@example.com	123-456-7891	456 Elm St	1990-11-22	2023-01-15	69
3	Cathy Brown	cathy@example.com	123-456-7892	789 Oak St	1995-02-10	2023-02-01	159
4	David Lee	david@example.com	123-456-7893	222 Maple St	1988-07-20	2023-03-01	79
5	Eva Green	eva@example.com	123-456-7894	444 Pine St	1992-04-12	2023-05-15	109
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6 View of Customer Table after Updating Loyalty Points

```sql
/* QUERY 1: Best Selling Products */
SELECT
    p.ProductID,
    p.ProductName,
    SUM(oi.Quantity) AS TotalQuantitySold
FROM
    Product p
JOIN
    OrderItems oi ON p.ProductID = oi.ProductID
GROUP BY
    p.ProductID, p.ProductName
ORDER BY
    TotalQuantitySold DESC
LIMIT 10;


/* QUERY 2: Customer Engagement Levels */
SELECT
    (SELECT COUNT(DISTINCT CustomerID)
     FROM `Order`
     WHERE CustomerID IN (
         SELECT CustomerID
         FROM `Order`
         GROUP BY CustomerID
         HAVING COUNT(OrderID) > 1
     )) * 100.0 /
    (SELECT COUNT(DISTINCT CustomerID)
     FROM `Order`) AS CustomerRetentionRate;



/* QUERY 3: Low-Performing Stores */
SELECT
    s.StoreID,
    s.Location,
    SUM(o.TotalAmount) AS QuarterlyRevenue
FROM
    Store s
JOIN
    `Order` o ON s.StoreID = o.StoreID
WHERE
    o.OrderDate BETWEEN DATE_SUB(CURDATE(), INTERVAL 3 MONTH) AND CURDATE()
GROUP BY
    s.StoreID, s.Location
HAVING
    QuarterlyRevenue < 700
ORDER BY
    QuarterlyRevenue ASC;
```

Figure 7 SQL Code for the 3 Queries

	ProductID	ProductName	TotalQuantitySold
▶	1	Smartphone	3
	2	Laptop	2
	3	Sofa	1
	4	Dining Table	1
	6	Tablet	1

Figure 8 Result for Query 1: Best-Selling Products

	CustomerRetentionRate
▶	20.00000

Figure 9 Result for Query 2: Customer Engagement Levels

	StoreID	Location	QuarterlyRevenue
▶	2	Uptown	699.99

Figure 10 Result for Query 3: Low-Performing Stores

Aspect	SQL (Relational Databases)	NoSQL (Non-Relational Databases)
Structure	Structured schema with predefined tables and relationships.	Flexible schema; handles structured, semi-structured, and unstructured data.
Scalability	Vertical scaling (adding more resources to a single server).	Horizontal scaling (adding more servers to distribute load).
Performance	Excellent for complex queries (e.g., joins, aggregations).	High performance for large-scale data operations and simple queries.
Transaction Support	Strong ACID compliance ensures reliable, consistent transactions.	Eventual consistency; some NoSQL databases support limited ACID compliance (e.g., Cassandra)
Data Volume	Efficient for moderate volumes of structured data.	Ideal for handling massive volumes of unstructured or semi-structured data.
Query Language	Uses SQL, a standardised and powerful query language.	Varies by database (e.g., MongoDB uses JSON-like queries, Cassandra uses CQL).
Flexibility	Less flexible; schema changes require careful planning and migration.	Highly flexible; supports evolving data models without downtime.
Complex Relationships	Excellent for modelling and managing complex relationships.	Better suited for simpler relationships; graph databases (e.g., Neo4j) excel in specific cases; storing semi and unstructured data is less complex.
Real-Time Processing	Limited real-time capabilities in high-volume scenarios.	Optimised for real-time analytics and dynamic data processing.
Cost	May require expensive hardware upgrades for scaling.	Cost-effective scaling using commodity hardware.
Tooling and Ecosystem	Mature ecosystem with extensive tools and community support.	Rapidly growing ecosystem; less mature but highly innovative.
Use Cases	Financial systems, ERP, inventory management, and structured data.	Big data, IoT, content management, social media, and unstructured data.
Learning Curve	Familiar and widely taught; easier for new developers.	Varies by database; may require specialised knowledge.

Figure 11 Comparison Between SQL and NoSQL Databases (Adapted from Phiri & Kunda, 2017, p.3, table 1)