Saw Myat Mon Hnin_P00202698_DBDD

by Saw Myat Mon Hnin

Submission date: 24-Jul-2024 05:09PM (UTC+0100)
Submission ID: 237323213
File name: copypaste.txt (19.48K)
Word count: 2537
Character count: 16427

Task-1

Scenario of Banking Loan System

Abaddon Bank, based in Yangon, Myanmar, started its journey in 2010. By 2019, the bank had expanded its services to include a loan system. Initially, all loan-related data was managed using Excel spreadsheets, which seemed practical at the time. However, as the number of loans and customers increased, this method led to numerous issues and errors, making it clear that a more reliable solution was needed. To address these challenges, Abaddon Bank decided to transition to a database system, aiming to improve data accuracy, streamline operations, and enhance security. This move was crucial for supporting the bank's growth and ensuring better for its customers.

Figure 1 Day by Day flow chart

At Abaddon Bank, Customer must register their collateral for loan. The collateral has collateral type and collateral detail information. Abaddon's staff approved or not the registration according to the rules set by the bank. After then, the bank calculates

the loan type and % for loan. In the end, the bank issue the loan and customer repay the loan monthly.

Entities

1. Customer
2. Customer_Type
3. Registration
4. Loan
5. Loan_Type
6. Loan_Rule
7. Rule
8. Staff
9. Staff_Type
10. Register_Collateral
11. Collateral
12. Collateral_Type

Transactions

- Customer entity- Insert, update or delete customer data.
- Customer_Type- Insert, update or delete customer type data.
- Registration- Insert, update or delete registration data.
- Loan- Insert, update or delete loan information.

- Loan_Type- Update loan type.
- Loan_Rule- Insert or delete loan rule link.
- Rule- Insert, update or delete rule.
- Staff- Insert, update or delete staff data.
- Staff_Type- Insert, update or delete staff type data.
- Register_Collateral- Insert or delete register collateral link.
- Collateral- Insert, update or delete collateral information.
- Collateral_Type- Insert or update collateral type.

Abaddon's Loan Form

The Loan Form for Abaddon Bank is designed to collect all necessary information for processing loan database. It includes registration information, customer data, staff data, collateral data, rules and loan information.

Limitations

It has several limitations despite the thorough design of Abaddon Bank's database system. While it focuses on detailed loan management that it doesn't track or display payment transactions. This gap restricts the ability to conduct detailed financial audits or analyze payment trends. It's impacting financial planning and the identification of

payment-related issues. Additionally, the system may lack advanced user access controls, permissions and potentially allowing unauthorized data access and modifications. Addressing these limitations will be crucial for enhancing the bank.

Task-2

Entity Relationship Diagram

Figure 2 Registration Form

Data Dictionary

Registration

Name	Datatype	Length	Constraints	Description
Register_Id	Char	10	Primary Key, Not Null	Registration id
Registration_Date	Date		Not Null	Registration date
Approve	Varchar	20	Not Null	Registration that approves for loan
Customer_Account_Id	Char	10	Foreign Key, Not Null	Customer bank account id
Loan_Id	Char	10	Foreign Key, Not Null	Loan id

Staff_Id	Char	10	Foreign Key, Not Null	Staff id

Customer

Name	Datatype	Length	Constraints	Description
Customer_Account_Id	Char	10	Primary Key, Not Null	Customer bank account id
Customer_Name	Varchar	50	Not Null	Customer's name
Address	Varchar	100	Not Null	Customer's address
NRC	Char	20	Not Null	Customer's NRC
Email	Varchar	20	Not Null	Customer's email
Customer_Type_Id	Char	10	Foreign Key, Not Null	Customer type's id

Customer Type

Name	Datatype	Length	Constraints	Description
Customer_Type_Id	Char	10	Primary Key, Not Null	Customer type's id
Customer_Type_Description	Varchar	50	Not Null	Description of customer type

Loan

Name	Datatype	Length	Constraints	Description
Loan_Id	Char	10	Primary Key, Not Null	Loan's id
Amount_Accepted_Percentage	int	5	Not Null	% for loan
Start_Date	Date		Not Null	Loan start date

End_Date	Date		Not Null	Loan end date
Status	Varchar	20	Not Null	Continue or complete
Loan_Type_Id	Char	10	Foreign Key, Not Null	Loan type's id

Loan Type

Name	Datatype	Length	Constraints	Description
Loan_Type_Id	Char	10	Primary Key, Not Null	Loan type's id
Type_Description	Varchar	20	Not Null	Type of Loan

Loan_Rule

Name	Datatype	Length	Constraints	Description
Loan_Id	Char	10	Primary Key, Foreign Key, Not Null	Loan's id
Rule_Id	Char	10	Primary Key, Foreign Key, Not Null	Rule's id

Rule

Name	Datatype	Length	Constraints	Description
Rule_Id	Char	10	Primary Key, Not Null	Rule's id
Rule_Description	Varchar	100	Not Null	Rule's Description

Staff

Name	Datatype	Length	Constraints	Description
Staff_Id	Char	10	Primary Key, Not Null	Staff's id
Staff_Name	Varchar	30	Not Null	Staff's name
Staff_Type_Id	Char	10	Foreign Key, Not Null	Staff type's id

Staff Type

Name	Datatype	Length	Constraints	Description
Staff_Type_Id	Char	10	Primary Key, Not Null	Staff type's id
Staff_Type_Description	Varchar	20	Not Null	Position of staff
Department	Varchar	20	Not Null	Department that staff work

Register_Collateral

Name	Datatype	Length	Constraints	Description
Collateral_Id	Char	10	Primary Key, Foreign Key, Not Null	Collateral's id
Register_Id	Char	10	Primary Key, Foreign Key, Not Null	Registration's id

Collateral

Name	Datatype	Length	Constraints	Description
Collateral_Id	Char	10	Primary Key, Not Null	Collateral's id
Collateral_Description	Varchar	30	Not Null	Description of collateral
Location	Varchar	50	Null	Collateral's location
Estimated_Value	Dec	15,2	Not Null	Collateral's value
Owenership_Status	Varchar	50	Not Null	To record the owner
Collateral_Type_Id	Char	10	Foreign Key, Not Null	Collateral type's id

Collateral_Type

Name	Datatype	Length	Constraints	Description

Collateral_Type_Id	Char	10	Primary Key, Not Null	Collateral type's id
Type_Description	Varchar	50	Not Null	Type of Collateral

Task-3

Normalization

Normalization helps organize a database to reduce duplicate data and keep it accurate. For Abaddon Bank, this means more reliable and consistent information. It improves performance, making data retrieval faster and easier. By eliminating errors, normalization ensures the bank can make decisions based on accurate data (JavaTPoint, 2021).

UNF 1NF 2NF 3NF

Register_Id

Registration_Date

Approve

Customer_Account_Id

Customer_Name

Address

Phone_Number

NRC

Email

Customer_Type_Id

Customer_Type_Description

Loan_Id

Amount_Accepted_Percentage

Start_Date

End_Date

Status

Loan_Type_Id

Type_Description

Rule_Id

Rule_Description

Staff_Id

Staff_Name

Staff_Type_Id

Staff_Type_Description

Department

Collateral_Id

Collateral_Description

Location

Estimated_Value

Ownership_Status

Collateral_Type_Id

Type_Description Register_Id

Registration_Date

Approve

Customer_Account_Id

Customer_Name

Address

Phone_Number

NRC

Email

Customer_Type_Id

Customer_Type_Description

Loan_Id

Amount_Accepted_Percentage

Start_Date

End_Date

Status

Loan_Type_Id

Type_Description

Rule_Id

Rule_Description

Staff_Id

Staff_Name

Staff_Type_Id

Staff_Type_Description

Department

Collateral_Id

Register_Id*

Collateral_Description

Location

Estimated_Value

Ownership_Status

Collateral_Type_Id

Type_Description Register_Id

Registration_Date

Approve

Customer_Account_Id

Customer_Name

Address

Phone_Number

NRC

Email

Customer_Type_Id

Customer_Type_Description

Loan_Id

Amount_Accepted_Percentage

Start_Date

End_Date

Status

Loan_Type_Id

Type_Description

Rule_Id

Rule_Description

Staff_Id

Staff_Name

Staff_Type_Id

Staff_Type_Description

Department

Collateral_Id

Register_Id*

Collateral_Id

Collateral_Description

Location

Estimated_Value

Ownership_Status

Collateral_Type_Id

Type_Description Registration

Register_Id

Customer_Account_Id*

Loan_Id*

Staff_Id*

Registration_Date

Approve

1

Customer

Customer_Account_Id

Customer_Type_Id*

Customer_Name

Address

Phone_Number

NRC

Email

Customer_Type

Customer_Type_Id

Customer_Type_Description

Loan

Loan_Id

Loan_Type_Id*

Amount_Accepted_Percentage

Start_Date

End_Date

Status

Loan_Type

Loan_Type_Id

Type_Description

Loan_Rule

Loan_Id*

Rule_Id*

Rule

Rule_Id

Rule_Description

2
Staff

Staff_Id

Staff_Type_Id*

Staff_Name

Staff_Type

Staff_Type_Id

Staff_Type_Description

Department

Register_Collateral

Collateral_Id*

Register_Id*

Collateral

Collateral_Id

Collateral_Type_Id*

Collateral_Description

Location

Estimated_Value

Ownership_Status

Collateral_Type

Collateral_Type_Id

Type_Description

Optimization Table

Registration

Register_Id (PK)

Customer_Account_Id (FK)

Loan_Id*

Staff_Id*

Registration_Date

Approve

1 Customer

Customer_Account_Id (PK)

Customer_Type_Id (FK)

Customer_Name

Address

Phone_Number

NRC

Email

Customer_Type

Customer_Type_Id (PK)

Customer_Type_Description

Loan

Loan_Id (PK)

Loan_Type_Id (FK)

Amount_Accepted_Percentage

Start_Date

End_Date

Status

Loan_Type

Loan_Type_Id (PK)

Type_Description

Loan_Rule

Loan_Id (PK)(FK)

Rule_Id (PK)(FK)

Rule

Rule_Id (PK)

Rule_Description

Staff

Staff_Id (PK)

Staff_Type_Id (FK)

Staff_Name

Staff_Type

Staff_Type_Id (PK)

Staff_Type_Description

Department

Register_Collateral

Collateral_Id (PK) (FK)

Register_Id (PK) (FK)

Collateral

Collateral_Id (PK)

Collateral_Description

Location

Estimated_Value

Ownership_Status

Collateral_Type

Collateral_Type_Id (PK)

Type_Description

Anomalies of Normalization

Normalization solved some of Abaddon Bank's problems. Some examples are:

1. Insert Anomaly

- Adding a new loan type without immediately linking it to a loan can create orphaned data entries.

2. Update Anomaly

- Updating "Manager" to "Senior Manager" involves modifying each staff record with this type, which can lead to inconsistencies if some rows are not updated correctly.

3. Delete Anomaly

- Removing a loan that is the only instance of a specific type (e.g., "Home Loan") would delete the loan type description, losing information about that loan type.

Task-4

Assessment of Design

For Abaddon Bank, the logical database design was turned into a physical design using SQL scripts. These logical entities, attributes, and relationships from the design phase were translated into physical tables, columns, and constraints. This ensured that the abstract design was accurately implemented, allowing for efficient data storage and retrieval. The transition involved specifying data types, and setting up primary and foreign keys to maintain data integrity (GeeksforGeeks, 2021).

The tables for Abaddon Bank were designed carefully to meet of MySQL requirements. Each table had appropriate columns, data types, and constraints to matched the logical model. Primary keys were set to uniquely identify each record, and foreign keys enforced relationships between tables. The SQL script created a strong database structure, capable of supporting the bank's data operations. The tables included Customer_Type, Loan_Type, Rule, Staff_Type, Collateral_Type, Customer, Loan, Staff, Collateral, Register, Register_Collateral, and Loan_Rule (www.tutorialspoint.com,n.d.).

Derived data such as data that can be calculated or inferred from existing data that was identified and represented through SQL queries and views. For example, the total outstanding loan amount for a customer could be calculate using aggregate functions. This kept the database normalized and avoided redundancy while providing necessary information through calculated results (www.tutorialspoint.com,n.d.).

De-normalization was also considered to enhance query performance and simplify data retrieval for Abaddon Bank. While the initial design followed normalization principles to maintain data integrity and avoid redundancy and some scenarios

necessitated de-normalization to optimize performance. The involved combining related tables and duplicating some data such as customer details and their loan information to ensure efficient query execution. While extensive de-normalization was not needed, the approach was planned to balance performance optimization and maintaining a logical database structure tailored to the bank's needs (SearchDataManagement, n.d.).

Task-5

Create Table

SQL scripts were developed to define the tables, columns, primary keys, foreign keys, and other database objects to create the database for Abaddon Bank. These scripts were run in the dbForge environment, using MySQL as the DBMS (Database Management System). The process started with the parent tables which do not have foreign keys to provent any foreign key constraint errors. These tables included Customer_Type, Loan_Type, Rule, Staff_Type, and Collateral_Type. After the parent

tables were set up, the next step was to create the child tables. These tables included foreign keys to ensure the correct relationships between them. These child tables were Customer, Loan, Staff, Collateral, Regiser, Register_Collateral, and Loan_Rule.

SQL commands were used in XAMPP to create tables, columns, and set keys during database development. This allowed for direct management of the database and quick resolution of any errors. Occasionally, syntax errors occurred, like incorrect data types or missing commas, but these were fixed by checking the code.

Using SQL made it easy to organize the database effectively. It ensured the database structure was well-designed and data integrity was maintained, making it reliable for bank operations.

Task-6

Insert Table

To insert data into the database for Abaddon Bank, a series of SQL INSERT statements were crafted to populate the tables with initial records. These statements were executed sequentially, starting with the parent tables (Customer_Type, Loan_Type, Rule, Staff_Type, and Collateral_Type) to respect foreign key constraints. Subsequently, data was inserted into the child tables (Customer, Loan, Staff, Collateral, Register, Register_Collateral, and Loan_Rule).

Task-7

Select Data

1. The total amount that the bank loaned to each customer

2. Find the staff who have handled the greatest number of approved loans and the total loan amount they managed

3. Find the loan type that has the highest total loan amount across all customers

4. Find customers with more than one loan

5. Show the total loan amount and number of loans for each customer who has taken loans of different types

6. Show the customers who have been approved for the highest number of loans with a specific rule applied

7. List the staff members who have approved loans and the total number of loans they approved

8. List the total value of collaterals provided by each customer

9. List the customers with loans ending in the current year

10. List customers who have provided collaterals valued more than the average collateral value

Task-8

Future development of a distributed database

In the future, Abaddon Bank might merge with other similar companies, a common strategy for business looking to expand. As Abaddon Bank grows and establishes branches worldwide, it will need a reliable way to manage all its data, such as loans, customer information, and financial transactions, across all its locations.

A distributed database system would allow each branch to handle its own information while also sharing it with a central database. This setup ensures that all branches use the same data and system. It's keeping everything accurate and consistent across the organization. Abaddon Bank can use a homogeneous distributed database system to maintain consistency and simplify data integration.

A homogeneous distributed database system ensures all branches use the same DBMS. This simplifies data integration and maintenance. This uniformity allows for easier management of the system and reduces the complexity associated with data synchronization.

Data fragmentation is important in a distributed database. It involves breaking the database into smaller parts or fragments which can be stored at different locations. Proper fragmentation ensures data is stored near where it is most frequently used, improving access times and reducing communication costs. However, poorly managed fragmentation can lead to inefficiencies and increased complexity in query processing.

The following are advantages and disadvantages of using distributed database for Abaddon Bank:

Advantages Disadvantages

Improved Data Management: Handles vast data across all branches efficiently.

Complex Setup: Required meticulous planning and expertise.

High Availability: Remains operational even if one part fails. Synchronization Issues: Keeping data synchronized is challenging.

Faster Access: Localized storage enables quicker information access. High Costs: Expensive to implement and maintain.

Scalability: Easily scales with bank growth. Security Concerns: Ensuring data security across locations is tough.

Fault Tolerance: Handles failures without disrupting operations. Fragmentation Challenges: Managing data fragmentation is difficult (www.linkedin.com,n.d.).

Task-9

Future Development of a Data Warehouse

A data warehouse could be a game-changer for Abaddon Bank, driven by the need for comprehensive data analysis and reporting. By consolidating data from various sources into a centralized repository, the bank can significantly enhance its decision-making capabilities. Executives and managers will benefit from a unified view of data across departments, such as customer accounts, loans, and staff performance, enabling more informed strategic decisions and timely responses to market changes. Furthermore, a data warehouse improves data quality and consistency by standardizing information from disparate systems, ensuring reliable analytics and reporting. The ability to store large volumes of historical data supports trend analysis and forecasting, providing insights into past transactions and customer behavior, which are invaluable for risk management and strategic planning, by offloading analytical processing from transactional databases, the data ware house boosts operational efficiency, allowing transactional systems to handle day-to-day operations without the burden of complex queries. This results in faster transaction processing and better customer service. Additionally, a data warehouse supports regulatory compliance by maintaining a comprehensive, easily accessible record of transactions and customer

interactions, facilitating quick responses to regulatory inquiries. Enhanced customer insights from consolidated data across various touchpoints enable personalized services and targeted marketing campaigns, improving customer satisfaction. The scalability and performance of a data warehouse ensure that as Abaddon Bank grows, it can handle increasing data loads and analytical demands effectively. Lastly, leveraging advanced analytics and data mining within a data warehouse helps the bank uncover hidden patterns and correlations, supporting initiatives like fraud detection, credit scoring, and customer segmentation. In essence, a data warehouse would provide Abaddon Bank with the tools necessary to drive strategic growth, operational efficiency, and enhanced customer experiences (www.quest-global.com, 2022).

Saw Myat Mon Hnin_P00202698_DBDD

ORIGINALITY REPORT

10%	0%	0%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to KMD Computer Center Student Paper	3%
2	Submitted to University of Greenwich Student Paper	1%
3	Submitted to RMIT University Student Paper	1%
4	Submitted to NCC Education Student Paper	1%
5	Submitted to University of Suffolk Student Paper	1%
6	Submitted to NACIT Blantyre Student Paper	1%
7	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
8	Submitted to University of Teesside Student Paper	<1%
9	Submitted to Entregado a MCC Training Institute el 2012-10-29	<1%

Student Paper

10	Submitted to University of Portsmouth Student Paper	<1%
11	Submitted to School of Accounting & Management Student Paper	<1%
12	Submitted to Gaborone Institute of Professional Studies Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		