CS-GY6083 PROJECT PART II

Course: CS-GY 6083

Section B

Submission Date: 05/05/2024

Name: Amarnadh Reddy Mettu
Student Net-ID: am14305

Name: Deva Kumar Katta
Student Net-ID: dk4945

Name: Shanmukeshwar Reddy Kanjula
Student Net-ID: sk11331

TABLE OF CONTENTS

EXECUTIVE SUMMARY

BUSINESS CASE

The database is designed for a Save and Fortune Excellence Bank Management. It aims to manage both, employee and customers, handle different account types (Savings, Checking, Loan(Student Loan, Home Loan)), apply discounts, track student details and insurance company details for student and home loans respectively, and manage bank accounts. This database will streamline operations, enhance customer service, and facilitate decision-making through data-driven insights.

DEVELOPMENT STRATEGY

Schema Design:

The design follows a relational model with clear primary and foreign key relationships.

Normalization:

Tables are normalized to reduce redundancy and improve data integrity.

Customer Segmentation:

Users are categorized into customers and employees, with respective tables (das_cust, das_employee) to handle specific attributes.

Account Management:

Separate tables for savings accounts (das_savings), checking accounts (das_check) and loan accounts (das_loan) to facilitate different operations.

Loan Account details Tracking:

Tables like das_student and das_home ensure accurate details of educational institutions incase of home loan and corporate home insurance companies for home loan.

Account Management: The das_accounts table captures details of each type of accounts, including savings, checking, and loan accounts.

IMPACT ON BUSINESS PERFORMANCE

Efficient Data Management:

The database schema objects are structured and designed optimally to ensure high performance in data retrieval, efficient storage, and overall management of data.

Customer Relationship Management:

By maintaining detailed customer information, the bank can offer personalized services.

Operational Efficiency:

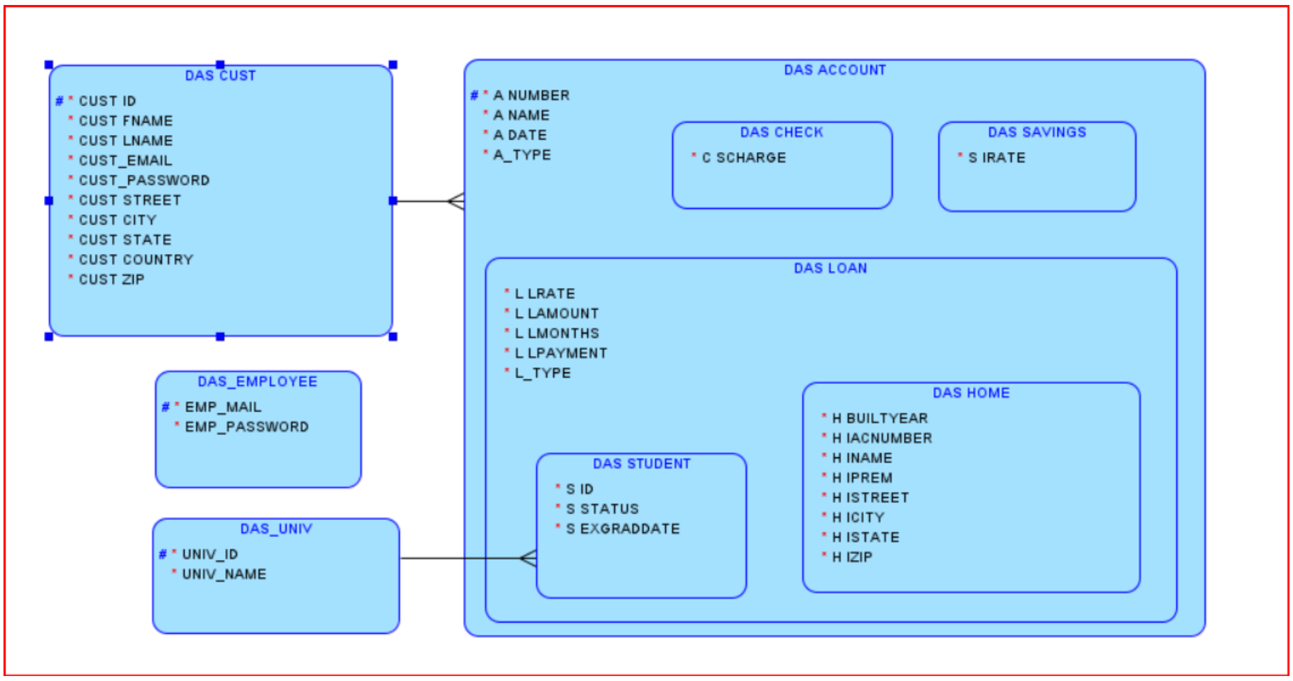
The database supports operational activities like creating bank account, updating personal details, deleting accounts, update rates such as loan interest rate, savings interest rate and service charge for checking accounts, and analysis about the type of account.

Data-Driven Decisions:

Aggregated data can be used for trend analysis, performance metrics, and strategic planning.

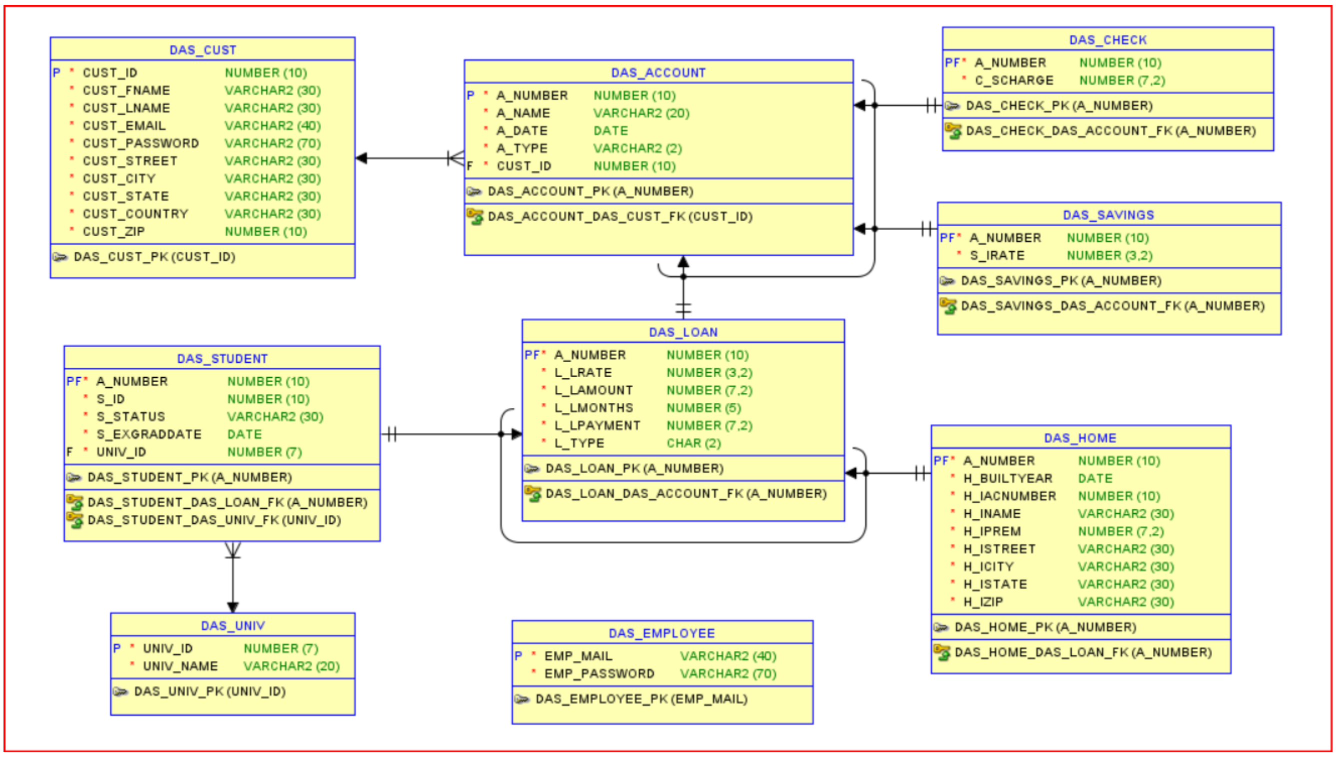
LOGICAL MODEL

The logical model outlines the structure of the database, including entities (tables), attributes (columns), and relationships (keys). It focuses on what data needs to be stored without delving into physical storage details.



RELATIONAL MODEL :

The relational model represents the logical model in a database format. It includes table creation with specified data types and constraints, and the establishment of relationships through primary and foreign keys.



ASSUMPTIONS

1. SAFE will only establish itself in United States and no other countries.
2. A_NUMBER is the account number.
3. A customer can have any number of accounts.
4. A_DATE is the account opening date.
5. Each customer can have many accounts and each account belongs to one customer.
6. A customer can have a loan account without savings and checking.
7. Insurance company details are mentioned in home table.
8. A_TYPE represents the type of account.
9. L_TYPE represent the type of loan account.
10. “S” for savings account, “C“ for checking account, “L“ for loan account, “ST“ for student loan account, “H“ for home loan account.
11. Every account has its own account number.
12. DAS_CHECK table represents Checking accounts.
13. Each registered User who has done registration through the Web Portal for SAFE services; can be either a customer or an employer.
14. Each User can register only once with their email address, with an email and password the user can login to SAFE Application.
15. As instance of this super-type DAS_ACCOUNTS could be only one of the subtypes i.e., either an Savings account or Checking account or Loan account, the disjoint rule is applied. Hence, A_NUMBER attribute is included in the DAS_ACCOUNTS entity, as this would be help to retrieve/update information from respective subtype entity.
16. Each customer DAS_CUST will have a CUST_EMAIL that is considered to be Unique.
17. Each employee DAS_EMPLOYEE will have a authorization code from the bank in order to sign up.
18. Each customer can have a student loan from one or more Universities. Also, each University can have many students with a student loan from SAFE. To resolve redundancy a separate entity for the University is created with DAS_UNIV having UNIV_ID and UNIV_NAME.
19. As instance of this super-type DAS_LOAN could be only one of the subtypes i.e., either a student loan or a home loan.
20. A customer cannot have two loan account at a time, for ex. if he had a student loan account, in order to get home loan account, he needs to close his student loan account. We make this assumption because if a customer has student loan, then he won’t need a home loan if he had home loan then he won’t be a student.
21. A customer in DAS_CUST can only update their personal details from the SAFE app
22. An employee in DAS_EMPLOYEE can only update the official details, i.e. interest rate and service charge.
23. If a customer want to delete his account then, an employee will delete that specific account using the account number from customer.
24. A Customer from DAS_CUST can view his personal details, all type of accounts and the details of his accounts with our SAFE application

OTHER CONSIDERATIONS

Data Integrity: Assumed the data entered into the system is valid and reliable.
User Role Management: Application serves predefined User-Roles with corresponding permissions and authorizations to certain functionalities.
Business Rules Compliance: Assumed that business rules are well-defined and enforced through constraints and triggers in the database.
Scalability: The design assumes that the system can scale to accommodate growing data and user demands. Implement robust security measures to protect sensitive customer and transaction data. Backup and **Recovery:** Regular backups and recovery plan exist for data safety.

SOFTWARE INFORMATION

Programming Languages, Databases, and Tools Used

MySQL

A widely-used open-source relational database management system (RDBMS), MySQL is essential for storing, retrieving, modifying, and administrating a database. It's known for its performance, reliability, and ease of use. MySQL is especially popular in Web Applications and is often used as part of the LAMP (Linux, Apache, MySQL, PHP / Perl / Python) stack.

PYTHON Programming

This high-level, interpreted programming language is celebrated for its readability and broad support for various programming paradigms, including object-oriented, procedural, and functional programming. Python is versatile, used in web development, data analysis, artificial intelligence, scientific computing, and more, thanks to its extensive libraries and frameworks.

Streamlit

Streamlit is an open-source Python library that simplifies the process of creating and sharing beautiful, custom web apps for machine learning and data science. In just a few lines of code, Streamlit allows you to build highly interactive web applications directly from your Python scripts. It is particularly popular among data scientists and analysts for its ease of use and flexibility in building data-driven web applications without the need for in-depth knowledge of web development frameworks.

Integration of MySQL, Python, Streamlit

Backend Processing with Python

Python acts as the core programming language, handling logic, data processing, and database operations. It interacts with MySQL for data storage and retrieval, ensuring efficient data management. MySQL manages the data storage, ensuring secure and efficient handling of database transactions, queries, and operations.

Frontend Interface with Streamlit

Streamlit enables the development of user-friendly web interfaces. It allows Python scripts to be turned into interactive web applications, which can be used to visualize data, input parameters, and display results.

DDL CODE

The below DDL statement was converted from Oracle generated DDL.sql to MySQL DDL.sql and modified appropriately to run on MySQL database without any error.

<u>Generated SQL</u>

```
-- SQLINES DEMO *** le SQL Developer Data Modeler 23.1.0.087.0806
-- SQLINES DEMO *** -05-04 14:47:08 GMT-04:00
-- SQLINES DEMO *** le Database 21c
-- SQLINES DEMO *** le Database 21c
-- SQLINES DEMO *** no DDL - MDSYS.SDO_GEOMETRY
-- SQLINES DEMO *** no DDL - XMLTYPE
-- SQLINES LICENSE FOR EVALUATION USE ONLY

CREATE TABLE das_account (
    a_number BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
    a_name   VARCHAR(20) NOT NULL COMMENT 'ACCOUNT NAME',
    a_date   DATETIME NOT NULL COMMENT 'ACCOUNT OPENING DATE
',
    a_type   VARCHAR(2) NOT NULL COMMENT 'TYPE OF THE ACCOUNT',
    cust_id  BIGINT NOT NULL
);

ALTER TABLE das_account
    ADD CONSTRAINT ch_inh_das_account CHECK ( a_type IN ( 'C', 'L', 'S' ) );

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_account.a_number IS
    'ACCOUNT NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_account.a_name IS
    'ACCOUNT NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_account.a_date IS
    'ACCOUNT OPENING DATE
'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_account.a_type IS
    'TYPE OF THE ACCOUNT'; */

ALTER TABLE das_account ADD CONSTRAINT das_account_pk PRIMARY KEY ( a_number );
```

```
-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_check (
    a_number  BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
    c_scharge DECIMAL(7, 2) NOT NULL COMMENT 'SERVICE CHARGE OF CHECKING ACCOUNT'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_check.a_number IS
    'ACCOUNT NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_check.c_scharge IS
    'SERVICE CHARGE OF CHECKING ACCOUNT'; */

ALTER TABLE das_check ADD CONSTRAINT das_check_pk PRIMARY KEY ( a_number );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_cust (
    cust_id       BIGINT NOT NULL COMMENT 'CUSTOMER IDENTIFICATION NUMBER',
    cust_fname    VARCHAR(30) NOT NULL COMMENT 'CUSTOMER''S FIRST NAME',
    cust_lname    VARCHAR(30) NOT NULL COMMENT 'CUSTOMER LAST NAME',
    cust_email    VARCHAR(40) NOT NULL COMMENT 'CUSTOMER EMAIL ADDRESS
',
    cust_password VARCHAR(70) NOT NULL COMMENT 'CUSTOMER PASSWORD',
    cust_street   VARCHAR(30) NOT NULL COMMENT 'CUSTOMER STREET NAME',
    cust_city     VARCHAR(30) NOT NULL COMMENT 'CUSTOMER CITY NAME',
    cust_state    VARCHAR(30) NOT NULL COMMENT 'CUSTOMER STATE NAME',
    cust_country  VARCHAR(30) NOT NULL COMMENT 'CUSTOMER COUNTRY NAME',
    cust_zip      BIGINT NOT NULL COMMENT 'CUSTOMER ZIP CODE
'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_id IS
    'CUSTOMER IDENTIFICATION NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_fname IS
    'CUSTOMER''S FIRST NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_lname IS
    'CUSTOMER LAST NAME'; */

/* Moved to CREATE TABLE
```

```
COMMENT ON COLUMN das_cust.cust_email IS
    'CUSTOMER EMAIL ADDRESS
'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_password IS
    'CUSTOMER PASSWORD'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_street IS
    'CUSTOMER STREET NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_city IS
    'CUSTOMER CITY NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_state IS
    'CUSTOMER STATE NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_country IS
    'CUSTOMER COUNTRY NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_cust.cust_zip IS
    'CUSTOMER ZIP CODE
'; */

ALTER TABLE das_cust ADD CONSTRAINT das_cust_pk PRIMARY KEY ( cust_id )
-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_employee (
    emp_mail     VARCHAR(40) NOT NULL COMMENT 'EMPLOYEE EMAIL ADDRESS',
    emp_password VARCHAR(70) NOT NULL COMMENT 'EMPLOYEE PASSWORD'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_employee.emp_mail IS
    'EMPLOYEE EMAIL ADDRESS'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_employee.emp_password IS
    'EMPLOYEE PASSWORD'; */

ALTER TABLE das_employee ADD CONSTRAINT das_employee_pk PRIMARY KEY ( emp_mail );
```

```
-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_home (
   a_number    BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
   h_builtyear DATETIME NOT NULL COMMENT 'BUILT DATE OF HOME',
   h_iacnumber BIGINT NOT NULL COMMENT 'INSURANCE ACCOUNT NUMBER OF HOME',
   h_iname     VARCHAR(30) NOT NULL COMMENT 'INSURANCE COMPANY NAME OF HOME',
   h_iprem     DECIMAL(7, 2) NOT NULL COMMENT 'MONTHLY INSURANCE PREMIUM OF HOME',
   h_istreet   VARCHAR(30) NOT NULL COMMENT 'INSURANCE COMPANY STREET NAME',
-- SQLINES DEMO *** Y CITY NAME
   h_icity     VARCHAR(30) NOT NULL COMMENT 'INSURANCE COMPANY CITY NAME
',
-- SQLINES DEMO *** Y STATE NAME
   h_istate    VARCHAR(30) NOT NULL COMMENT 'INSURANCE COMPANY STATE NAME
',
-- SQLINES DEMO *** Y ZIP CODE
   h_izip      VARCHAR(30) NOT NULL COMMENT 'INSURANCE COMPANY ZIP CODE
'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.a_number IS
   'ACCOUNT NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_builtyear IS
   'BUILT DATE OF HOME'; */
/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_iacnumber IS
   'INSURANCE ACCOUNT NUMBER OF HOME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_iname IS
   'INSURANCE COMPANY NAME OF HOME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_iprem IS
   'MONTHLY INSURANCE PREMIUM OF HOME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_istreet IS
   'INSURANCE COMPANY STREET NAME'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_icity IS
   'INSURANCE COMPANY CITY NAME
'; */
```

```
/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_istate IS
    'INSURANCE COMPANY STATE NAME
'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_home.h_izip IS
    'INSURANCE COMPANY ZIP CODE
'; */

ALTER TABLE das_home ADD CONSTRAINT das_home_pk PRIMARY KEY ( a_number );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_loan (
    a_number   BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
    l_lrate    DECIMAL(3, 2) NOT NULL COMMENT 'LOAN RATE OF ACCOUNT',
    l_lamount  DECIMAL(7, 2) NOT NULL COMMENT 'TOTAL LOAN AMOUNT ',
    l_lmonths  INT NOT NULL COMMENT 'DURATION OF THE LOAN IN MONTHS',
    l_lpayment DECIMAL(7, 2) NOT NULL COMMENT 'MONTHLY PAYMENT OF LOAN',
    l_type     CHAR(2) NOT NULL COMMENT 'TYPE OF LOAN ACCOUNT
'
);

ALTER TABLE das_loan
    ADD CONSTRAINT ch_inh_das_loan CHECK ( l_type IN ( 'H', 'ST' ) );

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.a_number IS
    'ACCOUNT NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.l_lrate IS
    'LOAN RATE OF ACCOUNT'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.l_lamount IS
    'TOTAL LOAN AMOUNT '; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.l_lmonths IS
    'DURATION OF THE LOAN IN MONTHS'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.l_lpayment IS
    'MONTHLY PAYMENT OF LOAN'; */
```

```
/* Moved to CREATE TABLE
COMMENT ON COLUMN das_loan.l_type IS
   'TYPE OF LOAN ACCOUNT
'; */

ALTER TABLE das_loan ADD CONSTRAINT das_loan_pk PRIMARY KEY ( a_number );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_savings (
   a_number BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
   s_irate  DECIMAL(3, 2) NOT NULL COMMENT 'INTEREST RATE OF SAVINGS ACCOUNT
'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_savings.a_number IS
   'ACCOUNT NUMBER'; */
/* Moved to CREATE TABLE
COMMENT ON COLUMN das_savings.s_irate IS
   'INTEREST RATE OF SAVINGS ACCOUNT
'; */

ALTER TABLE das_savings ADD CONSTRAINT das_savings_pk PRIMARY KEY ( a_number );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_student (
   a_number     BIGINT NOT NULL COMMENT 'ACCOUNT NUMBER',
   s_id         BIGINT NOT NULL COMMENT 'STUDENT IDENTIFICATION NUMBER',
   s_status     VARCHAR(30) NOT NULL COMMENT 'GRAD OR UNDER GRAD STATUS OF THE STUDENT',
   s_exgraddate DATETIME NOT NULL COMMENT 'EXPECTED GRADUATION DATE OF THE STUDENT',
   univ_id      INT NOT NULL
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_student.a_number IS
   'ACCOUNT NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_student.s_id IS
   'STUDENT IDENTIFICATION NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_student.s_status IS
   'GRAD OR UNDER GRAD STATUS OF THE STUDENT'; */
```

```
/* Moved to CREATE TABLE
COMMENT ON COLUMN das_student.s_exgraddate IS
    'EXPECTED GRADUATION DATE OF THE STUDENT'; */

ALTER TABLE das_student ADD CONSTRAINT das_student_pk PRIMARY KEY ( a_number );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
CREATE TABLE das_univ (
    univ_id   INT NOT NULL COMMENT 'UNIVERSITY IDENTIFICATION NUMBER',
    univ_name VARCHAR(20) NOT NULL COMMENT 'STUDENT UNIVERSITY NAME'
);

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_univ.univ_id IS
    'UNIVERSITY IDENTIFICATION NUMBER'; */

/* Moved to CREATE TABLE
COMMENT ON COLUMN das_univ.univ_name IS
    'STUDENT UNIVERSITY NAME'; */

ALTER TABLE das_univ ADD CONSTRAINT das_univ_pk PRIMARY KEY ( univ_id );

ALTER TABLE das_account
    ADD CONSTRAINT das_account_das_cust_fk FOREIGN KEY ( cust_id )
        REFERENCES das_cust ( cust_id );

ALTER TABLE das_check
    ADD CONSTRAINT das_check_das_account_fk FOREIGN KEY ( a_number )
        REFERENCES das_account ( a_number );

ALTER TABLE das_home
    ADD CONSTRAINT das_home_das_loan_fk FOREIGN KEY ( a_number )
        REFERENCES das_loan ( a_number );

ALTER TABLE das_loan
    ADD CONSTRAINT das_loan_das_account_fk FOREIGN KEY ( a_number )
        REFERENCES das_account ( a_number );

ALTER TABLE das_savings
    ADD CONSTRAINT das_savings_das_account_fk FOREIGN KEY ( a_number )
        REFERENCES das_account ( a_number );

ALTER TABLE das_student
    ADD CONSTRAINT das_student_das_loan_fk FOREIGN KEY ( a_number )
        REFERENCES das_loan ( a_number );
```

```
ALTER TABLE das_student
   ADD CONSTRAINT das_student_das_univ_fk FOREIGN KEY ( univ_id )
      REFERENCES das_univ ( univ_id );

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_check;

DELIMITER //

CREATE TRIGGER arc_fkarc_9_das_check BEFORE
 INSERT ON das_check
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'C' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FK DAS_CHECK_DAS_ACCOUNT_FK in Table DAS_CHECK violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''C''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //

CREATE TRIGGER arc_fkarc_9_das_check BEFORE
   UPDATE ON das_check
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'C' ) THEN
```

```
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FK DAS_CHECK_DAS_ACCOUNT_FK in Table DAS_CHECK violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''C''';
   END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_savings;

DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_savings BEFORE
   INSERT ON das_savings
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'S' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_SAVINGS_DAS_ACCOUNT_FK in Table DAS_SAVINGS violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''S''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_savings BEFORE
   UPDATE ON das_savings
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;
   IF ( d IS NULL OR d <> 'S' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_SAVINGS_DAS_ACCOUNT_FK in Table DAS_SAVINGS violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''S''';
   END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_loan;

DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_loan BEFORE
   INSERT ON das_loan
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'L' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_LOAN_DAS_ACCOUNT_FK in Table DAS_LOAN violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''L''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_loan BEFORE
   UPDATE ON das_loan
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
```

```
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'L' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_LOAN_DAS_ACCOUNT_FK in Table DAS_LOAN
violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''L''';
    END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_8_das_home;

DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_home BEFORE
    INSERT ON das_home
    FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
    SELECT
        a.l_type
    INTO d
    FROM
        das_loan a
    WHERE
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'H' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_HOME_DAS_LOAN_FK in Table DAS_HOME
violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value ''H''';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_home BEFORE
    UPDATE ON das_home
    FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
    SELECT
        a.l_type
    INTO d
    FROM
        das_loan a
    WHERE
```

```
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'H' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_HOME_DAS_LOAN_FK in Table DAS_HOME
violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value ''H''';
    END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_8_das_student;

DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_student BEFORE
    INSERT ON das_student
    FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
    SELECT
        a.l_type
    INTO d
    FROM
        das_loan a
    WHERE
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'ST' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_STUDENT_DAS_LOAN_FK in Table
DAS_STUDENT violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value
''ST''';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_student BEFORE
    UPDATE ON das_student
    FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
    SELECT
        a.l_type
    INTO d
    FROM
        das_loan a
    WHERE
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'ST' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_STUDENT_DAS_LOAN_FK in Table
DAS_STUDENT violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value
''ST''';
    END IF;
END;
//
DELIMITER ;

-- SQLINES DEMO *** per Data Modeler Summary Report:
--
-- SQLINES DEMO ***                    9
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                   18
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO *** DY                 0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    5
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***  TYPE              0
-- SQLINES DEMO ***  TYPE              0
-- SQLINES DEMO ***  TYPE BODY  0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO *** EGMENT             0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO *** ED VIEW            0
-- SQLINES DEMO *** ED VIEW LOG 0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
--
-- SQLINES DEMO ***                    0
-- SQLINES DEMO ***                    0
```

TRIGGER CODE

```
-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_check;

DELIMITER //

CREATE TRIGGER arc_fkarc_9_das_check BEFORE
   INSERT ON das_check
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'C' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FK DAS_CHECK_DAS_ACCOUNT_FK in Table DAS_CHECK violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''C''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //

CREATE TRIGGER arc_fkarc_9_das_check BEFORE
   UPDATE ON das_check
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'C' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'FK DAS_CHECK_DAS_ACCOUNT_FK in Table DAS_CHECK violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''C''';
   END IF;
END;
//
DELIMITER ;
-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_savings;

DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_savings BEFORE
   INSERT ON das_savings
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'S' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_SAVINGS_DAS_ACCOUNT_FK in Table DAS_SAVINGS violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''S''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_savings BEFORE
   UPDATE ON das_savings
   FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
   SELECT
      a.a_type
   INTO d
   FROM
      das_account a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'S' ) THEN
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_SAVINGS_DAS_ACCOUNT_FK in Table
DAS_SAVINGS violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have
value ''S''';
    END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_9_das_loan;

DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_loan BEFORE
    INSERT ON das_loan
    FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
    SELECT
        a.a_type
    INTO d
    FROM
        das_account a
    WHERE
        a.a_number = new.a_number;

    IF ( d IS NULL OR d <> 'L' ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_LOAN_DAS_ACCOUNT_FK in Table DAS_LOAN
violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''L''';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_9_das_loan BEFORE
    UPDATE ON das_loan
    FOR EACH ROW
BEGIN
DECLARE d VARCHAR(2);
    SELECT
        a.a_type
    INTO d
    FROM
        das_account a
    WHERE
        a.a_number = new.a_number;
```

```
   IF ( d IS NULL OR d <> 'L' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_LOAN_DAS_ACCOUNT_FK in Table DAS_LOAN
violates Arc constraint on Table DAS_ACCOUNT - discriminator column A_TYPE doesn''t have value ''L''';
   END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_8_das_home;

DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_home BEFORE
   INSERT ON das_home
   FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
   SELECT
      a.l_type
   INTO d
   FROM
      das_loan a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'H' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_HOME_DAS_LOAN_FK in Table DAS_HOME
violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value ''H''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_home BEFORE
   UPDATE ON das_home
   FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
   SELECT
      a.l_type
   INTO d
   FROM
      das_loan a
   WHERE
      a.a_number = new.a_number;
   IF ( d IS NULL OR d <> 'H' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_HOME_DAS_LOAN_FK in Table DAS_HOME
violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value ''H''';
   END IF;
END;
//
DELIMITER ;

-- SQLINES LICENSE FOR EVALUATION USE ONLY
DROP TRIGGER IF EXISTS arc_fkarc_8_das_student;

DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_student BEFORE
   INSERT ON das_student
   FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
   SELECT
      a.l_type
   INTO d
   FROM
      das_loan a
   WHERE
      a.a_number = new.a_number;

   IF ( d IS NULL OR d <> 'ST' ) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_STUDENT_DAS_LOAN_FK in Table
DAS_STUDENT violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value
''ST''';
   END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER arc_fkarc_8_das_student BEFORE
   UPDATE ON das_student
   FOR EACH ROW
BEGIN
DECLARE d CHAR(2);
   SELECT
      a.l_type
   INTO d
   FROM
      das_loan a
   WHERE
      a.a_number = new.a_number;
```

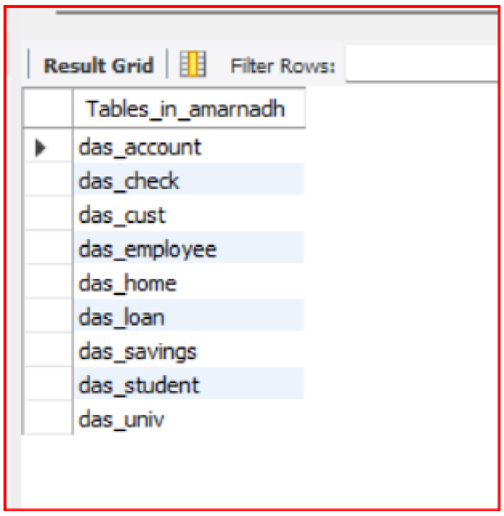
```
  IF ( d IS NULL OR d <> 'ST' ) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='FK DAS_STUDENT_DAS_LOAN_FK in Table DAS_STUDENT violates Arc constraint on Table DAS_LOAN - discriminator column L_TYPE doesn''t have value ''ST''';
  END IF;
END;
//
DELIMITER ;
```

LIST OF TABLES AND DATA RECORDS PER TABLE

I. List of Tables

❖ **Data Dictionary query to list all tables using MySQL.**

SHOW TABLES;

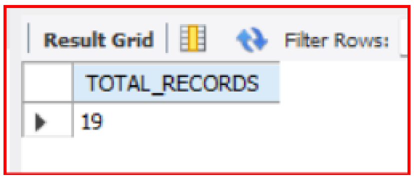


```
Tables_in_amarnadh
das_account
das_check
das_cust
das_employee
das_home
das_loan
das_savings
das_student
das_univ
```

II. List of Total number of Data Records per Table

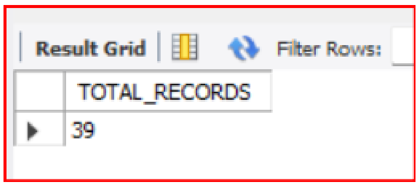
COUNT (*) query and result for each table.

SELECT COUNT(*) AS Total_records FROM DAS_CUST;



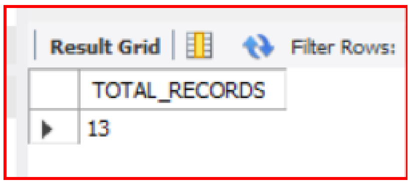
```
TOTAL_RECORDS
19
```

SELECT COUNT(*) AS Total_records FROM DAS_ACCOUNTS;



```
TOTAL_RECORDS
39
```

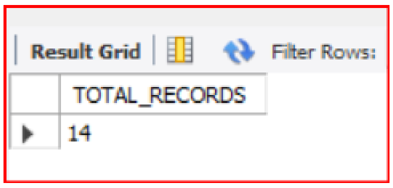
SELECT COUNT(*) AS Total_records FROM DAS_CHECK;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
13
```

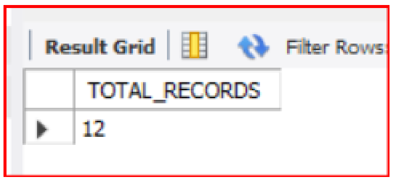
SELECT COUNT(*) AS Total_records FROM DAS_LOAN;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
14
```

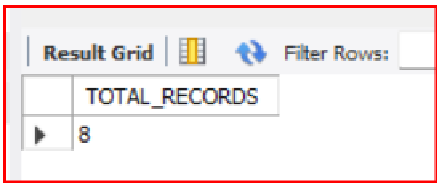
SELECT COUNT(*) AS Total_records FROM DAS_SAVINGS;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
12
```

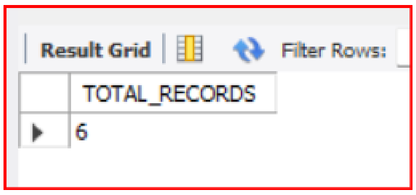
SELECT COUNT(*) AS Total_records FROM DAS_STUDENT;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
8
```

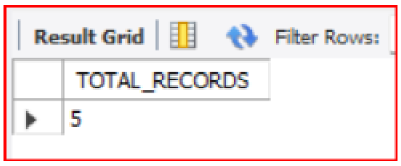
SELECT COUNT(*) AS Total_records FROM DAS_HOME;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
6
```

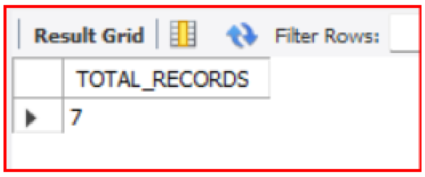
SELECT COUNT(*) AS Total_records FROM DAS_UNIV;



Result Grid | Filter Rows:

```
TOTAL_RECORDS
5
```

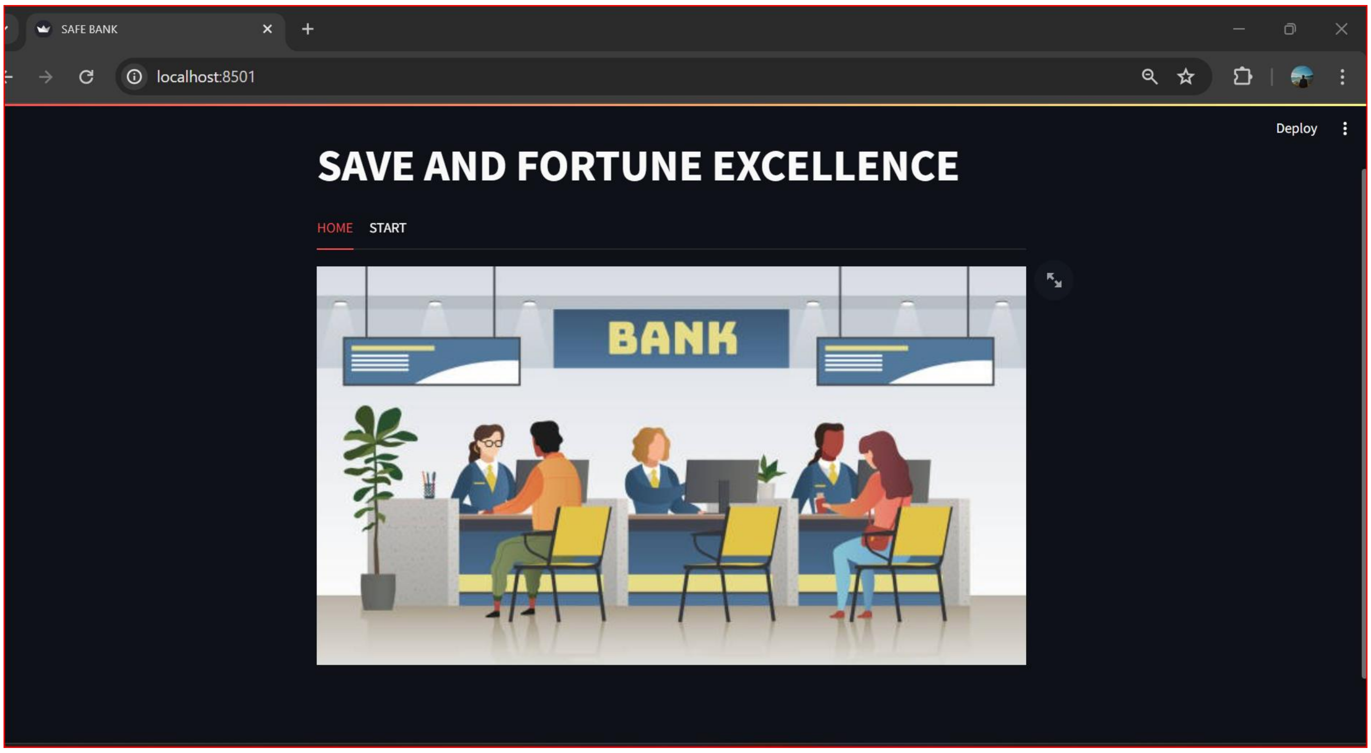
SELECT COUNT(*) AS Total_records FROM DAS_EMPLOYEE;

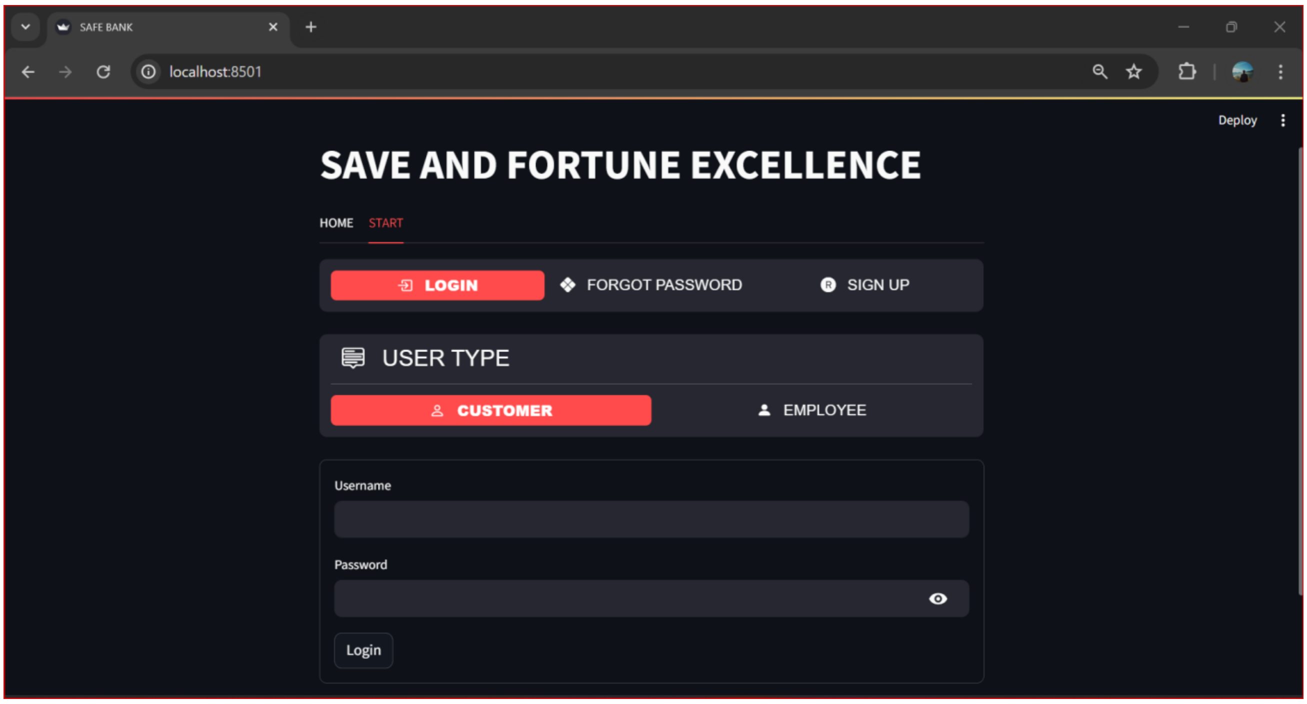


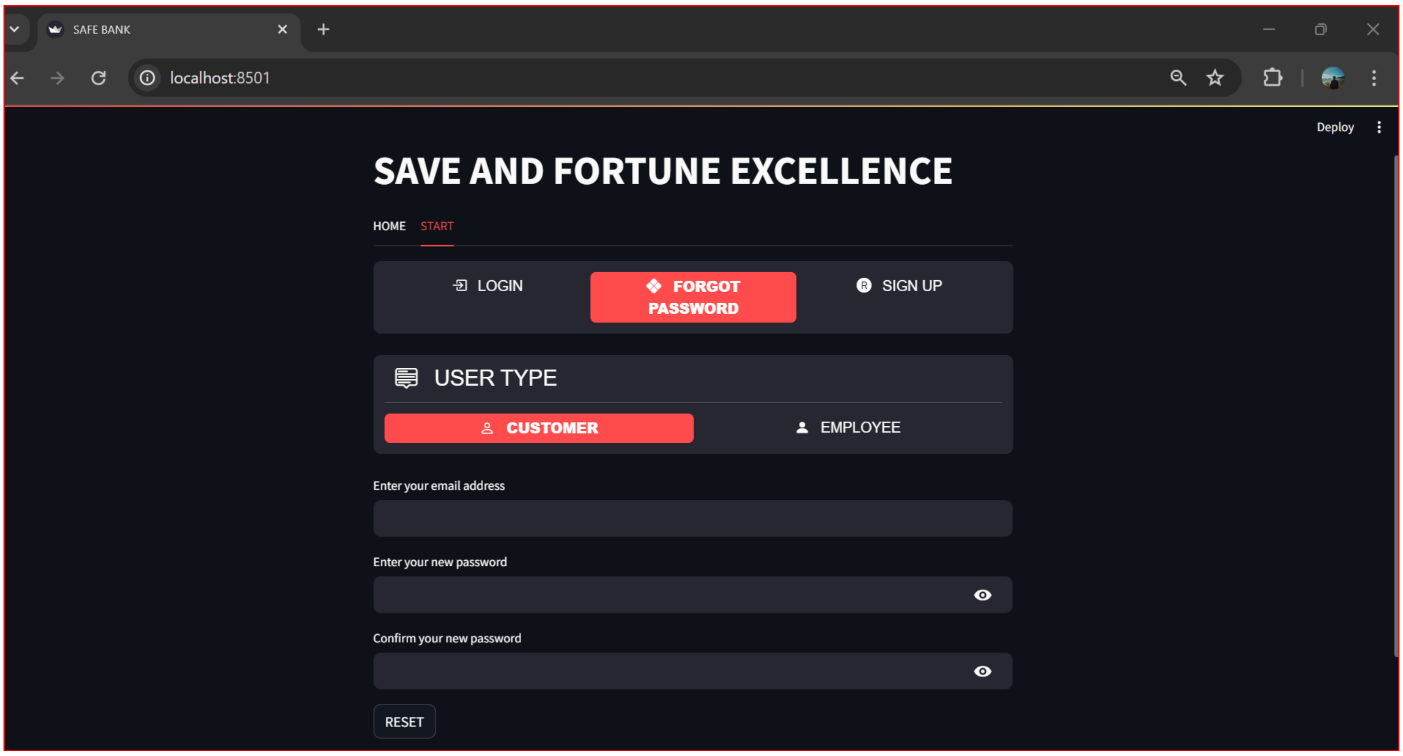
Result Grid | Filter Rows:

```
TOTAL_RECORDS
7
```

SCREENSHOTS







SAFE BANK

localhost:8501

Deploy

SAVE AND FORTUNE EXCELLENCE

HOME START

LOGIN FORGOT PASSWORD SIGN UP

USER TYPE

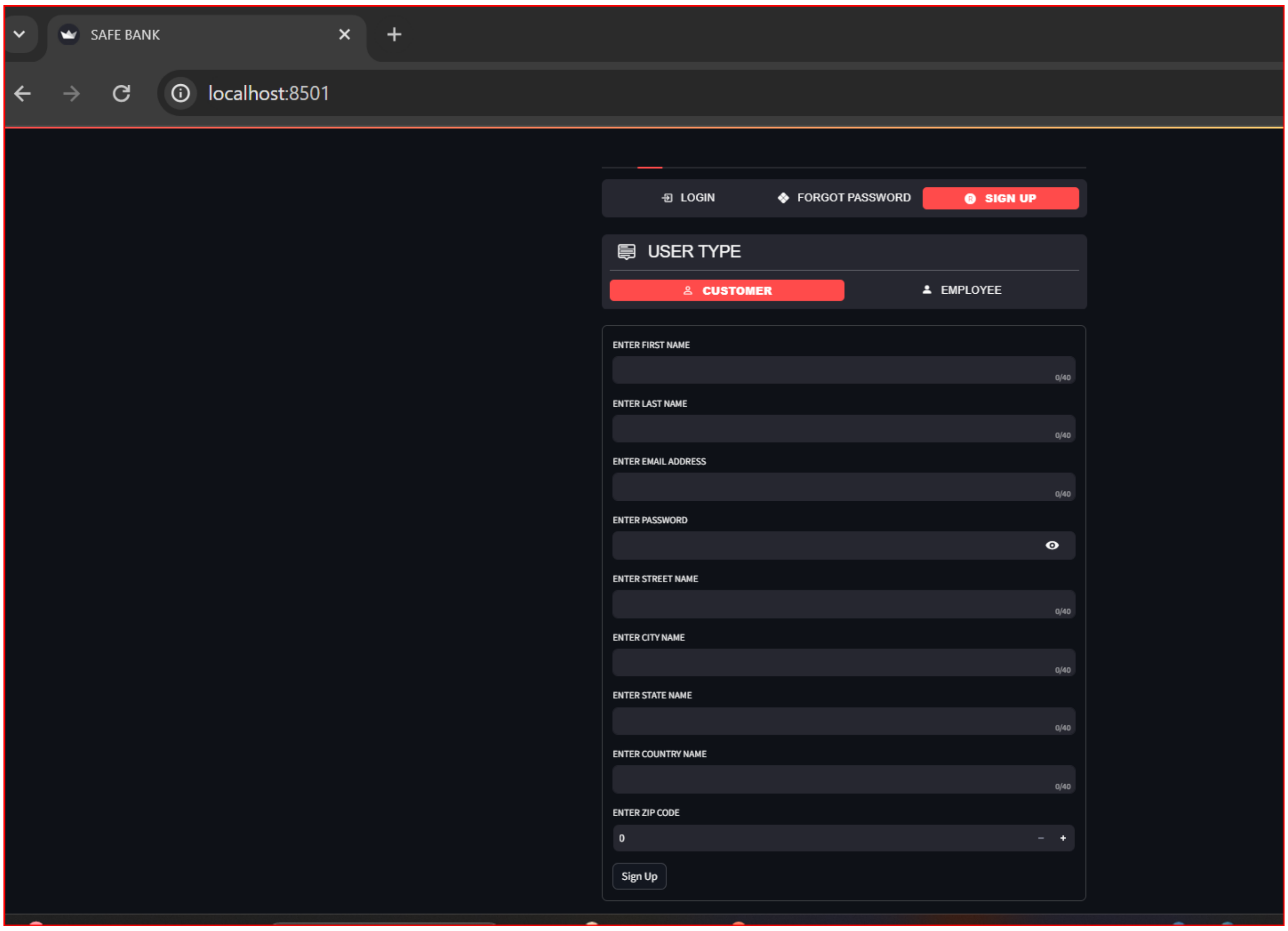
CUSTOMER EMPLOYEE

Enter your email address

Enter your new password

Confirm your new password

RESET



SAFE BANK

localhost:8501

LOGIN FORGOT PASSWORD SIGN UP

USER TYPE

CUSTOMER EMPLOYEE

ENTER FIRST NAME 0/40

ENTER LAST NAME 0/40

ENTER EMAIL ADDRESS 0/40

ENTER PASSWORD

ENTER STREET NAME 0/40

ENTER CITY NAME 0/40

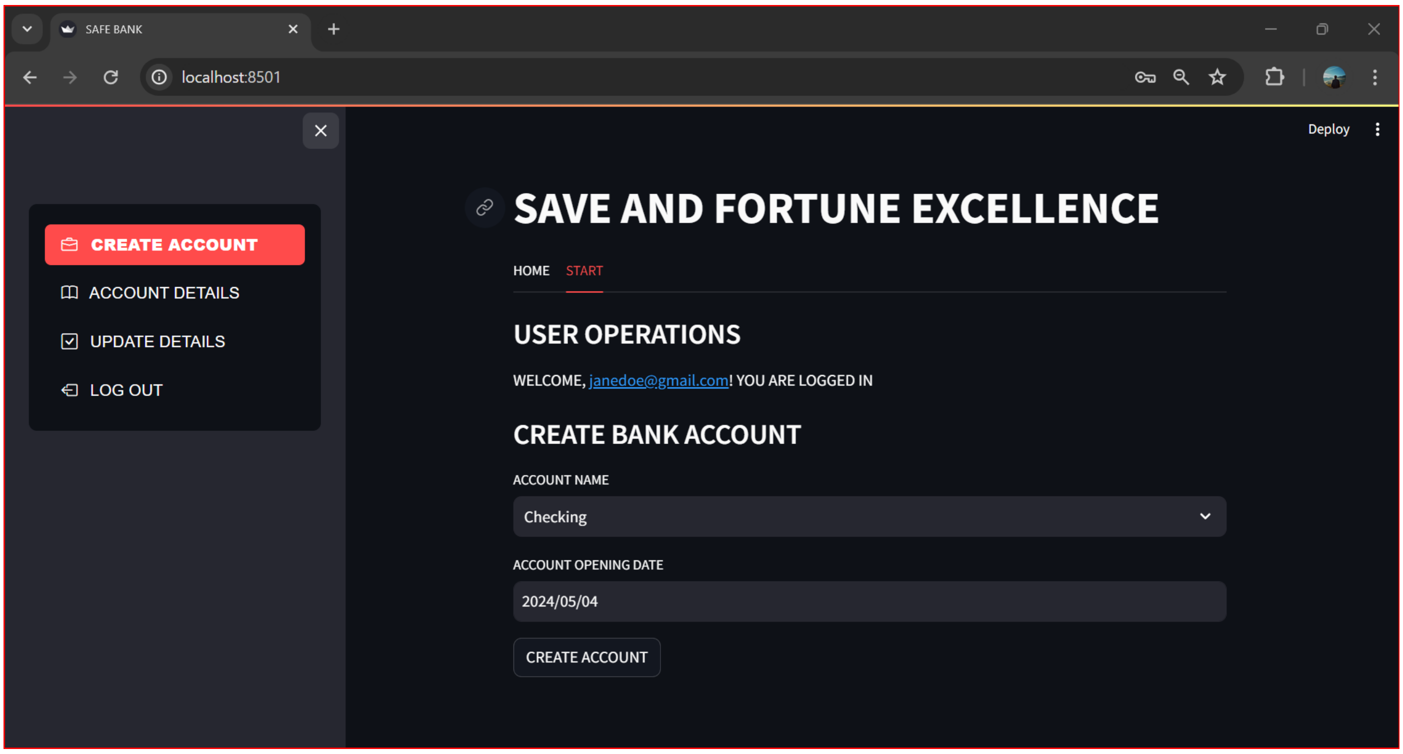
ENTER STATE NAME 0/40

ENTER COUNTRY NAME 0/40

ENTER ZIP CODE

0

Sign Up



SAFE BANK

localhost:8501

Deploy

CREATE ACCOUNT
ACCOUNT DETAILS
UPDATE DETAILS
LOG OUT

SAVE AND FORTUNE EXCELLENCE

HOME START

USER OPERATIONS

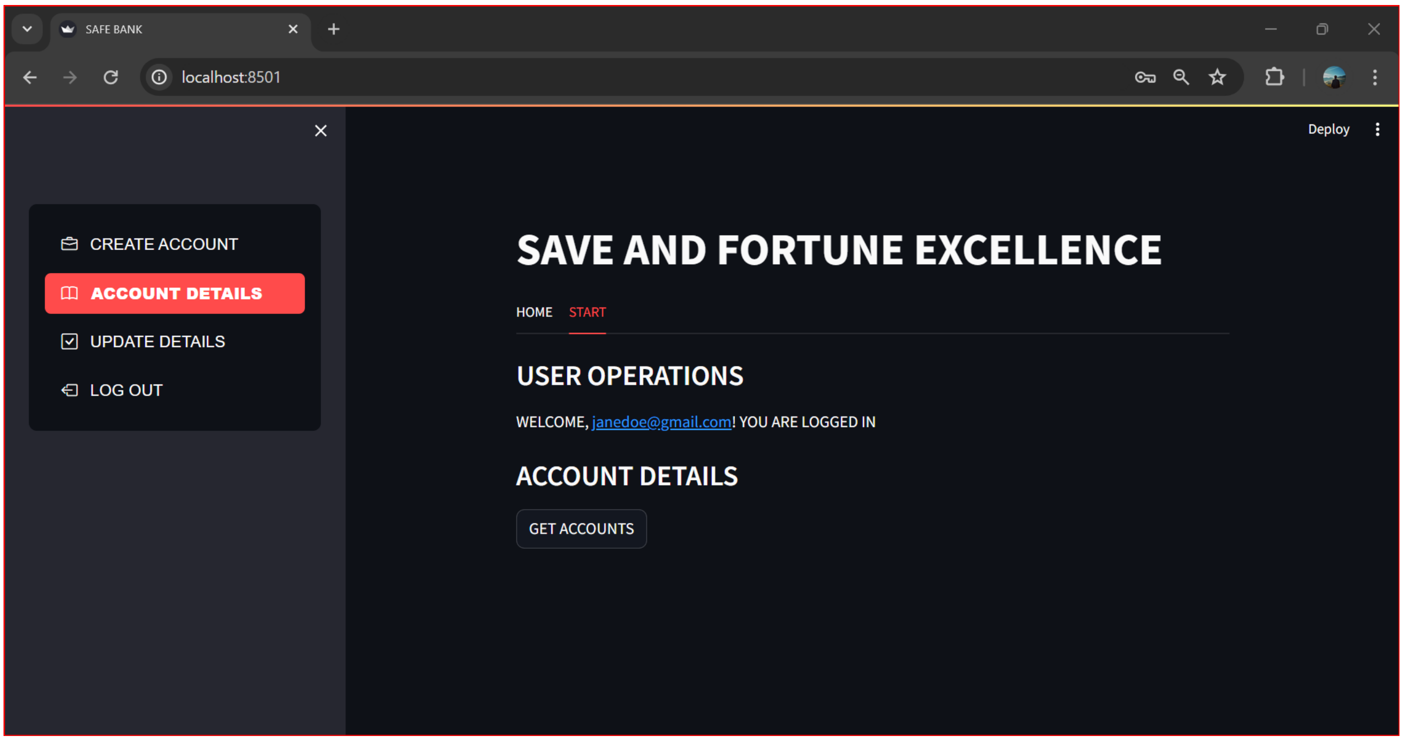
WELCOME, janedoe@gmail.com! YOU ARE LOGGED IN

CREATE BANK ACCOUNT

ACCOUNT NAME: Checking

ACCOUNT OPENING DATE: 2024/05/04

CREATE ACCOUNT



SAFE BANK

localhost:8501

Deploy

CREATE ACCOUNT
ACCOUNT DETAILS
UPDATE DETAILS
LOG OUT

SAVE AND FORTUNE EXCELLENCE

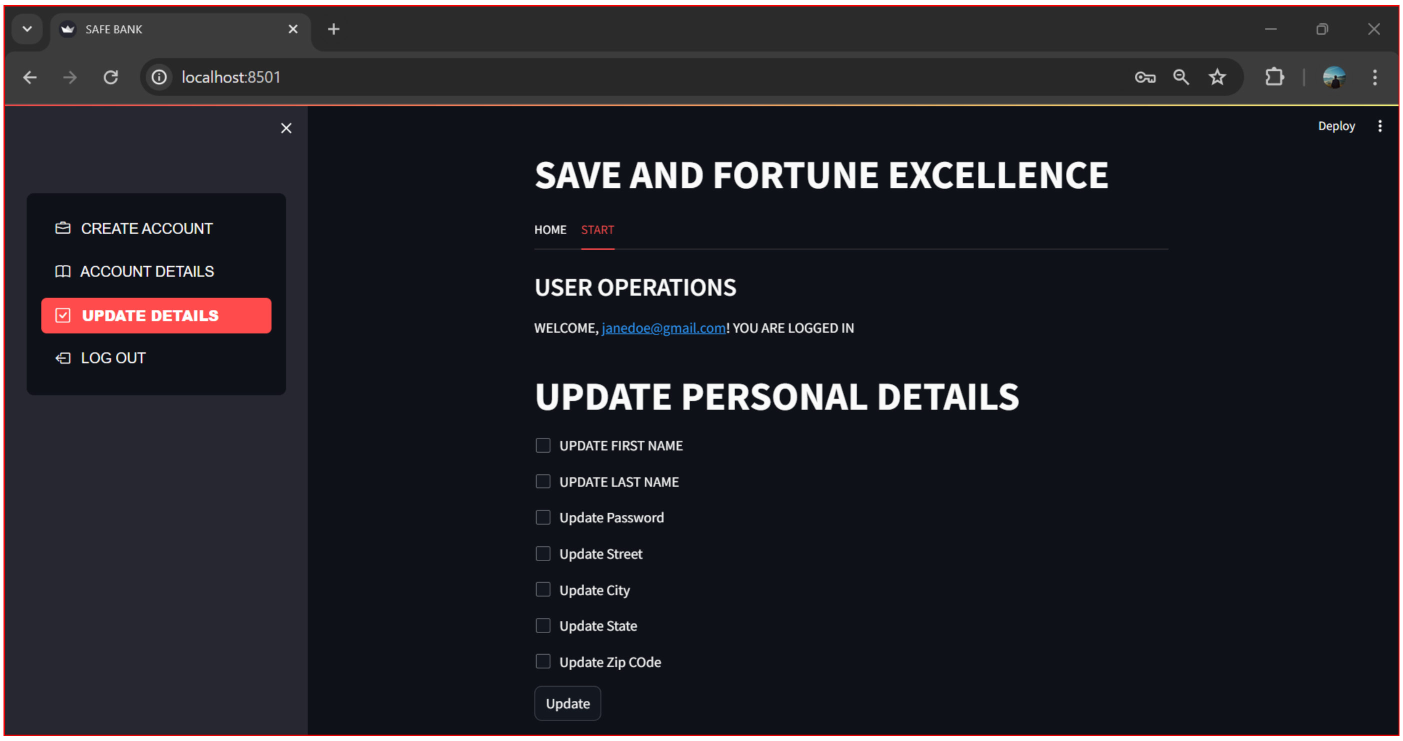
HOME START

USER OPERATIONS

WELCOME, janedoe@gmail.com! YOU ARE LOGGED IN

ACCOUNT DETAILS

GET ACCOUNTS


SAFE BANK

localhost:8501

Deploy

CREATE ACCOUNT
ACCOUNT DETAILS
UPDATE DETAILS
LOG OUT

SAVE AND FORTUNE EXCELLENCE

HOME START

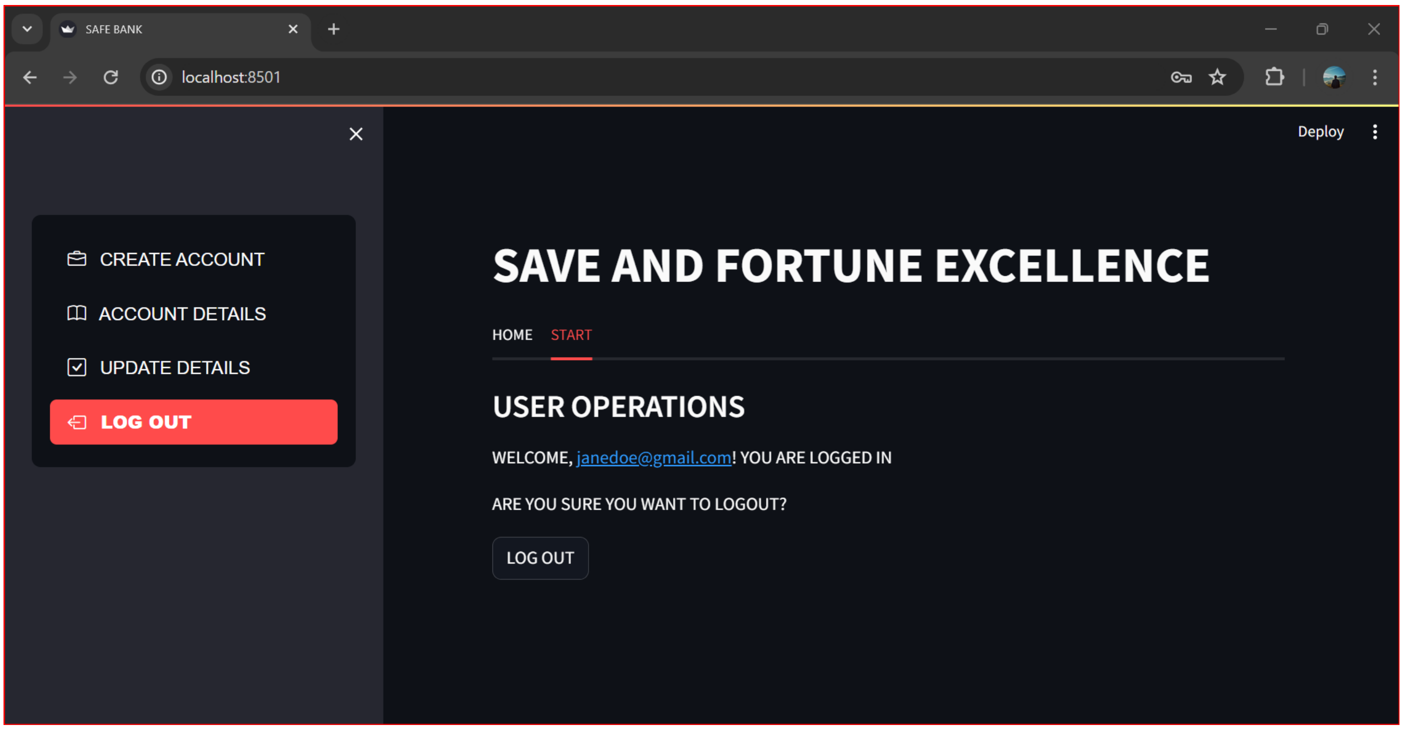
USER OPERATIONS

WELCOME, janedoe@gmail.com! YOU ARE LOGGED IN

UPDATE PERSONAL DETAILS

- [ ] UPDATE FIRST NAME
- [ ] UPDATE LAST NAME
- [ ] Update Password
- [ ] Update Street
- [ ] Update City
- [ ] Update State
- [ ] Update Zip COde

Update


SAFE BANK

localhost:8501

Deploy

CREATE ACCOUNT
ACCOUNT DETAILS
UPDATE DETAILS
LOG OUT

SAVE AND FORTUNE EXCELLENCE

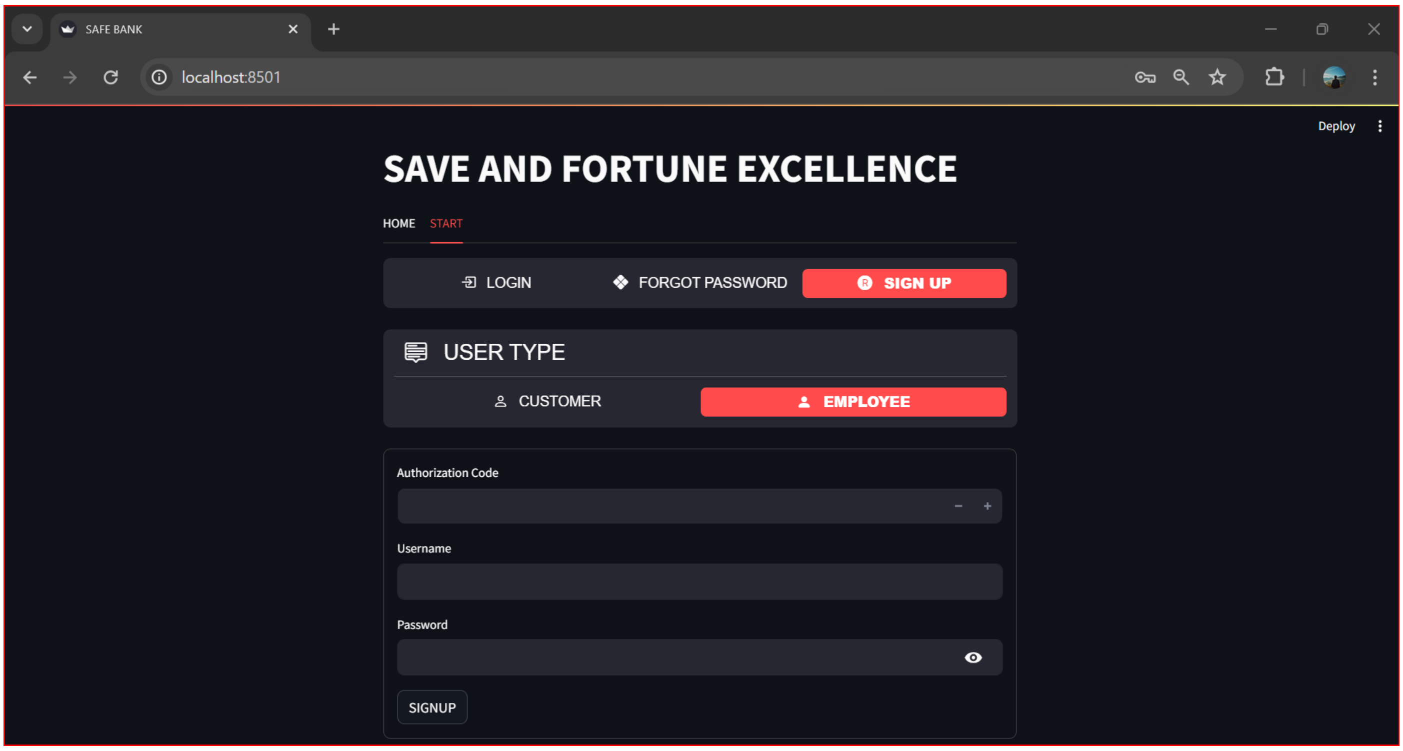
HOME START

USER OPERATIONS

WELCOME, janedoe@gmail.com! YOU ARE LOGGED IN

ARE YOU SURE YOU WANT TO LOGOUT?

LOG OUT



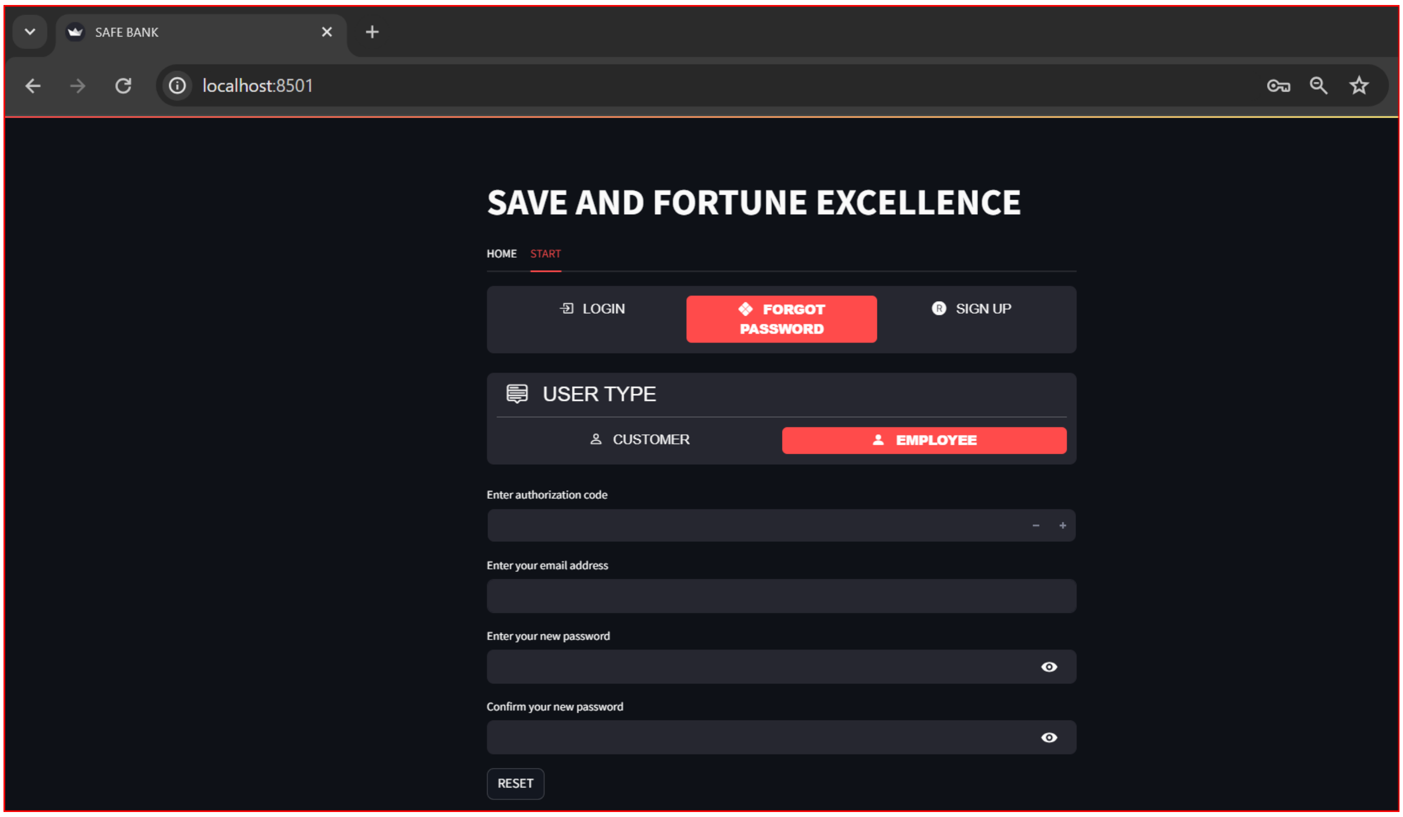
SAVE AND FORTUNE EXCELLENCE

USER TYPE

Authorization Code

Username

Password



SAVE AND FORTUNE EXCELLENCE

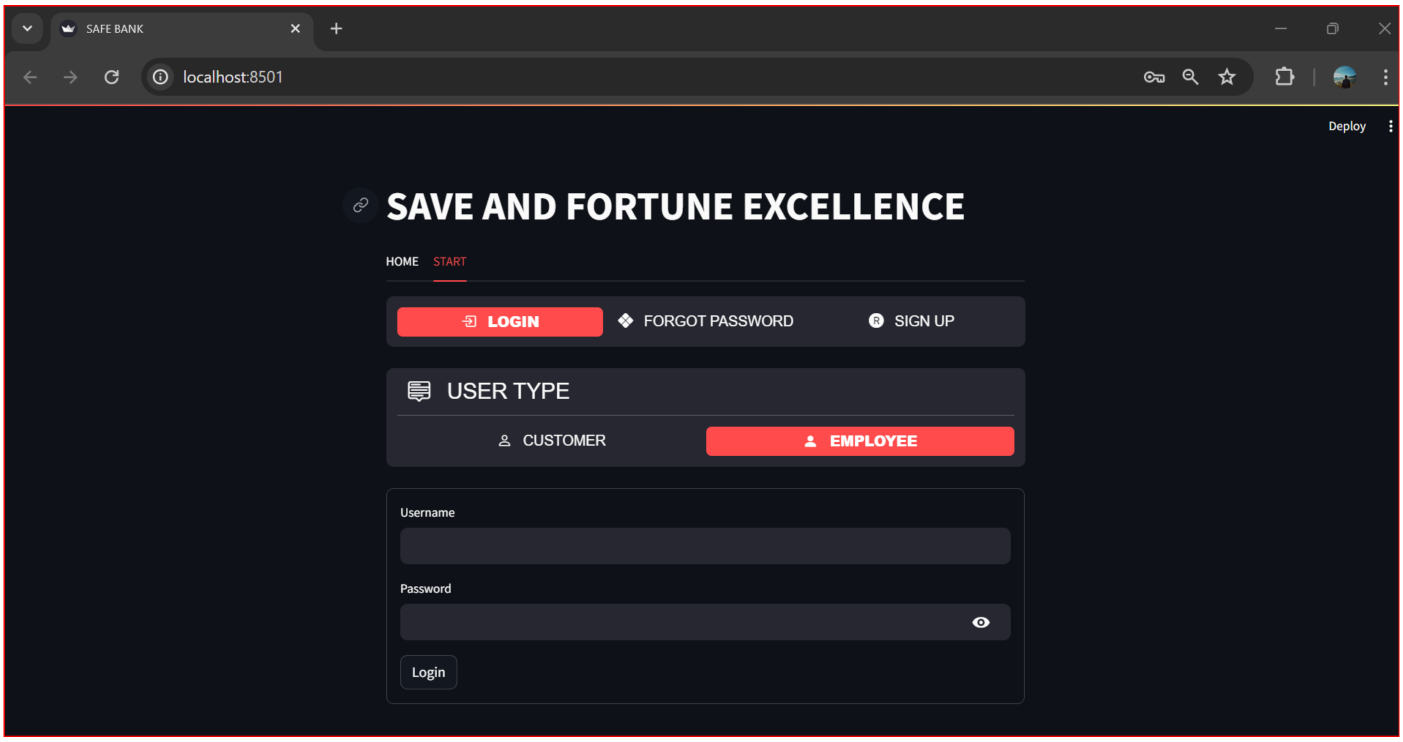
USER TYPE

Enter authorization code

Enter your email address

Enter your new password

Confirm your new password

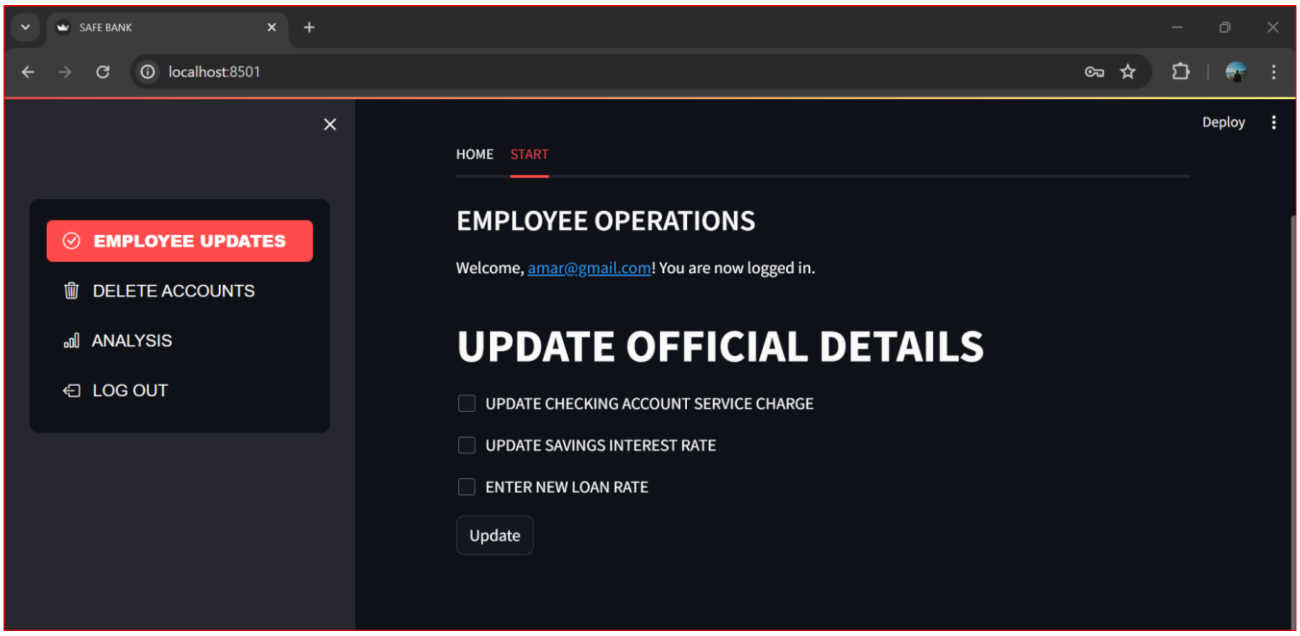


SAVE AND FORTUNE EXCELLENCE

USER TYPE

Username

Password

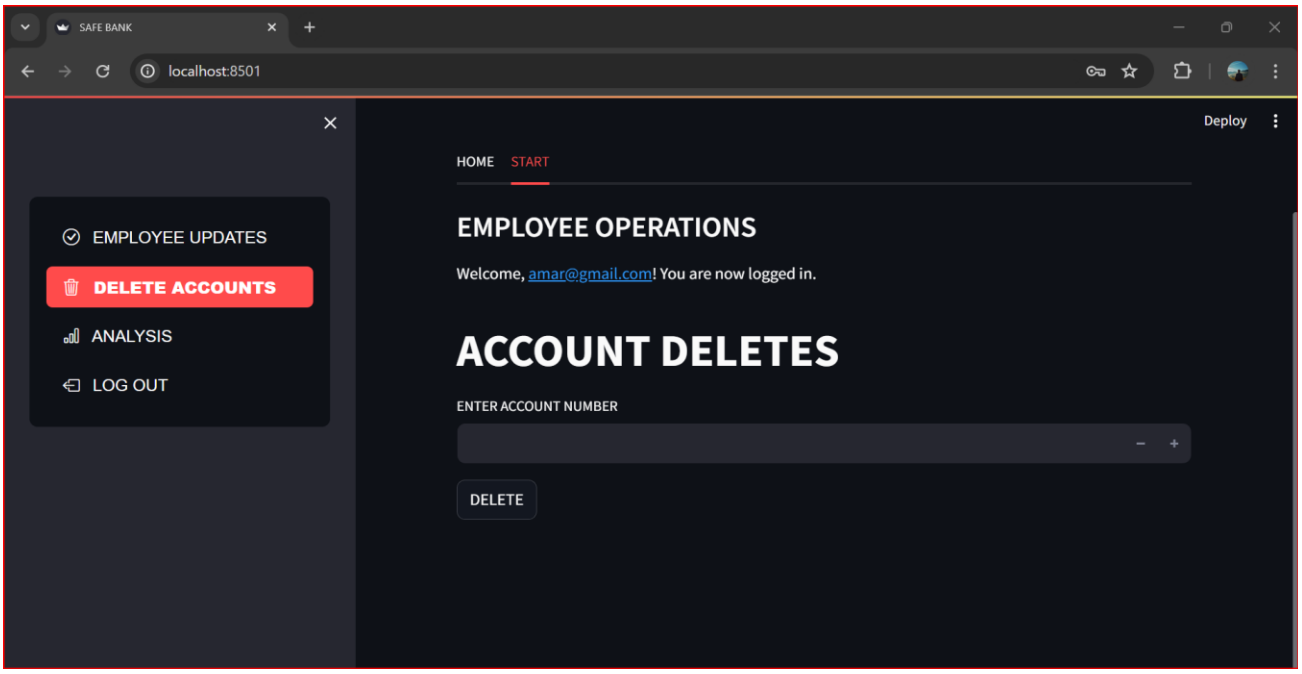


EMPLOYEE OPERATIONS

Welcome, amar@gmail.com! You are now logged in.

UPDATE OFFICIAL DETAILS

- [ ] UPDATE CHECKING ACCOUNT SERVICE CHARGE
- [ ] UPDATE SAVINGS INTEREST RATE
- [ ] ENTER NEW LOAN RATE



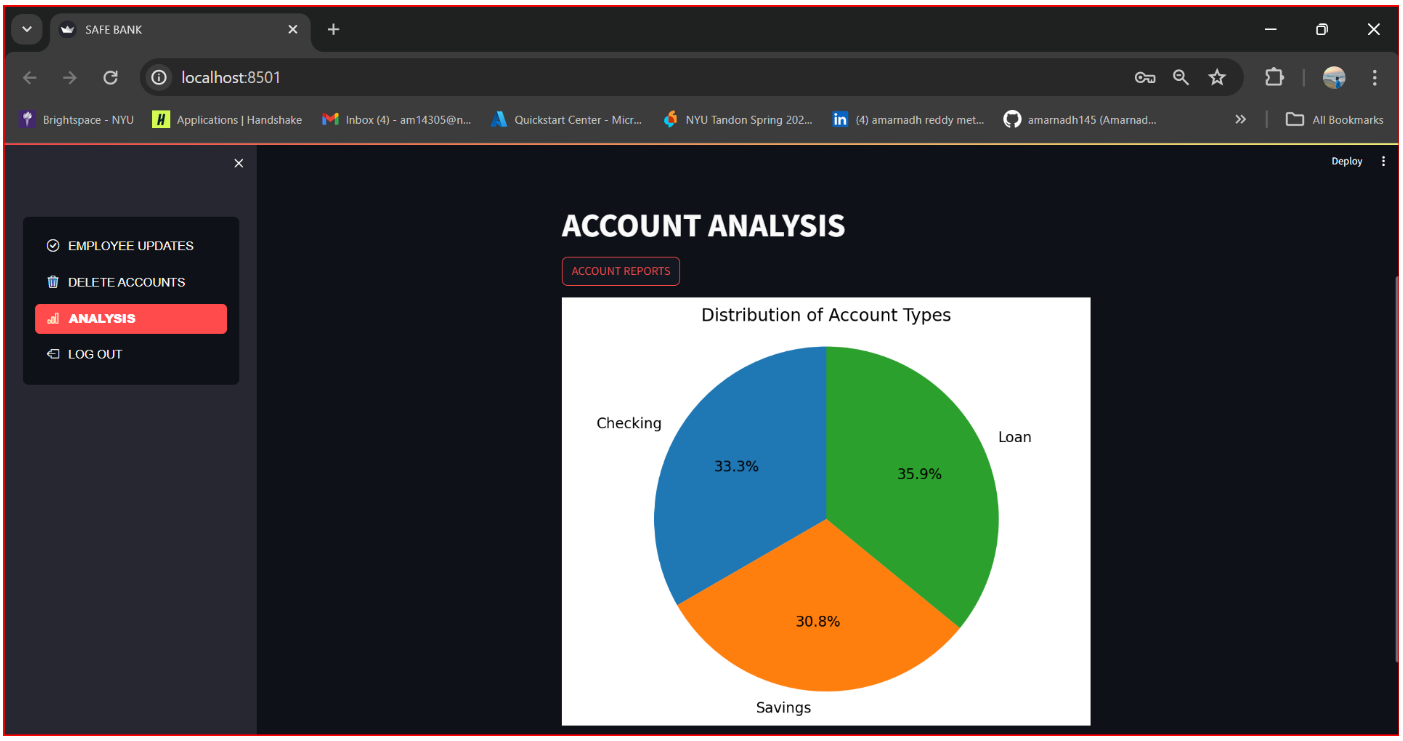
EMPLOYEE OPERATIONS

Welcome, amar@gmail.com! You are now logged in.

ACCOUNT DELETES

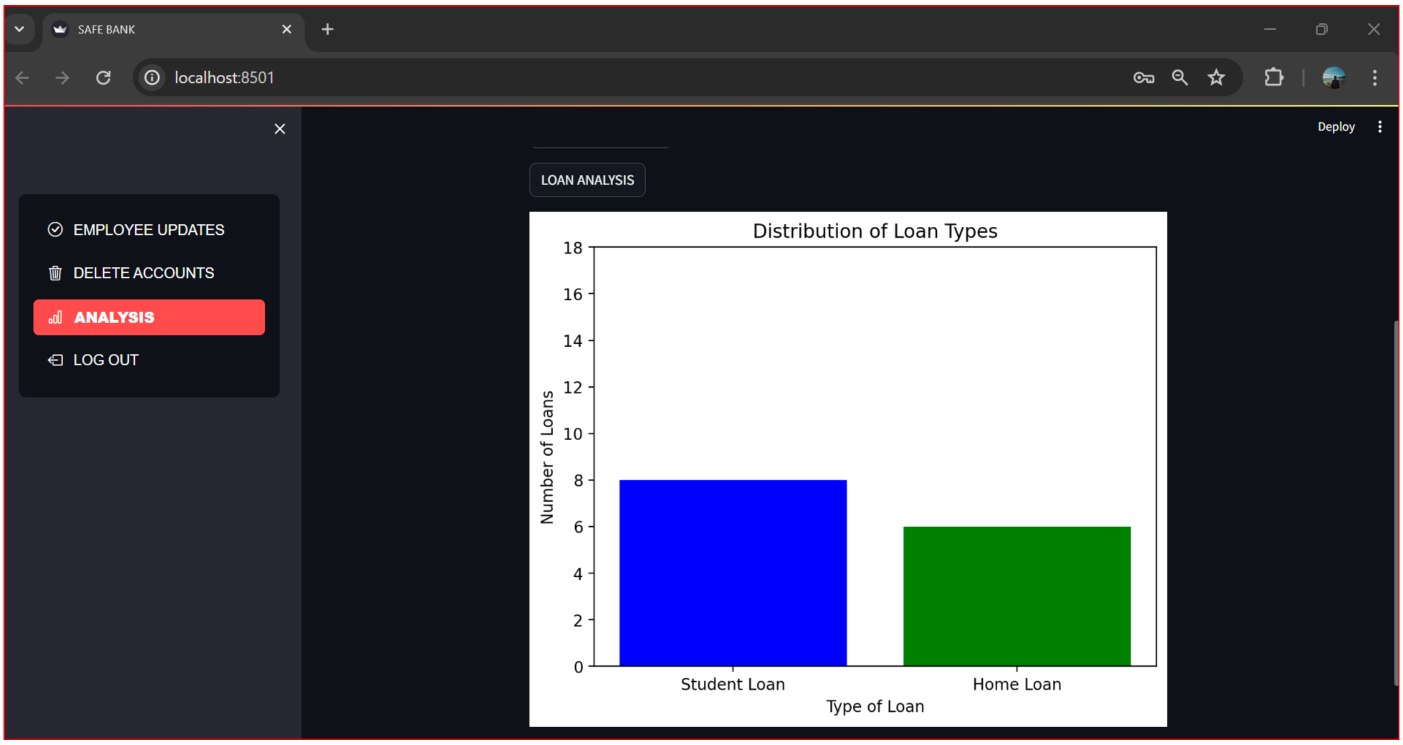
ENTER ACCOUNT NUMBER

DELETE



ACCOUNT ANALYSIS

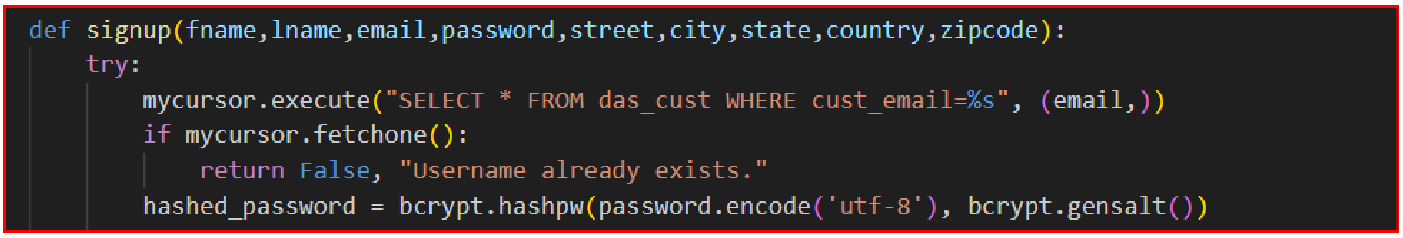
ACCOUNT REPORTS



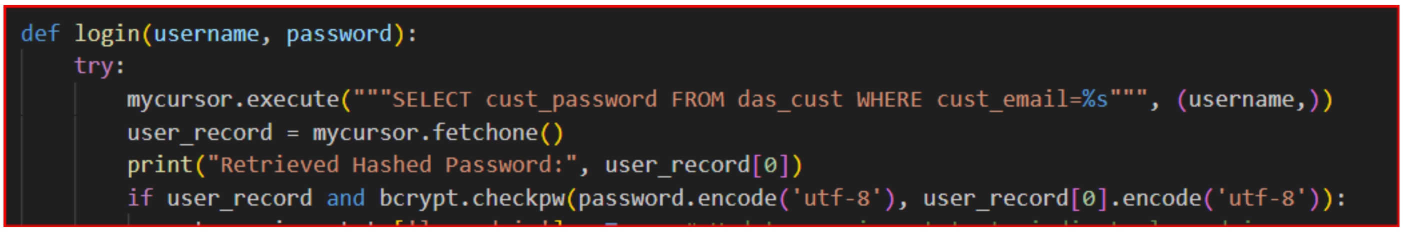
SECURITY FEATURES

(1) PASSWORD ENCRYPTION

The bcrypt.hashpw() function securely hashes passwords using bcrypt, incorporating a randomly generated salt to enhance security. On the other hand, bcrypt.checkpw() compares passwords against their hashed counterparts, ensuring their validity without exposing sensitive information. Together, these functions offer robust password hashing and verification, crucial for safeguarding user credentials in applications against various security threats.



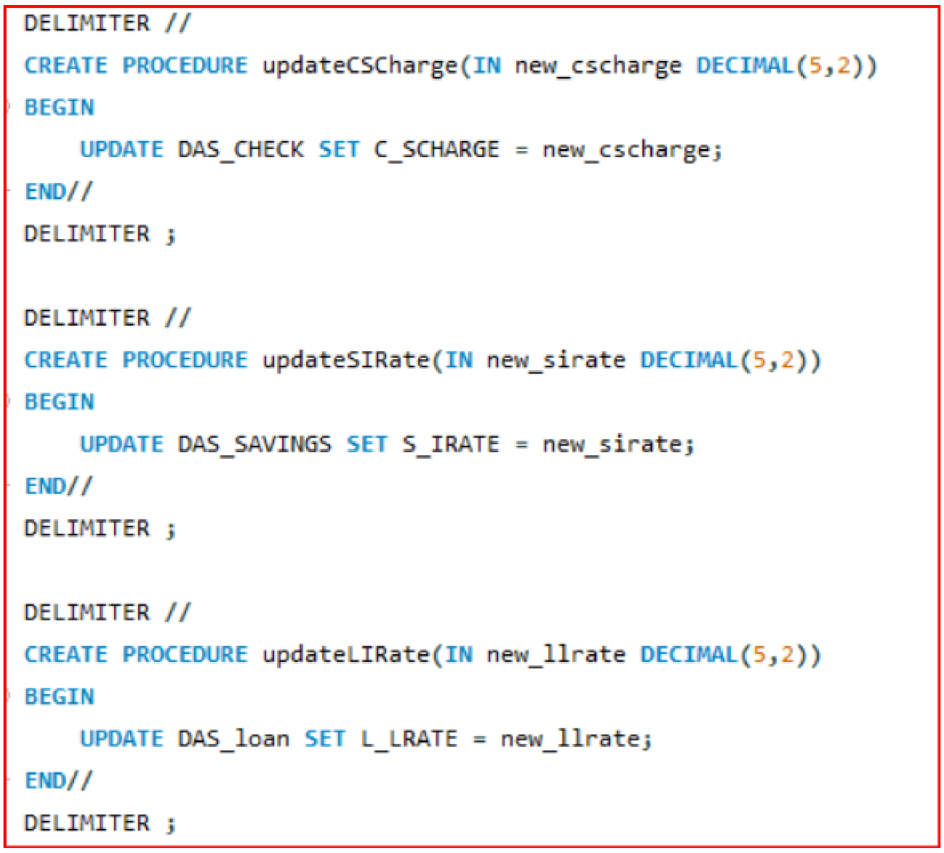
```
def signup(fname,lname,email,password,street,city,state,country,zipcode):
    try:
        mycursor.execute("SELECT * FROM das_cust WHERE cust_email=%s", (email,))
        if mycursor.fetchone():
            return False, "Username already exists."
        hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
```



```
def login(username, password):
    try:
        mycursor.execute("""SELECT cust_password FROM das_cust WHERE cust_email=%s""", (username,))
        user_record = mycursor.fetchone()
        print("Retrieved Hashed Password:", user_record[0])
        if user_record and bcrypt.checkpw(password.encode('utf-8'), user_record[0].encode('utf-8')):
```

(2) STORED PROCEDURES TO PREVENT SQL INJECTION

Implementing stored procedures for admin functionalities such as changing checking account service charge, savings account interest rate and loan accounts interest rate is a proactive security measure in database management. These procedures help safeguard against potential security threats like SQL injection attacks by encapsulating SQL logic within predefined procedures. By utilizing stored procedures, input parameters are validated within the procedure itself, reducing the risk of unauthorized or malicious database operations. Furthermore, stored procedures can enforce access control, limiting direct user access to sensitive data and ensuring that only authorized users with appropriate privileges can execute these administrative functions. This approach enhances overall database security and maintains the integrity of the system, offering a layer of protection against various security vulnerabilities.



```
DELIMITER //
CREATE PROCEDURE updateCSCharge(IN new_cscharge DECIMAL(5,2))
BEGIN
    UPDATE DAS_CHECK SET C_SCHARGE = new_cscharge;
END//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE updateSIRate(IN new_sirate DECIMAL(5,2))
BEGIN
    UPDATE DAS_SAVINGS SET S_IRATE = new_sirate;
END//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE updateLIRate(IN new_llrate DECIMAL(5,2))
BEGIN
    UPDATE DAS_loan SET L_LRATE = new_llrate;
END//
DELIMITER ;
```

LESSONS LEARNED

ADAPTABILITY IN TECHNOLOGY CHOICES

The need to switch from PHP to Python and Streamlit highlighted the importance of selecting technologies that align with the team's skill set. This experience underscores the value of flexibility and adaptability in technology choices, especially in a team environment.

IMPORTANCE OF PRELIMINARY SKILL ASSESSMENT

The challenge with PHP due to a lack of familiarity among team members stresses the importance of assessing team members' skills and experience before deciding on the technology stack.

CONTINUOUS LEARNING

The project illustrated the necessity and value of continuous learning and upskilling, especially in a field as dynamic as software development.

WHAT WENT WELL

DATABASE RECONSTRUCTION

Successfully reconstructing the database to better suit the project's needs was a significant achievement. It demonstrates the team's ability to identify and rectify structural issues, leading to a more efficient and effective database design.

ADOPTION OF STREAMLIT

Leveraging Python and Streamlit for web development was a strategic move. It played to the team's strengths and allowed for the efficient creation of a user-friendly web interface. Streamlit's ease of use for building interactive web applications made it an excellent choice for the project.

TEAM COLLABORATION AND PROBLEM-SOLVING

The team's ability to identify issues with the initial approach and pivot to a more suitable solution is commendable. It reflects strong problem-solving skills and effective collaboration.

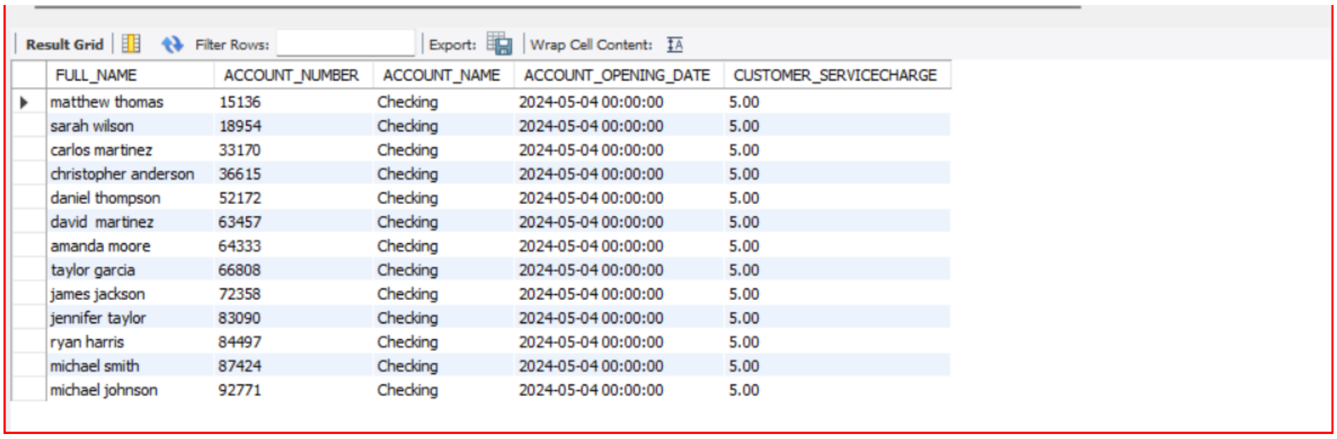
CONCLUSION

This project was a practical learning experience in many aspects of software development, from technology selection and team management to problem-solving and adaptability. The success in reconstructing the database and effectively utilizing Python and Streamlit for web development, despite the initial setbacks, is a testament to the team's resilience and capacity to overcome challenges. Moving forward, these lessons can be applied to future projects for more streamlined and effective project execution.

BUSINESS ANALYSIS WITH 6 SQL STATEMENTS USING PROJECT DATA

[1] Table joins with at least 3 tables in join

```
SELECT concat(b.cust_fname," ",b.cust_lname) as FULL_NAME,
A.A_NUMBER AS ACCOUNT_NUMBER,A.A_NAME AS ACCOUNT_NAME,A_DATE AS
ACCOUNT_OPENING_DATE,C.C_SCHARGE AS CUSTOMER_SERVICECHARGE
FROM DAS_ACCOUNT A JOIN DAS_CUST B ON A.CUST_ID=B.CUST_ID JOIN DAS_CHECK C ON
A.A_NUMBER=C.A_NUMBER where A.a_type='C';
```

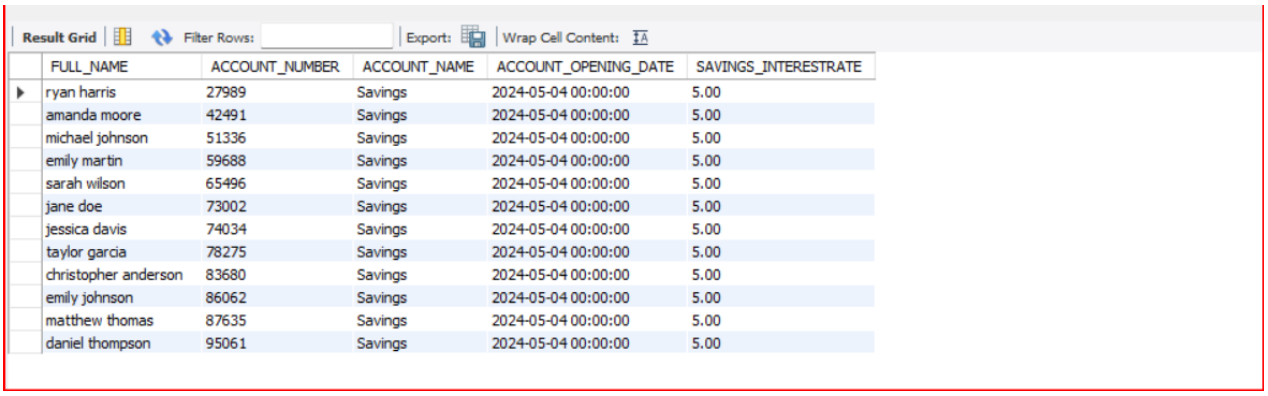


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FULL_NAME	ACCOUNT_NUMBER	ACCOUNT_NAME	ACCOUNT_OPENING_DATE	CUSTOMER_SERVICECHARGE
matthew thomas	15136	Checking	2024-05-04 00:00:00	5.00
sarah wilson	18954	Checking	2024-05-04 00:00:00	5.00
carlos martinez	33170	Checking	2024-05-04 00:00:00	5.00
christopher anderson	36615	Checking	2024-05-04 00:00:00	5.00
daniel thompson	52172	Checking	2024-05-04 00:00:00	5.00
david martinez	63457	Checking	2024-05-04 00:00:00	5.00
amanda moore	64333	Checking	2024-05-04 00:00:00	5.00
taylor garcia	66808	Checking	2024-05-04 00:00:00	5.00
james jackson	72358	Checking	2024-05-04 00:00:00	5.00
jennifer taylor	83090	Checking	2024-05-04 00:00:00	5.00
ryan harris	84497	Checking	2024-05-04 00:00:00	5.00
michael smith	87424	Checking	2024-05-04 00:00:00	5.00
michael johnson	92771	Checking	2024-05-04 00:00:00	5.00

To obtain the account holder name and their corresponding checking accounts. we join customer table with the account table to obtain all the checking accounts and then join with checking table to get the details such as service charge of the checking account.

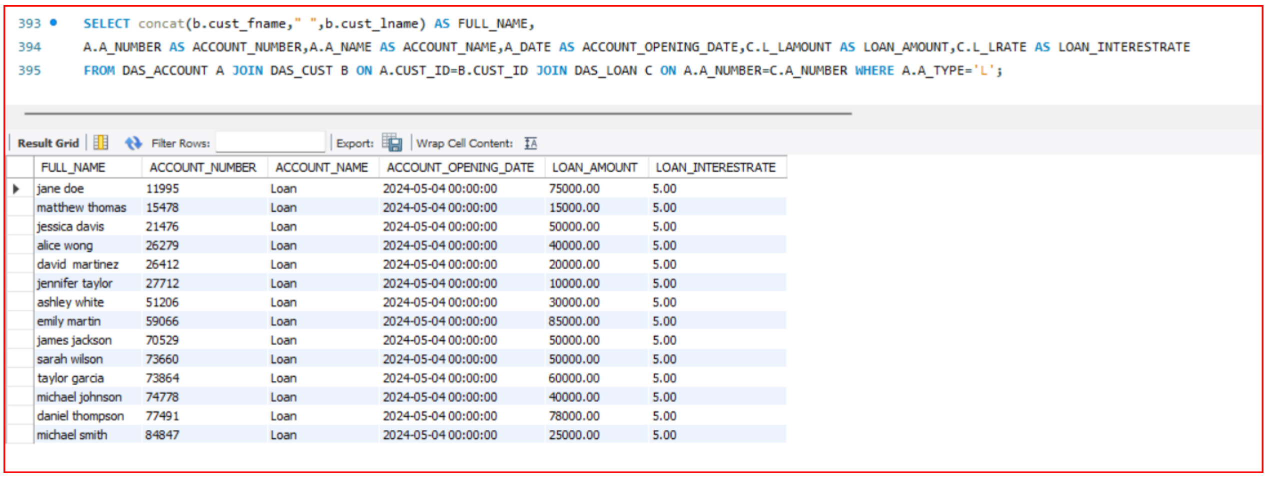
SELECT concat(b.cust_fname," ",b.cust_lname) as FULL_NAME,
A.A_NUMBER as ACCOUNT_NUMBER,A.A_NAME AS ACCOUNT_NAME,A_DATE AS ACCOUNT_OPENING_DATE,C.S_IRATE AS SAVINGS_INTERESTRATE
FROM DAS_ACCOUNT A JOIN DAS_CUST B ON A.CUST_ID=B.CUST_ID JOIN DAS_SAVINGS C ON A.A_NUMBER=C.A_NUMBER WHERE A.A_TYPE='S';



FULL_NAME	ACCOUNT_NUMBER	ACCOUNT_NAME	ACCOUNT_OPENING_DATE	SAVINGS_INTERESTRATE
ryan harris	27989	Savings	2024-05-04 00:00:00	5.00
amanda moore	42491	Savings	2024-05-04 00:00:00	5.00
michael johnson	51336	Savings	2024-05-04 00:00:00	5.00
emily martin	59688	Savings	2024-05-04 00:00:00	5.00
sarah wilson	65496	Savings	2024-05-04 00:00:00	5.00
jane doe	73002	Savings	2024-05-04 00:00:00	5.00
jessica davis	74034	Savings	2024-05-04 00:00:00	5.00
taylor garcia	78275	Savings	2024-05-04 00:00:00	5.00
christopher anderson	83680	Savings	2024-05-04 00:00:00	5.00
emily johnson	86062	Savings	2024-05-04 00:00:00	5.00
matthew thomas	87635	Savings	2024-05-04 00:00:00	5.00
daniel thompson	95061	Savings	2024-05-04 00:00:00	5.00

To obtain the account holder name and their corresponding savings accounts. we join customer table with the account table to obtain all the savings accounts and then join with savings table to get the details such as savings interest rate of the savings account.

SELECT concat(b.cust_fname," ",b.cust_lname) AS FULL_NAME,
A.A_NUMBER AS ACCOUNT_NUMBER,A.A_NAME AS ACCOUNT_NAME,A_DATE AS ACCOUNT_OPENING_DATE,C.L_LAMOUNT AS LOAN_AMOUNT,C.L_LRATE AS LOAN_INTERESTRATE
FROM DAS_ACCOUNT A JOIN DAS_CUST B ON A.CUST_ID=B.CUST_ID JOIN DAS_LOAN C ON A.A_NUMBER=C.A_NUMBER WHERE A.A_TYPE='L';



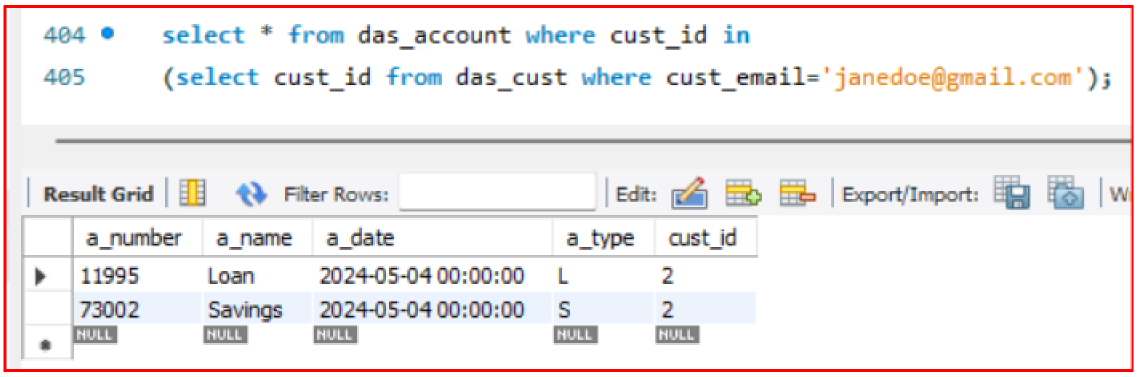
```
SELECT concat(b.cust_fname," ",b.cust_lname) AS FULL_NAME,
A.A_NUMBER AS ACCOUNT_NUMBER,A.A_NAME AS ACCOUNT_NAME,A_DATE AS ACCOUNT_OPENING_DATE,C.L_LAMOUNT AS LOAN_AMOUNT,C.L_LRATE AS LOAN_INTERESTRATE
FROM DAS_ACCOUNT A JOIN DAS_CUST B ON A.CUST_ID=B.CUST_ID JOIN DAS_LOAN C ON A.A_NUMBER=C.A_NUMBER WHERE A.A_TYPE='L';
```

FULL_NAME	ACCOUNT_NUMBER	ACCOUNT_NAME	ACCOUNT_OPENING_DATE	LOAN_AMOUNT	LOAN_INTERESTRATE
jane doe	11995	Loan	2024-05-04 00:00:00	75000.00	5.00
matthew thomas	15478	Loan	2024-05-04 00:00:00	15000.00	5.00
jessica davis	21476	Loan	2024-05-04 00:00:00	50000.00	5.00
alice wong	26279	Loan	2024-05-04 00:00:00	40000.00	5.00
david martinez	26412	Loan	2024-05-04 00:00:00	20000.00	5.00
jennifer taylor	27712	Loan	2024-05-04 00:00:00	10000.00	5.00
ashley white	51206	Loan	2024-05-04 00:00:00	30000.00	5.00
emily martin	59066	Loan	2024-05-04 00:00:00	85000.00	5.00
james jackson	70529	Loan	2024-05-04 00:00:00	50000.00	5.00
sarah wilson	73660	Loan	2024-05-04 00:00:00	50000.00	5.00
taylor garcia	73864	Loan	2024-05-04 00:00:00	60000.00	5.00
michael johnson	74778	Loan	2024-05-04 00:00:00	40000.00	5.00
daniel thompson	77491	Loan	2024-05-04 00:00:00	78000.00	5.00
michael smith	84847	Loan	2024-05-04 00:00:00	25000.00	5.00

To obtain the account holder name and their corresponding loan accounts. we join customer table with the account table to obtain all the loan accounts and then join with loan table to get the details such as loan interest rate and loan amount of the loan account.

Multi-row subquery

select * from das_account where cust_id in
(select cust_id from das_cust where cust_email='janedoe@gmail.com');



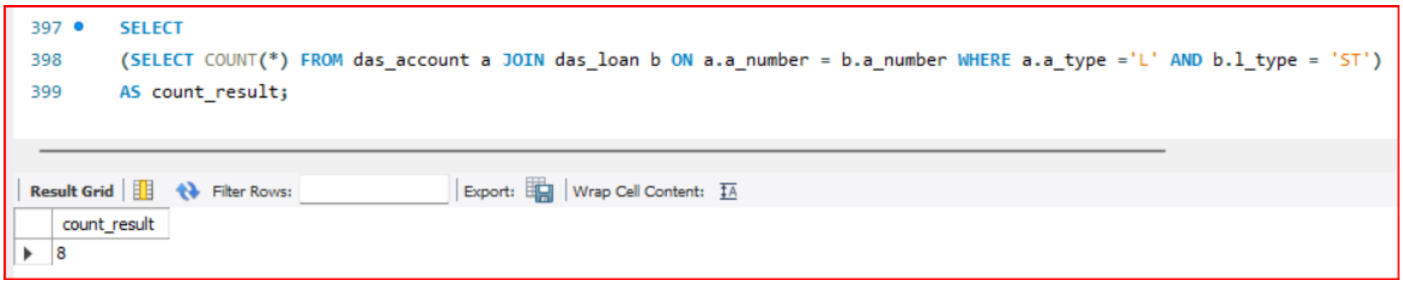
```
select * from das_account where cust_id in
(select cust_id from das_cust where cust_email='janedoe@gmail.com');
```

a_number	a_name	a_date	a_type	cust_id
11995	Loan	2024-05-04 00:00:00	L	2
73002	Savings	2024-05-04 00:00:00	S	2
NULL	NULL	NULL	NULL	NULL

To obtain all the details of all types of account related to a customer. We find the cust_id from customer table using the email address then retrieving all the details of all type of accounts that he own using account table.

Correlated subquery

SELECT (SELECT COUNT(*) FROM das_account a JOIN das_loan b ON a.a_number = b.a_number WHERE a.a_type = ‘L’ AND b.l_type = ‘ST’) AS count_result;

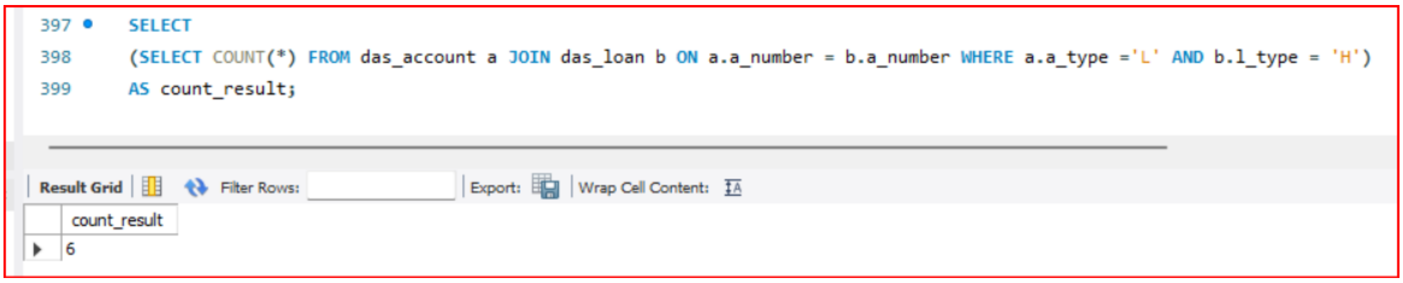


```
SELECT
(SELECT COUNT(*) FROM das_account a JOIN das_loan b ON a.a_number = b.a_number WHERE a.a_type ='L' AND b.l_type = 'ST')
AS count_result;
```

count_result
8

To get the analysis of how many loan accounts are student loan from all the loan accounts. We join accounts table with loan to find all the student loan accounts among all.

SELECT (SELECT COUNT(*) FROM das_account a JOIN das_loan b ON a.a_number = b.a_number WHERE a.a_type ='L' AND b.l_type = 'H') AS count_result;



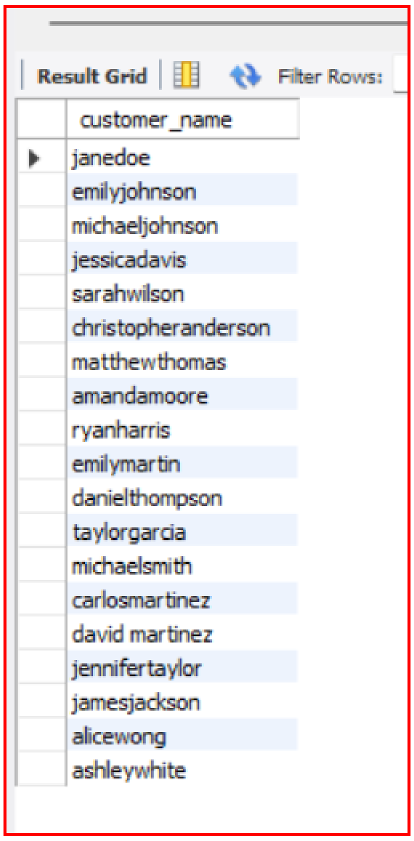
```
SELECT
(SELECT COUNT(*) FROM das_account a JOIN das_loan b ON a.a_number = b.a_number WHERE a.a_type ='L' AND b.l_type = 'H')
AS count_result;
```

count_result
6

To get the analysis of how many loan accounts are home loan from all the loan accounts. We join accounts table with loan to find all the home loan accounts among all.

SET query:

Select concat(a.cust_fname,"",a.cust_lname) as customer_name from das_cust a join das_account b where a.cust_id=b.cust_id and b.a_type='S' Union
Select concat(a.cust_fname,"",a.cust_lname) as customer_name from das_cust a join das_account b where a.cust_id=b.cust_id and b.a_type='C' union
select concat(a.cust_fname,"",a.cust_lname) as customer_name from das_cust a join das_account b where a.cust_id=b.cust_id and b.a_type='L';

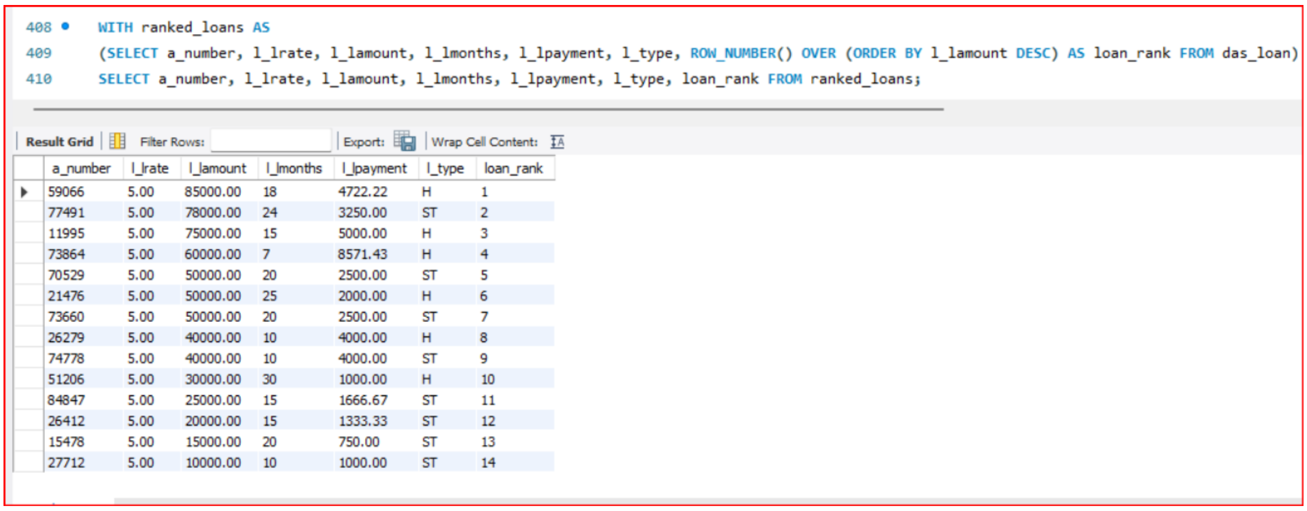


customer_name
janedoe
emilyjohnson
michaeljohnson
jessicadavis
sarahwilson
christopheranderson
matthewthomas
amandamoore
ryanharris
emilymartin
danielthompson
taylorgarcia
michaelsmith
carlosmartinez
david martinez
jennifertaylor
jamesjackson
alicewong
ashleywhite

This is for our business analysis. To get a comprehensive list of all the customers who atleast have an account with the safe bank.

Query with in-line view or WITH clause

WITH ranked_loans AS
(SELECT a_number, l_lrate, l_lamount, l_lmonths, l_lpayment, l_type, ROW_NUMBER() OVER (ORDER BY l_lamount DESC) AS loan_rank FROM das_loan)
SELECT a_number, l_lrate, l_lamount, l_lmonths, l_lpayment, l_type, loan_rank FROM ranked_loans;



```sql
WITH ranked_loans AS
(SELECT a_number, l_lrate, l_lamount, l_lmonths, l_lpayment, l_type, ROW_NUMBER() OVER (ORDER BY l_lamount DESC) AS loan_rank FROM das_loan)
SELECT a_number, l_lrate, l_lamount, l_lmonths, l_lpayment, l_type, loan_rank FROM ranked_loans;
```

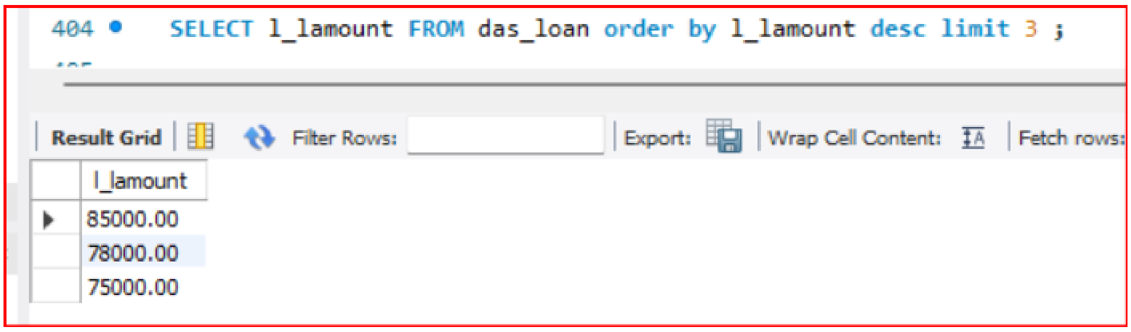
a_number	l_lrate	l_lamount	l_lmonths	l_lpayment	l_type	loan_rank
59066	5.00	85000.00	18	4722.22	H	1
77491	5.00	78000.00	24	3250.00	ST	2
11995	5.00	75000.00	15	5000.00	H	3
73864	5.00	60000.00	7	8571.43	H	4
70529	5.00	50000.00	20	2500.00	ST	5
21476	5.00	50000.00	25	2000.00	H	6
73660	5.00	50000.00	20	2500.00	ST	7
26279	5.00	40000.00	10	4000.00	H	8
74778	5.00	40000.00	10	4000.00	ST	9
51206	5.00	30000.00	30	1000.00	H	10
84847	5.00	25000.00	15	1666.67	ST	11
26412	5.00	20000.00	15	1333.33	ST	12
15478	5.00	15000.00	20	750.00	ST	13
27712	5.00	10000.00	10	1000.00	ST	14

This for our business analysis , to obtain all the loan accounts ranked according to their loan amounts so that the company can have analysis on them.

TOP-N/BOTTOM-N query

In my sql we can get the top and bottom records of a table using limit

Select l_amount from das_loan order by l_lamount desc limit 3;



```sql
SELECT l_lamount FROM das_loan order by l_lamount desc limit 3 ;
```

l_lamount
85000.00
78000.00
75000.00

This for our business analysis , to get the highest loan amount among all the customers