

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

“KHARKIV POLYTECHNIC INSTITUTE”

Department of Software Engineering and Intelligent Management Technology

COURSE WORK

in the discipline “\_ Design and development of databases\_”

Supervisor:

Head of the Department of SEMIT Dept.

\_ Kopp A. M. \_

Author:

student of the group KH-222ia.e\_

\_Uyar B.B.\_

Kharkiv – 2024

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE  
NATIONAL TECHNICAL UNIVERSITY  
“KHARKIV POLYTECHNIC INSTITUTE”

Department of Software Engineering and Intelligent Management Technology

Mark

-

Head of comission,

- /

/

«\_\_\_\_» 2024\_.

COURSE WORK

Topic: «\_ Design and Development of a Database for Bank Management  
System\_»

Supervisor:

Head of the Department of SEMIT Dept. /\_ Kopp A. M. \_/

«21» \_05\_2024\_.

Author:

student of the group KH-222ia.e /\_ Uyar B.B.\_/

«21\_» \_05\_ 2024\_.

Kharkiv – 2024

# Task for a course work

### Summary of the work:

### c) Design And Development Of The Database Application

Course work supervisor / \_\_ Kopp A. M. \_\_ /

## REVIEW

for the course work

authored by student of the group KH-22ia.e\_

\_\_\_ Uyar B.B. \_\_\_

The course work is devoted to solving the problem of designing and implementing a database system for a bank management system.

During the execution of the work there was comprehensive analysis conducted on the subject domain, including the identification of entities, attributes, and business rules governing the bank management system. Additionally, meticulous attention was given to the design and development phases, resulting in the creation of logical and physical data models that accurately represent the system's structure.

During the execution of the work was developed encompassing tables for Clients, Accounts, Transactions, and Loans, along with appropriate relationships and constraints to ensure data integrity.

At the final stage of course work the implementation of the database in MySQL was successfully carried out, with the creation of tables, insertion of initial records, and development of queries, stored procedures, and triggers to facilitate efficient data management and retrieval.

The coursework fully meets the requirements and deserves an evaluation of understanding and proficiency in database design and development principles. Therefore, it deserves an evaluation of «\_ excellent \_».

CW supervisor  
/\_\_\_ Kopp. A. M.\_ \_\_\_/

\_\_\_ Head of the Department of SEMIT \_\_\_

## ABSTRACT

Report of implementation of SW: \_\_\_\_ p., \_\_\_\_ fig., \_\_\_\_ tab., \_\_\_\_ sources

*Keywords:* Banking System, Database, MySQL, Information Modeling.

The object of work is the development of a database for a bank management system.

The subject of work is the creation of a database that allows storing and processing information about clients, accounts, transactions, and loans in the banking system.

The purpose of the work is to increase the productivity of storing and processing information about product supply through the development of an appropriate database.

Thus, to achieve the goal, the following tasks were solved in the work:

1. Analysis of the subject domain and determination of business rules.
2. Development of logical and physical data models of the database.
3. Creation of a database in the MySQL database management system.
4. Population of the database with initial records.
5. Development of SQL queries, stored procedures, and triggers to automate operations and enforce business rules.
6. Development of a database application for convenient usage and interaction with the database.

## CONTENT

ABBREVIATIONS .....	8
INTRODUCTION .....	9
1 ANALYSIS OF THE SUBJECT DOMAIN .....	10
1.1 The system of business rules .....	10
1.2 Modeling of the subject area .....	11
1.3 Problem statement .....	12
2 DATABASE DESIGN AND DEVELOPMENT .....	14
2.1 Development of logical and physical data models .....	14
2.2 Description of the database structure .....	15
2.3 Implementation of the database in the MySQL DBMS .....	15
2.4 Filling the database with initial records .....	17
2.5 Creating database queries .....	21
3 DESIGN AND DEVELOPMENT OF THE DATABASE APPLICATION .....	24
3.1 Database application design .....	24
3.2 Database application usage example .....	25
CONCLUSIONS .....	27
REFERENCES .....	28

## **ABBREVIATIONS**

DB – database.

DBMS – database management system.

DDL – Data Definition Language.

ER – Entity-Relationship.

SQL – Structured Query Language.

## **INTRODUCTION**

Some companies buy goods from different suppliers (both legal entities and individuals). The purchase of goods is carried out in batches and is executed in the form of supply contracts. Each contract for the supply of goods has a unique number and can be concluded with only one supplier. The documents for each contract for each product indicate: name, size of the delivered lot and price (in UAH).

To store and process such information, it is necessary to develop a database (DB), which will take into account the specifics of considered business rules.

The object of work – the process of supplying products that are performed in a commercial enterprise.

The subject of the work is a database that will allow storing and processing information on the supply of products in a commercial enterprise.

The purpose of the work is to increase the productivity of storage and processing of information on the supply of products through the development of an appropriate database.

## **1 ANALYSIS OF THE SUBJECT DOMAIN**



## 1.1 The system of business rules

The bank management system operates within a structured framework governed by various entities and attributes. Bank management system based on the provided system of business rules:

Facts:

- Clients are identified by unique account numbers and store personal information such as name, address, and contact details.
- Each client can have multiple accounts, each with a unique account number, account type, and balance.
- Transactions are categorized by types (e.g., deposit, withdrawal, transfer) and are associated with specific accounts, including transaction details like amount and date/time.
- Loans are issued based on unique contracts with clients, with each loan agreement having a unique loan ID and associated loan details.

Restrictions:

- Clients can be individuals or organizations (legal entities).
- Tax identification numbers must be unique for legal entity clients.
- Each loan agreement must be associated with a specific client, and missing client information is not permitted.
- The start date of a loan agreement must be specified, defaulting to the current date if not provided.
- Loan amounts must always be indicated and cannot be zero or negative.

Activators:

When adding data about a legal entity client, verification is required to ensure data about them as an individual does not already exist, and vice versa.

Conclusions:

Loans are considered delinquent if payments are not received within 30 calendar days from the due date.

Loans are considered overdue if a client fails to meet their payment obligations within five days of the due date.

## 1.2 Modeling of the subject area

To model the bank management system's subject area, an Entity-Relationship (ER) diagram is utilized to depict entities, attributes, and their relationships.

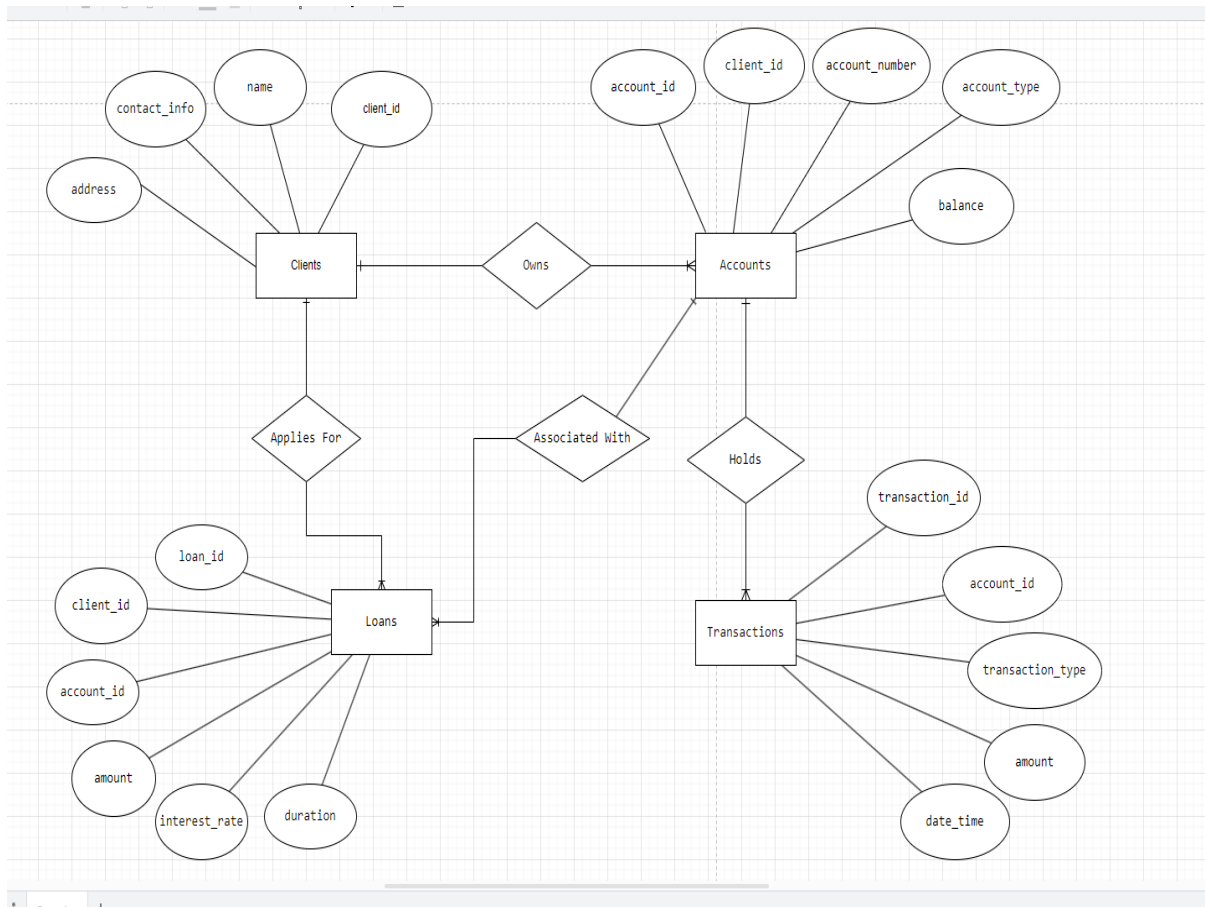


Figure 1.1 – ER-model of the subject area

### Entities:

1. Clients
2. Accounts
3. Transactions
4. Loans

### Attributes:

1. Clients: client\_id (Primary Key), name, address, contact\_info
2. Accounts: account\_id (Primary Key), client\_id (Foreign Key), account\_number, account\_type, balance

3. Transactions: transaction\_id (Primary Key), account\_id (Foreign Key), transaction\_type, amount, date\_time

4. Loans: loan\_id (Primary Key), client\_id (Foreign Key), account\_id (Foreign Key), amount, interest\_rate, duration

### **Relationships:**

#### **1. Clients - Accounts (One-to-Many):**

- Each client can have multiple accounts, but each account belongs to only one client.

- Represented by the client\_id attribute in the Accounts entity, acting as a foreign key referencing the client\_id attribute in the Clients entity.

#### **2. Accounts - Transactions (One-to-Many):**

- Each account can have multiple transactions, but each transaction is associated with only one account.

- Represented by the account\_id attribute in the Transactions entity, acting as a foreign key referencing the account\_id attribute in the Accounts entity.

#### **3. Clients - Loans (One-to-Many):**

- Each client can have multiple loans, but each loan is associated with only one client.

- Represented by the client\_id attribute in the Loans entity, acting as a foreign key referencing the client\_id attribute in the Clients entity.

#### **4. Accounts - Loans (One-to-Many):**

- Each account can have multiple loans, but each loan is associated with only one account.

- Represented by the account\_id attribute in the Loans entity, acting as a foreign key referencing the account\_id attribute in the Accounts entity.

The Entity-Relationship (ER) diagram for the bank management system is a visual representation of the relationships between its core entities:

Clients, Accounts, Transactions, and Loans. Clients are uniquely identified by a `client_id` and are associated with personal details such as name, address, and contact information.

Accounts, each identified by an `account_id`, belong to clients and hold attributes like `account_number`, `account_type`, and `balance`.

Transactions record activities within accounts, including `transaction_type`, `amount`, and `date_time`, with each transaction linked to a specific account.

Loans, distinguished by a `loan_id`, involve clients and accounts, detailing loan amounts, interest rates, and durations. These entities are interconnected through various relationships, enabling efficient management and tracking of banking activities. Clients may possess multiple accounts, each account can have multiple transactions, and clients can also hold multiple loans.

Additionally, each loan is associated with a single account, enhancing the clarity and structure of the system's design.

### **1.3 Problem statement**

To achieve the goal of the work it is necessary to perform the following tasks:

- 1 Develop a logical model of the database.
- 2 Develop a physical database model.
- 3 Describe the structure of the database.
- 4 Implement the database in the MySQL database management system (DBMS) using DDL (Data Definition Language) commands.
- 5 Fill the database with initial records and develop queries in SQL (Structured Query Language).
- 6 Develop the database application.
- 7 Demonstrate the usage example of the database application.

## 2 DATABASE DESIGN AND DEVELOPMENT

### 2.1 Development of logical and physical data models

The logical data model in the IDEF1X notation is shown in Figure 2.1.

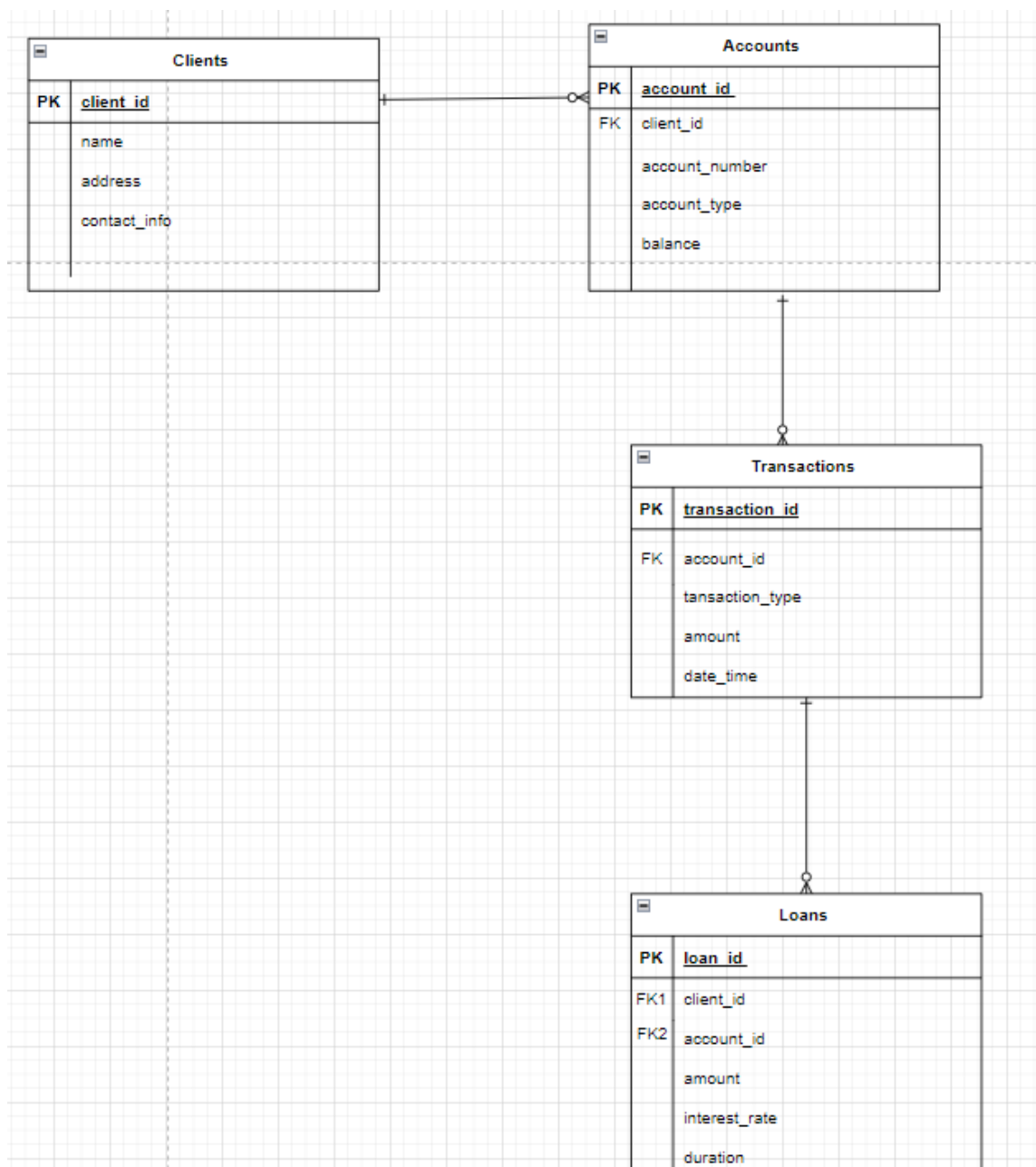


Figure 2.1 – Logical data model

The physical data model in the IDEF1X notation is shown in Figure 2.2.

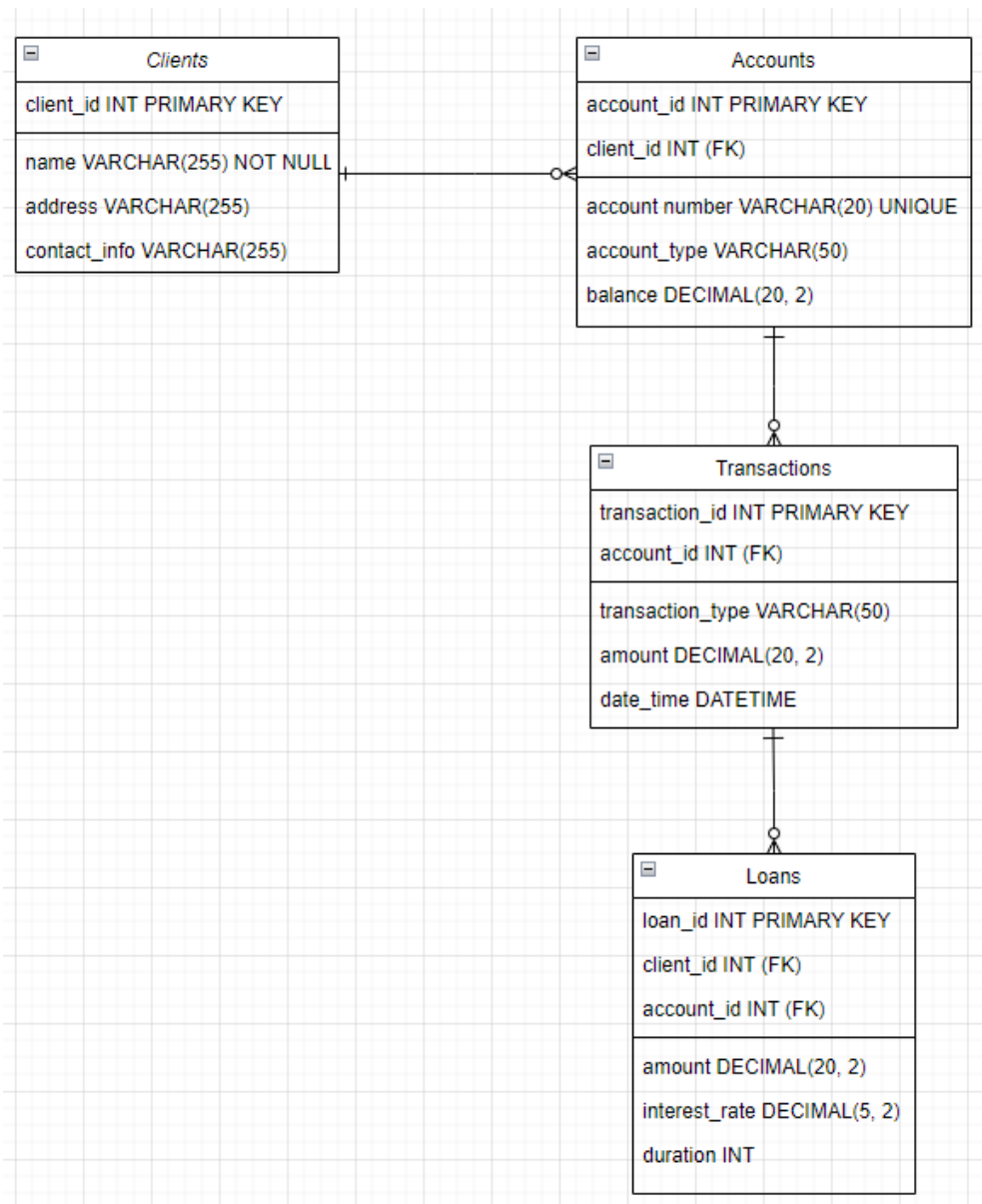


Figure 2.2 – Physical data model

This ER model captures the relationships between clients, accounts, transactions, and loans in a bank management system, allowing for efficient data management and retrieval.

## 2.2 Description of the database structure

The Clients table stores information about the clients of the bank. Each client is uniquely identified by a client\_id and has associated personal details such as their name, address, and contact information.

Table 2.1 – Description of the structure of the table "Clients":

Key	Field name	Data type	Field size	Description
PK	client_id	INT	Whole	Numerical client identifier (Primary Key)
	name	VARCHAR	255	Name of the client (NOT NULL)
	address	VARCHAR	255	Address of the client
	contact_info	VARCHAR	255	Contact information of the client

The Accounts table maintains records of the accounts held by the bank's clients. Each account is identified by an account\_id and is associated with a specific client through the client\_id foreign key. It stores information about the account number, type (e.g., savings, checking), and balance.

Table 2.2 – Description of the structure of the table "Accounts":

Key	Field name	Data type	Field size	Description
PK	account_id	INT	Whole	Numerical account identifier (Primary Key)
FK	client_id	INT	Whole	Numerical client identifier (Foreign Key referencing Clients table)
	account_number	VARCHAR	20	Unique account number

	account_type	VARCHAR	50	Type of account (e.g., savings, checking)
	balance	DECIMAL	20, 2	Balance of the account

The Loans table contains information about the loans provided by the bank to its clients. Each loan is uniquely identified by a loan\_id and is associated with a specific client and account through foreign keys. It stores details such as the loan amount, interest rate, and duration.

Table 2.3 – Description of the structure of the table "Loans":

Key	Field name	Data type	Field size	Description
PK	loan_id	INT	Whole	Numerical loan identifier (Primary Key)
FK	client_id	INT	Whole	Numerical client identifier (Foreign Key referencing Clients table)
FK	account_id	INT	Whole	Numerical account identifier (Foreign Key referencing Accounts table)
	amount	DECIMAL	20, 2	Amount of the loan
	interest_rate	DECIMAL	5, 2	Interest rate of the loan
	duration	INT		Duration of the loan (in months)

The Transactions table records all transactions made within the bank. Each transaction is uniquely identified by a transaction\_id and is associated with a specific account through the account\_id foreign key. It includes information about the transaction type (e.g., deposit, withdrawal, transfer), the transaction amount, and the date and time of the transaction.

Table 2.4 – Description of the structure of the table "Transactions":



Key	Field name	Data type	Field size	Description
PK	transaction_id	INT	Whole	Numerical transaction identifier (Primary Key)
FK	account_id	INT	Whole	Numerical account identifier (Foreign Key referencing Accounts table)
	transaction_type	VARCHAR	50	Type of transaction (e.g., deposit, withdrawal, transfer)
	amount	DECIMAL	20, 2	Amount of the transaction
	date_time	DATETIME	Short date format	Date and time of the transaction

### 2.3 Implementation of the database in the MySQL DBMS

The data scheme for the database, which is implemented by MySQL database, is shown in Figure 2.3.

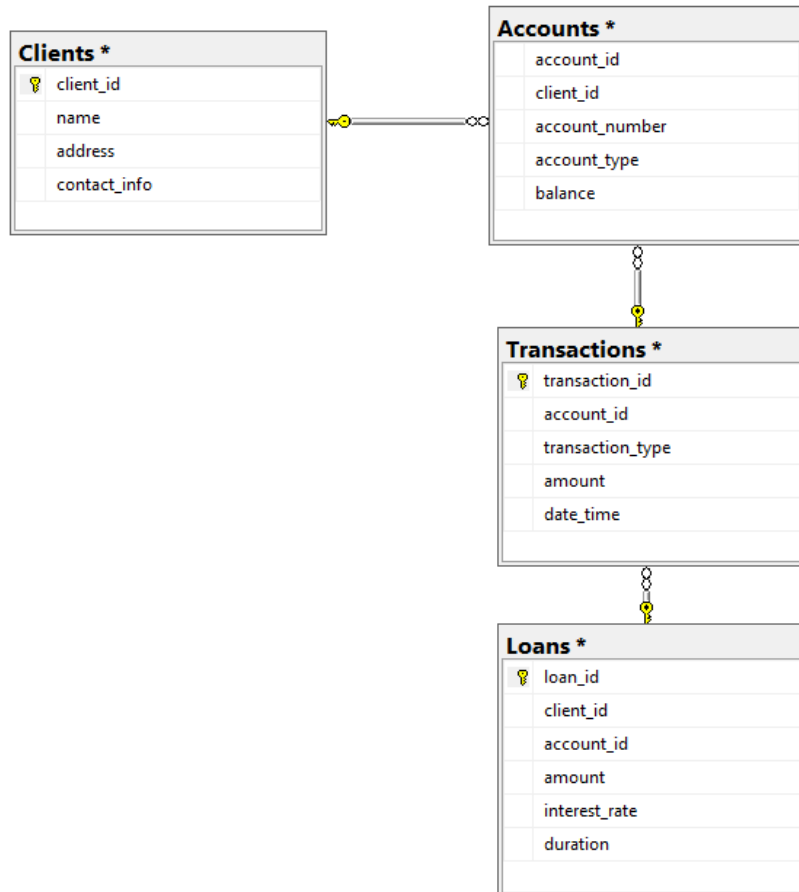


Figure 2.3 – Scheme of the developed database

The following DDL commands were used to create a database in MySQL:

```

-- Create Clients table
CREATE TABLE Clients (
    client_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    contact_info VARCHAR(255)
);

-- Create Accounts table
CREATE TABLE Accounts (
    account_id INT PRIMARY KEY,
    client_id INT,
    account_number VARCHAR(20) UNIQUE,
    account_type VARCHAR(50),
    balance DECIMAL(20, 2),
    FOREIGN KEY (client_id) REFERENCES Clients(client_id)
);
  
```

Figure 2.4 – Codes of the developed database

```

-- Create Transactions table
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(50),
    amount DECIMAL(20, 2),
    date_time DATETIME,
    FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);

-- Create Loans table
CREATE TABLE Loans (
    loan_id INT PRIMARY KEY,
    client_id INT,
    account_id INT,
    amount DECIMAL(20, 2),
    interest_rate DECIMAL(5, 2),
    duration INT,
    FOREIGN KEY (client_id) REFERENCES Clients(client_id),
    FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);

```

Figure 2.5 – Codes of the developed database

## 2.4 Filling the database with initial records

The following commands insert initial data about clients, accounts, loans, and transactions into the created database. (Figure 2.6, 2.7, 2.8, 2.9):

Stores information about clients, including their name, address, and contact details.(Figure 2.6)

```

-- Insert records into Clients table
INSERT INTO Clients (client_id, name, address, contact_info) VALUES
(1, 'John Doe', '123 Main St, Anytown', 'john@example.com'),
(2, 'Jane Smith', '456 Elm St, Othertown', 'jane@example.com');

```

Results		Messages		
	client_id	name	address	contact_info
1	1	John Doe	123 Main St, Anytown	john@example.com
2	2	Jane Smith	456 Elm St, Othertown	jane@example.com

Figure 2.6 – Inserts Records Of Clients

Manages details of accounts held by clients, including account number, type, and balance. (Figure 2.7)

```
-- Insert records into Accounts table
INSERT INTO Accounts (account_id, client_id, account_number, account_type, balance) VALUES
(1, 1, '1234567890', 'Savings', 100.00),
(2, 2, '9876543210', 'Checking', 500.00);
```

account_id	client_id	account_num...	account_type	balance
1	1	1234567890	Savings	100
2	2	9876543210	Checking	500
NULL	NULL	NULL	NULL	NULL

Figure 2.7 – Inserts Records Of Accounts

Tracks information about loans provided to clients, including loan amount, interest rate, and duration. (Figure 2.8)

```
-- Insert records into Loans table
INSERT INTO Loans (loan_id, client_id, account_id, amount, interest_rate, duration) VALUES
(14, 1, 1, 10000.00, 5.0, 24),
(15, 2, 2, 5000.00, 6.0, 12);
```

loan_id	client_id	account_id	amount	interest_rate	duration
14	1	1	10000,00	5,00	24
15	2	2	5000,00	6,00	12
NULL	NULL	NULL	NULL	NULL	NULL

Figure 2.8 – Inserts Records Of Loans

Records all transactions made within the bank, including transaction type, amount, and date/time. (Figure 2.9)

```
-- Insert records into Transactions table
INSERT INTO Transactions (transaction_id, account_id, transaction_type, amount, date_time) VALUES
(12,1, 'Deposit', 500.00, '2024-03-10 09:00:00'),
(13,2, 'Withdrawal', 100.00, '2024-03-11 14:30:00');
```

transaction_id	account_id	transaction_type	amount	date_time
12	1	Deposit	500	2024-03-10 09:0...
13	2	Withdrawal	100	2024-03-11 14:3...
NULL	NULL	NULL	NULL	NULL

Figure 2.9 – Inserts Records Of Transactions

## 2.5 Creating database queries

The following SQL queries have been developed for a Bank Management System. This system manages clients, accounts, transactions, and loans within the bank.

### 2.5.1 Creation of Database Views

#### 1. Create a View to Retrieve Information About Clients and Their Accounts

- This view combines client and account information, allowing easy access to details about which accounts belong to which clients. It includes client IDs, client names, account IDs, account numbers, account types, and balances.

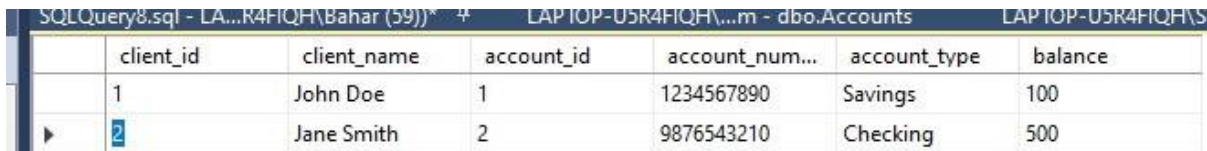
-- View to retrieve information about clients and their accounts

CREATE VIEW ClientAccounts AS

SELECT c.client\_id, c.name AS client\_name, a.account\_id, a.account\_number,  
a.account\_type, a.balance

FROM Clients c

JOIN Accounts a ON c.client\_id = a.client\_id;



client_id	client_name	account_id	account_num...	account_type	balance
1	John Doe	1	1234567890	Savings	100
2	Jane Smith	2	9876543210	Checking	500

Figure 2.10 – Code and result of creating the ClientAccounts view

The result displays client details along with their associated accounts.

## 2. Create a View to Retrieve Information About Transactions Associated with Each Account

- This view simplifies the retrieval of transaction details for each account. It includes account IDs, account numbers, transaction IDs, transaction types, amounts, and timestamps. View to retrieve information about transactions associated with each account



account_id	account_num...	transaction_id	transaction_type	amount	date_time
1	1234567890	12	Deposit	500	2024-03-10 09:0...
2	9876543210	13	Withdrawal	100	2024-03-11 14:3...
NULL	NULL	NULL	NULL	NULL	NULL

Figure 2.11 – Code and result of creating the AccountTransactions view

```
CREATE VIEW AccountTransactions AS
SELECT a.account_id, a.account_number, t.transaction_id, t.transaction_type,
t.amount, t.date_time FROM Accounts a
JOIN Transactions t ON a.account_id = t.account_id;
```

The result shows transactions associated with each account, providing details on transaction types and amounts.

## 3. Create a View to Retrieve Information About Loans Along with Associated Client and Account Details

- This view provides a comprehensive look at loan details, including the loan amount, interest rate, duration, client names, and account numbers associated with each loan.

-- View to retrieve information about loans along with associated client and account details

CREATE VIEW LoanDetails AS

```
SELECT l.loan_id, l.client_id, l.account_id, l.amount AS loan_amount,
l.interest_rate, l.duration,
c.name AS client_name, a.account_number FROM Loans l
```

```
JOIN Clients c ON l.client_id = c.client_id
```

```
JOIN Accounts a ON l.account_id = a.account_id;
```

SQLQuery8.sql - LA...R4FIQH\Bahar (59))*			LAPTOP-U5R4FIQH\S...- dbo.LoanDetails		SQLQuery12.sql - L...R4FIQH\Bahar (62))*			SQLQue
	loan_id	client_id	account_id	loan_amount	interest_rate	duration	client_name	account_num...
▶	14	1	1	10000,00	5,00	24	John Doe	1234567890
	15	2	2	5000,00	6,00	12	Jane Smith	9876543210

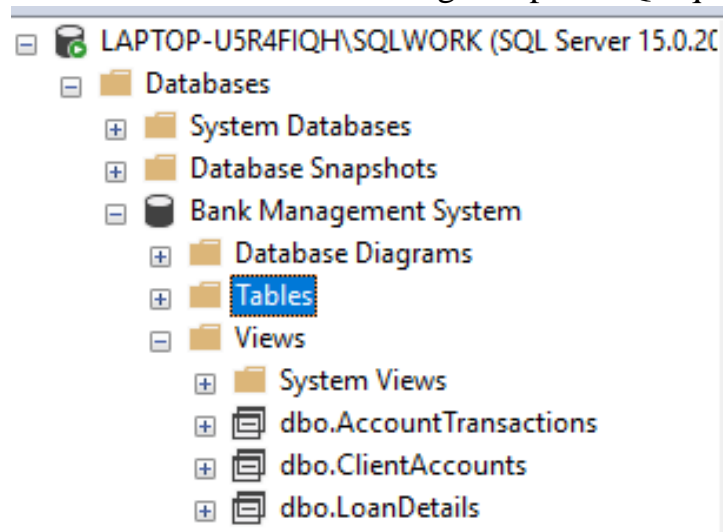
Figure 2.12 – Code and result of creating the LoanDetails view

The result includes detailed information on loans, along with associated client and account details.

## Results of Creating Views and Their Use

After executing the above queries, the views are created in the database. These views provide simplified access to complex data structures, making it easier to

retrieve relevant information without writing complex SQL queries each time.



## 2.5.2 Create stored functions, procedures, and triggers in the database.

The selected domain for this laboratory work is a Bank Management System. This system manages various aspects of banking operations including client information, account management, and transaction handling.

### 1. Create a Function to Calculate the Balance for a Given Account

- This function computes the total balance for a given account ID by summing up all transaction amounts associated with that account.

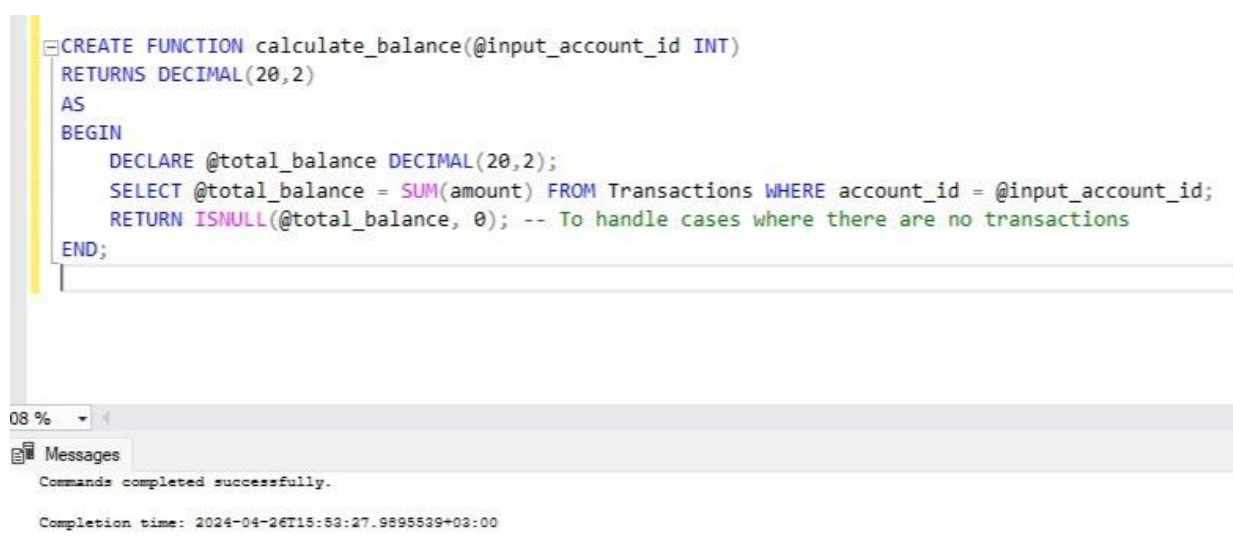




Figure 2.13 – Code and result of creating the calculate\_balance function

The result shows the total balance calculation for a specified account.

## 2. Create a Stored Procedure to Retrieve Transactions by Amount

- This stored procedure retrieves transactions with amounts greater than or equal to a specified minimum amount. It includes error handling to prevent negative input values and returns transaction details meeting the criteria

```
SELECT
  Clients.name AS client_name,
  Clients.address AS client_address,
  Clients.contact_info AS client_contact_info,
  Accounts.account_number,
  Accounts.account_type,
  Accounts.balance,
  Transactions.transaction_type,
  Transactions.amount,
  Transactions.date_time
FROM
  Clients
INNER JOIN
  Accounts ON Clients.client_id = Accounts.client_id
INNER JOIN
  Transactions ON Accounts.account_id = Transactions.account_id
WHERE
  Transactions.amount >= 80;
```

Figure 2.14 – Code of Queries to Create Stored Procedures 1

```

CREATE PROCEDURE GetTransactionsByAmount
@min_amount DECIMAL(20,2)
AS
BEGIN
    IF @min_amount < 0
    BEGIN
        RAISERROR ('Minimum amount cannot be negative.', 16, 1);
        RETURN;
    END;

    SELECT
        Clients.name AS client_name,
        Clients.address AS client_address,
        Clients.contact_info AS client_contact_info,
        Accounts.account_number,
        Accounts.account_type,
        Accounts.balance,
        Transactions.transaction_type,
        Transactions.amount,
        Transactions.date_time
    FROM
        Clients
    INNER JOIN
        Accounts ON Clients.client_id = Accounts.client_id
    INNER JOIN
        Transactions ON Accounts.account_id = Transactions.account_id
    WHERE
        Transactions.amount >= @min_amount;
END;

```

108 %

Messages

Commands completed successfully.

Completion time: 2024-04-26T16:41:56.4562594+03:00

Figure 2.15 – Code of Queries to Create Stored Procedures 2

### Results of Creating Stored Procedures and Their Use:

The stored procedure **GetTransactionsByAmount** is designed to retrieve transactions with amounts greater than or equal to a specified minimum amount. It includes error handling to prevent negative input values and returns transaction details meeting the criteria

```

FROM
    Clients
INNER JOIN
    Accounts ON Clients.client_id = Accounts.client_id
INNER JOIN
    Transactions ON Accounts.account_id = Transactions.account_id
WHERE
    Transactions.amount >= @min_amount;
END;

EXEC GetTransactionsByAmount 80;
EXEC GetTransactionsByAmount -50;

```

108 %

Results Messages

(4 rows affected)

	client_name	client_address	client_contact_info	account_number	account_type	balance	transaction_type	amount	date_time
1	John Doe	123 Main St, Anytown	john@example.com	1234567890	Savings	100	Deposit	500	2024-03-10 09:00:00.000
2	Jane Smith	456 Elm St, Othertown	jane@example.com	9876543210	Checking	500	Withdrawal	100	2024-03-11 14:30:00.000
3	John Doe	123 Main St, Anytown	john@example.com	1234567890	Savings	100	Deposit	500	2024-03-10 09:00:00.000
4	Jane Smith	456 Elm St, Othertown	jane@example.com	9876543210	Checking	500	Withdrawal	100	2024-03-11 14:30:00.000

Figure 2.16 – Results Of Query Procedure Execution

### 3. Create a Trigger to Prevent Negative Account Balances

- This trigger prevents transactions that would result in a negative account balance. It validates transactions before insertion into the Transactions table.

```

CREATE TRIGGER check_balance
ON Transactions
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @current_balance DECIMAL(20, 2);

    -- Get the current balance of the account
    SELECT @current_balance = A.balance
    FROM Accounts A
    INNER JOIN inserted I ON A.account_id = I.account_id;

    -- Check if the transaction will cause the balance to become negative
    IF EXISTS (SELECT 1 FROM inserted I WHERE I.transaction_type = 'debit' AND @current_balance - I.amount < 0)
    BEGIN
        RAISERROR ('Transaction cannot be completed. Insufficient balance.', 16, 1);
        ROLLBACK TRANSACTION;
    END;

    -- Add a condition to handle the case where the account doesn't exist
    IF EXISTS (SELECT 1 FROM inserted I LEFT JOIN Accounts A ON I.account_id = A.account_id WHERE A.account_id IS NULL)
    BEGIN
        RAISERROR ('Transaction cannot be completed. Account does not exist.', 16, 1);
        ROLLBACK TRANSACTION;
    END;
    ELSE
    BEGIN
        INSERT INTO Transactions (account_id, transaction_type, amount)
        SELECT I.account_id, I.transaction_type, I.amount
        FROM inserted I;
    END;
END;

```

107 %

Messages

Commands completed successfully.

Completion time: 2024-04-27T14:26:13.4447030+03:00

Figure 2.17 – Code of Queries to Create Trigger

```

-- Get the current balance of the account
SELECT @current_balance = A.balance
FROM Accounts A
INNER JOIN inserted I ON A.account_id = I.account_id;

-- Check if the transaction will cause the balance to become negative
IF EXISTS (SELECT 1 FROM inserted I WHERE I.transaction_type = 'debit' AND @current_balance - I.amount < 0)
BEGIN
    RAISERROR ('Transaction cannot be completed. Insufficient balance.', 16, 1);
    ROLLBACK TRANSACTION;
END;

-- Add a condition to handle the case where the account doesn't exist
IF EXISTS (SELECT 1 FROM inserted I LEFT JOIN Accounts A ON I.account_id = A.account_id WHERE A.account_id IS NULL)
BEGIN
    RAISERROR ('Transaction cannot be completed. Account does not exist.', 16, 1);
    ROLLBACK TRANSACTION;
END;
ELSE
BEGIN
    INSERT INTO Transactions (account_id, transaction_type, amount)
    SELECT I.account_id, I.transaction_type, I.amount
    FROM inserted I;
END;

```

107 %

Messages

Commands completed successfully.

Completion time: 2024-04-27T14:26:13.4447030+03:00

Figure 2.18 – Code of creating the check\_balance trigger

## Results of Creating Triggers and Their Performance:

The trigger **check\_balance** is implemented to prevent transactions that would result in a negative account balance or involve non-existent accounts. It validates transactions before insertion into the **Transactions** table. An attempt to insert invalid data triggers appropriate error messages.



Figure 2.19 – Results Of Query Trigger Execution

The trigger **check\_balance** is implemented to prevent transactions that would result in a negative account balance or involve non-existent accounts.

### 2.5.3 Working with Transaction Mechanism in the Database

#### 1. Name and Description of the Selected Domain:

- The chosen domain for this laboratory work is a Bank Management System. This system manages various banking operations including account management, transactions, and balances.

#### 2. Queries to Perform Transactions:

- Perform a Transaction to Transfer Funds Between Accounts

```

LAPTOP-U5R4FIQH\S...dbo.Transactions  SQLQuery4_FOR_LA...4FIQH\Bahar (60))*  SQLQuery1_FOR_LA...4FIQH\Bahar (69))

-- Start transaction
BEGIN TRANSACTION;

-- Attempt to transfer $100 from account with ID 1 to account with ID 2
UPDATE Accounts
SET balance = balance - 100
WHERE account_id = 1;

UPDATE Accounts
SET balance = balance + 100
WHERE account_id = 2;

-- Check if both updates were successful
IF (SELECT COUNT(*) FROM Accounts WHERE account_id = 1 AND balance >= 0) = 1
AND (SELECT COUNT(*) FROM Accounts WHERE account_id = 2) = 1
BEGIN
    -- If successful, insert a record into the Transactions table
    INSERT INTO Transactions (account_id, transaction_type, amount)
    VALUES (1, 'Deposit', 100),
            (2, 'Withdrawal', 100);

    -- Commit the transaction
    COMMIT TRANSACTION;

    -- Output success message
    PRINT 'Transaction completed successfully.';
END
ELSE
BEGIN
    -- If any update failed or balances are invalid, rollback the transaction
    ROLLBACK TRANSACTION;

```

Figure 2.20 – Code and result of performing the transfer transaction 1



```

LAPTOP-U5R4FIQH\S...dbo.Transactions  SQLQuery4_FOR_LA...4FIQH\Bahar (60))  SQLQuery1_FOR_LA...4FIQH\Bahar (69))

SET balance = balance + 100
WHERE account_id = 2;

-- Check if both updates were successful
IF (SELECT COUNT(*) FROM Accounts WHERE account_id = 1 AND balance >= 0) = 1
AND (SELECT COUNT(*) FROM Accounts WHERE account_id = 2) = 1
BEGIN
    -- If successful, insert a record into the Transactions table
    INSERT INTO Transactions (account_id, transaction_type, amount)
    VALUES (1, 'Deposit', 100),
            (2, 'Withdrawal', 100);

    -- Commit the transaction
    COMMIT TRANSACTION;

    -- Output success message
    PRINT 'Transaction completed successfully.';
END
ELSE
BEGIN
    -- If any update failed or balances are invalid, rollback the transaction
    ROLLBACK TRANSACTION;

    -- Output error message
    PRINT 'Transaction failed. Insufficient balance or invalid account.';
END;

```

107 %

Messages

```

(1 row affected)
(1 row affected)
(2 rows affected)
(2 rows affected)
Transaction completed successfully.
Completion time: 2024-04-27T17:33:40.5180415+03:00

```

Figure 2.21 – Code and result of performing the transfer transaction 2

## Table Contents Before, During, and After Execution of Transactions:

### 1. Before Transaction:

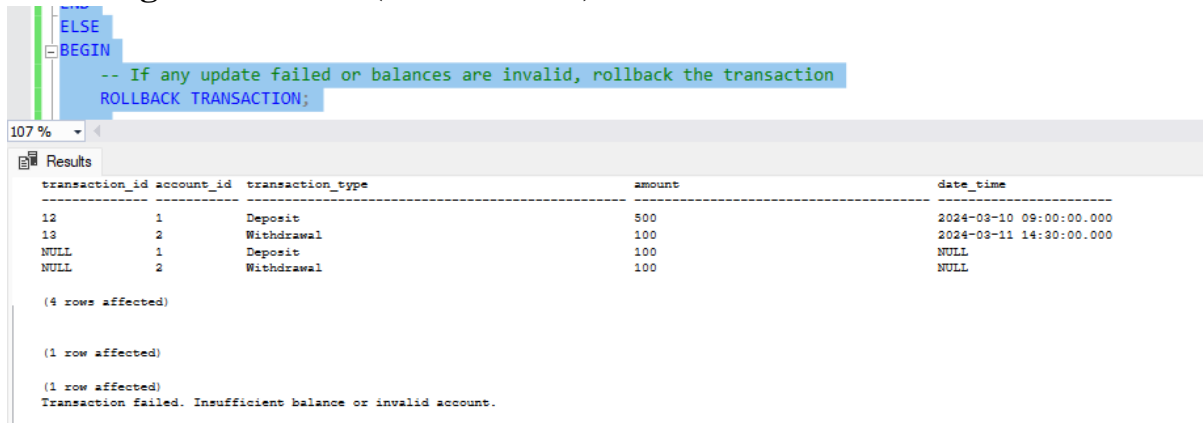
	account_id	client_id	account_num...	account_type	balance
	1	1	1234567890	Savings	100
	2	2	9876543210	Checking	500
▶▶	NULL	NULL	NULL	NULL	NULL

	transaction_id	account_id	transaction_type	amount	date_time
▶	12	1	Deposit	500	2024-03-10 09:0...
	13	2	Withdrawal	100	2024-03-11 14:3...

Figure 2.22 – Account balances before the transaction

*The account balances before the transaction is executed.*

## 2. During Transaction (If Successful):



```

ELSE
BEGIN
-- If any update failed or balances are invalid, rollback the transaction
ROLLBACK TRANSACTION;

```

transaction_id	account_id	transaction_type	amount	date_time
12	1	Deposit	500	2024-03-10 09:00:00.000
13	2	Withdrawal	100	2024-03-11 14:30:00.000
NULL	1	Deposit	100	NULL
NULL	2	Withdrawal	100	NULL

(4 rows affected)

(1 row affected)

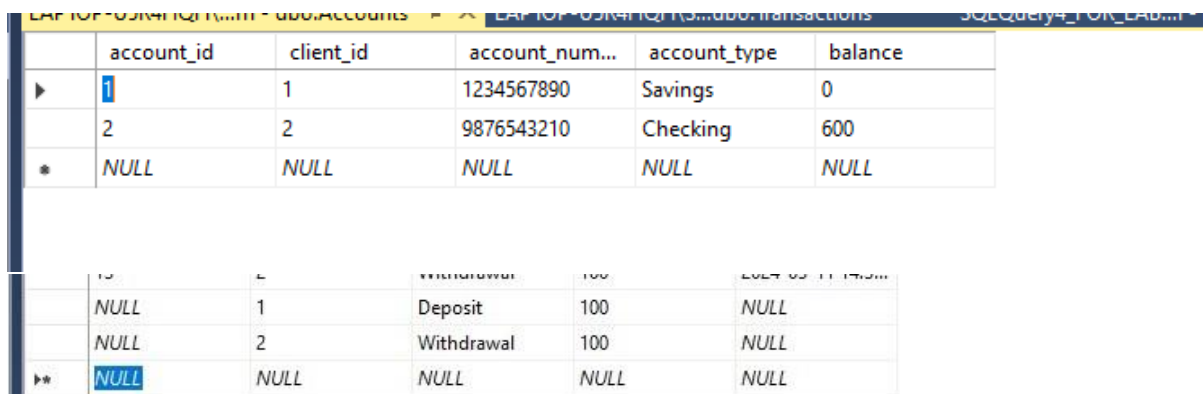
(1 row affected)

Transaction failed. Insufficient balance or invalid account.

Figure 2.23 – Account balances during the transaction

*The account balances are being updated during the transaction.*

## 3 . After Transaction (If Successful):



account_id	client_id	account_num...	account_type	balance
1	1	1234567890	Savings	0
2	2	9876543210	Checking	600
NULL	NULL	NULL	NULL	NULL

transaction_id	account_id	transaction_type	amount	date_time
NULL	1	Deposit	100	NULL
NULL	2	Withdrawal	100	NULL
NULL	NULL	NULL	NULL	NULL

Figure 2.24 – Account balances after the transaction

*The account balances after the transaction is completed.*



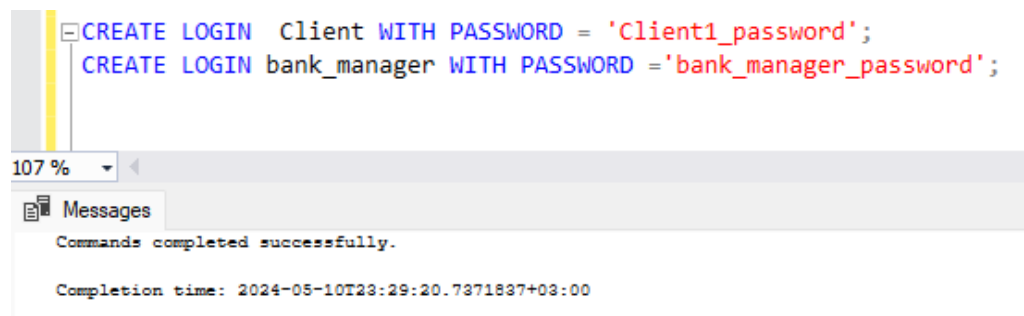
## 2.5.4 Setting Up User Rights and Mechanisms for Ensuring the Integrity of the Database

### 1. Domain Description:

- The selected domain for this laboratory work is a Bank Management System. This system manages various aspects of banking operations such as client accounts, loans, transactions, and administrative tasks. It involves multiple user roles with different privileges to ensure secure and efficient operation of the system.

### 2. Queries to Create Login Accounts and Assign Privileges:

- Create Login Accounts and Assign Privileges



```

CREATE LOGIN Client WITH PASSWORD = 'Client1_password';
CREATE LOGIN bank_manager WITH PASSWORD = 'bank_manager_password';

```

107 %

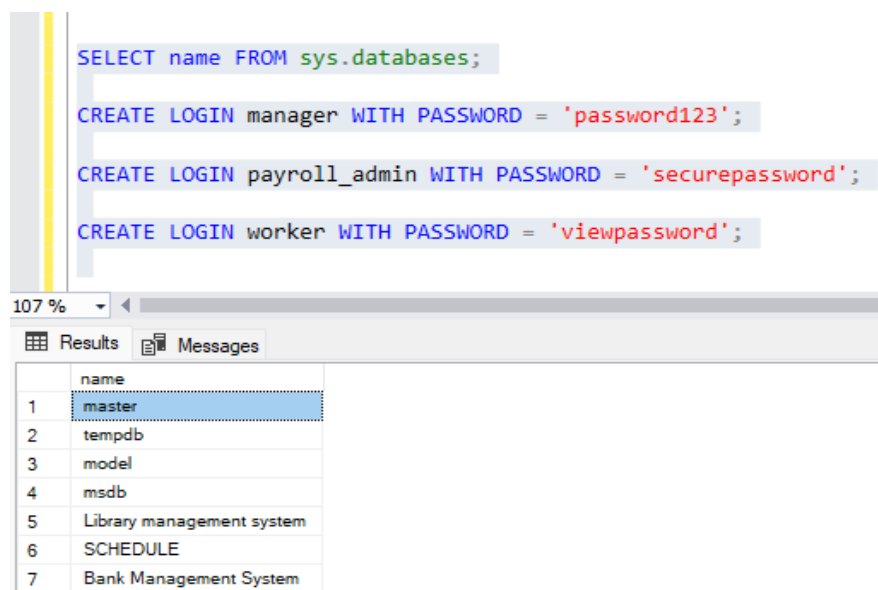
Messages

Commands completed successfully.

Completion time: 2024-05-10T23:29:20.7371837+03:00

Figure 2.25 – Code of creating Login accounts and assigning privileges

### 3. Results of Account Creation and Privileges:



```

SELECT name FROM sys.databases;
CREATE LOGIN manager WITH PASSWORD = 'password123';
CREATE LOGIN payroll_admin WITH PASSWORD = 'securepassword';
CREATE LOGIN worker WITH PASSWORD = 'viewpassword';

```

107 %

Results Messages

	name
1	master
2	tempdb
3	model
4	msdb
5	Library management system
6	SCHEDULE
7	Bank Management System

Figure 2.26 – Result of Login account creation and privilege assignment

```
EXEC sp_addrolemember 'db_datareader', Client;
EXEC sp_addrolemember 'db_datawriter', bank_manager;

-- Check the existing constraints on the Accounts table
EXEC sp_helpconstraint Accounts;
```

117 %

Messages

Commands completed successfully.

Completion time: 2024-05-12T16:45:26.5381228+03:00

Figure 2.27 – Result of Login account creation and privilege assignment

The result confirms the successful creation of user accounts and assignment of privileges.

After executing the above queries, the specified user accounts are created successfully, and the necessary privileges are assigned accordingly. The user roles `Client`, `bank\_manager`, and `worker\_view` are granted appropriate permissions to interact with relevant tables in the Bank Management System.

```
SQLQuery_FOR_LAB_...4FIQH\Bahar (52)) * X
SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'dbo' AND TABLE_NAME IN ('Accounts', 'Loans', 'Transactions');

GRANT SELECT, UPDATE ON dbo.Accounts TO worker_view ;
GRANT SELECT, INSERT, UPDATE ON dbo.Loans TO bank_manager;
GRANT SELECT, INSERT, UPDATE ON dbo.Transactions TO admin;

-- Check privileges of 'bank_manager'
SHOW GRANTS FOR 'bank_manager'@'localhost';
```

117 %

Results Messages

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	Bank Management System	dbo	Accounts	BASE TABLE
2	Bank Management System	dbo	Loans	BASE TABLE
3	Bank Management System	dbo	Transactions	BASE TABLE

Figure 2.28 – Code of modifying foreign key constraints

#### 4. Requests to Modify Referential Integrity Mechanisms:

```

SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'dbo' AND TABLE_NAME IN ('Accounts', 'Loans', 'Transactions');

-- Adjusting an existing foreign key mechanism
ALTER TABLE Loans
DROP CONSTRAINT FK__Loans__account_i__6383C8BA;

ALTER TABLE Loans
ADD FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

```

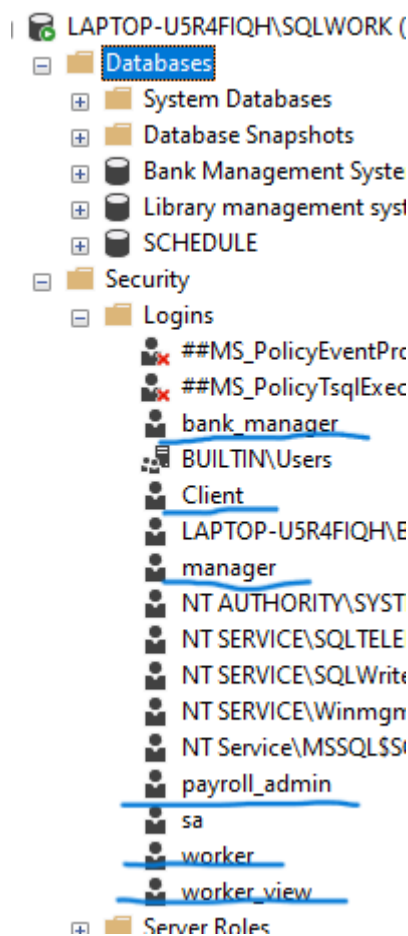
117 %

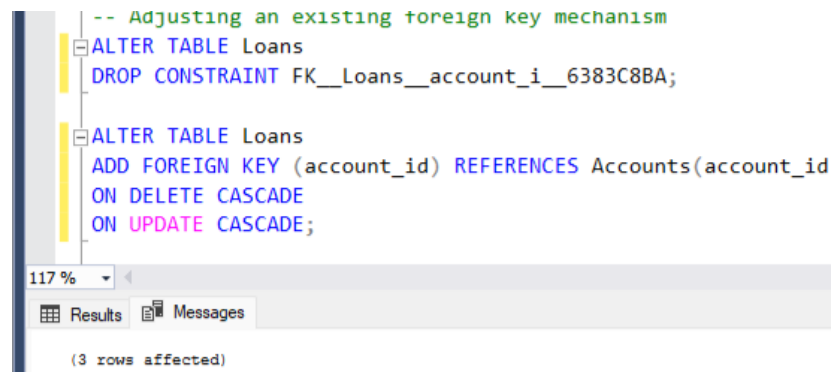
Results Messages

(3 rows affected)

## 5. Confirmation of Referential Integrity Mechanisms:

The referential integrity mechanisms have been successfully modified as per the requests. The foreign key constraint between the **Loans** table and the **Accounts** table has been adjusted to ensure data consistency and integrity.





```
-- Adjusting an existing foreign key mechanism
ALTER TABLE Loans
DROP CONSTRAINT FK__Loans__account_i__6383C8BA;

ALTER TABLE Loans
ADD FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

117 %

Results Messages

(3 rows affected)

Figure 2.29 – Confirmation of referential integrity mechanisms

### 3 DESIGN AND DEVELOPMENT OF THE DATABASE APPLICATION

#### 3.1 Database application design

The use case diagram of the database application is shown in Figure 3.1.

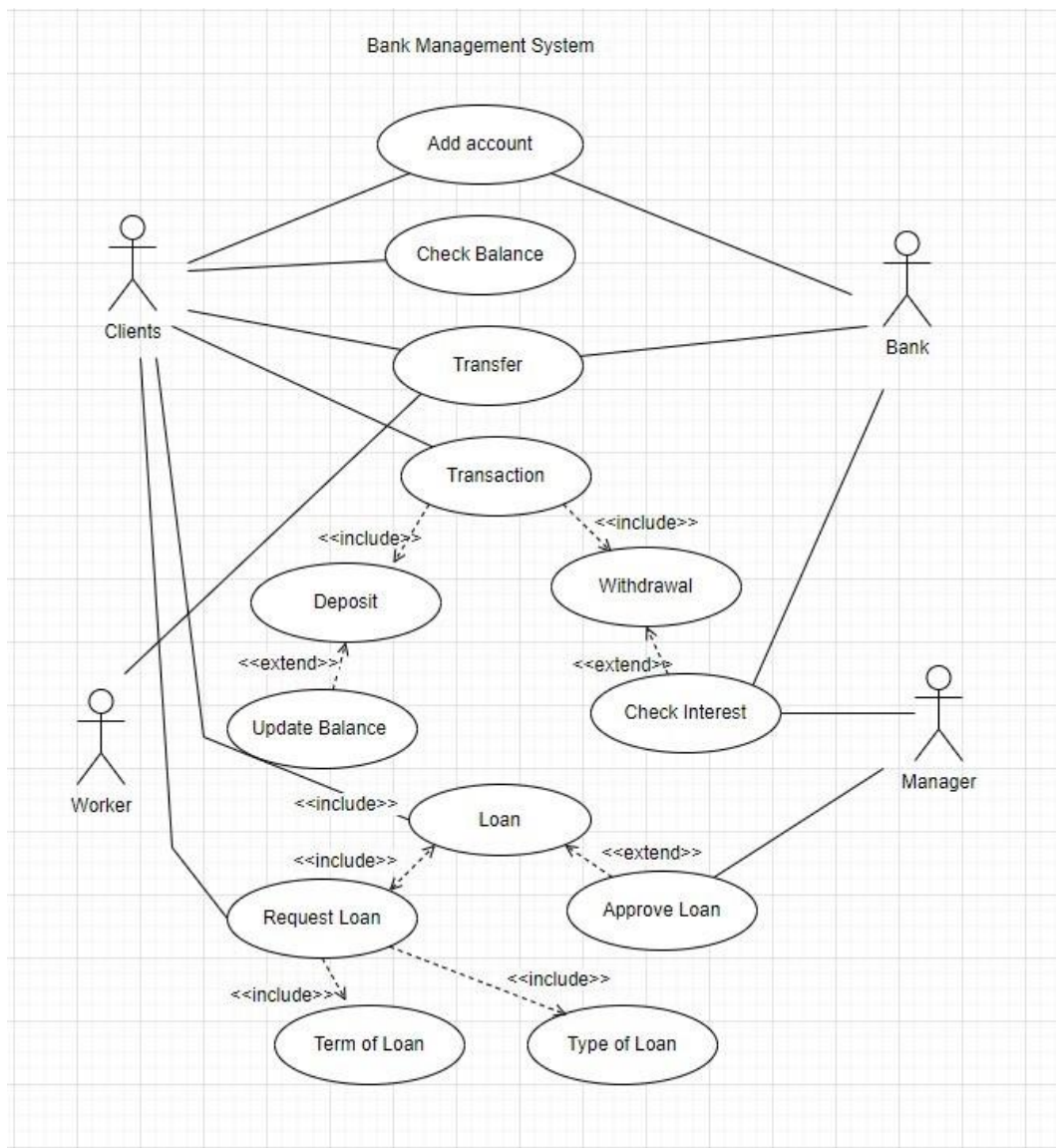


Figure 3.1 – The use case diagram

The deployment diagram, depicted in Figure 3.2, provides insights into the physical deployment of the database application. It illustrates the distribution of

components across different nodes or hardware devices, elucidating how the system architecture ensures scalability, reliability, and performance.

The deployment diagram of the database application is shown in Figure 3.2.

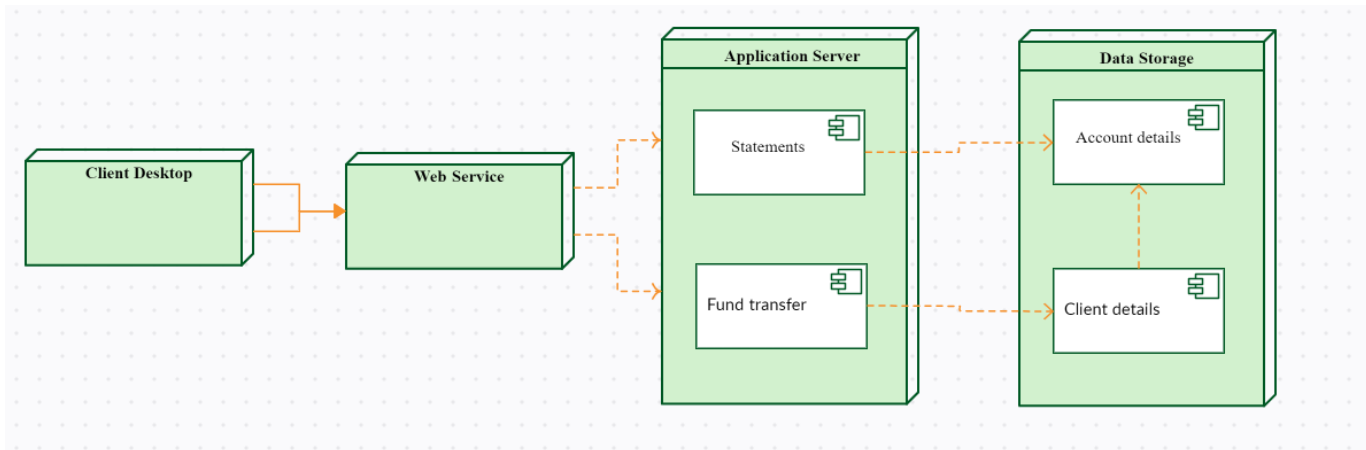


Figure 3.2 – The deployment diagram

### 3.2 Database application usage example

Examples of using the application to work with the database are shown in Figures 3.3 and 3.4.

Figure 3.3 displays a form designed for interacting with the client ATM platform. This interface enables users to perform various operations related to client management, such as account creation, balance inquiries, and transaction history retrieval.

Figure 3.3 – Form for working with client atm platform

Figure 3.4 exhibits a form tailored for managing delivered goods within the system. This interface empowers users to track and manage the delivery process efficiently, facilitating tasks such as order processing, inventory management, and shipment tracking.

Figure 3.4 – Form for working with delivered goods

These examples highlight the intuitive design and practical functionality of the database application, underscoring its role in streamlining operations, enhancing productivity, and fostering seamless interaction with the underlying database system.

## CONCLUSIONS

During the course work, a comprehensive study and development of a database system tailored for a commercial enterprise that procures goods from various suppliers were undertaken. The aim was to design a database that effectively handles the storage and processing of product supply information, adhering to specific business rules. The following key tasks were completed to achieve this objective:

1. **Development of a Logical Model:** The course work began with developing a logical model of the database, utilizing the Entity-Relationship (ER) diagram. This model identified the key entities involved—Clients, Accounts, Transactions, and Loans—and defined the relationships and attributes associated with each entity.
2. **Creation of a Physical Model:** Building on the logical model, a physical model was developed to outline the actual structure and storage mechanisms of the database. This model ensured that the database could be efficiently implemented and maintained in a real-world setting.
3. **Description of Database Structure:** A detailed description of the database structure was provided, specifying the tables, fields, data types, and constraints. This description serves as a blueprint for the database design and implementation.



4. **Implementation in MySQL:** The database was implemented using MySQL, employing Data Definition Language (DDL) commands. This step involved creating tables and defining relationships, constraints, and indexes to ensure data integrity and performance.
5. **Initial Data Population and Query Development:** The database was populated with initial records for clients, accounts, loans, and transactions. Additionally, SQL queries were developed to perform various operations on the database, including stored procedures and triggers to automate tasks and enforce business rules.
6. **Development of a Database Application:** To facilitate interaction with the database, a database application was designed and developed. This application includes forms for managing client information, accounts, loans, and transactions, providing an intuitive interface for users.
7. **Demonstration of Application Usage:** The usage of the database application was demonstrated through examples, showcasing how it can be used to manage and retrieve data efficiently. This included form interfaces for entering and viewing data, as well as executing queries and procedures.

Throughout the course work, the complexities of managing product supply information within a commercial enterprise were addressed. The developed database system enhances the productivity and accuracy of storing and processing such information. By following a systematic approach to database design, development, and implementation, the final product meets the specified requirements and provides a robust solution for the enterprise.

This course work not only achieved its objectives but also provided valuable insights into the practical aspects of database management and application development, laying a strong foundation for future projects in this domain.

## REFERENCES

- 1 Halpin T., Morgan T. Information modeling and relational databases. - Morgan Kaufmann, 2010.
- 2 Date CJ An Introduction to Database Systems. - Pearson Education India, 2006.
- 3 Tahaghoghi SMM, Williams HE Learning MySQL: Get a Handle on Your Data. - O'Reilly Media, Inc., 2006.
- 4 Murach J. Murach's MySQL. - Mike Murach & Associates, Incorporated, 2015.
- 5 DuBois P. MySQL Cookbook: Solutions for Database Developers and Administrators. - "O'Reilly Media, Inc.", 2014.
- 6 Nixon R. Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites. - "O'Reilly Media, Inc.", 2012.
- 7 Boronczyk T. et al. Beginning PHP 6, Apache, MySQL 6 Web Development. - Wrox Press Ltd., 2009.
- 8 Schwartz B., Zaitsev P., Tkachenko V. High performance MySQL: optimization, backups, and replication. - "O'Reilly Media, Inc.", 2012.
- 9 Taylor AG SQL for Dummies. - John Wiley & Sons, 2011.
- 10 Becker J., Kugeler M., Rosemann M. (ed.). Process management: a guide for the design of business processes. - Springer Science & Business Media, 2013.
- 11 Jukic N., Vrbsky S., Nestorov S. Database systems: Introduction to databases and data warehouses. - Pearson, 2014. - P. 400.