CS-GY 6083
Principles of Database Systems
Section B

NEW YORK UNIVERSITY
Tandon School of Engineering
Spring 2024

Group Project – Part 2 Submission

GROUP MEMBERS

Durga Avinash Kodavalla	|| dk4852	|| N11128244
Chirag Chopra	|| cc7083	|| N18722294
Arushi Ojha	|| ao2618	|| N14118398

Date of Submission
05/05/2024

TABLE OF CONTENTS

1. Summary

Project Context

This report outlines the development of a comprehensive database system and web interface for SAFE Bank. The project was structured into two main segments, initially focusing on the creation of a robust database design, followed by the development of a web interface that facilitates user interactions.

Objectives

The main objective was to engineer a reliable database system to manage detailed customer and account data for SAFE Bank. This includes handling various account types such as checking, savings, and loan accounts, with specific features for student and home loans. The secondary objective was to develop a secure, web-based interface that supports various user operations.

Approach and Technologies Used

Database Design:

The project employed sqlite3 for the database back-end, supported by a schema designed using Oracle SQL Data Modeler to accommodate entities covering customers, accounts, and loans.

Web Interface Development:

A web interface was developed using Django, a high-level Python web framework, ensuring robust support for user authentication, session management, and CRUD operations. This choice was instrumental in facilitating seamless integration with the SQLite database.

Security Measures:

Security was a paramount concern, addressed by encrypting passwords using Fernet module from cryptography library to safeguard user data. Additionally, the application was designed to resist SQL injection attacks, enhancing the overall security posture of the system.

Results and Improvements

The system effectively manages SAFE Bank's data with an emphasis on security and operational efficiency. During the project's progression, initial database designs were iteratively refined to better meet the practical requirements of the bank's day-to-day operations.

Detailed CRUD Operations

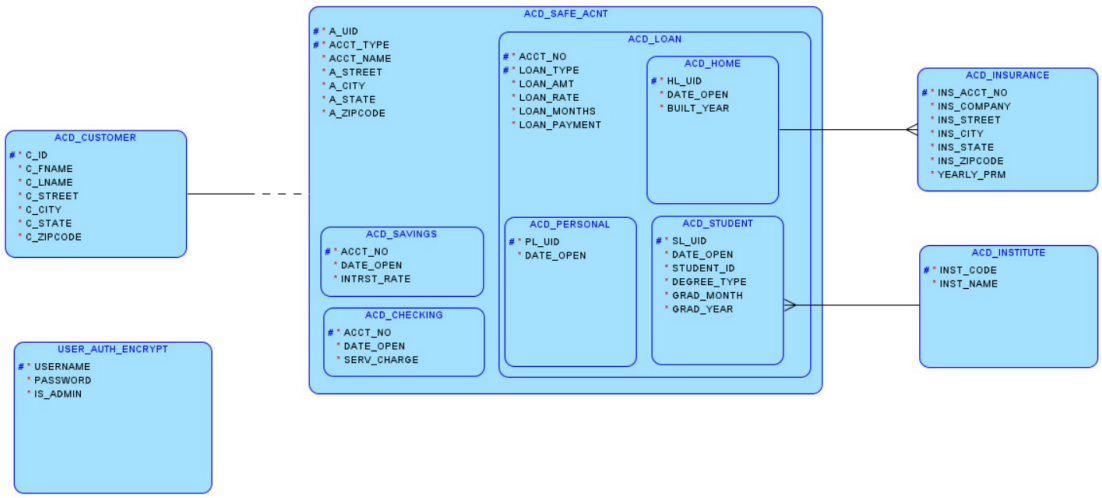
The web interface developed provides comprehensive CRUD (Create, Read, Update, Delete) functionalities, which are essential for managing bank operations:

Create: Admin user can create new customers and admin users
Read: Customers can view information of their bank account
Update: Customers & admin users can update passwords, customer can update their address
Delete: Admin user can delete other admin user accounts and customer accounts

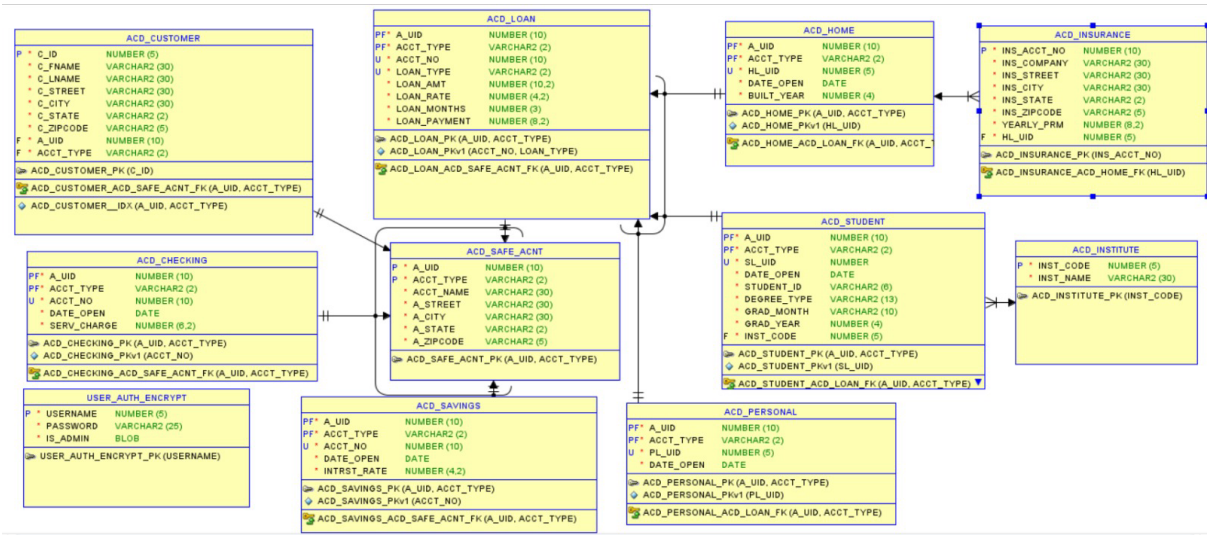
Conclusion

The project successfully melded theoretical database system principles with practical application development, resulting in a functional, secure, and user-friendly banking system. This endeavor not only demonstrated the capability of Django and SQLite in creating effective web applications but also emphasized the importance of security in contemporary software development. The robust implementation of CRUD operations further enhanced the system's utility by ensuring dynamic data management aligned with user needs and banking regulations.

2. Logical E-R Model



3. Relational Model



4. Assumptions

- A customer will have unique account ID (A_UID in ACD_SAFE_ACNT) and will have different account numbers for different types of accounts in it.
- A customer can also have multiple loan types (student/home/personal) at once in a single loan account with their own unique combination of account number and loan type.
- Each type of loan has their unique ID (HL_UID, SL_UID, PL_UID).
- A_UID & ACNT_TYPE in ACD_SAFE_ACNT and ACCT_NO & LOAN_TYPE in ACD_LOAN is a pair primary key to have one/many subtypes condition.
- For home loans, it is assumed to have an insurance (only one & mandatory).
- Each customer in ACD_CUSTOMER is assigned with their unique ID.
- Insurance account number (INS_ACCT_NO in ACD_INSURANCE) is assumed to be unique irrespective of the insurance companies available in the market.
- Applied a check constraint for undergraduate and graduate type of degree, and for all account types.
- Created a new table USERAUTHENCRYPT to store username, passwords (encrypted) of all customer and admin users.

5. Details of Software, Programming Language, Database In Use

For designing our logical and relational database models, we turned to Oracle SQL Data Modeler. This powerful software allowed us to visualize and structure our database efficiently. It also provided the capability to generate DDL (Data Definition Language) code, streamlining the process of implementing our database schema.

For programming, we opted for Python as our primary language. Python's simplicity and versatility made it an ideal choice, especially for web development. Additionally, we utilized Django, a high-level Python web framework, for the development process. Django offers a wide range of built-in features, including an ORM (Object-Relational Mapping) system, which seamlessly integrates with our chosen database.

For databases, we selected SQLite3 [RDBMS] as our database management system. SQLite3, being a lightweight, perfectly suited our project requirements. Its simplicity and ease of use, coupled with Django's excellent support, allowed us to efficiently manage and interact with our data.

6. DDL Code Generated from Relational Model with Check Constraints

```
-- Generated by Oracle SQL Developer Data Modeler 23.1.0.087.0806
--   at:        2024-05-04 16:30:57 GMT-04:00
--   site:      Oracle Database 21c
--   type:      Oracle Database 21c



-- predefined type, no DDL - MDSYS.SDO_GEOMETRY

-- predefined type, no DDL - XMLTYPE

CREATE TABLE acd_checking (
    a_uid       NUMBER(10) NOT NULL,
    acct_type   VARCHAR2(2) NOT NULL,
    acct_no     NUMBER(10) NOT NULL,
    date_open   DATE NOT NULL,
    serv_charge NUMBER(6, 2) NOT NULL
);

COMMENT ON COLUMN acd_checking.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_checking.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_checking.acct_no IS
    'Checking Account Unique Account Number';

COMMENT ON COLUMN acd_checking.date_open IS
    'Account Opening Date';

COMMENT ON COLUMN acd_checking.serv_charge IS
    'Account Service Charge';

ALTER TABLE acd_checking ADD CONSTRAINT acd_checking_pk PRIMARY KEY ( a_uid,
                                                                      acct_type );

ALTER TABLE acd_checking ADD CONSTRAINT acd_checking_pkv1 UNIQUE ( acct_no );

CREATE TABLE acd_customer (
    c_id      NUMBER(5) NOT NULL,
    c_fname   VARCHAR2(30) NOT NULL,
    c_lname   VARCHAR2(30) NOT NULL,
    c_street  VARCHAR2(30) NOT NULL,
    c_city    VARCHAR2(30) NOT NULL,
    c_state   VARCHAR2(2) NOT NULL,
    c_zipcode VARCHAR2(5) NOT NULL,
    a_uid     NUMBER(10) NOT NULL,
    acct_type VARCHAR2(2) NOT NULL
```

```
);

COMMENT ON COLUMN acd_customer.c_id IS
    'Unique Customer ID';

COMMENT ON COLUMN acd_customer.c_fname IS
    'Customer First Name';

COMMENT ON COLUMN acd_customer.c_lname IS
    'Customer Last Name';

COMMENT ON COLUMN acd_customer.c_street IS
    'Customer Street - Address Line 1';

COMMENT ON COLUMN acd_customer.c_city IS
    'Customer City Location';

COMMENT ON COLUMN acd_customer.c_state IS
    'Customer State Code - Ex: New York: NY';

COMMENT ON COLUMN acd_customer.c_zipcode IS
    'Customer Zipcode';

CREATE UNIQUE INDEX acd_customer__idx ON
    acd_customer (
        a_uid
    ASC,
        acct_type
    ASC );

ALTER TABLE acd_customer ADD CONSTRAINT acd_customer_pk PRIMARY KEY ( c_id );

CREATE TABLE acd_home (
    a_uid      NUMBER(10) NOT NULL,
    acct_type  VARCHAR2(2) NOT NULL,
    hl_uid     NUMBER(5) NOT NULL,
    date_open  DATE NOT NULL,
    built_year NUMBER(4) NOT NULL
);

COMMENT ON COLUMN acd_home.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_home.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_home.hl_uid IS
    'Home Loan Unique Loan ID';

COMMENT ON COLUMN acd_home.date_open IS
    'Home Loan Account Opening Date';
```

```
COMMENT ON COLUMN acd_home.built_year IS
    'Home Built Year';

ALTER TABLE acd_home ADD CONSTRAINT acd_home_pk PRIMARY KEY ( a_uid,
                                                             acct_type );

ALTER TABLE acd_home ADD CONSTRAINT acd_home_pkv1 UNIQUE ( hl_uid );

CREATE TABLE acd_institute (
    inst_code NUMBER(5) NOT NULL,
    inst_name VARCHAR2(30) NOT NULL
);

COMMENT ON COLUMN acd_institute.inst_code IS
    'Unique Institute Code';

COMMENT ON COLUMN acd_institute.inst_name IS
    'Institute Name (University Name)';

ALTER TABLE acd_institute ADD CONSTRAINT acd_institute_pk PRIMARY KEY ( inst_code
);

CREATE TABLE acd_insurance (
    ins_acct_no NUMBER(10) NOT NULL,
    ins_company VARCHAR2(30) NOT NULL,
    ins_street  VARCHAR2(30) NOT NULL,
    ins_city    VARCHAR2(30) NOT NULL,
    ins_state   VARCHAR2(2) NOT NULL,
    ins_zipcode VARCHAR2(5) NOT NULL,
    yearly_prm  NUMBER(8, 2) NOT NULL,
    hl_uid      NUMBER(5) NOT NULL
);

COMMENT ON COLUMN acd_insurance.ins_acct_no IS
    'Home Insurance Unique Account Number';

COMMENT ON COLUMN acd_insurance.ins_company IS
    'Insurance Company Name';

COMMENT ON COLUMN acd_insurance.ins_street IS
    'Insurance Street Address';

COMMENT ON COLUMN acd_insurance.ins_city IS
    'Insurance City Location';

COMMENT ON COLUMN acd_insurance.ins_state IS
    'Insurance State Code- Ex: New York: NY';

COMMENT ON COLUMN acd_insurance.ins_zipcode IS
    'Insurance Company''s Zipcode';
```

```
COMMENT ON COLUMN acd_insurance.yearly_prm IS
    'Insurance Yearly Premium';

ALTER TABLE acd_insurance ADD CONSTRAINT acd_insurance_pk PRIMARY KEY ( ins_acct_no
);

CREATE TABLE acd_loan (
    a_uid        NUMBER(10) NOT NULL,
    acct_type    VARCHAR2(2) NOT NULL,
    acct_no      NUMBER(10) NOT NULL,
    loan_type    VARCHAR2(2) NOT NULL,
    loan_amt     NUMBER(10, 2) NOT NULL,
    loan_rate    NUMBER(4, 2) NOT NULL,
    loan_months  NUMBER(3) NOT NULL,
    loan_payment NUMBER(8, 2) NOT NULL
);

ALTER TABLE acd_loan
    ADD CONSTRAINT ch_inh_acd_loan CHECK ( loan_type IN ( 'HL', 'PL', 'SL' ) );

COMMENT ON COLUMN acd_loan.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_loan.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_loan.acct_no IS
    'Loan Account Unique Account Number';

COMMENT ON COLUMN acd_loan.loan_type IS
    'Loan Type - Student (SL)/ Home (HL)/ Personal (PL)';

COMMENT ON COLUMN acd_loan.loan_amt IS
    'OVERALL LOAN AMOUNT ';

COMMENT ON COLUMN acd_loan.loan_rate IS
    'LOAN INTEREST RATE';

COMMENT ON COLUMN acd_loan.loan_months IS
    'LOAN DURATION';

COMMENT ON COLUMN acd_loan.loan_payment IS
    'LOAN REPAYMENT PER MONTH';

ALTER TABLE acd_loan ADD CONSTRAINT acd_loan_pk PRIMARY KEY ( a_uid,
                                                              acct_type );

ALTER TABLE acd_loan ADD CONSTRAINT acd_loan_pkv1 UNIQUE ( acct_no,
                                                           loan_type );
```

```
CREATE TABLE acd_personal (
    a_uid     NUMBER(10) NOT NULL,
    acct_type VARCHAR2(2) NOT NULL,
    pl_uid    NUMBER(5) NOT NULL,
    date_open DATE NOT NULL
);

COMMENT ON COLUMN acd_personal.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_personal.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_personal.pl_uid IS
    'Personal Loan Unique ID';

COMMENT ON COLUMN acd_personal.date_open IS
    'Personal Loan Account Opening Date';

ALTER TABLE acd_personal ADD CONSTRAINT acd_personal_pk PRIMARY KEY ( a_uid,
                                                                      acct_type );

ALTER TABLE acd_personal ADD CONSTRAINT acd_personal_pkv1 UNIQUE ( pl_uid );

CREATE TABLE acd_safe_acnt (
    a_uid     NUMBER(10) NOT NULL,
    acct_type VARCHAR2(2) NOT NULL,
    acct_name VARCHAR2(30) NOT NULL,
    a_street  VARCHAR2(30) NOT NULL,
    a_city    VARCHAR2(30) NOT NULL,
    a_state   VARCHAR2(2) NOT NULL,
    a_zipcode VARCHAR2(5) NOT NULL
);

ALTER TABLE acd_safe_acnt
    ADD CONSTRAINT ch_inh_acd_safe_acnt CHECK ( acct_type IN ( 'C', 'HL', 'L',
'PL', 'S', 'SL' ) );

COMMENT ON COLUMN acd_safe_acnt.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_safe_acnt.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_safe_acnt.acct_name IS
    'Account Name';

COMMENT ON COLUMN acd_safe_acnt.a_street IS
    'Account Street Address';

COMMENT ON COLUMN acd_safe_acnt.a_city IS
```

```
    'Account City Location';

COMMENT ON COLUMN acd_safe_acnt.a_state IS
    'Account State Code - Ex: New York: NY';

COMMENT ON COLUMN acd_safe_acnt.a_zipcode IS
    'Account Zipcode';

ALTER TABLE acd_safe_acnt ADD CONSTRAINT acd_safe_acnt_pk PRIMARY KEY ( a_uid,
                                                                       acct_type
);

CREATE TABLE acd_savings (
    a_uid       NUMBER(10) NOT NULL,
    acct_type   VARCHAR2(2) NOT NULL,
    acct_no     NUMBER(10) NOT NULL,
    date_open   DATE NOT NULL,
    intrst_rate NUMBER(4, 2) NOT NULL
);

COMMENT ON COLUMN acd_savings.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_savings.acct_type IS
    'Account Type - Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_savings.acct_no IS
    'Savings Account Unique Account Number';

COMMENT ON COLUMN acd_savings.date_open IS
    'Savings Account Opening Date';

COMMENT ON COLUMN acd_savings.intrst_rate IS
    'Interest Rate';

ALTER TABLE acd_savings ADD CONSTRAINT acd_savings_pk PRIMARY KEY ( a_uid,
                                                                   acct_type );

ALTER TABLE acd_savings ADD CONSTRAINT acd_savings_pkv1 UNIQUE ( acct_no );

CREATE TABLE acd_student (
    a_uid       NUMBER(10) NOT NULL,
    acct_type   VARCHAR2(2) NOT NULL,
    sl_uid      NUMBER NOT NULL,
    date_open   DATE NOT NULL,
    student_id  VARCHAR2(6) NOT NULL,
    degree_type VARCHAR2(13) NOT NULL,
    grad_month  VARCHAR2(10) NOT NULL,
    grad_year   NUMBER(4) NOT NULL,
    inst_code   NUMBER(5) NOT NULL
);
```

```
ALTER TABLE acd_student
ADD CONSTRAINT chk_degree_type CHECK (degree_type IN ('Undergraduate',
'Graduate'));

COMMENT ON COLUMN acd_student.a_uid IS
    'SAFE Unique Account ID';

COMMENT ON COLUMN acd_student.acct_type IS
    'Account Type – Checking (C)/ Savings (S)/ Loan (L)';

COMMENT ON COLUMN acd_student.sl_uid IS
    'Student Loan Unique Loan ID';

COMMENT ON COLUMN acd_student.date_open IS
    'Student Loan Account Opening Date';

COMMENT ON COLUMN acd_student.student_id IS
    'Student ID From Institute';

COMMENT ON COLUMN acd_student.degree_type IS
    'Degree Type – Graduate / Undergraduate';

COMMENT ON COLUMN acd_student.grad_month IS
    'Graduation Month';

COMMENT ON COLUMN acd_student.grad_year IS
    'Graduation Year';

ALTER TABLE acd_student ADD CONSTRAINT acd_student_pk PRIMARY KEY ( a_uid,
                                                                    acct_type );

ALTER TABLE acd_student ADD CONSTRAINT acd_student_pkv1 UNIQUE ( sl_uid );

CREATE TABLE user_auth_encrypt (
    username NUMBER(5) NOT NULL,
    password VARCHAR2(25) NOT NULL,
    is_admin RAW(2000) NOT NULL
);

COMMENT ON COLUMN user_auth_encrypt.username IS
    'Primary key';

COMMENT ON COLUMN user_auth_encrypt.password IS
    'password field';

COMMENT ON COLUMN user_auth_encrypt.is_admin IS
    'binary isAdmin field';

ALTER TABLE user_auth_encrypt ADD CONSTRAINT user_auth_encrypt_pk PRIMARY KEY (
username );
```

```
ALTER TABLE acd_checking
    ADD CONSTRAINT acd_checking_acd_safe_acnt_fk FOREIGN KEY ( a_uid,
                                                               acct_type )
        REFERENCES acd_safe_acnt ( a_uid,
                                   acct_type );

ALTER TABLE acd_customer
    ADD CONSTRAINT acd_customer_acd_safe_acnt_fk FOREIGN KEY ( a_uid,
                                                               acct_type )
        REFERENCES acd_safe_acnt ( a_uid,
                                   acct_type );

ALTER TABLE acd_home
    ADD CONSTRAINT acd_home_acd_loan_fk FOREIGN KEY ( a_uid,
                                                      acct_type )
        REFERENCES acd_loan ( a_uid,
                              acct_type );

ALTER TABLE acd_insurance
    ADD CONSTRAINT acd_insurance_acd_home_fk FOREIGN KEY ( hl_uid )
        REFERENCES acd_home ( hl_uid );

ALTER TABLE acd_loan
    ADD CONSTRAINT acd_loan_acd_safe_acnt_fk FOREIGN KEY ( a_uid,
                                                           acct_type )
        REFERENCES acd_safe_acnt ( a_uid,
                                   acct_type );

ALTER TABLE acd_personal
    ADD CONSTRAINT acd_personal_acd_loan_fk FOREIGN KEY ( a_uid,
                                                          acct_type )
        REFERENCES acd_loan ( a_uid,
                              acct_type );

ALTER TABLE acd_savings
    ADD CONSTRAINT acd_savings_acd_safe_acnt_fk FOREIGN KEY ( a_uid,
                                                              acct_type )
        REFERENCES acd_safe_acnt ( a_uid,
                                   acct_type );

ALTER TABLE acd_student
    ADD CONSTRAINT acd_student_acd_institute_fk FOREIGN KEY ( inst_code )
        REFERENCES acd_institute ( inst_code );

ALTER TABLE acd_student
    ADD CONSTRAINT acd_student_acd_loan_fk FOREIGN KEY ( a_uid,
                                                         acct_type )
        REFERENCES acd_loan ( a_uid,
                              acct_type );
```

```
CREATE OR REPLACE TRIGGER arc_fkarc_1_acd_personal BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_personal
    FOR EACH ROW
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.loan_type
    INTO d
    FROM
        acd_loan a
    WHERE
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'PL' ) THEN
        raise_application_error(-20223, 'FK ACD_PERSONAL_ACD_LOAN_FK in Table
ACD_PERSONAL violates Arc constraint on Table ACD_LOAN - discriminator column
LOAN_TYPE doesn''t have value ''PL'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_1_acd_student BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_student
    FOR EACH ROW
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.loan_type
    INTO d
    FROM
        acd_loan a
    WHERE
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'SL' ) THEN
        raise_application_error(-20223, 'FK ACD_STUDENT_ACD_LOAN_FK in Table
ACD_STUDENT violates Arc constraint on Table ACD_LOAN - discriminator column
LOAN_TYPE doesn''t have value ''SL'''
        );
    END IF;
```

```
EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_1_acd_home BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_home
    FOR EACH ROW
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.loan_type
    INTO d
    FROM
        acd_loan a
    WHERE
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'HL' ) THEN
        raise_application_error(-20223, 'FK ACD_HOME_ACD_LOAN_FK in Table ACD_HOME
violates Arc constraint on Table ACD_LOAN - discriminator column LOAN_TYPE doesn''t
have value ''HL'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_2_acd_savings BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_savings
    FOR EACH ROW
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.acct_type
    INTO d
    FROM
        acd_safe_acnt a
    WHERE
```

```
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'S' ) THEN
        raise_application_error(-20223, 'FK ACD_SAVINGS_ACD_SAFE_ACNT_FK in Table
ACD_SAVINGS violates Arc constraint on Table ACD_SAFE_ACNT - discriminator column
ACCT_TYPE doesn''t have value ''S'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_2_acd_checking BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_checking
    FOR EACH ROW
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.acct_type
    INTO d
    FROM
        acd_safe_acnt a
    WHERE
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'C' ) THEN
        raise_application_error(-20223, 'FK ACD_CHECKING_ACD_SAFE_ACNT_FK in Table
ACD_CHECKING violates Arc constraint on Table ACD_SAFE_ACNT - discriminator column
ACCT_TYPE doesn''t have value ''C'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_2_acd_loan BEFORE
    INSERT OR UPDATE OF a_uid, acct_type ON acd_loan
    FOR EACH ROW
```

```
DECLARE
    d VARCHAR2(2);
BEGIN
    SELECT
        a.acct_type
    INTO d
    FROM
        acd_safe_acnt a
    WHERE
            a.a_uid = :new.a_uid
        AND a.acct_type = :new.acct_type;

    IF ( d IS NULL OR d <> 'L' ) THEN
        raise_application_error(-20223, 'FK ACD_LOAN_ACD_SAFE_ACNT_FK in Table
ACD_LOAN violates Arc constraint on Table ACD_SAFE_ACNT - discriminator column
ACCT_TYPE doesn''t have value ''L'''
        );
    END IF;

EXCEPTION
    WHEN no_data_found THEN
        NULL;
    WHEN OTHERS THEN
        RAISE;
END;
/



-- Oracle SQL Developer Data Modeler Summary Report:
--
-- CREATE TABLE                            11
-- CREATE INDEX                             1
-- ALTER TABLE                             28
-- CREATE VIEW                              0
-- ALTER VIEW                               0
-- CREATE PACKAGE                           0
-- CREATE PACKAGE BODY                      0
-- CREATE PROCEDURE                         0
-- CREATE FUNCTION                          0
-- CREATE TRIGGER                           6
-- ALTER TRIGGER                            0
-- CREATE COLLECTION TYPE                   0
-- CREATE STRUCTURED TYPE                   0
-- CREATE STRUCTURED TYPE BODY              0
-- CREATE CLUSTER                           0
-- CREATE CONTEXT                           0
-- CREATE DATABASE                          0
-- CREATE DIMENSION                         0
-- CREATE DIRECTORY                         0
-- CREATE DISK GROUP                        0
```

```
-- CREATE ROLE                          0
-- CREATE ROLLBACK SEGMENT              0
-- CREATE SEQUENCE                      0
-- CREATE MATERIALIZED VIEW             0
-- CREATE MATERIALIZED VIEW LOG         0
-- CREATE SYNONYM                       0
-- CREATE TABLESPACE                    0
-- CREATE USER                          0
--
-- DROP TABLESPACE                      0
-- DROP DATABASE                        0
--
-- REDACTION POLICY                     0
--
-- ORDS DROP SCHEMA                     0
-- ORDS ENABLE SCHEMA                   0
-- ORDS ENABLE OBJECT                   0
--
-- ERRORS                               0
-- WARNINGS                             0
```

7. RDBMS (Django sqlite3 models.py code)

```python
from django.db import models

# SAFE Bank Accounts
class AcdSafeAcnt(models.Model):
    a_uid = models.BigIntegerField()
    acct_type = models.CharField(max_length=2)
    acct_name = models.CharField(max_length=30)
    a_street = models.CharField(max_length=30)
    a_city = models.CharField(max_length=30)
    a_state = models.CharField(max_length=2)
    a_zipcode = models.CharField(max_length=5)
    surrogate_key = models.AutoField(primary_key=True)

    class Meta:
        # managed = False
        # db_table = 'AcdSafeAcnt'
        unique_together = ('a_uid', 'acct_type')

# Customer Details
class AcdCustomer(models.Model):
    c_id = models.IntegerField(primary_key=True)
    c_fname = models.CharField(max_length=30)
    c_lname = models.CharField(max_length=30)
    c_street = models.CharField(max_length=30)
    c_city = models.CharField(max_length=30)
    c_state = models.CharField(max_length=2)
```

```
    c_zipcode = models.CharField(max_length=5)
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Customer_Info')

    '''
    class Meta:
        managed = False
        db_table = 'AcdCustomer'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tCustomer Unique
ID: {self.c_id}"

# Checking Accounts
class AcdChecking(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Checking_Accounts')
    acct_no = models.BigIntegerField(unique=True)
    date_open = models.DateTimeField()
    serv_charge = models.DecimalField(max_digits=6, decimal_places=2)

    '''
    class Meta:
        managed = False
        db_table = 'AcdChecking'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Savings Accounts
class AcdSavings(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Savings_Accounts')
    acct_no = models.BigIntegerField(unique=True)
    date_open = models.DateTimeField()
    intrst_rate = models.DecimalField(max_digits=4, decimal_places=2)

    '''
    class Meta:
        managed = False
        db_table = 'AcdSavings'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Loan Accounts
```

```
class AcdLoan(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Loan_Accounts')
    acct_no = models.BigIntegerField()
    loan_type = models.CharField(max_length=2)
    loan_amt = models.DecimalField(max_digits=10, decimal_places=2)
    loan_rate = models.DecimalField(max_digits=4, decimal_places=2)
    loan_months = models.SmallIntegerField()
    loan_payment = models.DecimalField(max_digits=8, decimal_places=2)

    class Meta:
        # managed = False
        # db_table = 'AcdLoan'
        unique_together = ('acct_no', 'loan_type')

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Personal Loan - From Loan Accounts
class AcdPersonal(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Personal_Loan')
    pl_uid = models.IntegerField(unique=True)
    date_open = models.DateTimeField()

    '''
    class Meta:
        managed = False
        db_table = 'AcdPersonal'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Institutes - For Student Loans
class AcdInstitute(models.Model):
    inst_code = models.IntegerField(primary_key=True)
    inst_name = models.CharField(max_length=30)

    '''
    class Meta:
        managed = False
        db_table = 'AcdInstitute'
    '''

# Student Loan - From Loan Accounts
class AcdStudent(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Student_Loan')
```

```
    sl_uid = models.FloatField(unique=True)
    date_open = models.DateTimeField()
    student_id = models.CharField(max_length=6)
    degree_type = models.CharField(max_length=13)
    grad_month = models.CharField(max_length=10)
    grad_year = models.SmallIntegerField()
    inst_code = models.ForeignKey(AcdInstitute, models.DO_NOTHING)

    '''
    class Meta:
        managed = False
        db_table = 'AcdStudent'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Home Loan - From Loan Accounts
class AcdHome(models.Model):
    a_safe_acnt = models.ForeignKey(AcdSafeAcnt, on_delete=models.CASCADE,
verbose_name='SAFE Unique Account', related_name='Home_Loan')
    hl_uid = models.IntegerField(unique=True)
    date_open = models.DateTimeField()
    built_year = models.SmallIntegerField()

    '''
    class Meta:
        managed = False
        db_table = 'AcdHome'
    '''

    def getSAFEAcntDetails(self):
        return f"SAFE Unique Accound ID: {self.a_safe_acnt.a_uid}\tAccount Type:
{self.a_safe_acnt.acct_type}"

# Insurance - For Home Loans
class AcdInsurance(models.Model):
    ins_acct_no = models.BigIntegerField(primary_key=True)
    ins_company = models.CharField(max_length=30)
    ins_street = models.CharField(max_length=30)
    ins_city = models.CharField(max_length=30)
    ins_state = models.CharField(max_length=2)
    ins_zipcode = models.CharField(max_length=5)
    yearly_prm = models.DecimalField(max_digits=8, decimal_places=2)
    hl_uid = models.ForeignKey(AcdHome, models.DO_NOTHING)

    '''
    class Meta:
        managed = False
        db_table = 'AcdInsurance'
```

```
    '''

# Login Details
class UserAuthEncrypt(models.Model):
    username = models.BigIntegerField(primary_key=True)
    password = models.BinaryField(null=False)
    is_admin = models.BooleanField(default=False, null=False)

    '''
    class Meta:
        managed = False
        db_table = 'UserAuth'
    '''
```

8. List of Tables and No of Records of Each Table

List of Tables

```
sqlite> .tables
acdbanking_acdchecking      auth_group
acdbanking_acdcustomer      auth_group_permissions
acdbanking_acdhome          auth_permission
acdbanking_acdinstitute     auth_user
acdbanking_acdinsurance     auth_user_groups
acdbanking_acdloan          auth_user_user_permissions
acdbanking_acdpersonal      django_admin_log
acdbanking_acdsafeacnt      django_content_type
acdbanking_acdsavings       django_migrations
acdbanking_acdstudent       django_session
acdbanking_userauthencrypt
sqlite>
```

Count of Records in ACDSAFEACNT Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdsafeacnt;
46
sqlite>
```

Count of Records in ACDCUSTOMER Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdcustomer;
34
sqlite>
```

Count of Records in ACDCHECKING Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdchecking;
15
sqlite>
```

Count of Records in ACDSAVINGS Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdsavings;
15
sqlite>
```

Count of Records in ACDLOAN Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdloan;
18
sqlite>
```

Count of Records in ACDPERSONAL Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdpersonal;
5
sqlite>
```

Count of Records in ACDSTUDENT Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdstudent;
6
sqlite>
```

Count of Records in ACDHOME Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdhome;
7
sqlite>
```

Count of Records in ACDINSTITUTE Table

```
sqlite> SELECT COUNT(*) FROM acdbanking_acdinstitute;
15
sqlite>
```

Count of Records in ACDINSURANCE Table

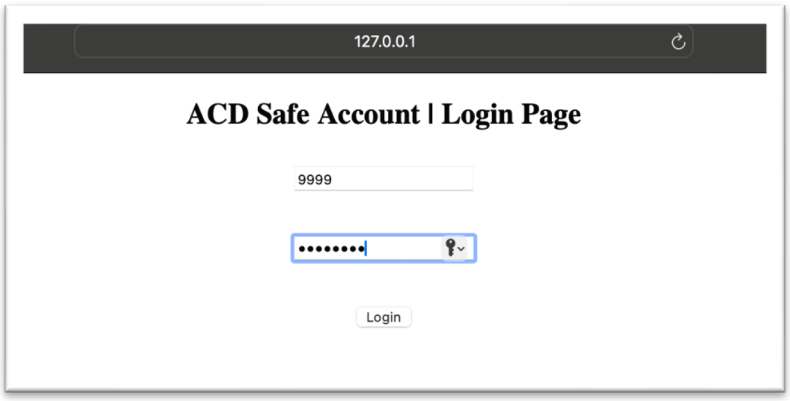
```
sqlite> SELECT COUNT(*) FROM acdbanking_acdinsurance;
7
sqlite>
```

Count of Records in USERAUTHENCRYPT Table

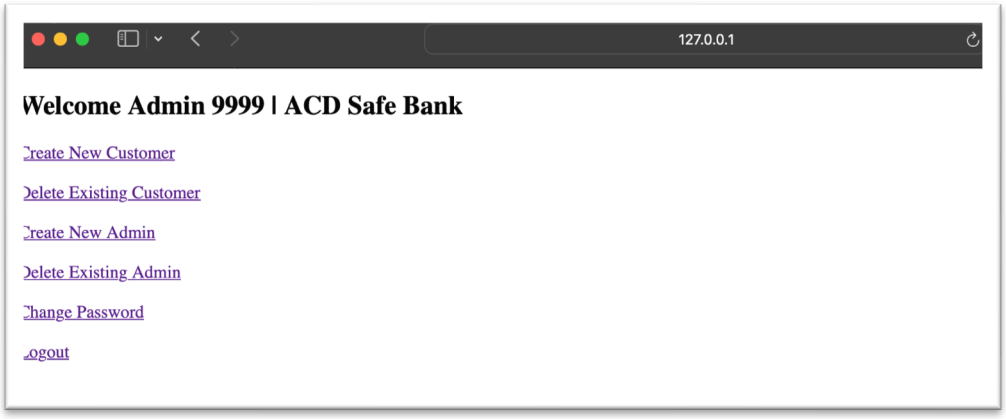
```
sqlite> SELECT COUNT(*) FROM acdbanking_userauthencrypt;
37
sqlite>
```

9. Screenshots of Web Application

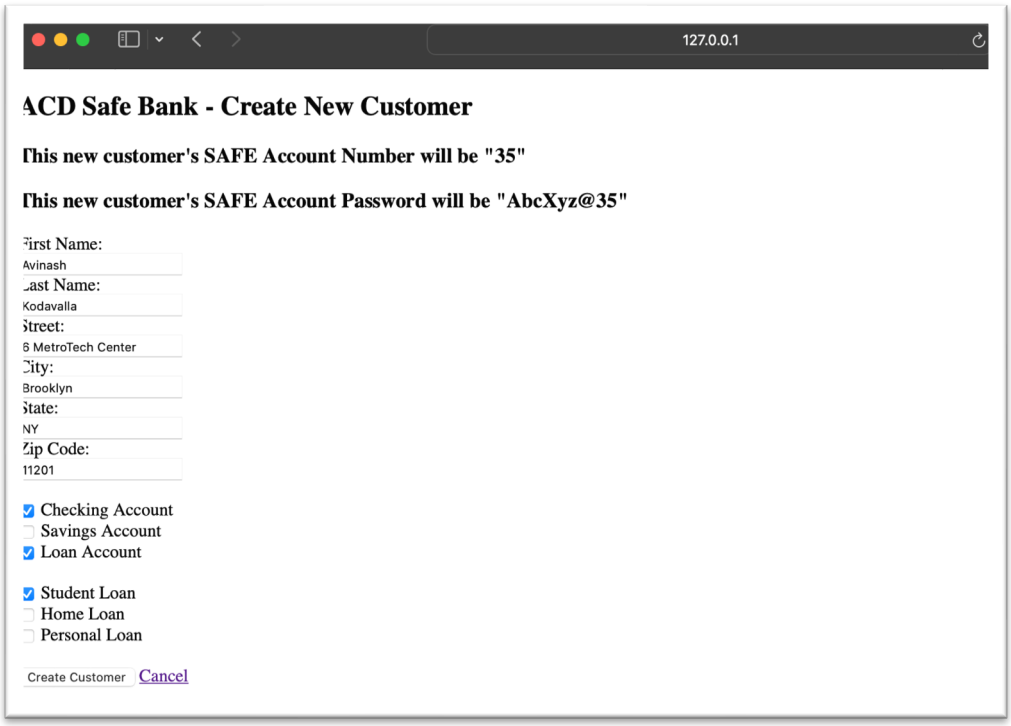
Login page (same page for both customers and admin users)



Admin User – Dashboard

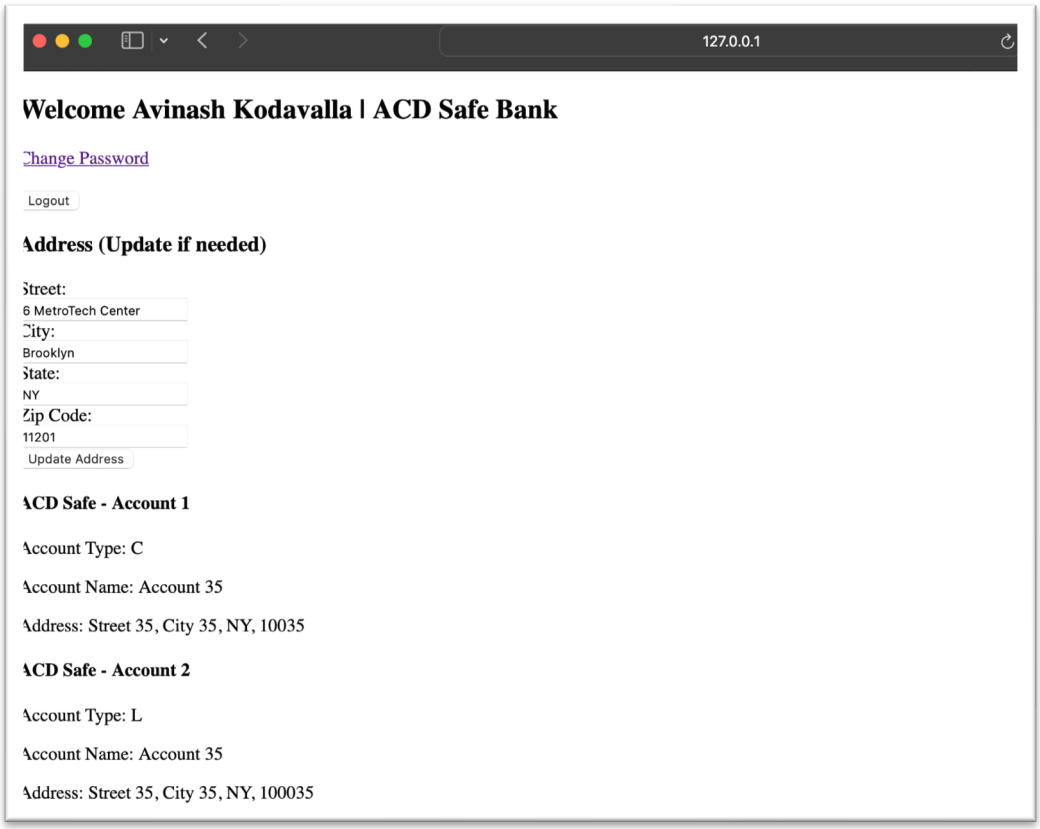


Creating New Customer by Admin User



Change Password Page

Customer – Dashboard



Display of Account Details of Customer

10. Security Features of Web Application

- Prevention of SQL Injection:
  Employed Django's security features to mitigate the risk of SQL injection attacks. Django utilizes parameterized queries and prepared statements, which effectively separate SQL code from user input. This ensures that user inputs are treated as data rather than executable SQL code.

  As you can see below how queries are parameterized:

```
admin_user = UserAuthEncrypt.objects.create(
    username=9999,
    password=fernet.encrypt('admin123'.encode()),
    is_admin=True
)
admin_user.save()
```

- Encryption and Decryption of User Passwords:
  Using the Fernet module from the cryptography library, we encrypt user passwords before storing them in the database. This ensures that even if the database is compromised, sensitive user data remains secure. Decryption is performed when authenticating users during login, ensuring seamless user authentication while maintaining data confidentiality.

  As you can see below how passwords are stored in database:

```
sqlite> SELECT * FROM acdbanking_userauthencrypt;
1|gAAAAABmNnjhG4uZ1LPM7s4iTijoku9S4aDSJVJk6IANjlEImxWhrl-Ibrb4gxK2xfc4RbCGW1EBQV3IddlBLuExtw2WJ9F1pQ==|0
2|gAAAAABmNnjhCY3N05-nI_jBwxPlF-iTIEtLruZqJXrDK3UC6PvpIr6DfA7RFn8Ig_zOz07VTrlWO8EmKzUS0LP4uAXM79T7hQ==|0
3|gAAAAABmNnjhzrtZqsz9SgW5R2LV0pJehRd4Jk6kmeWbaU0KEzFvXJIrVQ-89mnzC3JYOH1Spqrtk0yx2TlrybP6ont_jN93IA==|0
4|gAAAAABmNnjhrgQXGjl9cEgSnjYR-9NoteSzRwKWtPjdapQ_7sfflfsgOMdd0YjpiL9d83VdXrzg5O0Wou6AKsWDLyqsoEVvvA==|0
5|gAAAAABmNnjhPlBmLV1CAd9o-mia-8pvyJn2mZ-r31eeGpZonxsfLZkYU4zpa_vZrCIAS0UCTceJZ1TdAxa90Sm5-oJlic5s5g==|0
6|gAAAAABmNnjhSXjJOqBlwRn21m77Ftvz9VXXUhvKRt60b8pEa_jFVxY4aG4bpqRMXHJLp9lYdbJqIkkpBJStLFhf2-rBVAMTEA==|0
```

- Concurrency and Multi-User Support:
  Django provides built-in support for handling multiple users concurrently, making our web application capable of serving multiple users simultaneously. Each user session is isolated and maintains its state, preventing data leakage or unauthorized access between users.

11. Lessons and Reflections from Project

Key Lessons:

Collaborative Development:
- Team collaboration was crucial, especially in aligning database and web application components.
- Using version control systems helped maintain code integrity and manage contributions efficiently.

Design and Iteration:
- Initial designs often need refinement; iterative design enabled us to refine the database schema and web functionalities to better meet the requirements.
- Regular review sessions helped identify potential issues early, reducing the need for major changes later.

Security Considerations:
- Implementing robust security measures from the start was essential. This project reinforced the importance of considering security in every phase of development.
- Learning to use Django's security features effectively was crucial for preventing common vulnerabilities such as SQL injection.

Performance Optimization:
- Understanding the impact of database design on performance was a key takeaway. Optimizing SQL queries and choosing appropriate indexes significantly improved the application's responsiveness.
- Django's ORM, while convenient, required careful usage to avoid generating inefficient queries.

Testing and Debugging:
- Comprehensive testing, including unit tests, integration tests, and user acceptance testing, was vital. It not only ensured that the system worked as intended but also that it met all user requirements.
- Debugging, especially in a complex system involving multiple technologies, taught us the importance of having a good strategy and tools for tracking down and fixing issues.

User-Centered Design:
- Feedback from potential users was invaluable in shaping the user interface. It ensured that the application was not only functional but also user-friendly.
- The importance of responsive design was highlighted, ensuring that the web application was accessible on various devices.

Reflections:

Adaptability:

Throughout the project, adapting to new tools and technologies was necessary. The learning curve was steep but rewarding, as each team member brought different skills and knowledge to the table.

Communication:

Effective communication among team members was essential for coordinating tasks and ensuring everyone was on the same page. Regular meetings and updates helped keep the project on track.

Future Improvements:

For future projects, we plan to incorporate more automated testing and continuous integration/continuous deployment (CI/CD) practices to streamline development and ensure higher code quality.

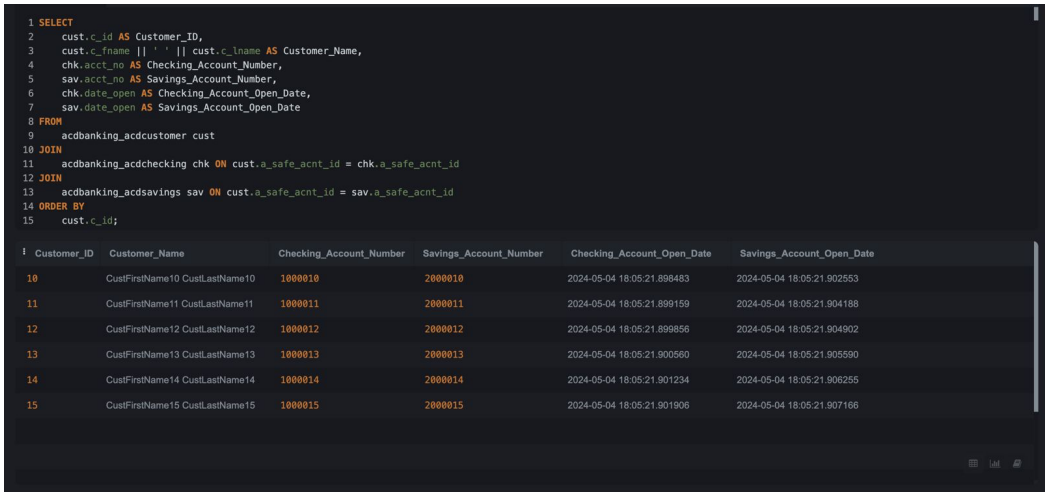
Personal Growth:

This project provided a platform to apply theoretical knowledge in a practical setting, enhancing our problem-solving and technical skills.

12. Business Analysis on Project Data

1) Business purpose:
This query provides a consolidated view of customer information across different account types (checking and savings), which is useful for understanding customer engagement with multiple banking services and for managing customer relations more effectively.

```
SELECT
    cust.c_id AS Customer_ID,
    cust.c_fname || ' ' || cust.c_lname AS Customer_Name,
    chk.acct_no AS Checking_Account_Number,
    sav.acct_no AS Savings_Account_Number,
    chk.date_open AS Checking_Account_Open_Date,
    sav.date_open AS Savings_Account_Open_Date
FROM
    acdbanking_acdcustomer cust
JOIN
    acdbanking_acdchecking chk ON cust.a_safe_acnt_id = chk.a_safe_acnt_id
JOIN
    acdbanking_acdsavings sav ON cust.a_safe_acnt_id = sav.a_safe_acnt_id
ORDER BY
    cust.c_id;
```

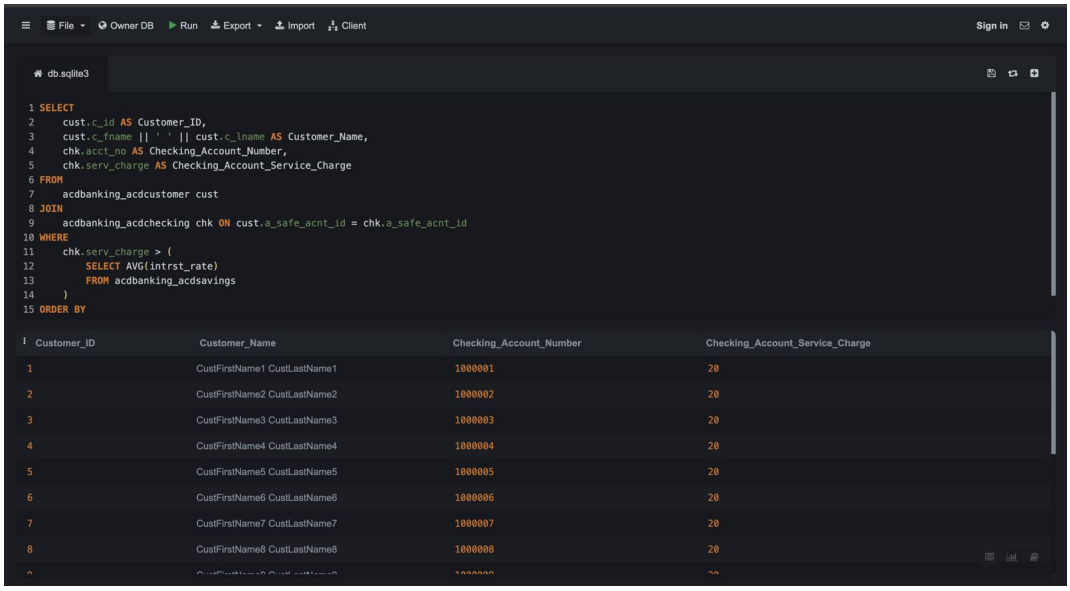


```
SELECT
    cust.c_id AS Customer_ID,
    cust.c_fname || ' ' || cust.c_lname AS Customer_Name,
    chk.acct_no AS Checking_Account_Number,
    sav.acct_no AS Savings_Account_Number,
    chk.date_open AS Checking_Account_Open_Date,
    sav.date_open AS Savings_Account_Open_Date
FROM
    acdbanking_acdcustomer cust
JOIN
    acdbanking_acdchecking chk ON cust.a_safe_acnt_id = chk.a_safe_acnt_id
JOIN
    acdbanking_acdsavings sav ON cust.a_safe_acnt_id = sav.a_safe_acnt_id
ORDER BY
    cust.c_id;
```

Customer_ID	Customer_Name	Checking_Account_Number	Savings_Account_Number	Checking_Account_Open_Date	Savings_Account_Open_Date
10	CustFirstName10 CustLastName10	1000010	2000010	2024-05-04 18:05:21.898483	2024-05-04 18:05:21.902553
11	CustFirstName11 CustLastName11	1000011	2000011	2024-05-04 18:05:21.899159	2024-05-04 18:05:21.904188
12	CustFirstName12 CustLastName12	1000012	2000012	2024-05-04 18:05:21.899856	2024-05-04 18:05:21.904902
13	CustFirstName13 CustLastName13	1000013	2000013	2024-05-04 18:05:21.900560	2024-05-04 18:05:21.905590
14	CustFirstName14 CustLastName14	1000014	2000014	2024-05-04 18:05:21.901234	2024-05-04 18:05:21.906255
15	CustFirstName15 CustLastName15	1000015	2000015	2024-05-04 18:05:21.901906	2024-05-04 18:05:21.907166

2) Business Purpose:
This query is useful for identifying customers who may be engaging in high volumes of transactions or maintaining high balances, as indicated by higher service charges. This information can be used for targeted marketing of premium services or offers that cater to more active account holders.

```
SELECT
    cust.c_id AS Customer_ID,
    cust.c_fname || ' ' || cust.c_lname AS Customer_Name,
    chk.acct_no AS Checking_Account_Number,
    chk.serv_charge AS Checking_Account_Service_Charge
FROM
    acdbanking_acdcustomer cust
JOIN
    acdbanking_acdchecking chk ON cust.a_safe_acnt_id = chk.a_safe_acnt_id
WHERE
    chk.serv_charge > (
        SELECT AVG(intrst_rate)
        FROM acdbanking_acdsavings
    )
ORDER BY
    chk.serv_charge DESC;
```



File · Owner DB · Run · Export · Import · Client · Sign in

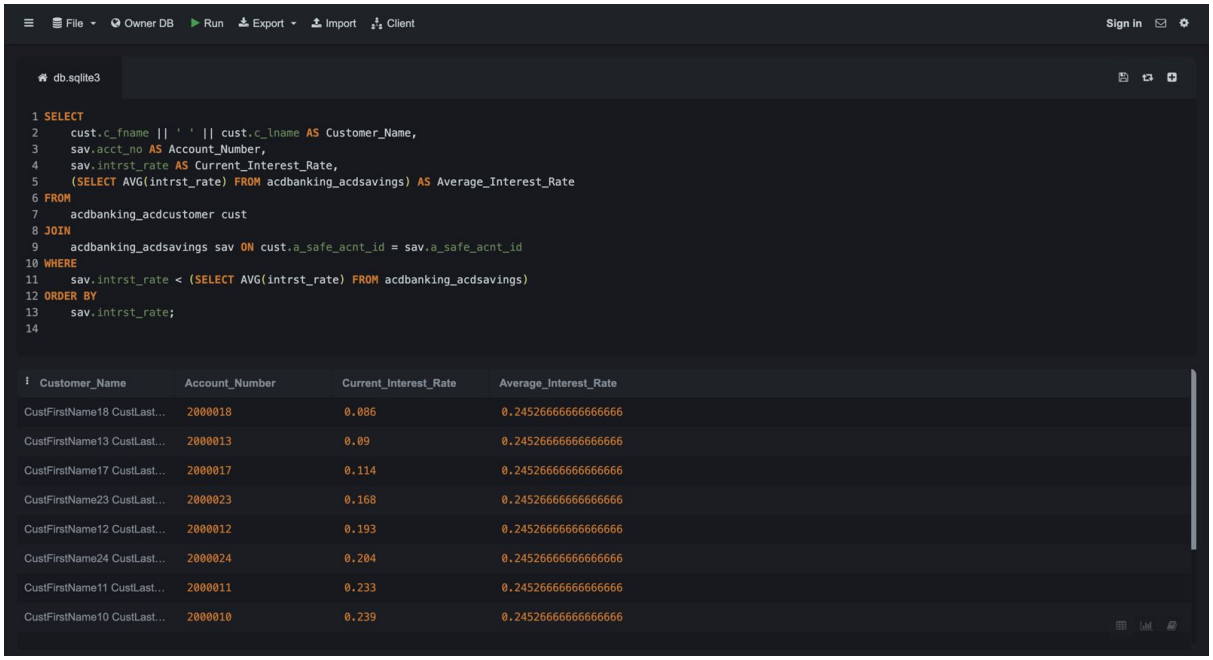
db.sqlite3

```
SELECT
    cust.c_id AS Customer_ID,
    cust.c_fname || ' ' || cust.c_lname AS Customer_Name,
    chk.acct_no AS Checking_Account_Number,
    chk.serv_charge AS Checking_Account_Service_Charge
FROM
    acdbanking_acdcustomer cust
JOIN
    acdbanking_acdchecking chk ON cust.a_safe_acnt_id = chk.a_safe_acnt_id
WHERE
    chk.serv_charge > (
        SELECT AVG(intrst_rate)
        FROM acdbanking_acdsavings
    )
ORDER BY
```

Customer_ID	Customer_Name	Checking_Account_Number	Checking_Account_Service_Charge
1	CustFirstName1 CustLastName1	1000001	20
2	CustFirstName2 CustLastName2	1000002	20
3	CustFirstName3 CustLastName3	1000003	20
4	CustFirstName4 CustLastName4	1000004	20
5	CustFirstName5 CustLastName5	1000005	20
6	CustFirstName6 CustLastName6	1000006	20
7	CustFirstName7 CustLastName7	1000007	20
8	CustFirstName8 CustLastName8	1000008	20

3) Business Purpose:
Identify customers whose latest savings account interest rate is below the average interest rate of all savings accounts at their bank. This could indicate customers receiving less competitive rates, which might impact customer satisfaction and retention.

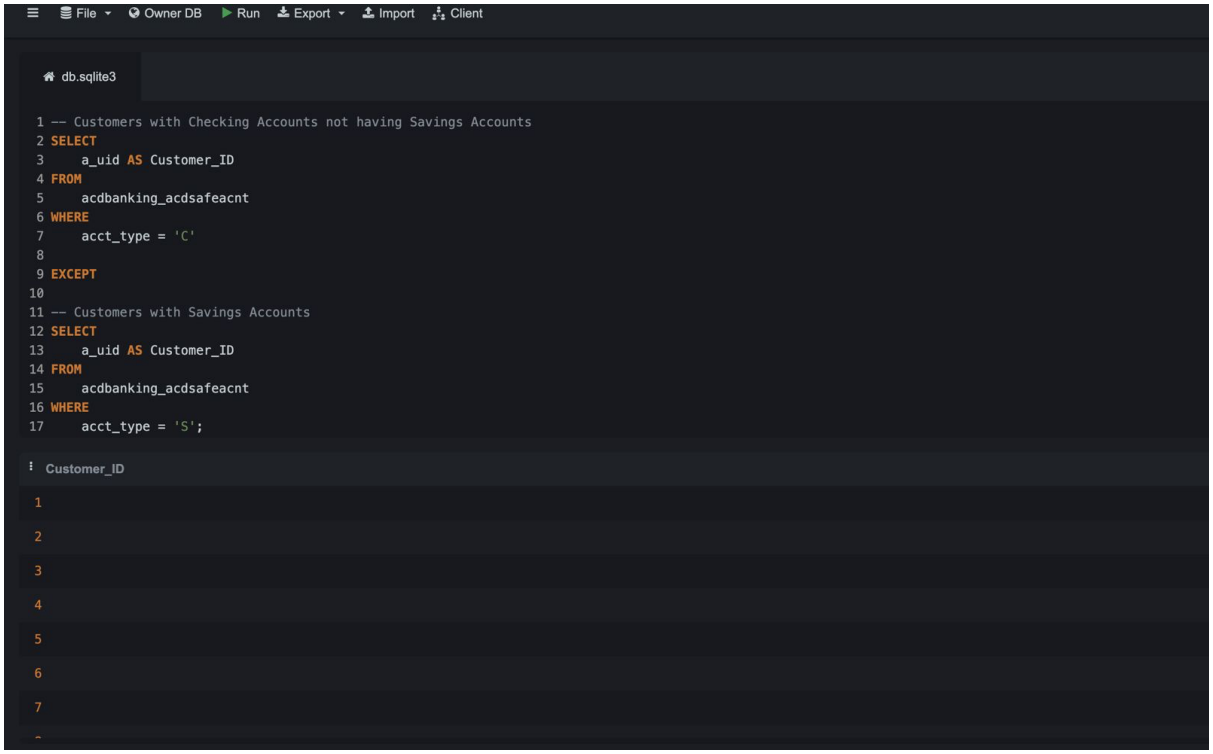
```
SELECT
    cust.c_fname || ' ' || cust.c_lname AS Customer_Name,
    sav.acct_no AS Account_Number,
    sav.intrst_rate AS Current_Interest_Rate,
    (SELECT AVG(intrst_rate) FROM acdbanking_acdsavings) AS Average_Interest_Rate
FROM
    acdbanking_acdcustomer cust
JOIN
    acdbanking_acdsavings sav ON cust.a_safe_acnt_id = sav.a_safe_acnt_id
WHERE
    sav.intrst_rate < (SELECT AVG(intrst_rate) FROM acdbanking_acdsavings)
ORDER BY
    sav.intrst_rate;
```



4) Business purpose:
We can use a SET operator to find out how many customers exclusively have checking accounts, savings accounts, or both. This analysis helps in understanding the distribution of product usage among customers, which can be crucial for targeted marketing and product development strategies.

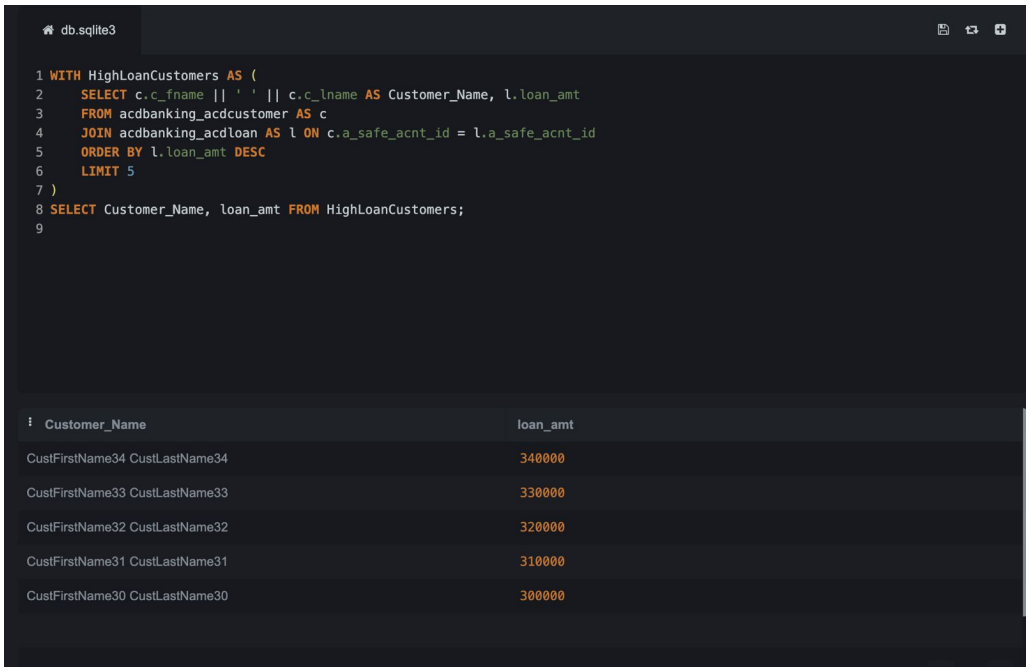
```
SELECT
    a_uid AS Customer_ID
FROM
    acdbanking_acdsafeacnt
WHERE
       acct_type = 'C'
EXCEPT

-- Customers with Savings Accounts
SELECT
    a_uid AS Customer_ID
FROM
    acdbanking_acdsafeacnt
WHERE
       acct_type = 'S';
```



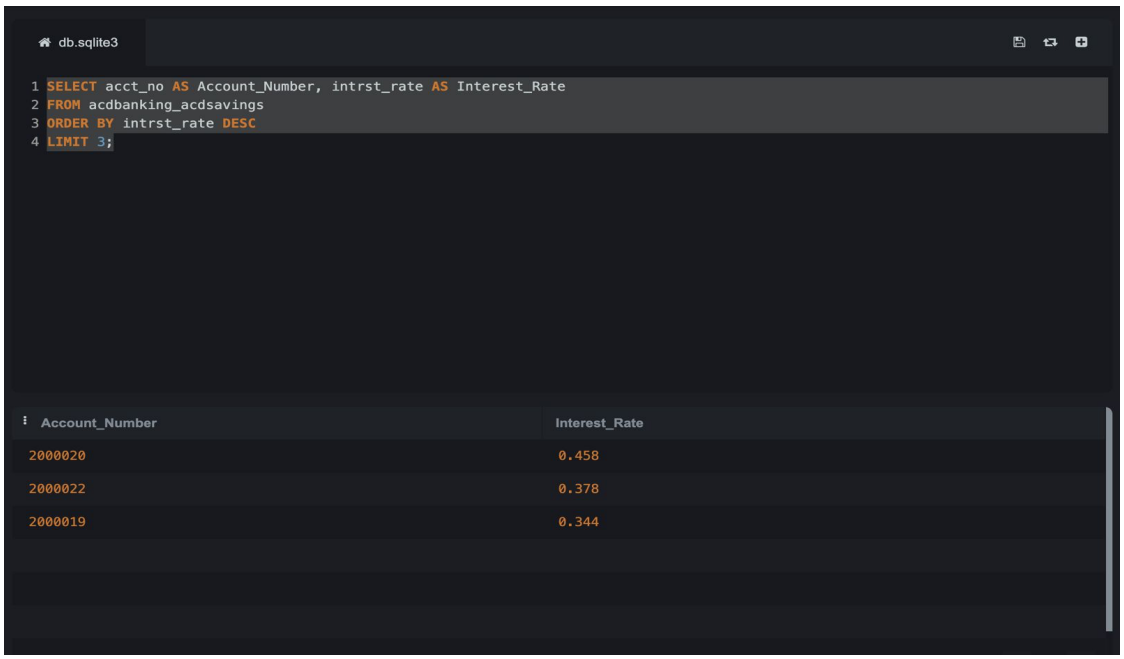
5. Business Purpose:
Identify customers with the highest loan amounts and their associated account details, aiding in targeted financial advising.

```
WITH HighLoanCustomers AS (
    SELECT c.c_fname || ' ' || c.c_lname AS Customer_Name, l.loan_amt
    FROM acdbanking_acdcustomer AS c
    JOIN acdbanking_acdloan AS l ON c.a_safe_acnt_id = l.a_safe_acnt_id
    ORDER BY l.loan_amt DESC
    LIMIT 5
)
SELECT Customer_Name, loan_amt FROM HighLoanCustomers;
```



6) Business Purpose:
List the top 3 savings accounts with the highest interest rates to highlight the best savings opportunities to customers

```
SELECT acct_no AS Account_Number, intrst_rate AS Interest_Rate
FROM acdbanking_acdsavings
ORDER BY intrst_rate DESC
LIMIT 3;
```



THE END