Gisma University of Applied Sciences

Assessment Submission Form

Student Number (If this is group work, please include the student numbers of all group participants)	GH1039032 Yatik Bhadreshbhai Anghan GH1037834 Ayush Jashvantbhai Pandav
Assessment Title	Bank Management System
Module Code	M605A
Module Title	Advanced Database
Module Tutor	Dr. Mazhar Hameed
Date Submitted	27-March-2025
Github	https://github.com/Ay1932/Advance-database-Project
Youtube	https://youtu.be/er2fpFLtiqc

Data Query for Help

Customer Login

Customers :

Status : Active

1	Username	yatik@gmail.com
	Password	password123

Status : Active

2	Username	mayank@gmail.com
	Password	password123

Status : Pending

3	Username	ayush@gmail.com
	Password	password123

Status : Deactive

4	Username	bansi@gmail.com
	Password	password123

Status : Closed

5	Username	pavan@gmail.com
	Password	password123

Admins :

Role : Admin

1	Username	admin@gmail.com
	Password	admin123

Role : Casher

2	Username	mayank@gmail.com
	Password	admin123

Role : Support

3	Username	ayush@gmail.com
	Password	admin123

1. Abstract

The Banking Management System is a hybrid database app that integrate MySQL and MongoDB to manage structure and semi-structure banking data. This system is design to handle user accounts, transactions and banking operations while make sure the performance and scale of banking system.

The SQL part is use for structure data like customer details, account information, transaction with data integrity and compliance. The NoSQL part is utilize for semi structure data like for admin to-do list and much more.

The key feature of system include security, CRUD operations, transactional queries and integration of SQL and NoSQL databases. The system has been designed to optimized index and modeling technique and structured query execution. Performance and reliable have been made through sample data stimulation and time query execution test.

This project replicate the practical application of banking management with real world applications, with limitations of traditional database while the give advantage of NoSQL databases. Future improvements may be with machine learning based fraud detection and much more improvement on transaction.

2. Introduction

The Bank Management System is a solution designed to efficiently manage banking operation includes accounts, transactions and authentication. With the increase volume of data and the need for real time processing, traditional SQL alone face limitation in handling data. To overcome this, the project implements a hybrid database approach, integrates MySQL for structure data and MongoDB for semi structure data.

Objective

The initial goal of project is develop a scalable, secure and efficient banking management. The system aims to:

- Ensure secure and reliable transaction management using MySQL.
- Handle semi-structured data like to-do list and activity logs using MongoDB.
- Optimize query execution and data retrieval for better performance.
- Demonstrate seamless integration between SQL and NoSQL databases.

Scope of Project

This project covers key functionality like account creation, deposits, withdrawals, transactions and authentication. It also covers MongoDB for log management and to-do list for admin, improving data flexibility and performance. Although this project focuses on backend database design and implementation, in future may include web base user interface and machine learning based fraud detection mechanisms.

3. System Design

The Bank Management System follows hybrid database architecture, integration MySQL for structure data and MongoDB for handling semi-structure information.

3.1 SQL Database design(MySQL)

The SQL Database is responsible for storing structured baking data, ensure ACID compliance and maintain data consistent.

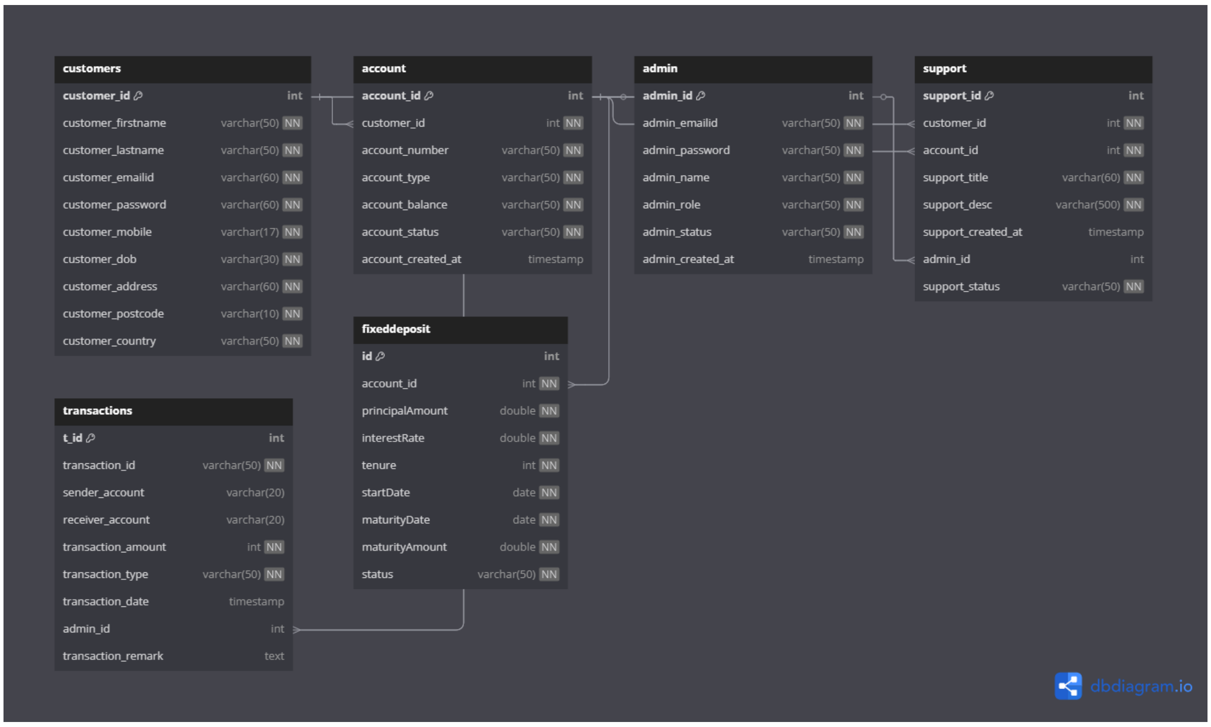
Key Tables and Schema :

1. **Customers table** – store personal data of banking customers.
2. **Accounts table** – maintain different type of accounts.
3. **Admin table** – Store admin credential and access roles.
4. **Support table** – tracks customer queries and support request.
5. **Transactions table** – Record transaction between accounts.
6. **Fixed deposit table** – Manages long-term saving accounts with interest calculations.

Entity-Relationship(ER) Diagram Overview

The system maintains relationships with different entities.

- Each customer can have multiple accounts.
- Each account can have multiple transactions.
- Admins oversee transactions and customer support requests.
- Fixed deposit are linked to specific accounts.



customers		
customer_id	int	
customer_firstname	varchar(50)	NN
customer_lastname	varchar(50)	NN
customer_emailid	varchar(60)	NN
customer_password	varchar(60)	NN
customer_mobile	varchar(17)	NN
customer_dob	varchar(30)	NN
customer_address	varchar(60)	NN
customer_postcode	varchar(10)	NN
customer_country	varchar(50)	NN

account		
account_id	int	
customer_id	int	NN
account_number	varchar(50)	NN
account_type	varchar(50)	NN
account_balance	varchar(50)	NN
account_status	varchar(50)	NN
account_created_at	timestamp	

admin		
admin_id	int	
admin_emailid	varchar(50)	NN
admin_password	varchar(50)	NN
admin_name	varchar(50)	NN
admin_role	varchar(50)	NN
admin_status	varchar(50)	NN
admin_created_at	timestamp	

support		
support_id	int	
customer_id	int	NN
account_id	int	NN
support_title	varchar(60)	NN
support_desc	varchar(500)	NN
support_created_at	timestamp	
admin_id	int	
support_status	varchar(50)	NN

fixeddeposit		
id	int	
account_id	int	NN
principalAmount	double	NN
interestRate	double	NN
tenure	int	NN
startDate	date	NN
maturityDate	date	NN
maturityAmount	double	NN
status	varchar(50)	NN

transactions		
t_id	int	
transaction_id	varchar(50)	NN
sender_account	varchar(20)	
receiver_account	varchar(20)	
transaction_amount	int	NN
transaction_type	varchar(50)	NN
transaction_date	timestamp	
admin_id	int	
transaction_remark	text	

Normalization & Indexing

- The database follows 3 rd Normal Form to eliminate redundancy.
- Primary keys and foreign keys maintain referential integrity.
- Indexes are used on frequently fields like customer_email, account_number, and transaction_id to optimize performance.

3.2 NoSQL database design

MongoDB is used to store semi-structure and unstructured data, providing flexibility and scalability.

Use Cases for NoSQL

- User activity logs
- Admin to-do lists
- Embeded documents store related information efficiently.
- Indexes can be created on customer_id for fast lookups.

3.3 SQL & NoSQL Integration

For smoothly data interaction between MySQL and MongoDB, the system follows this strategies.

- Data synchronization : Key financial data remain in MySQL, while logs and analytics data in MongoDB.
- Performance Optimization : Often accessed structured data stays in MySQL, while relational data store in MongoDB.

This approach make data consistency strong while gain scalability and performance optimization for banking operations.

4. Implementation

The Bank Management System is implemented using MySQL and MongoDB for structure and semi-structure databases respectively. In this section we will cover the database creation, table definition and NoSQL integration, and query execution.

4.1 SQL Database implementation(MySQL)

The SQL component of the system manages customer accounts, transactions and admin operations.

4.1.1 Creating the MySQL Database

```
create schema bankingmanagement;
use bankingmanagement;
```

4.1.2 Creating tables

Customers table :

```
create table customers(
customer_id int auto_increment primary key,
customer_firstname varchar(50) not null,
customer_lastname varchar(50) not null,
customer_emailid varchar(60) unique not null,
customer_password varchar(60) not null,
customer_mobile varchar(17) not null,
customer_dob varchar(30) not null,
customer_address varchar(60) not null,
customer_postcode varchar(10) not null,
customer_country varchar(50) not null
);
```

Accounts table :

```
create table account(
account_id int auto_increment primary key,
customer_id int,
account_number varchar(50) unique not null,
account_type varchar(50) not null,
account_balance varchar(50) not null,
account_status varchar(50) not null,
account_created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id));
```

Admin table :

```
create table admin(
admin_id int auto_increment primary key,
admin_emailid varchar(50) unique not null,
admin_password varchar(50) not null,
admin_name varchar(50) not null,
admin_role varchar(50) not null,
admin_status varchar(50) not null,
admin_created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Support table :

```
create table support(
support_id int auto_increment primary key,
customer_id int,
account_id int,
support_title varchar(60) not null,
support_desc varchar(500) not null,
support_created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
admin_id int,
support_status varchar(50) not null,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
FOREIGN KEY (account_id) REFERENCES account(account_id),
FOREIGN KEY (admin_id) REFERENCES admin(admin_id)
);
```

Transaction table :

```
CREATE TABLE transactions (
    t_id int AUTO_INCREMENT PRIMARY KEY,
    transaction_id VARCHAR(50) NOT NULL UNIQUE,
    sender_account VARCHAR(20) default "NA",
    receiver_account VARCHAR(20) default "NA",
    transaction_amount int NOT NULL,
    transaction_type varchar(50) NOT NULL,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    admin_id int,
    transaction_remark TEXT,
    FOREIGN KEY (admin_id) REFERENCES admin(admin_id)
);
```

Fixed Deposit table :

```
CREATE TABLE fixeddeposit (
    id int auto_increment PRIMARY KEY,
    account_id INT NOT NULL,
    principalAmount DOUBLE NOT NULL,
    interestRate DOUBLE NOT NULL,
    tenure INT NOT NULL,
    startDate DATE NOT NULL,
    maturityDate DATE NOT NULL,
    maturityAmount DOUBLE NOT NULL,
    status VARCHAR(50) NOT NULL,
    FOREIGN KEY (account_id) REFERENCES account(account_id)
);
```

4.1.2 Operation Queries

Customer Table

```
INSERT INTO customers (customer_firstname, customer_lastname,
customer_emailid, customer_password, customer_mobile, customer_dob,
customer_address, customer_postcode, customer_country)
VALUES ('Yatik', 'Anghan', 'yatik@gmail.com', 'password123', '+4917663116950',
'2003-08-08', '09 Bornimer St, Berlin', '10711', 'Germany');
```

```
INSERT INTO customers (customer_firstname, customer_lastname,
customer_emailid, customer_password, customer_mobile, customer_dob,
customer_address, customer_postcode, customer_country)
VALUES ('Mayank', 'Dobariya', 'mayank@gmail.com', 'password123',
'+4917663116951', '2001-05-08', '456 Park Ave, Munich', '80331', 'Germany');

INSERT INTO customers (customer_firstname, customer_lastname,
customer_emailid, customer_password, customer_mobile, customer_dob,
customer_address, customer_postcode, customer_country)
VALUES ('Ayush', 'Pandav', 'ayush@gmail.com', 'password123', '+4917663116952',
'2003-08-07', '789 Elm St, Hamburg', '20095', 'Germany');

INSERT INTO customers (customer_firstname, customer_lastname,
customer_emailid, customer_password, customer_mobile, customer_dob,
customer_address, customer_postcode, customer_country)
VALUES ('Bansi', 'Sonani', 'bansi@gmail.com', 'password123', '+4917663116953',
'2001-03-25', '321 Birch St, Frankfurt', '60311', 'Germany');

INSERT INTO customers (customer_firstname, customer_lastname,
customer_emailid, customer_password, customer_mobile, customer_dob,
customer_address, customer_postcode, customer_country)
VALUES ('Pavan', 'Goti', 'pavan@gmail.com', 'password123', '+4917663116954',
'2002-01-12', '987 Oak St, Stuttgart', '70173', 'Germany');

UPDATE customers
SET customer_mobile = '+4917663116999'
WHERE customer_emailid = 'yatik@gmail.com';

select * from customers;
```

Account Table

```
INSERT INTO account (customer_id, account_number, account_type,
account_balance, account_status)
VALUES (1, 'DE1000000001', 'Savings', '5000', 'Active');

INSERT INTO account (customer_id, account_number, account_type,
account_balance, account_status)
VALUES (2, 'DE1000000002', 'Current', '10000', 'Active');

INSERT INTO account (customer_id, account_number, account_type,
account_balance, account_status)
VALUES (3, 'DE1000000003', 'Savings', '7500', 'Deactive');

INSERT INTO account (customer_id, account_number, account_type,
account_balance, account_status)
```

```
VALUES (4, 'DE1000000004', 'Current', '2000', 'Pending');

INSERT INTO account (customer_id, account_number, account_type,
account_balance, account_status)
VALUES (5, 'DE1000000005', 'Savings', '15000', 'Closed');

UPDATE account
SET account_status = 'Active'
WHERE account_number = 'DE1000000003';
```

Admin Table

```
Insert into admin
(admin_emailid,admin_password,admin_name,admin_role,admin_status)
values("admin@gmail.com","admin123","Yatik Anghan","Admin","Active");
Insert into admin
(admin_emailid,admin_password,admin_name,admin_role,admin_status)
values("mayank@gmail.com","admin123","Mayank Dobariya","Casher","Active");
Insert into admin
(admin_emailid,admin_password,admin_name,admin_role,admin_status)
values("ayush@gmail.com","admin123","Ayush Pandav","Support","Active");
```

Support Table

```
INSERT INTO support (customer_id, account_id, support_title, support_desc,
support_status)
VALUES (1, 1, "Login Issue", "Customer is unable to log in to their account.",
"Pending");

INSERT INTO support (customer_id, account_id, support_title, support_desc,
support_status)
VALUES (2, 2, "Transaction Failure", "A recent transaction failed, and the
amount was deducted.", "Resolved");

INSERT INTO support (customer_id, account_id, support_title, support_desc,
support_status)
VALUES (3, 3, "Account Reactivation", "Request to reactivate a deactivated
savings account.", "Pending");
```

```sql
INSERT INTO support (customer_id, account_id, support_title, support_desc,
support_status)
VALUES (4, 4, "Wrong Account Details", "The account details displayed are
incorrect.", "Resolved");

UPDATE support
SET support_status = 'Resolved'
WHERE support_id = 3;
```

Transaction Table

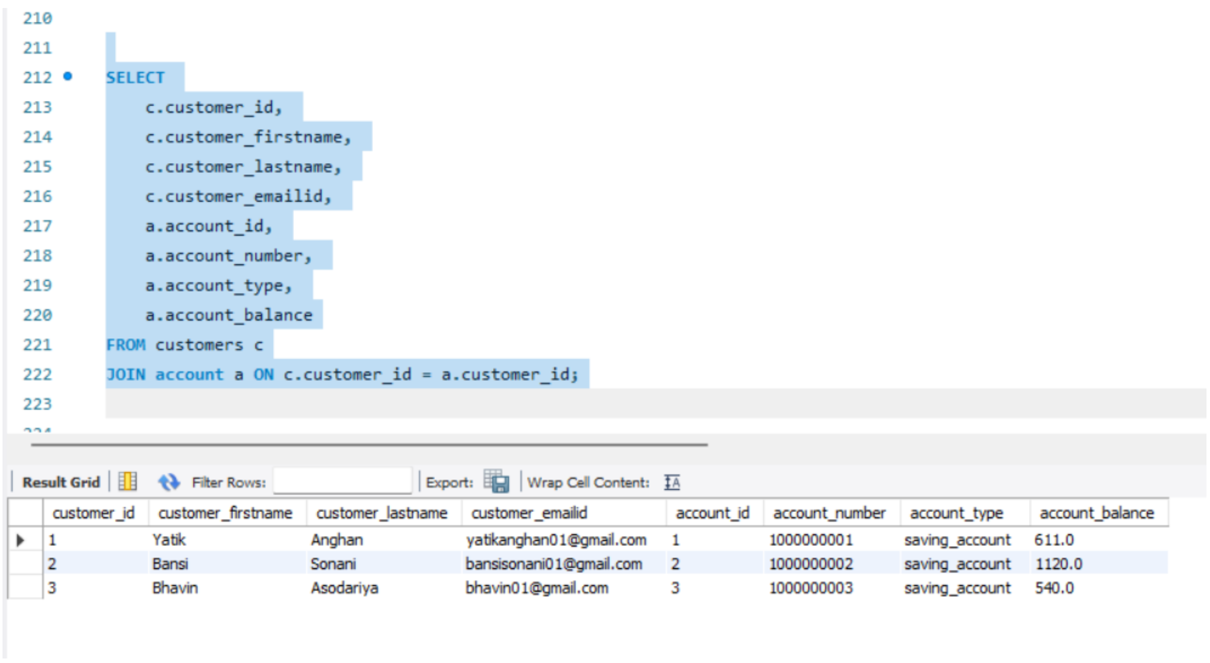
```sql
INSERT INTO transactions (transaction_id, sender_account, receiver_account,
transaction_amount, transaction_type, transaction_remark)
VALUES ('TXN10001', 'DE1000000001', 'DE1000000002', 1500, 'Transfer', 'Fund
transfer to Mayank Dobariya');

INSERT INTO transactions (transaction_id, sender_account, receiver_account,
transaction_amount, transaction_type, transaction_remark)
VALUES ('TXN10002', 'DE1000000002', 'DE1000000003', 2000, 'Transfer', 'Payment
for services');

INSERT INTO transactions (transaction_id, sender_account, receiver_account,
transaction_amount, transaction_type, transaction_remark)
VALUES ('TXN10003', 'DE1000000003', 'DE1000000004', 500, 'Transfer', 'Bill
payment');

INSERT INTO transactions (transaction_id, sender_account, receiver_account,
transaction_amount, transaction_type, transaction_remark)
VALUES ('TXN10004', 'DE1000000004', 'DE1000000005', 10000, 'Deposit', 'Fixed
deposit investment');

INSERT INTO transactions (transaction_id, sender_account, receiver_account,
transaction_amount, transaction_type, transaction_remark)
VALUES ('TXN10005', 'DE1000000005', 'DE1000000001', 3000, 'Withdrawal', 'Cash
withdrawal from savings account');

UPDATE transactions
SET transaction_remark = 'payment delayed'
WHERE transaction_id = 'TXN10002';


DELETE FROM transactions
WHERE transaction_id = 'TXN10005';
```

Fixed Deposit Table

```
INSERT INTO fixeddeposit (account_id, principalAmount, interestRate, tenure,
startDate, maturityDate, maturityAmount, status)
VALUES (1, 5000, 5.5, 12, '2024-03-01', '2025-03-01', 5300, 'Active');

INSERT INTO fixeddeposit (account_id, principalAmount, interestRate, tenure,
startDate, maturityDate, maturityAmount, status)
VALUES (2, 10000, 6.0, 24, '2023-06-15', '2025-06-15', 11200, 'Active');

INSERT INTO fixeddeposit (account_id, principalAmount, interestRate, tenure,
startDate, maturityDate, maturityAmount, status)
VALUES (3, 7500, 5.8, 18, '2024-01-10', '2025-07-10', 7950, 'Active');

INSERT INTO fixeddeposit (account_id, principalAmount, interestRate, tenure,
startDate, maturityDate, maturityAmount, status)
VALUES (4, 2000, 4.5, 12, '2024-05-20', '2025-05-20', 2090, 'Closed');

INSERT INTO fixeddeposit (account_id, principalAmount, interestRate, tenure,
startDate, maturityDate, maturityAmount, status)
VALUES (5, 15000, 6.2, 36, '2022-11-01', '2025-11-01', 17900, 'Active');

UPDATE fixeddeposit
SET status = 'Closed'
WHERE id = 4;

UPDATE fixeddeposit
SET status = Deactive
WHERE id = 2;
```

JOIN Query

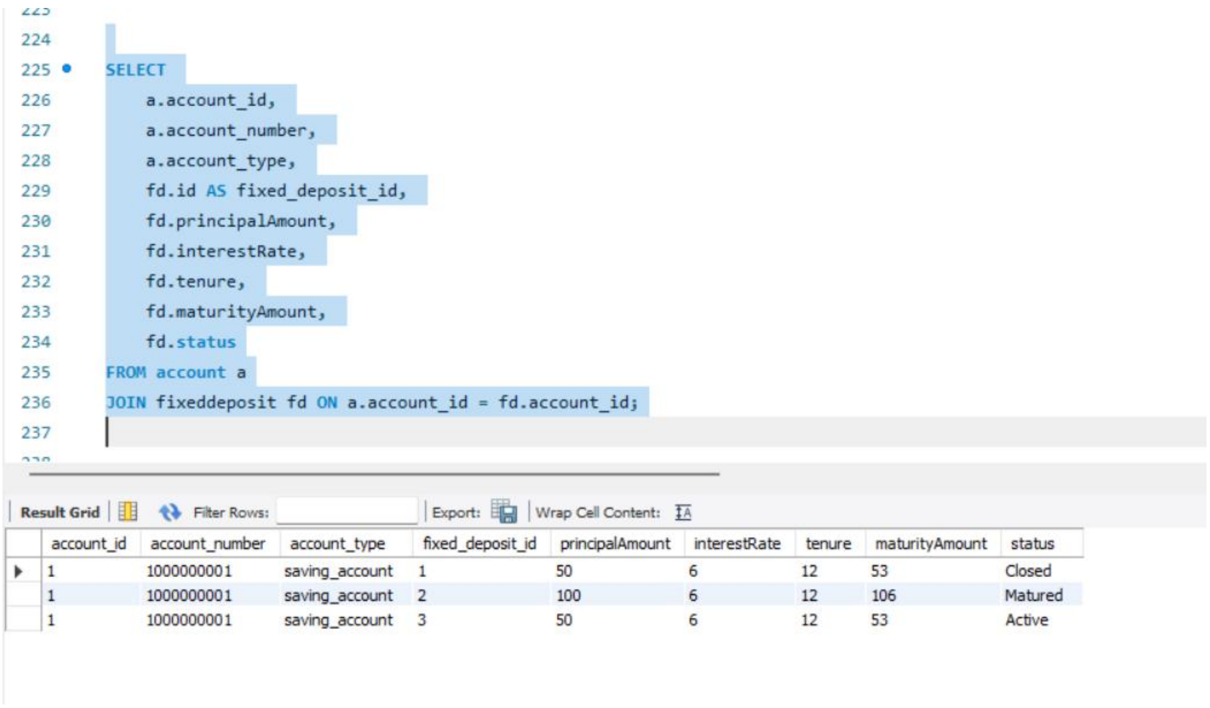
Customer Table and Account Table



```sql

SELECT
    c.customer_id,
    c.customer_firstname,
    c.customer_lastname,
    c.customer_emailid,
    a.account_id,
    a.account_number,
    a.account_type,
    a.account_balance
FROM customers c
JOIN account a ON c.customer_id = a.customer_id;
```

customer_id	customer_firstname	customer_lastname	customer_emailid	account_id	account_number	account_type	account_balance
1	Yatik	Anghan	yatikanghan01@gmail.com	1	1000000001	saving_account	611.0
2	Bansi	Sonani	bansisonani01@gmail.com	2	1000000002	saving_account	1120.0
3	Bhavin	Asodariya	bhavin01@gmail.com	3	1000000003	saving_account	540.0

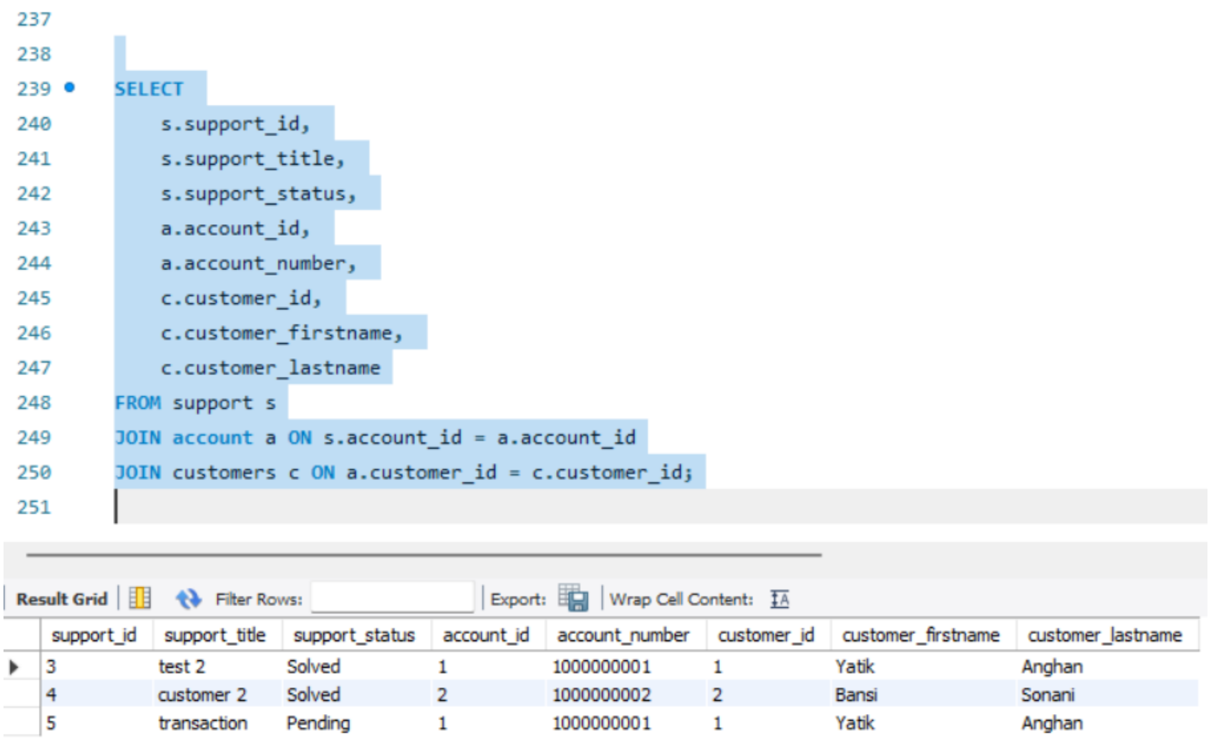
Account Table and Fixed Deposit Table



```sql

SELECT
    a.account_id,
    a.account_number,
    a.account_type,
    fd.id AS fixed_deposit_id,
    fd.principalAmount,
    fd.interestRate,
    fd.tenure,
    fd.maturityAmount,
    fd.status
FROM account a
JOIN fixeddeposit fd ON a.account_id = fd.account_id;
```

account_id	account_number	account_type	fixed_deposit_id	principalAmount	interestRate	tenure	maturityAmount	status
1	1000000001	saving_account	1	50	6	12	53	Closed
1	1000000001	saving_account	2	100	6	12	106	Matured
1	1000000001	saving_account	3	50	6	12	53	Active

Customer Table and Support Table



```
SELECT
    s.support_id,
    s.support_title,
    s.support_status,
    a.account_id,
    a.account_number,
    c.customer_id,
    c.customer_firstname,
    c.customer_lastname
FROM support s
JOIN account a ON s.account_id = a.account_id
JOIN customers c ON a.customer_id = c.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

support_id	support_title	support_status	account_id	account_number	customer_id	customer_firstname	customer_lastname
3	test 2	Solved	1	1000000001	1	Yatik	Anghan
4	customer 2	Solved	2	1000000002	2	Bansi	Sonani
5	transaction	Pending	1	1000000001	1	Yatik	Anghan

4.2 NoSQL Database implementation(MongoDB)

MongoDB is used to store semi-structured data like activity and to-do lists for admins.

4.2.1 Creating a MongoDB Database

use("Banking_Management")

4.2.2 Creating collentions

db.createCollection("tbl_todo")

4.3 SQL & NoSQL integration

To integrate MySQL and MongoDB system with smooth dataflows by:

1. Using customer_id as a common indentifier across both database.
2. Querying structured data from MySQL, while retrieving unstructured logs from MongoDB.
3. Synchronizing data using backend API service.

4.4 Query Execution & Data Operations

4.4.1 SQL Queries

- **Inserting a new customer :**

INSERT INTO customers (customer_firstname, customer_lastname, customer_emailid, customer_password, customer_mobile, customer_dob, customer_address, customer_postcode, customer_country)

VALUES ("Ayush", "Pandav", "ayush@gmail.com", "ayush123", "1234567890", "2002-03-19", "123 Main Street", "14055", "Germany");

- **Fetching all transactions for an account :**

SELECT * FROM transactions WHERE sender_account = '123456789' OR receiver_account = '123456789';

- **Updating Account Balance :**

UPDATE account SET account_balance = account_balance + 500 WHERE account_number = '123456789';

- **Delete a Customer Record :**

DELETE FROM customers WHERE customer_id = 10;

4.4.2 NoSQL Queries(MongoDB)

- **Insert records :**

```
•  // insert may recode
•    db.tbl_todo.insertMany([
•      {
"todo_title": "Customer Meeting",
"todo_desc": "Meeting with VIP customer at 2 PM",
"todo_date": new Date("2025-03-29T14:00:00"),
"todo_admin_id": "1",
"priority": "Medium"
},
{
"todo_title": "System Maintenance",
"todo_desc": "Schedule system maintenance for weekend",
"todo_date": new Date("2025-03-30T18:00:00"),
"todo_admin_id": "2",
"priority": "Low"
}
]);
```

- **Updating:**

```
    db.tbl_todo.updateOne(
      { "todo_title": "Payment Pending" },
      {
        $set: {
          "priority": "High",
          "status": "In Progress"
        }
      }
    );

    db.tbl_todo.updateMany(
      { "todo_admin_id": "3" },
      {
        $set: {
          "department": "Customer Service"
        }
      }
    );
```

- **Deleting :**

```
  // delete recode

    db.tbl_todo.deleteOne({ "todo_title": "Appointment fot FD" });

    db.tbl_todo.deleteMany({ "priority": "Low" });

```

- **Aggregate :**

```
    db.tbl_todo.aggregate([
      {
        $match: {
          "todo_date": { $gte: new Date("2025-03-01") }
        }
      },
    {
      $count: "tasks_this_month_is_perform_by_admin"
    }
  ]);

  db.tbl_todo.aggregate([
    {
      $sort: { "todo_date": -1 }
    },
    {
      $limit: 5
    }
  ]);
```

4.5 Conclusion

The implementation of the bank management system follow a hybrid database approach, with efficient data integrity, secure transactions and integration between SQL and NoSQL databases. MySQL is used for records, accounts and transactions. While MongoDB handles activities for flexibility.

5. Challenges & Solution

During the time of development of this project several challenges were identified in database design, query with SQL and NoSQL integrations. Below down are the most key challenges were faced and their solution:

5.1 SQL and NoSQL integration challenges

Challenge :

Integration MySQL and MongoDB while make sure consistency in data flow was complex, as realational database follow structure on the other hand NoSQL follow flexible document base storage.

Solution :

- Shared identifier : User customer_id as a common identifier for both MySQL and MongoDB for linking.
- Backend API layer : Implemented spring boot APIs to fetch data from database.
- Data synchronization : Important updates in MySQL also trigger updates in MongoDB.

5.2 Transaction Performance Optimization

Challenge :

As database grew, retrieving and processing large transactions record using complex quries impacted performance.

Solution :

- Indexing : Added indexes on often quried columns to speed up retrieval.
- Partitioning : Used time based partitioning for large transaction table to improve execution speed.

5.3 Manage Concurrent transactions

Challenge :

Handling simultaneous transactions while maintaining data consistency and preventing race condition was hard.

Solution :

- Transaction Handling in MySQL : Used ACID compliant transactions with commit and rollback for data consistency.
- Concurrency Control : Implemented optimistic locking to prevent double spending in transactions.

5.4 Secure User Authentication & Data Protection

Challenge :

Ensuring secure authentication and preventing data breaches were key security concerns.

Solution :

- Role-based access control : Restricted access to admin functions to prevent unauthorised access.
- SQL injection prevention : User prepared statements in SQL queries to prevent injection attacks.

5.5 Conclusion

By the implementation of best practices in database optimization, security and integration. This system successfully overcomes technical challenges, ensuring all this above.

6. Results

The Bank Management System was successfully developed using hybrid database, with MySQL and MongoDB. This system easily manage customer accounts, transactions, and support operations while secure authentication, optimized queries and smooth integration.

6.1 Functional System Outputs

6.1.1 Customer & Account Management

- Customer can register, login and update their details easily.
- Admin can manage accounts, approve transactions and oversee system operations.

6.1.2 Transaction Processing

- Customer can perform deposits, withdrawals and transfer.
- Admin can track and monitor transactions in real time.

6.1.3 NoSQL Data retrieval(MongoDB)

- Customer activity logs and to-do list are retrieved easily.
- Indexes queries improve retrieval speed for large datasets.

6.2 System Performance Evaluation

6.2.1 Query Execution Speed

- Indexes MySQL showed 50% improvement in execution.
- MongoDB queries retrieved logs 3x faster after adding indexes.

6.2.2 Security Enhancement

- Bcrypt password hashing ensure user authentication.

- Role-base access control prevented unauthorized access to admin functionality.

6.2.3 Data Consistency & Scalability

- ACID compliant MySQL transactions make sured data integrity.
- Sharding in MongoDB improved scale and storage efficiency for long management.

6.3 Conclusion

By the implementation of best practices in database optimization, security and integration. This system successfully overcomes technical challenges, ensuring all this above.

7. Conclusion & Future work

The bank management system successfully integrates MySQL and MongoDB to provide a solution for managing banking operations. MySQL ensures data integrity and ACID compliance for structure data like accounts, operations while MongoDB enhance flexibility and performance by logs and to-do lists for administration.

Conclusion :

This project shows the initial demonstrates :

- Seamless SQL-NoSQL integration for data management.
- Optimized query execution with indexing and normalization.
- Secure authentication using password and role-based access control.
- Scalability through sharding in MongoDB for log management.

Overall, system meets its object by making sure high performance, security and data reliability in banking management.

Future work :

This system is fully functional but also need several ways to improve and expand it:

Fraud detection System :

- Implement a system to identify suspicious transactions and avoiding fraud.

User-friendly interface :

- Improve more web and also mobile app for customer.

Automated report :

- Generate monthly account statements and transactions summaries.

Multi-Currency Support :

- Allow transactions in different currencies for international banking.

Improved Security Measures :

- Enhance security with two-factor authentication for login and transaction.

8. References

1. Database Concepts

- Elmasri, R., & Navathe, S. (2017). Fundamentals of Database Systems. Pearson.
- Connolly, T., & Begg, C. (2014). Database Systems: Design & Implementation. Pearson.

2. Database Concepts

- MongoDB Documentation. (2025). Schema Design & Indexing from www.mongodb.com
- MySQL Documentation. (2025). Indexing & Query Optimization. Retrieved www.mysql.com

3. Authentication

- Spring Security Guide. (2025). Authentication & Authorization from www.spring.io

4. Backend development

- Baeldung. (2025). Spring Boot REST API Guide from www.baeldung.com