ASSIGNMENT

A diagram of a company

Description automatically generated with medium confidence

Rationale Behind Schema Design

My database schema was created to offer a normalised, efficient, and scalable framework for handling a financial system that includes various interconnected entities like customers, accounts, transactions, loans, payments, and administrative information. This design guarantees data integrity, facilitates complex queries, and reduces redundancy.

It entails an Entity-Specific Design Considerations such as;

1. Customers: Stores personal and contact details which forms the core of the system, also linking to accounts, loans and consents ensures a view of the customer.
2. Accounts: This table tracks account types (e.g savings etc) with opening dates and balances. Also supports multiple accounts per customer.
3. Transaction: Supports and records all activities in the account, providing traceability and transparency.
4. Admins: This enables oversight and links to account for more detailed control
5. Loans and Payments: These tracks loans and payments associated with it, supports interest rate and loan schedules, ensuring financial accuracy.

Advantages of My Design

* Reduced Redundancy: Lowers storage needs and enhances consistency by removing duplicate information.
* Improved Performance: The relational framework and indexing support efficient query processing.
* Flexibility: The modular design enables easy incorporation of new features or tables (such as insurance and rewards programs).
* Preparedness for Compliance: Guarantees that the system can effectively handle data privacy and auditing requirements.

Data Integrity

Foreign key constraints enforce relationships between tables:

Customers and Accounts via CustID

Accounts and Transactions via AcctID

Loans and Payments via LoanID

Cascading ON DELETE CASCADE ensures that related records are cleaned up automatically, maintaining consistency.

Scalability:

My database schema is designed to handle growth by isolating data into modular tables, each responsible for a specific domain (eg, loans, payments).

Use of AUTO\_INCREMENT for primary keys ensures the system can efficiently generate unique IDs.

.

.

TASK 2

Database Normalisation: Before Normalisation

Screens screenshot of a computer

Description automatically generated

After Normalisation

A diagram of a company

Description automatically generated

Normalization involves structuring data within a database to minimize duplication and enhance data accuracy. The advantages of normalization consist of:

* Reduce Redundancy (Redundancy is unnecessary duplication):

\* It eliminates duplicate data by dividing the database into smaller and related tables.

\* Enables pieces of data to be stored only once, thereby reducing data storage requirements and inconsistencies caused by data duplication.

* Enhances Data Integrity:

\*Guarantees that the information kept in the database stays precise and uniform.

\*Changes such as updates, deletions, or additions are transmitted accurately without leading to any inconsistencies in data.

* Simplifies Database Maintenance:

\* Makes the database easier to update and manage.

\* Avoids issues like update, insertion, and deletion anomalies.

* Enhances Data Security:
* Sensitive information can be separated into designated tables, enabling more precise management of access permissions.
* Relationships prevent unauthorised modifications in one table from impacting other tables.
* Enables future expansion while maintaining the integrity of the current data.
* Improves Query Performance:
* Normalisation reduces redundancy, which decreases the volume of data that must be scanned or processed during queries.
* Indexes are more efficient when used with smaller tables that contain targeted data.
* Support Scalability
* A normalised database design is modular and allows for easy expansion by incorporating additional tables or relationships.

SQL QUERIES;

1. Query to List customers with loan payments due in the first 7 days of  
   the month. Ensure the query includes both customer and  
   account details.

SELECT

c.CustID,

c.FirstName,

c.LastName,

c.Email,

c.PhoneNum,

c.Address,

a.AcctNum,

a.AcctType,

a.Balance,

l.LoanID,

l.LoanType,

l.LoanAmount,

p.PaymentID,

p.PaymentDate,

p.PaymentAmount

FROM

Customers c

JOIN

Loans l ON c.CustID = l.CustID

JOIN

Payments p ON l.LoanID = p.LoanID

JOIN

Accounts a ON c.CustID = a.CustID

WHERE

DAY(p.PaymentDate) <= 7

ORDER BY

p.PaymentDate;

A screenshot of a computer

Description automatically generated

1. Query to Extract all transactions that occurred in the past 5 days.

SELECT \*

FROM Transactions

WHERE TransactDate >= CURDATE() - INTERVAL 5 DAY;

A screenshot of a computer

Description automatically generated

No transaction was made in the last seven days, thus the output of the query.

1. Query to List customers with balances exceed £5000. Ensure that at  
   least one customer has a balance greater than £5000.

SELECT

c.CustID,

c.FirstName,

c.LastName,

c.Email,

c.PhoneNum,

c.Address,

a.AcctNum,

a.AcctType,

a.Balance

FROM

Customers c

JOIN

Accounts a ON c.CustID = a.CustID

WHERE

a.Balance > 5000;

A screenshot of a computer

Description automatically generated

1. Query to Calculate and list the bank’s Total Outstanding, which is  
   the sum of the up-to-date balances across all accounts. The  
   total should be calculated by adding the opening balance  
   of each account, subtracting all outgoing payments, and  
   adding all incoming payments.

SELECT

SUM(a.Balance)

- COALESCE(SUM(CASE WHEN t.TransacType IN ('Withdrawal', 'Transfer') THEN t.Amount ELSE 0 END), 0)

+ COALESCE(SUM(CASE WHEN t.TransacType = 'Deposit' THEN t.Amount ELSE 0 END), 0) AS TotalOutstandings

FROM

Accounts a

LEFT JOIN

Transactions t ON a.AcctID = t.AcctID;

A screenshot of a computer

Description automatically generated

1. Evidence of at least two queries of your choice against  
   YOUR database that use the SQL JOIN clause – choose TWO  
   from each of Inner, Left, or Right JOIN.

INNER JOIN QUERY;

SELECT

t.TransactID,

t.TransactDate,

t.TransacType,

t.Amount,

t.Description,

c.FirstName,

c.LastName,

c.Email

FROM

Transactions t

INNER JOIN

Accounts a ON t.AcctID = a.AcctID

INNER JOIN

Customers c ON a.CustID = c.CustID;

A screenshot of a computer

Description automatically generated

INNER JOIN

Customers c ON a.CustID = c.CustID;

LEFT JOIN QUERY

SELECT

c.CustID,

c.FirstName,

c.LastName,

l.LoanID,

l.LoanType,

l.LoanAmount,

l.InterestRate

FROM

Customers c

LEFT JOIN

Loans l ON c.CustID = l.CustID;

A screenshot of a computer

Description automatically generated

4. Security and Scalability

Securing a database is critical to maintain integrity, confidentiality, and availability of data. There are different threats that can affect a database security. Technical approach to secure database is as follows:

* Access Control:

Role-based access control: Not everyone should have access to the account, each customer and accounts should be assigned to Admins and unauthorised personnel should be restricted from it. e.g the admin table regulates this. Users and applications should only have access to limited required resources. Enforces complex passwords and use strong password managers.

Employee training on data security policies and practices. Auditing and monitoring data access and sharing activities. Using encryption for sensitive data. Strong password policies should be put in place, like using multi-factor authentication. Access control includes Discretionary Access control and Mandatory Access control.

* Regular Updates:

Regular updating of the Database system and software is critical aspect in maintaining and securing a reliable database.

* Updating and patching the database system improve stability and compliance and adherence to industry standards and regulations such as GDPR etc.
* New features can be introduced to deprecate outdated functionality improving usability.
* Database Auditing and Monitoring:

This is an essential practice for ensuring data integrity, security, and compliance. It involves tracking database activities and anomalies maintaining a record of operations. Incidents can easily be detected and reported and investigated, unusual forceful logins can also be detected. Ensures changes to data are authorised and traceable. Database monitoring helps to monitor performance across queries, identifying slow-running queries and deadlocks. Track usage patterns to predict future scaling.

Real time tools to track database metrics, alerts, and notification (this include unusual query patterns and multiple failed logins).

Backups and Recovery

A strong backup and recovery plan is vital for safeguarding the safety and integrity of your database. In the face of disasters such as system failures, cyber-attacks, human errors, or natural disasters, a clearly defined backup and recovery strategy is crucial for minimizing data loss and maintaining business continuity. Effective backup strategies should consider aspects like frequency, security, storage, and recovery procedures to ensure that data can be restored quickly and accurately when needed.

Regular Backups

Performing regular backups is a key practice for data protection. This guarantees that copies of the database are always available, allowing for quick recovery in case of data loss. Frequent backups help reduce the risk of losing critical data, especially in fast-paced environments where updates and transactions occur continuously.

Backup Frequency: The frequency of backups should align with the importance and rate of change of the data. For essential applications, real-time or daily backups may be necessary, while less critical systems might only need weekly backups. Incremental backups (which save only the changes made since the last backup) and differential backups (which capture changes since the last full backup) can help minimize the time and storage required for backups without compromising data security.

Backup Retention Policy: Creating a backup retention policy is crucial for determining how long backup copies are kept. Unneeded data can be deleted to free up storage, but important backups should be retained for longer periods to comply with regulations and potential future needs.

Automated Backups: Automating the backup process ensures consistency and lowers the risk of human error. With automated backups, organizations can schedule them at regular intervals, ensuring that no backup opportunities are missed. Automated systems can also send notifications or alerts in the event of backup failures, prompting immediate action.

SCALABILITY:

Scalability refers to a system's capability to handle an increasing workload or larger data volumes without sacrificing performance. The main benefits of scalability include its ability to handle diverse workloads, minimize downtime, improve system efficiency, and enhance customer satisfaction. To achieve scalability, various strategies can be implemented within the database to ensure consistent performance as the system grows. These strategies include optimizing the database, ensuring high availability, leveraging cloud infrastructure, improving the application layer, and fine-tuning database architecture.

There are different ways my database can be scaled.

1. Database Optimisation  
   a. Indexing

Implementing indexes on commonly queried fields such as CustID, AcctID, and LoanID to enhance query performance. Utilise composite indexes for queries that incolve multiple fields (for instance, CustID and Email).

b. Partitioning

Horizontal Partitioning (sharding): Distribute the Customers, Accounts, and Loans tables across various database servers based on shard key, like CustID or geographic area.

-Vertical Partitioning: Move less frequency accessed columns (such as ConsentDetails) to a separate table to minimise the size of the data that is frequently queried.

c. Query Optimisation

Revise complex queries following best practices to ensure they run efficiently. Use EXPLAIN or query planners to pinpoint performance issues.

1. Ensuring High Availability and Reliability
2. Database Replication

Establish master-slave replication to enhance read scalability and provide fault tolerance. All write operations are directed to the master database, while read operations can be spread across replicas.

1. Load Balancing

Utilise load balancers to evenly distribute traffic among several database servers, preventing anyone sever from becoming overwhelmed.

1. Backup and Recovery

Conduct regular database backups to maintain data integrity and integrity and facilitate swift recovery in the event of failures.

1. Storage and Infrastructure
2. Cloud Scaling

Utilise cloud platforms such as AWS RDS, Azure SQL Database, or Google Cloud SQL to take advantage of their managed scaling features, which offer both vertical and horizontal scaling on demand.

1. Distributed Data Storage

Keep extensive data (such as logs and consent history) in distributed storage solutions like Amazon S3 or Hadoop, which can be connected to the database for easy access when required.

1. Enhancements to the Application Layer
2. Caching

Implement caching tools such as Redis or Memcached for data that is accessed often (for instance, customer information and account balances). Store query results for dashboards and reports that experience high traffic.

1. Asynchronous Processing

Delegate lengthy tasks, such as creating detailed reports, to asynchronous processes or background jobs.

1. Improvements in Database Architecture
2. Integration of NoSQL

Utilise NoSQL database such as MongoDB or Cassandra for handling unstructured or semi-structured data, like transaction logs or consent histories. This approach allows for quicker access to non-relational data while still using a relational database for essential financial information.

1. Microservices Framework

Break the application down into smaller, independent services. For instance:

A service dedicated to managing customer profiles.

A service focused on processing transactions and payments.

A service for managing loans.

1. Data Warehousing Solutions

Implement data warehouse like snowflake Amazon Redshift to relieve the production database of analytical and reporting responsibilities.

1. Oversight and Performance Indicators

Utilise monitoring solutions such as New Relic or Prometheus to access database performance, focusing on aspects like query response times, CPU consumption, and memory usage. Set up alerts to detect and address performance problems in advance.

By applying these strategies, the database can efficiently expand to accommodate a substantial rise in customers while preserving performance and dependability. Careful planning and the use of contemporary database technologies will facilitate seamless growth and effective functioning.

References

1. AWS RDS (for cloud database scaling) [AWS RDS Documentation](https://aws.amazon.com/rds/)
2. Azure SQL Database [Azure SQL Database Overview](https://azure.microsoft.com/en-us/services/sql-database/)
3. Google Cloud SQL - Google Cloud SQL Overview
4. Redis Caching – Redis Documentation
5. MongoDB NoSQL Database [MongoDB Documentation](https://www.mongodb.com/docs/)
6. Prometheus Monitoring – Prometheus Documentation