An Indecisive Banking System

By Hoai-Nam Phung, Kunwarpreet Singh Behar, and Ben Mahanloo

Team Indecisive

3. Database schema in the form of SQL CREATE TABLE. Specify key constraints and foreign key constraints in the schema.

Table Name	Attributes	Constraints	Notes
Users	userID, firstName, lastName, email, pw, updatedAt	AUTO_INCREMENT PRIMARY KEY(userID)	Used for account creation/login. pw = SHA-256 encrypted.
Accounts	userID, bankName, accType, balance	PRIMARY KEY(userID, bankName, accType), FOREIGN KEY(userID) REFERENCES Users(userID), FOREIGN KEY(bankName) REFERENCES Banks(bankName)	Users can have three types of accounts per bank. accType = “Checking”/”Savings”/”Loans”
Transactions	transID, userID, bankName, accType, transDateTime, location, summary, transType, amount, netBalance	PRIMARY KEY(transID), FOREIGN KEY(userID, bankName, accType) REFERENCES Accounts(userID, bankName, accType)	transType = “Deposit”/”Withdrawal”
Banks	bankName, balance	PRIMARY KEY(bankName)	balance = amount of money the bank has, not including money users have deposited into the bank.
Loans	loanID, userId, bankName, accType, amount, loanDate, dueDate	PRIMARY KEY(loanID), FOREIGN KEY(bankName) REFERENCES Banks(bankName), FOREIGN KEY(userId, bankName, accType) REFERENCES Accounts(userID, bankName, accType)	Doesn’t implement interest for the sake of simplicity.

DDL for Database Initialization

```sql
DROP DATABASE bank_system;
CREATE DATABASE IF NOT EXISTS bank_system;
USE bank_system;

/* Delete the tables if they already exist */
DROP TABLE IF EXISTS Users;
DROP TABLE IF EXISTS Accounts;
DROP TABLE IF EXISTS Transactions;
DROP TABLE IF EXISTS Banks;
DROP TABLE IF EXISTS Loans;
DROP TABLE IF EXISTS UsersArchive;

/* Create the schema for our tables */
CREATE TABLE Users (
      userID TINYINT UNSIGNED AUTO_INCREMENT,
      firstName VARCHAR(36), lastName VARCHAR(36),
      email VARCHAR(256), pw BINARY(32),
      updatedAt TIMESTAMP,
      PRIMARY KEY(userID),
      UNIQUE KEY(email)
);
CREATE TABLE Banks (
      bankName VARCHAR(256),
      balance DECIMAL(15, 2),
      PRIMARY KEY(bankName)
);
CREATE TABLE Accounts (
      userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8),
      balance DECIMAL(15, 2),
      PRIMARY KEY(userID, bankName, accType),
      FOREIGN KEY(userID) REFERENCES Users(userID),
      FOREIGN KEY(bankName) REFERENCES Banks(bankName)
);
CREATE TABLE Transactions (
      transID TINYINT UNSIGNED AUTO_INCREMENT,
      userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8),
      transDateTime DATETIME,
      location VARCHAR(256),
      summary VARCHAR(256),
      transType VARCHAR(10),
      amount DECIMAL(15, 2),
      netBalance DECIMAL(15, 2),
      PRIMARY KEY(transID),
      FOREIGN KEY(userID, bankName, accType) REFERENCES Accounts(userID, bankName, accType)
);
CREATE TABLE Loans (
      loanID TINYINT UNSIGNED AUTO_INCREMENT,
      userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8),
      amount DECIMAL(15, 2),
      loanDate DATETIME, dueDate DATETIME,
      PRIMARY KEY(loanID),
      FOREIGN KEY(bankName) REFERENCES Banks(bankName),
      FOREIGN KEY(userId, bankName, accType) REFERENCES Accounts(userID, bankName, accType)
);
CREATE TABLE ArchivedUsers (
      userID TINYINT UNSIGNED,
      firstName VARCHAR(36), lastName VARCHAR(36),
      email VARCHAR(256), pw BINARY(32),
      updatedAt TIMESTAMP,
      PRIMARY KEY(userID)
);

/* Create the stored procedures */
DELIMITER $$
CREATE PROCEDURE ArchiveUsers (cutoffLoginDate TIMESTAMP)
BEGIN
      REPLACE INTO ArchivedUsers(userId, firstName, lastName, email, pw, updatedAt) (
      SELECT u.userId, u.firstName, u.lastName, u.email, u.pw, u.updatedAt FROM Users u WHERE u.updatedAt
<= cutoffLoginDate);
END$$
CREATE PROCEDURE CreateUser (firstName VARCHAR(36), lastName VARCHAR(36), email VARCHAR(256), pw
BINARY(32))
BEGIN
      INSERT INTO Users(firstName, lastName, email, pw, updatedAt) VALUES(firstName, lastName, email, pw,
CURRENT_TIMESTAMP());
END$$
CREATE PROCEDURE GetUserID (email VARCHAR(256), pw BINARY(32), OUT userID TINYINT UNSIGNED)
BEGIN
      SELECT Users.userID INTO userID FROM Users WHERE Users.email=email AND Users.pw=pw;
      UPDATE Users SET updatedAt = CURRENT_TIMESTAMP();
END$$
CREATE PROCEDURE DeleteUser (userID TINYINT UNSIGNED)
BEGIN
      DELETE FROM Users WHERE Users.userID = userID;
END$$
CREATE PROCEDURE CreateBankAccount (bankName VARCHAR(256), accType VARCHAR(8), balance DECIMAL(15, 2),
userID TINYINT UNSIGNED)
BEGIN
      INSERT INTO Accounts(bankName, accType, balance, userID) VALUES(bankName, accType, balance,
userId);
END$$
CREATE PROCEDURE DeleteBankAccount (bankName VARCHAR(256), accType VARCHAR(8), userID TINYINT UNSIGNED)
BEGIN
      DELETE FROM Accounts WHERE Accounts.userID = userID AND Accounts.bankName = bankName AND
Accounts.accType = accType;
END$$
CREATE PROCEDURE GetAllUserBankAccountsAtBank (bankName VARCHAR(256), userID TINYINT UNSIGNED)
BEGIN
      SELECT firstName, accType, balance FROM Accounts LEFT JOIN Users ON (Accounts.userId =
Users.userId) WHERE Accounts.userId = userId;
END$$
CREATE PROCEDURE GetBankAccountBalance (bankName VARCHAR(256), accType VARCHAR(8), userID TINYINT
UNSIGNED)
BEGIN
      SELECT balance FROM Accounts WHERE Accounts.bankName = bankName AND Accounts.accType = accType AND
Accounts.userID = userID;
END$$
CREATE PROCEDURE CalculateNetWorth (userID TINYINT UNSIGNED)
BEGIN
      SELECT SUM(balance) as NetWorth FROM Accounts WHERE Accounts.userID = userID;
END$$
CREATE PROCEDURE GetAllBanks ()
BEGIN
      SELECT bankName, balance from Banks;
END$$
CREATE PROCEDURE CreateBank (bankName VARCHAR(256), balance DECIMAL(15, 2))
BEGIN
      INSERT INTO Banks(bankName, balance) VALUES(bankName, balance);
END$$
CREATE PROCEDURE GetBanksBalance (bankName VARCHAR(256))
BEGIN
      SELECT balance FROM Banks WHERE Banks.bankName = bankName;
END$$
CREATE PROCEDURE GetRecentTransactions (userID TINYINT UNSIGNED, bankName VARCHAR(256), accType
VARCHAR(8))
BEGIN
      SELECT * FROM (SELECT * FROM Transactions ORDER BY transDateTime DESC LIMIT 10) AS t1
      WHERE (t1.userId, t1.bankName, t1.accType) IN
      (SELECT t2.userId, t2.bankName, t2.accType FROM
            (SELECT * FROM Transactions WHERE Transactions.userID = userID AND Transactions.bankName =
bankName AND Transactions.accType = accType) AS t2);
END$$
CREATE PROCEDURE GetMonthlyTransactions (userID TINYINT UNSIGNED, bankName VARCHAR(256), accType
VARCHAR(8), filterDate DATETIME)
BEGIN
      SELECT * FROM Transactions WHERE Transactions.userID = userID AND Transactions.bankName = bankName
AND Transactions.accType = accType
      GROUP BY Transactions.transDateTime HAVING MONTH(Transactions.transDateTime) = MONTH(filterDate)
AND YEAR(Transactions.transDateTime) = YEAR(filterDate);
END$$
CREATE PROCEDURE GetLoans(userID TINYINT UNSIGNED)
BEGIN
      SELECT * FROM Loans WHERE userID = userID;
END$$
CREATE PROCEDURE Deposit(userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8),transType
VARCHAR(10),amount DECIMAL(15, 2))
BEGIN
      INSERT INTO Transactions (userID, bankName, accType, transType, amount, transDateTime)
VALUES(userID, bankName, accType, "Deposit", amount, NOW());
END$$
CREATE PROCEDURE Withdraw(userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8),transType
VARCHAR(10),amount DECIMAL(15, 2))
BEGIN
      INSERT INTO Transactions (userID, bankName, accType, transType, amount) VALUES(userID, bankName,
accType, "Withdrawal", amount);
END$$
CREATE PROCEDURE CreateLoan(userID TINYINT UNSIGNED, bankName VARCHAR(256), accType VARCHAR(8), amount
DECIMAL(15, 2))
BEGIN
      INSERT INTO Loans(userID, bankName, accType, amount) VALUES(userID, bankName, accType, amount);
END$$


DELIMITER ;

/* Create triggers */
DELIMITER $$
CREATE TRIGGER update_account_on_new_transaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
      UPDATE Accounts a SET a.balance =
      CASE new.transType
      WHEN "Withdrawal" THEN a.balance - new.amount
      WHEN "Deposit" THEN a.balance + new.amount
      END
      WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType;
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER update_account_on_new_loan
AFTER INSERT ON Loans
FOR EACH ROW
BEGIN
      IF new.accType = "Loans"
      THEN UPDATE Accounts a SET a.balance = a.balance + new.amount
      WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType;
      ELSE
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Not a Loans account; insertion failed.";
      END IF;
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER update_account_on_update_loan
AFTER UPDATE ON Loans
FOR EACH ROW
BEGIN
      IF new.accType = "Loans"
      THEN UPDATE Accounts a SET a.balance = a.balance + new.amount - old.amount
      WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType;
      IF new.amount = 0
            THEN DELETE FROM Loans WHERE Loans.userID = new.userID AND Loans.bankName = new.bankName AND
Loans.accType = new.accType;
      END IF;
      ELSE
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Not a Loans account; insertion failed.";
      END IF;
END $$
DELIMITER ;

/* Populate tables */
CALL CreateUser('Hoai', 'Phung', 'hoai-nam.phung@sjsu.edu', '12345678');
CALL CreateUser('Hoai-Nam', 'Phung', 'hoainamphung2000@gmail.edu', '12345678');
CALL CreateUser('Ben', 'Mahanloo', 'ben.mahanloo@sjsu.edu', '12345678');
CALL CreateUser('John', 'Smith', 'johnsmith@gmail.com', '12345678');

CALL CreateBank("CHASE", 0);
CALL CreateBank("CITI", 0);
CALL CreateBank("WELLS FARGO", 0);
CALL CreateBank("BANK OF AMERICA", 0);

INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("CHASE", "Savings", 0, 1);
INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("CHASE", "Checking", 0, 1);
INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("CITI", "Checking", 0, 2);
INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("WELLS FARGO", "Savings", 0, 3);
INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("BANK OF AMERICA", "Loans", 0, 4);
INSERT INTO Accounts(bankName, accType, balance, userID)
VALUES("WELLS FARGO", "Loans", 0, 3);

INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (1, "CHASE", "Checking", "2021-12-23 12:00:00", "San Jose", "", "Deposit", 1000, 1000);
INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (1, "CHASE", "Checking", "2021-12-24 12:00:00", "San Jose", "", "Withdraw", 50, 950);
INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (2, "CITI", "Checking", "2021-11-20 10:00:00", "San Jose", "", "Deposit", 250, 250);
INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (3, "WELLS FARGO", "Savings", "2021-11-20 10:00:00", "San Jose", "", "Deposit", 10, 10);
INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (1, "CHASE", "Checking", "2020-12-23 12:00:00", "San Jose", "", "Deposit", 1000, 1000);
INSERT INTO Transactions(userID, bankName, accType, transDateTime, location, summary, transType, amount,
netBalance)
VALUES (1, "CHASE", "Checking", "2021-4-14 10:00:00", "Santa Cruz", "", "Withdraw", 50, 950);

INSERT INTO Loans(userID, bankName, accType, amount, loanDate, dueDate)
VALUES(4, "BANK OF AMERICA", "Loans", 15000, "2021-12-23 12:00:00", "2021-12-25 12:00:00");
INSERT INTO Loans(userID, bankName, accType, amount, loanDate, dueDate)
VALUES(4, "BANK OF AMERICA", "Loans", 5000, "2021-12-23 12:00:00", "2021-12-25 12:00:00");
INSERT INTO Loans(userID, bankName, accType, amount, loanDate, dueDate)
VALUES(3, "WELLS FARGO", "Loans", 1200, "2020-10-13 12:00:00", "2021-10-13 12:00:00");
```

4. Indicate if your team used a public data set or not. If a public data set is used, specify its URL.

Our team did not use a public dataset

5. Descriptions to show relation schemas are in BCNF or in 3NF.

1. Users (userID, firstName, lastName, email, pw, updatedAt)

   FD = {userID → userID, userID → firstName, userID → lastName, userID → email, userID → pw, userID → updatedAt}

{firstName}	{firstName}	Trivial
{lastName}	{lastName}	Trivial
{email}	{email}	Trivial
{userID}	{userID, firstName, lastName, email, pw, updatedAt}	Superkey
{pw}	{pw}	Trivial
{updatedAt}	{updatedAt}	Trivial

- 1NF
  - Every column in the userID table is atomic
  - Every column is the userID table is the same type
- 2NF
  - UserID is the primary key for the table and the columns will rely on the user for the information of the user.
  - The table is 1NF
- 3NF
  - Table is 1NF and 2NF
  - The non trivial functional dependency is either superkey or prime

Throughout the table, userID remains the superkey, any attribute without userID is trivial, making this table 3NF.

2. Accounts (userID, bankName, accType, balance)

   FD = {userID → userID, userID → bankName, userID → accType, userID → balance}

{bankName}	{bankName}	Trivial
{accType}	{accType}	Trivial
{balance}	{balance}	Trivial
{userID}	{userID, bankName, accType, balance}	Superkey

- 1NF
  - Every column in the Accounts table is atomic
  - Every column is the Accounts table is the same type
- 2NF
  - UserID is the primary key for the table and the columns will rely on the userID for the information of the Account.
  - The table is 1NF
- 3NF
  - Table is 1NF and 2NF
  - The non trivial functional dependency is either superkey or prime

Throughout the table, userID remains the superkey, any attribute without userID is trivial, making this table 3NF.

3. Transactions (transID, userID, bankName, accType, transDateTime, location, summary, transType, amount, netBalance)

   FD = {transID → transID, transID → userID, transID → bankName, transID → accType, transID → transDateTime, transID → location, transID → summary, transID → transType, transID → amount, transID → netBalance}

{transID}	{transID, userID, bankName, accType, transDateTime, location, summary, transType, amount, netBalance}	Superkey
{userID}	{userID}	Trivial
{bankName}	{bankName}	Trivial
{accType}	{accType}	Trivial
{transDateTime}	{transDateTime}	Trivial
{location}	{location}	Trivial
{summary}	{summary}	Trivial
{transType}	{transType}	Trivial
{amount}	{amount}	Trivial
{netBalance}	{netBalance}	Trivial

- 1NF
  - Every column in the Transactions table is atomic
  - Every column is the Transactions table is the same type
- 2NF
  - TransID is the primary key for the table and the columns will rely on the transID for the information of the Transaction.
  - The table is 1NF
- 3NF
  - Table is 1NF and 2NF
  - The non trivial functional dependency is either superkey or prime

Throughout the table, transID remains the superkey, any attribute without transID is trivial, making this table 3NF.

4. Banks (bankName, balance)

   FD = {bankName → bankName, bankName → balance}

{bankName}	{bankName, balance}	Superkey
{balance}	{balance}	Trivial

- 1NF
  - Every column in the Banks table is atomic
  - Every column is the Banks table is the same type
- 2NF
  - UserID is the primary key for the table and the columns will rely on the bankName for the information of the Bank.
  - The table is 1NF
- 3NF
  - Table is 1NF and 2NF
  - The non trivial functional dependency is either superkey or prime

Throughout the table, bankName remains the superkey, any attribute without bankName is trivial, making this table 3NF.

5. Loans (loanID, userId, bankName, accType, amount, loanDate, dueDate)

   FD = {loanID → loanID, loanID → userId, loanID → bankName, loanID → accType, loanID → amount, loanID → loanDate, loanID → dueDate}

{loanID}	{loanID, userId, bankName, accType, amount, loanDate, dueDate}	Superkey
{userId}	{userId}	Trivial
{bankName}	{bankName}	Trivial
{accType}	{accType}	Trivial

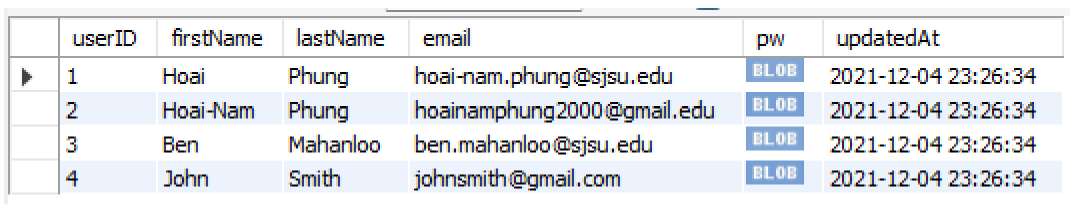
{amount}	{amount}	Trivial
{loanDate}	{loanDate}	Trivial
{dueDate}	{dueDate}	Trivial

- 1NF
  - Every column in the Loans table is atomic
  - Every column is the Loans table is the same type
- 2NF
  - LoanID is the primary key for the table and the columns will rely on the loanID for the information of the Loan.
  - The table is 1NF
- 3NF
  - Table is 1NF and 2NF
  - The non trivial functional dependency is either superkey or prime

Throughout the table, loanID remains the superkey, any attribute without loanID is trivial, making this table 3NF.

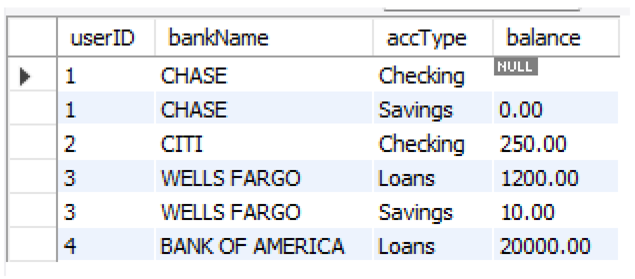
6. Screenshots of all relations after populating initial data. Label each relation clearly.

Users Table



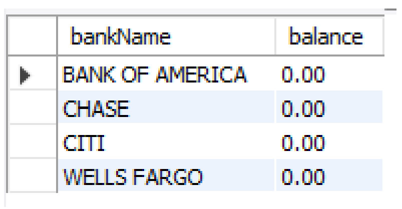
userID	firstName	lastName	email	pw	updatedAt
1	Hoai	Phung	hoai-nam.phung@sjsu.edu	BLOB	2021-12-04 23:26:34
2	Hoai-Nam	Phung	hoainamphung2000@gmail.edu	BLOB	2021-12-04 23:26:34
3	Ben	Mahanloo	ben.mahanloo@sjsu.edu	BLOB	2021-12-04 23:26:34
4	John	Smith	johnsmith@gmail.com	BLOB	2021-12-04 23:26:34

Accounts Table



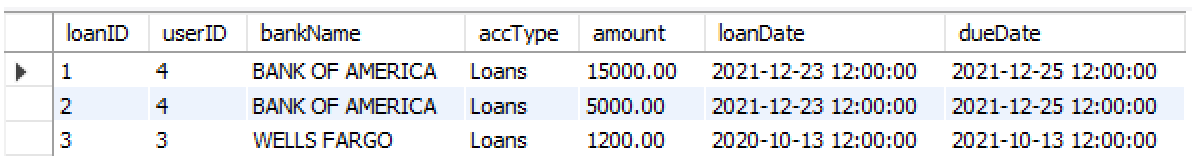
userID	bankName	accType	balance
1	CHASE	Checking	NULL
1	CHASE	Savings	0.00
2	CITI	Checking	250.00
3	WELLS FARGO	Loans	1200.00
3	WELLS FARGO	Savings	10.00
4	BANK OF AMERICA	Loans	20000.00

Banks Table



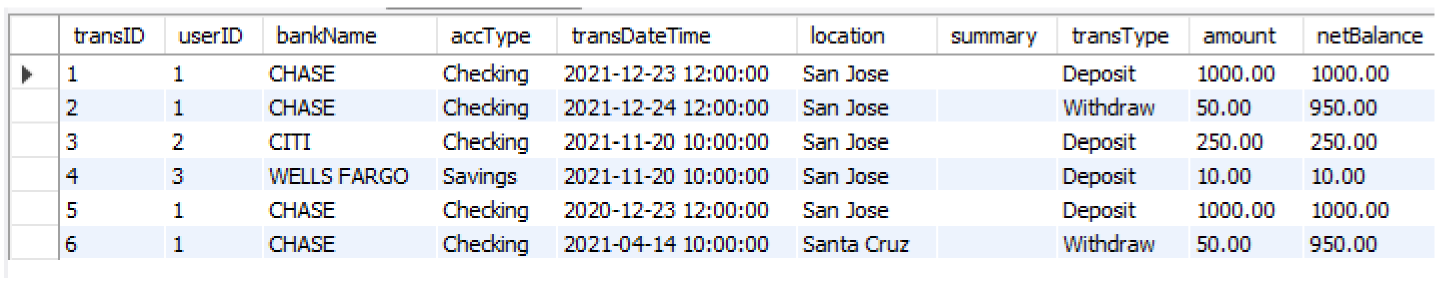
bankName	balance
BANK OF AMERICA	0.00
CHASE	0.00
CITI	0.00
WELLS FARGO	0.00

Loans Table



loanID	userID	bankName	accType	amount	loanDate	dueDate
1	4	BANK OF AMERICA	Loans	15000.00	2021-12-23 12:00:00	2021-12-25 12:00:00
2	4	BANK OF AMERICA	Loans	5000.00	2021-12-23 12:00:00	2021-12-25 12:00:00
3	3	WELLS FARGO	Loans	1200.00	2020-10-13 12:00:00	2021-10-13 12:00:00

Transactions Table



transID	userID	bankName	accType	transDateTime	location	summary	transType	amount	netBalance
1	1	CHASE	Checking	2021-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
2	1	CHASE	Checking	2021-12-24 12:00:00	San Jose		Withdraw	50.00	950.00
3	2	CITI	Checking	2021-11-20 10:00:00	San Jose		Deposit	250.00	250.00
4	3	WELLS FARGO	Savings	2021-11-20 10:00:00	San Jose		Deposit	10.00	10.00
5	1	CHASE	Checking	2020-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
6	1	CHASE	Checking	2021-04-14 10:00:00	Santa Cruz		Withdraw	50.00	950.00

7. List of at least 15 distinct functions excluding functions done by DBA. List each function in English and include all associated SQL statements, triggers, and stored procedures to support that particular function.

1. Archive users that haven't logged in since a certain date.
2. Register to be a user
3. Sign in as a user/Sign out as a user
4. Delete user
5. Create account

6. Close (delete) account
7. Display user's accounts at a given bank.
8. Retrieve account balance
9. Retrieve user's net worth (cumulative balance across all accounts)
10. Deposit money into an account
11. Withdraw money from an account
12. Show 10 most recent transactions on a bank account
13. Show all transactions on a bank account for a given month
14. Create a bank
15. Retrieve bank balance
16. Take out a loan from a bank as a user.
17. Pay back (part) of a specific loan from a bank as a user.
18. Display all of a user's loans.

8. The following requirements are for me to check there are at least 5 significantly different queries involving different relations and attributes.

- **Please highlight in bold the five significantly different types of SQL (one correlated subquery, group by and having, aggregation, outer join, and mathematical set operation)**
    1. **#0 = Correlated Subquery**
    2. **#6 = LEFT JOIN (instead of OUTER JOIN)**
    3. **#8 = Aggregation**
    4. **#11 = Mathematical Set Operation *(DIFFERENCE isn't allowed in SQL, so we used a substitution as seen in class)***
    5. **#12 = GROUP BY … HAVING**
- **List all SQL select statements**
- **List all SQL update statements**
- **List all SQL delete statements**
- **List all SQL insert statements**
- **List all SQL triggers**
- **List all SQL stored procedures**

#	Category	Functional Requirement	Stored Procedure	MySQL Query
0	**ArchivedUsers, Users**	**Archive users that haven't logged in since a certain date.**	**ArchiveUsers (cutoffLoginDate)**	**`REPLACE INTO ArchivedUsers(userId, firstName, lastName, email, pw, updatedAt) ( SELECT u.userId, u.firstName, u.lastName, u.email, u.pw, u.updatedAt FROM Users u WHERE u.updatedAt <= cutoffLoginDate);`**
1	Users	Register to be a user	CreateUser(firstName, lastName, email, pw)	INSERT INTO Users(firstName, lastName, email, pw, updatedAt) VALUES(firstName, lastName, email, pw, CURRENT_TIMESTAMP());
2	Users	Sign in as a user	GetUserID(email, pw)	SELECT Users.userID INTO userID FROM Users WHERE Users.email=email AND Users.pw=pw;
N/A	Users	Sign out as a user	N/A	N/A
3	Users	Delete user	DeleteUser(userID)	DELETE FROM Users WHERE Users.userID = userID;
4	Users, Accounts	Create account	CreateBankAccount(bankName, accType, balance, userID)	`INSERT INTO Accounts(bankName, accType, balance, userID) VALUES(bankName, accType, balance, userId);`
5	Accounts	Close (delete) account	DeleteBankAccount(bankName, accType, userID)	`DELETE FROM Accounts WHERE Accounts.userID = userID AND Accounts.bankName = bankName AND Accounts.accType = accType;`
6	**Users, Accounts, Banks**	**Display user's accounts at a given bank.**	**GetAllUserBankAccountsAtBank (bankName, userID)**	**`SELECT firstName, accType, balance FROM Accounts LEFT JOIN Users ON (Accounts.userId = Users.userId)`**
7	Accounts	Retrieve account balance	GetBankAccountBalance (bankName, accType, userID)	`SELECT balance FROM Accounts WHERE Accounts.bankName = bankName AND Accounts.accType = accType AND Accounts.userID = userID;`
8	**Users, Accounts**	**Retrieve user's net worth (cumulative balance across all accounts)**	**CalculateNetWorth (userID)**	**`SELECT SUM(balance) as NetWorth FROM Accounts WHERE Accounts.userID = userID;`**

9	Transactions, Accounts	Deposit money into an account	Deposit(userID, bankName, accType, amount)	INSERT INTO Transactions (userID, bankName, accType, transType, amount) VALUES(userID, bankName, accType, "Deposit", amount)
10	Transactions, Accounts	Withdraw money from an account	Withdraw(userID, bankName, accType, amount)	INSERT INTO Transactions (userID, bankName, accType, transType, amount) VALUES(userID, bankName, accType, "Withdrawal", amount)

11	**Transactions, Accounts**	**Show 10 most recent transactions on a bank account**	**GetRecentTransactions(userID, bankName, accType)**	**SELECT * FROM Transactions ORDER BY transDateTime[ASC] LIMIT 10
				INTERSECT
				SELECT * FROM Transactions WHERE userID = userID AND bankName = bankName AND accType = accType
				
				becomes
				
				SELECT * FROM (SELECT TOP 10 * FROM Transactions ORDER BY transDateTime[ASC] LIMIT 10 AS t1) WHERE (t1.userId, t1.bankName, t1.accType) IN (SELECT t2.userId, t2.bankName, t2.accType FROM (SELECT * FROM Transactions WHERE userID = userID AND bankName = bankName AND accType = accType AS t2))**

12	**Transactions, Accounts**	**Show all transactions on a bank account for a given month**	**GetMonthlyTransactions(userID, bankName, accType, filterDate)**	**SELECT * FROM Transactions WHERE userID = userID AND bankName = bankName AND accType = accType GROUP BY transDateTime HAVING MONTH(transDateTime) = MONTH(filterDate) AND YEAR(transDateTime) = YEAR(filterDate);**
13	Banks	Create a bank	CreateBank(bankName, balance)	INSERT INTO Banks(bankName, balance) VALUES(bankName, balance)
14	Banks	Retrieve bank balance	GetBankBalance(bankName)	INSERT INTO Banks(bankName, balance) VALUES(bankName, balance)
15	Banks, Loans, Accounts, Transactions	Take out a loan from a bank as a user.	NewLoan(userID, bankName, accType, amount, loanDateTime, dueDateTime)	INSERT INTO Loans(userID, bankName, accType, amount, loanDateTime, dueDateTime) VALUES(userID, bankName, accType, amount, loanDateTime, dueDateTime);
16	Users, Accounts	Display all of a user's loans.	GetAllLoanUsers(userID)	SELECT * FROM Loans WHERE userID = userID;

Three Trigger(s) in a valid SQL syntax - Loans and Transactions

Triggers

#	Trigger	MySQL Query

1	On Transaction insertion/deletion/update -> update account balance	DELIMITER $$
		CREATE TRIGGER update_account_on_new_transaction
		AFTER INSERT ON Transactions
		FOR EACH ROW
		BEGIN
		    UPDATE Accounts a SET a.balance =
		    CASE new.transType
		       WHEN "Withdrawal" THEN a.balance - new.amount
		       WHEN "Deposit" THEN a.balance + new.amount
		    END
		    WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType;
		END $$
		DELIMITER ;

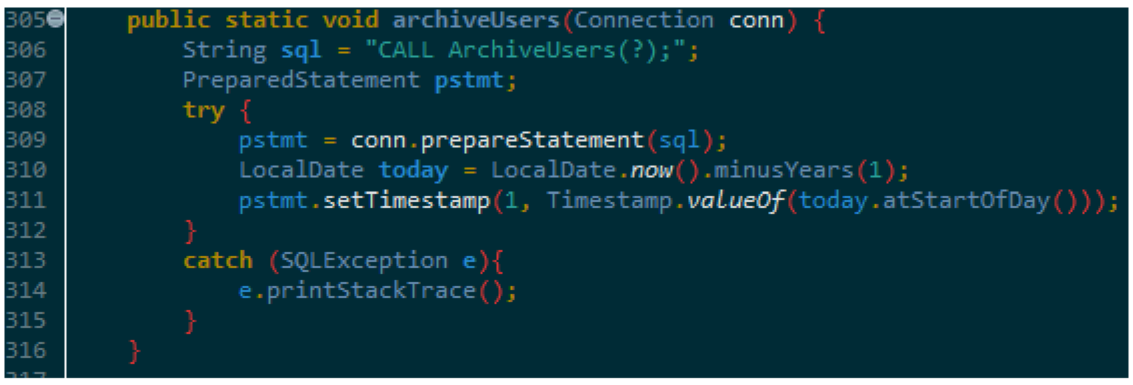
2	On Loan insertion -> update account balance	DELIMITER $$
		CREATE TRIGGER update_account_on_new_loan
		AFTER INSERT ON Loans
		FOR EACH ROW
		BEGIN
		    IF new.accType = "Loans"
		       THEN UPDATE Accounts a SET a.balance = a.balance + new.amount

		`WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType; ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Not a Loans account; insertion failed."; END IF; END $$ DELIMITER ;`
3	On Loan update -> update account balance, delete Loan if paid off	`DELIMITER $$ CREATE TRIGGER update_account_on_update_loan AFTER UPDATE ON Loans FOR EACH ROW BEGIN IF new.accType = "Loans" THEN UPDATE Accounts a SET a.balance = a.balance + new.amount - old.amount WHERE a.userID = new.userID AND a.bankName = new.bankName AND a.accType = new.accType; IF new.amount = 0 THEN DELETE FROM Loans WHERE Loans.userID = new.userID AND Loans.bankName = new.bankName AND Loans.accType = new.accType; END IF; ELSE SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Not a Loans account; insertion failed."; END IF; END $$ DELIMITER ;`

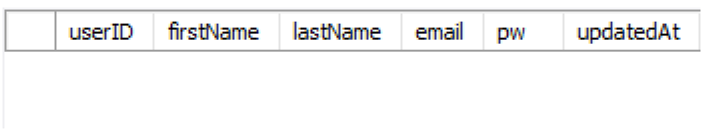
9. Screenshots to demonstrate the following functionality. Label each screenshot clearly.

- **each 15 functions - for each function, include a screenshot at Java level and another screenshot of corresponding table contents to show the function reads from or write on the database.**
- **archiving - one screenshot at Java level and another screenshot to show the table is achieved at the database level.**
- **key constraint and foreign key constraint violations - for each constraint, a screenshot(s) to show SQLException and the reason for the exception. For example, a screenshot showing java.sql.SQLIntegrityConstraintViolationException: Duplicate entry '1' for key 'PRIMARY' or something similar is enough for the primary key constraint violation.**

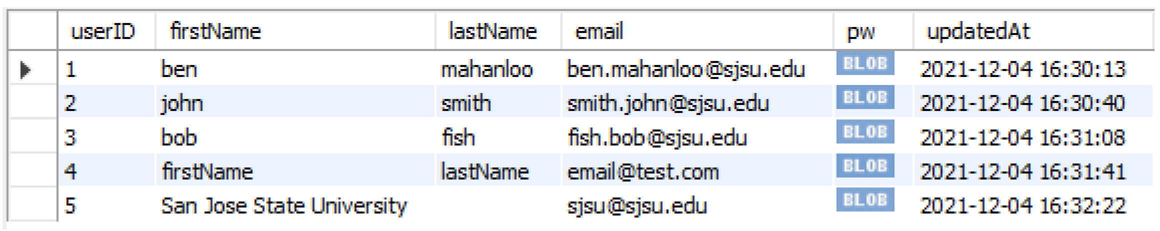
Archiving



```
    public static void archiveUsers(Connection conn) {
        String sql = "CALL ArchiveUsers(?);";
        PreparedStatement pstmt;
        try {
            pstmt = conn.prepareStatement(sql);
            LocalDate today = LocalDate.now().minusYears(1);
            pstmt.setTimestamp(1, Timestamp.valueOf(today.atStartOfDay()));
        }
        catch (SQLException e){
            e.printStackTrace();
        }
    }
```

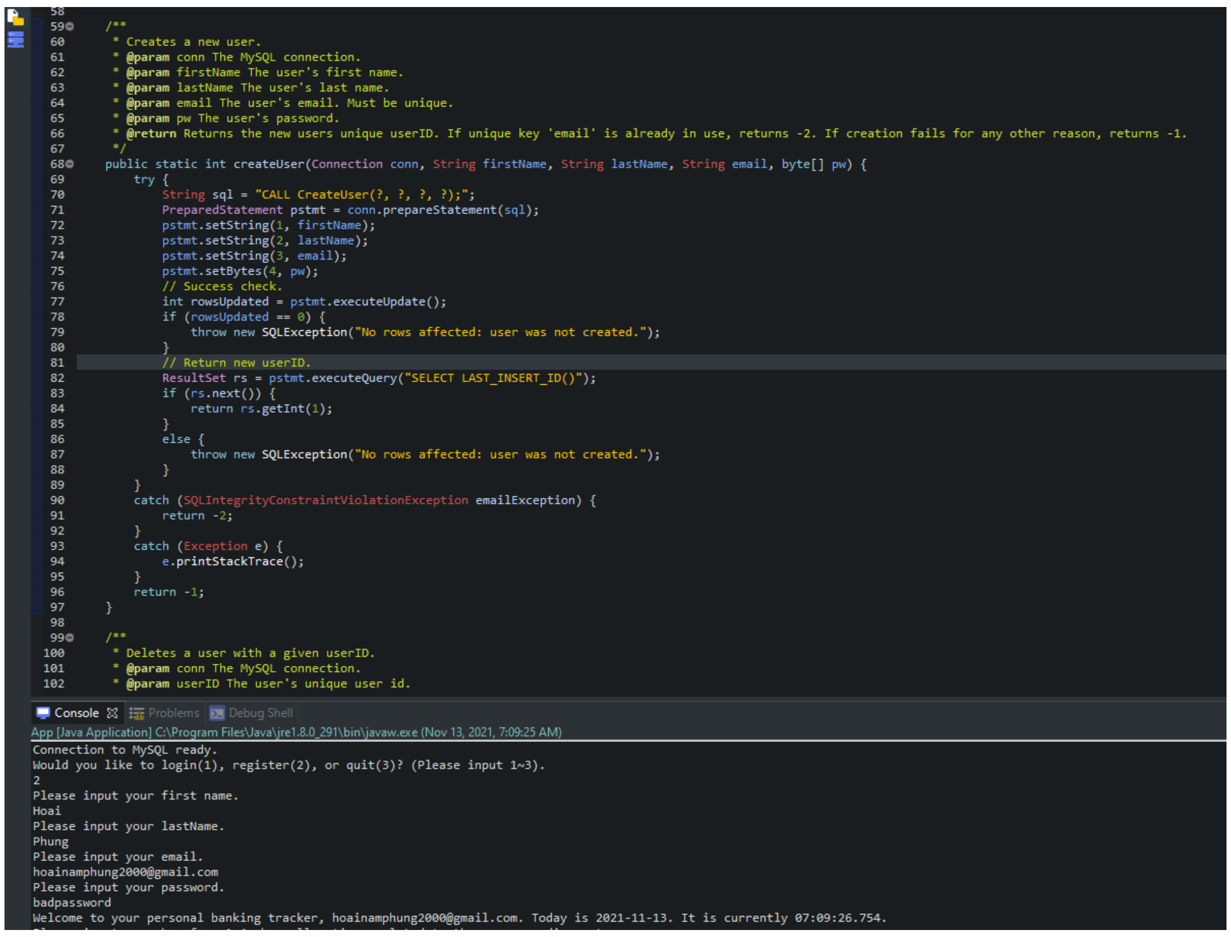


userID	firstName	lastName	email	pw	updatedAt



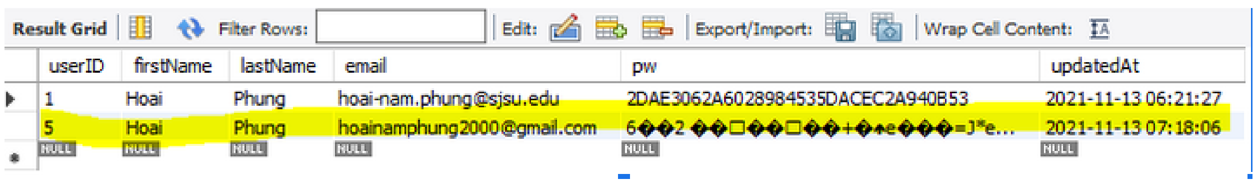
userID	firstName	lastName	email	pw	updatedAt
1	ben	mahanloo	ben.mahanloo@sjsu.edu	BLOB	2021-12-04 16:30:13
2	john	smith	smith.john@sjsu.edu	BLOB	2021-12-04 16:30:40
3	bob	fish	fish.bob@sjsu.edu	BLOB	2021-12-04 16:31:08
4	firstName	lastName	email@test.com	BLOB	2021-12-04 16:31:41
5	San Jose State University		sjsu@sjsu.edu	BLOB	2021-12-04 16:32:22

1. **User Registration (Success)**



```
    /**
     * Creates a new user.
     * @param conn The MySQL connection.
     * @param firstName The user's first name.
     * @param lastName The user's last name.
     * @param email The user's email. Must be unique.
     * @param pw The user's password.
     * @return Returns the new users unique userID. If unique key 'email' is already in use, returns -2. If creation fails for any other reason, returns -1.
     */
    public static int createUser(Connection conn, String firstName, String lastName, String email, byte[] pw) {
        try {
            String sql = "CALL CreateUser(?, ?, ?, ?);";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, firstName);
            pstmt.setString(2, lastName);
            pstmt.setString(3, email);
            pstmt.setBytes(4, pw);
            // Success check.
            int rowsUpdated = pstmt.executeUpdate();
            if (rowsUpdated == 0) {
                throw new SQLException("No rows affected: user was not created.");
            }
            // Return new userID.
            ResultSet rs = pstmt.executeQuery("SELECT LAST_INSERT_ID()");
            if (rs.next()) {
                return rs.getInt(1);
            }
            else {
                throw new SQLException("No rows affected: user was not created.");
            }
        }
        catch (SQLIntegrityConstraintViolationException emailException) {
            return -2;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    /**
     * Deletes a user with a given userID.
     * @param conn The MySQL connection.
     * @param userID The user's unique user id.
```

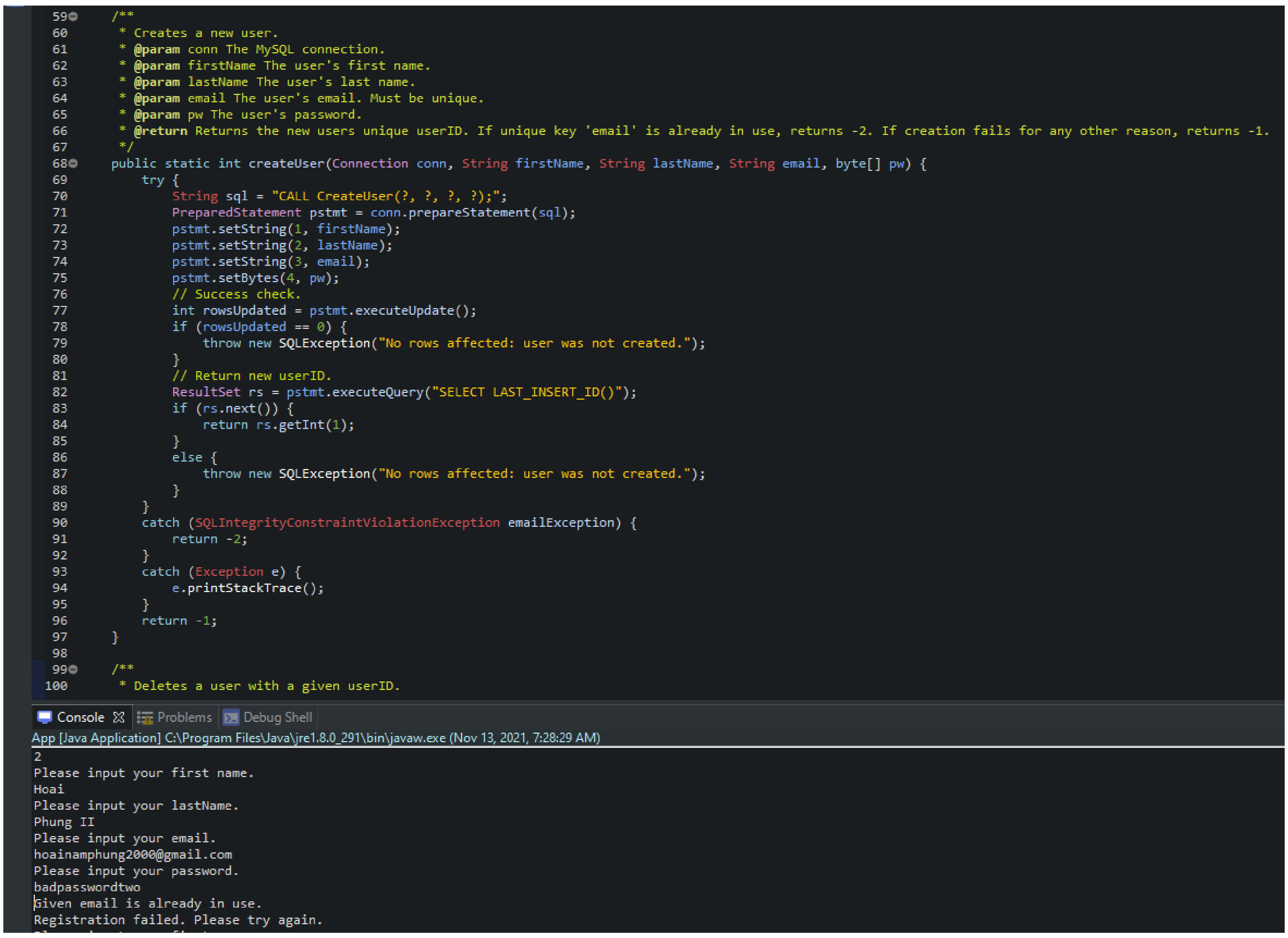
```
App [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Nov 13, 2021, 7:09:25 AM)
Connection to MySQL ready.
Would you like to login(1), register(2), or quit(3)? (Please input 1~3).
2
Please input your first name.
Hoai
Please input your lastName.
Phung
Please input your email.
hoainamphung2000@gmail.com
Please input your password.
badpassword
Welcome to your personal banking tracker, hoainamphung2000@gmail.com. Today is 2021-11-13. It is currently 07:09:26.754.
```



Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:

userID	firstName	lastName	email	pw	updatedAt
1	Hoai	Phung	hoai-nam.phung@sjsu.edu	2DAE3062A6028984535DACEC2A940B53	2021-11-13 06:21:27
5	Hoai	Phung	hoainamphung2000@gmail.com	6��2 ��□��□��+�♠e���=J*e...	2021-11-13 07:18:06
NULL	NULL	NULL	NULL	NULL	NULL

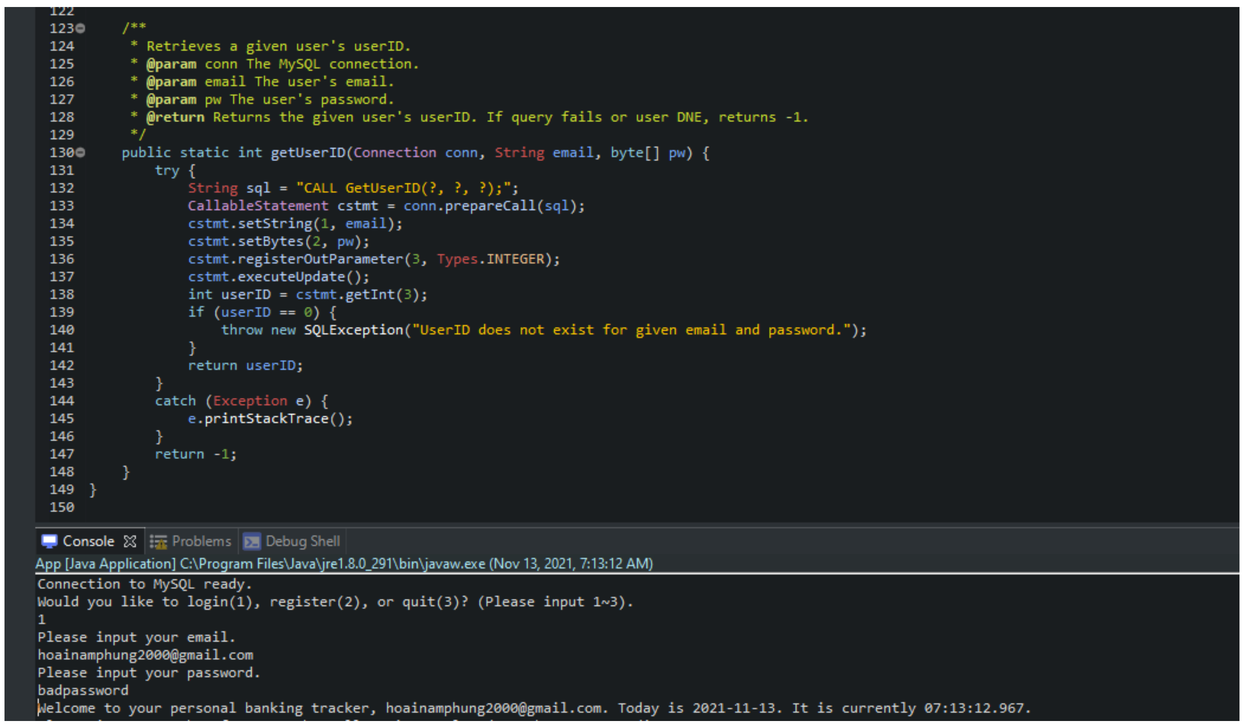
User Registration (Failure, UNIQUE KEY constraint broken for given email)



```
    /**
     * Creates a new user.
     * @param conn The MySQL connection.
     * @param firstName The user's first name.
     * @param lastName The user's last name.
     * @param email The user's email. Must be unique.
     * @param pw The user's password.
     * @return Returns the new users unique userID. If unique key 'email' is already in use, returns -2. If creation fails for any other reason, returns -1.
     */
    public static int createUser(Connection conn, String firstName, String lastName, String email, byte[] pw) {
        try {
            String sql = "CALL CreateUser(?, ?, ?, ?);";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, firstName);
            pstmt.setString(2, lastName);
            pstmt.setString(3, email);
            pstmt.setBytes(4, pw);
            // Success check.
            int rowsUpdated = pstmt.executeUpdate();
            if (rowsUpdated == 0) {
                throw new SQLException("No rows affected: user was not created.");
            }
            // Return new userID.
            ResultSet rs = pstmt.executeQuery("SELECT LAST_INSERT_ID()");
            if (rs.next()) {
                return rs.getInt(1);
            }
            else {
                throw new SQLException("No rows affected: user was not created.");
            }
        }
        catch (SQLIntegrityConstraintViolationException emailException) {
            return -2;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    /**
     * Deletes a user with a given userID.
```

```
App [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Nov 13, 2021, 7:28:29 AM)
2
Please input your first name.
Hoai
Please input your lastName.
Phung II
Please input your email.
hoainamphung2000@gmail.com
Please input your password.
badpasswordtwo
Given email is already in use.
Registration failed. Please try again.
```

2. **User Sign In (Success)**



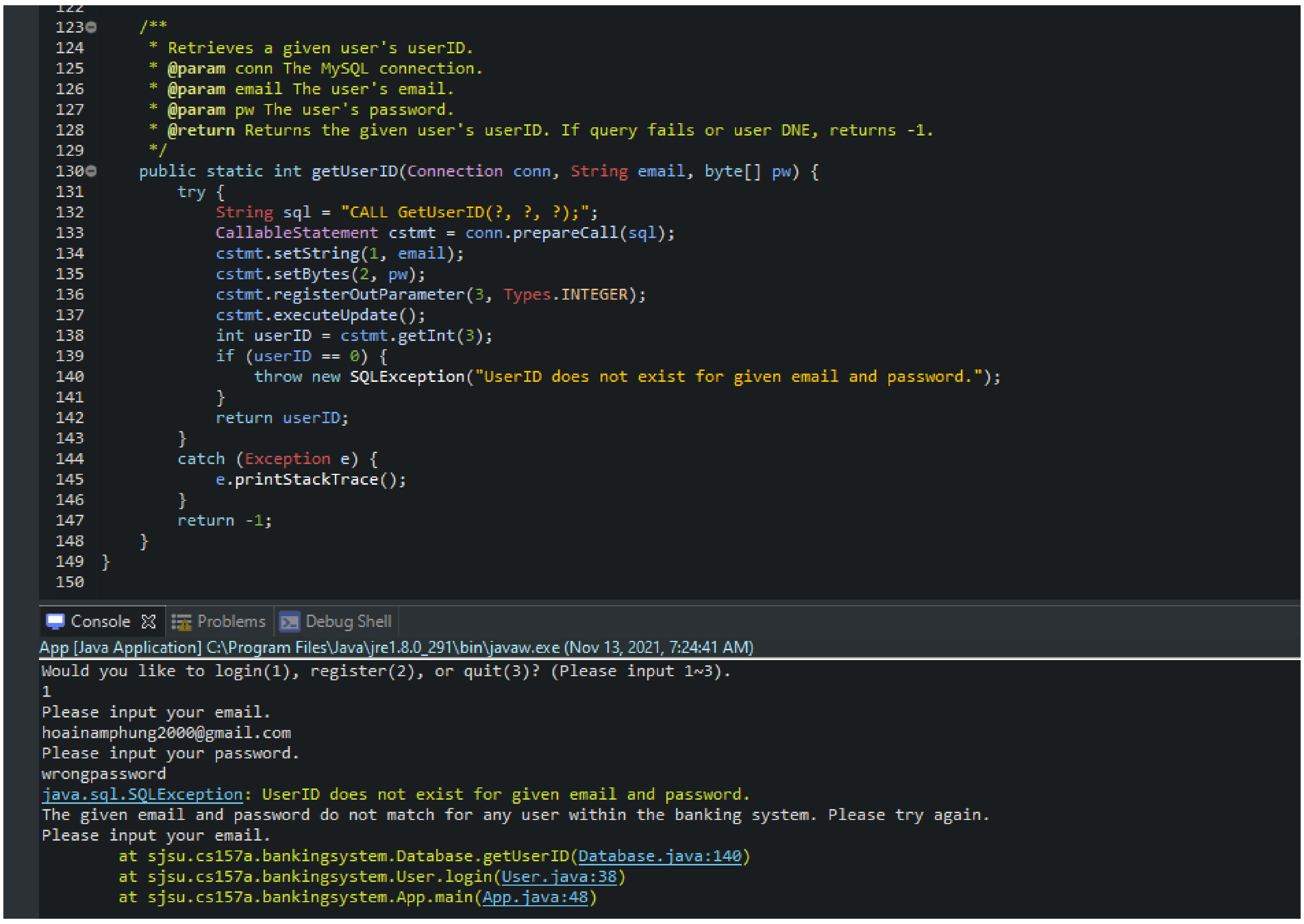
```java
    /**
     * Retrieves a given user's userID.
     * @param conn The MySQL connection.
     * @param email The user's email.
     * @param pw The user's password.
     * @return Returns the given user's userID. If query fails or user DNE, returns -1.
     */
    public static int getUserID(Connection conn, String email, byte[] pw) {
        try {
            String sql = "CALL GetUserID(?, ?, ?);";
            CallableStatement cstmt = conn.prepareCall(sql);
            cstmt.setString(1, email);
            cstmt.setBytes(2, pw);
            cstmt.registerOutParameter(3, Types.INTEGER);
            cstmt.executeUpdate();
            int userID = cstmt.getInt(3);
            if (userID == 0) {
                throw new SQLException("UserID does not exist for given email and password.");
            }
            return userID;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }
}
```

Console | Problems | Debug Shell

App [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Nov 13, 2021, 7:13:12 AM)

```
Connection to MySQL ready.
Would you like to login(1), register(2), or quit(3)? (Please input 1~3).
1
Please input your email.
hoainamphung2000@gmail.com
Please input your password.
badpassword
Welcome to your personal banking tracker, hoainamphung2000@gmail.com. Today is 2021-11-13. It is currently 07:13:12.967.
```

User Sign In (Failure, SELECT returns nothing for given email-password combination.)



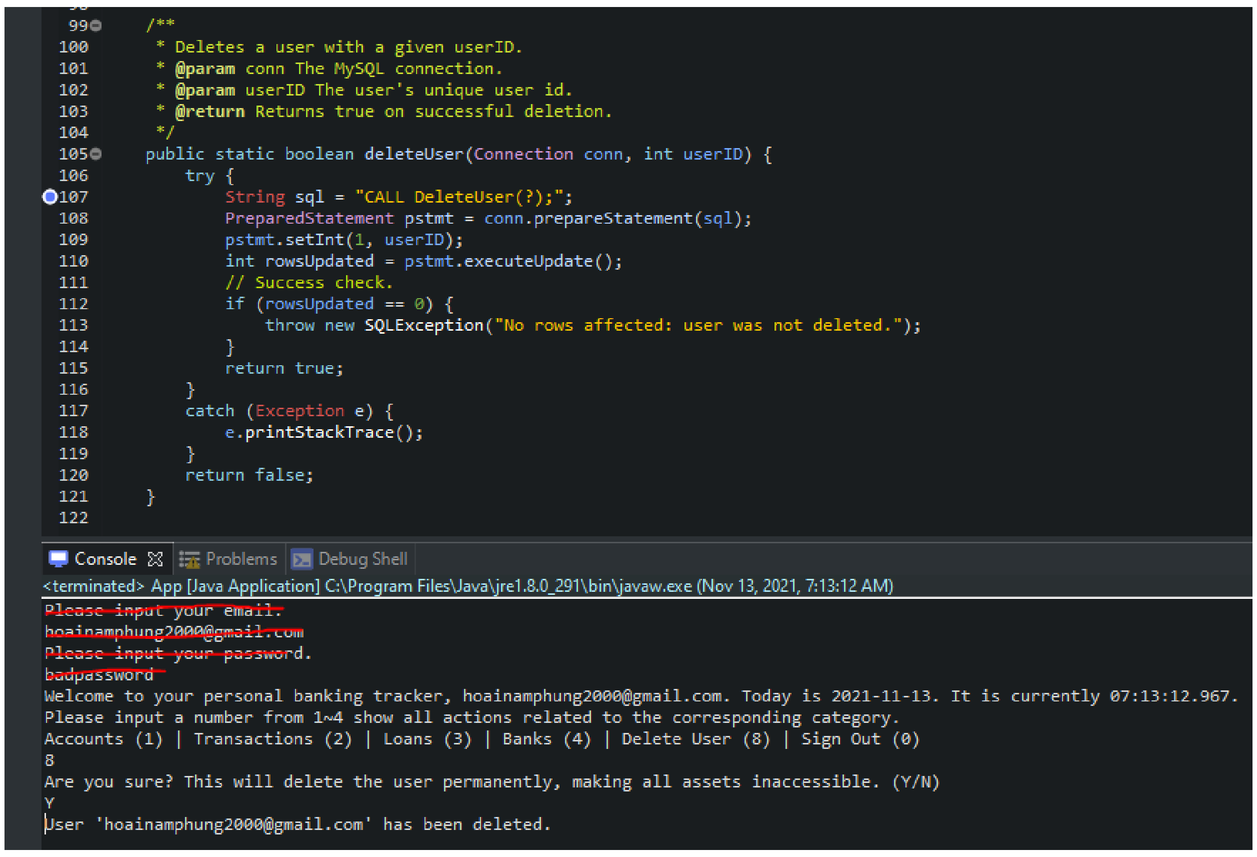
```java
    /**
     * Retrieves a given user's userID.
     * @param conn The MySQL connection.
     * @param email The user's email.
     * @param pw The user's password.
     * @return Returns the given user's userID. If query fails or user DNE, returns -1.
     */
    public static int getUserID(Connection conn, String email, byte[] pw) {
        try {
            String sql = "CALL GetUserID(?, ?, ?);";
            CallableStatement cstmt = conn.prepareCall(sql);
            cstmt.setString(1, email);
            cstmt.setBytes(2, pw);
            cstmt.registerOutParameter(3, Types.INTEGER);
            cstmt.executeUpdate();
            int userID = cstmt.getInt(3);
            if (userID == 0) {
                throw new SQLException("UserID does not exist for given email and password.");
            }
            return userID;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }
}
```

Console | Problems | Debug Shell

App [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Nov 13, 2021, 7:24:41 AM)

```
Would you like to login(1), register(2), or quit(3)? (Please input 1~3).
1
Please input your email.
hoainamphung2000@gmail.com
Please input your password.
wrongpassword
java.sql.SQLException: UserID does not exist for given email and password.
The given email and password do not match for any user within the banking system. Please try again.
Please input your email.
	at sjsu.cs157a.bankingsystem.Database.getUserID(Database.java:140)
	at sjsu.cs157a.bankingsystem.User.login(User.java:38)
	at sjsu.cs157a.bankingsystem.App.main(App.java:48)
```

3. User Deletion

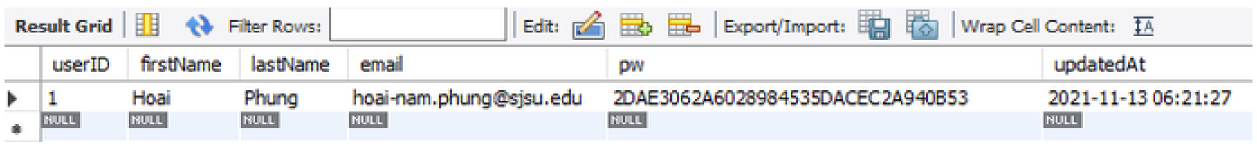


```java
    /**
     * Deletes a user with a given userID.
     * @param conn The MySQL connection.
     * @param userID The user's unique user id.
     * @return Returns true on successful deletion.
     */
    public static boolean deleteUser(Connection conn, int userID) {
        try {
            String sql = "CALL DeleteUser(?);";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, userID);
            int rowsUpdated = pstmt.executeUpdate();
            // Success check.
            if (rowsUpdated == 0) {
                throw new SQLException("No rows affected: user was not deleted.");
            }
            return true;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }
```

Console ☒ | Problems | Debug Shell

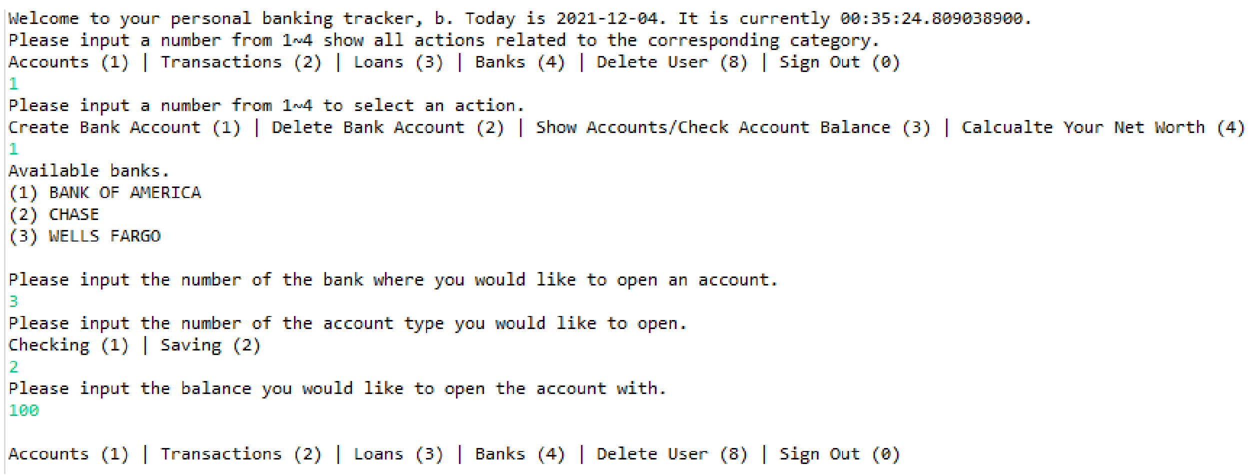
<terminated> App [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Nov 13, 2021, 7:13:12 AM)

```
~~Please input your email.~~
~~hoainamphung2000@gmail.com~~
~~Please input your password.~~
~~badpassword~~
Welcome to your personal banking tracker, hoainamphung2000@gmail.com. Today is 2021-11-13. It is currently 07:13:12.967.
Please input a number from 1~4 show all actions related to the corresponding category.
Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
8
Are you sure? This will delete the user permanently, making all assets inaccessible. (Y/N)
Y
User 'hoainamphung2000@gmail.com' has been deleted.
```

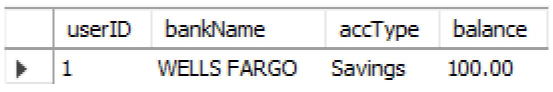


userID	firstName	lastName	email	pw	updatedAt
1	Hoai	Phung	hoai-nam.phung@sjsu.edu	2DAE3062A6028984535DACEC2A940B53	2021-11-13 06:21:27
NULL	NULL	NULL	NULL	NULL	NULL

4. Bank Account Creation

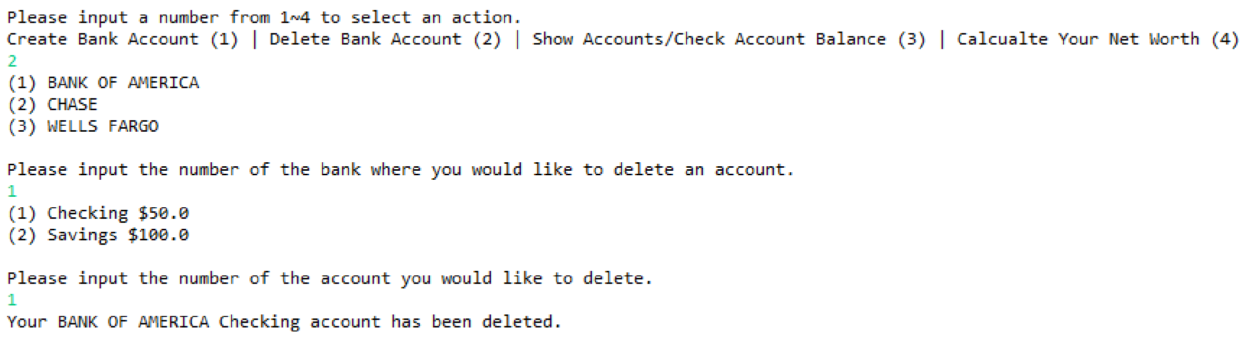


```
Welcome to your personal banking tracker, b. Today is 2021-12-04. It is currently 00:35:24.809038900.
Please input a number from 1~4 show all actions related to the corresponding category.
Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
1
Please input a number from 1~4 to select an action.
Create Bank Account (1) | Delete Bank Account (2) | Show Accounts/Check Account Balance (3) | Calcualte Your Net Worth (4)
1
Available banks.
(1) BANK OF AMERICA
(2) CHASE
(3) WELLS FARGO

Please input the number of the bank where you would like to open an account.
3
Please input the number of the account type you would like to open.
Checking (1) | Saving (2)
2
Please input the balance you would like to open the account with.
100

Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
```

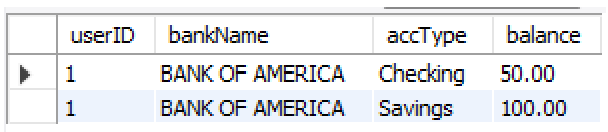


userID	bankName	accType	balance
1	WELLS FARGO	Savings	100.00

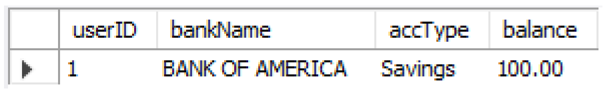
5. Bank Account Deletion



```
Please input a number from 1~4 to select an action.
Create Bank Account (1) | Delete Bank Account (2) | Show Accounts/Check Account Balance (3) | Calcualte Your Net Worth (4)
2
(1) BANK OF AMERICA
(2) CHASE
(3) WELLS FARGO

Please input the number of the bank where you would like to delete an account.
1
(1) Checking $50.0
(2) Savings $100.0

Please input the number of the account you would like to delete.
1
Your BANK OF AMERICA Checking account has been deleted.
```

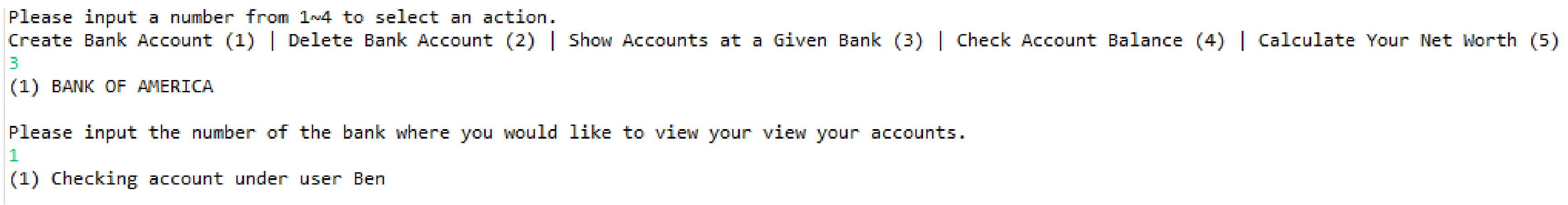


userID	bankName	accType	balance
1	BANK OF AMERICA	Checking	50.00
1	BANK OF AMERICA	Savings	100.00



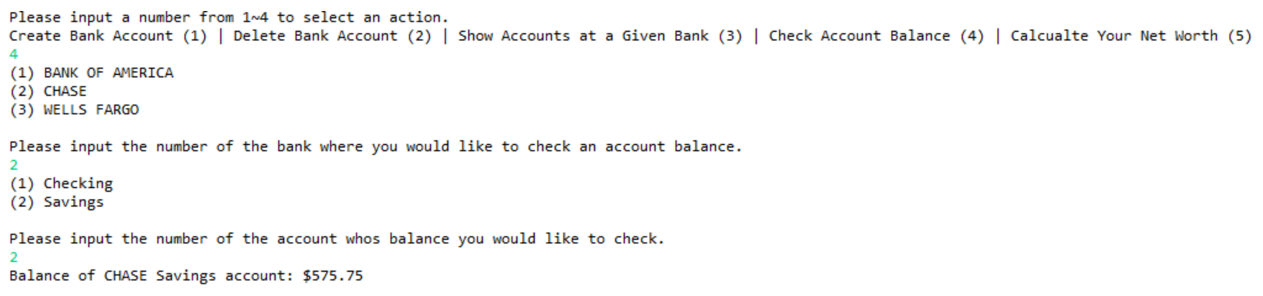
userID	bankName	accType	balance
1	BANK OF AMERICA	Savings	100.00

6. Show All Accounts at a Given Bank



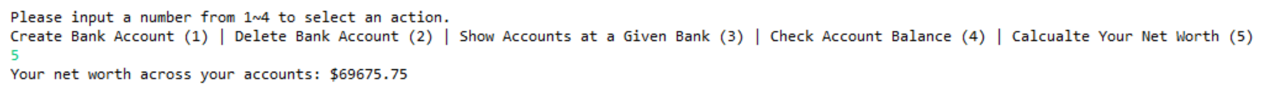
```
Please input a number from 1~4 to select an action.
Create Bank Account (1) | Delete Bank Account (2) | Show Accounts at a Given Bank (3) | Check Account Balance (4) | Calculate Your Net Worth (5)
3
(1) BANK OF AMERICA

Please input the number of the bank where you would like to view your view your accounts.
1
(1) Checking account under user Ben
```

7. Retrieve Account Balance



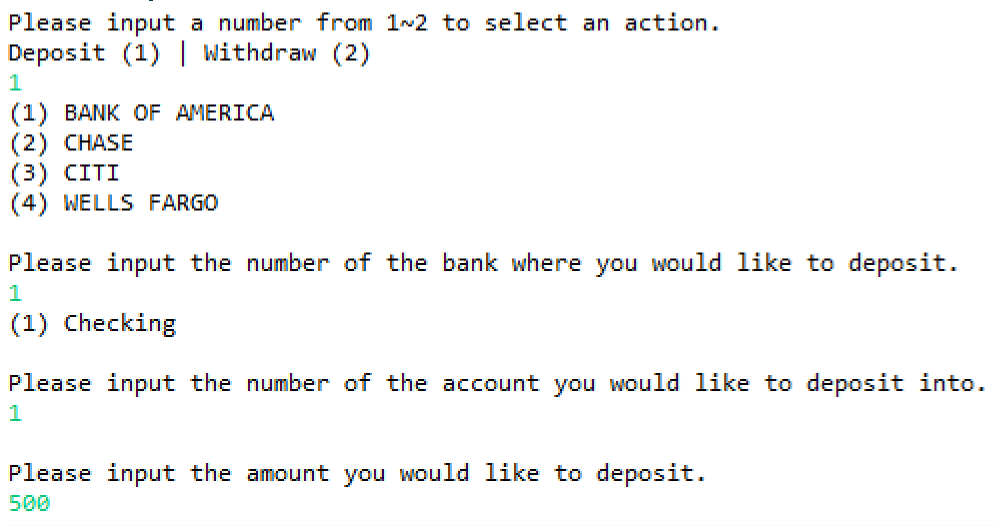
```
Please input a number from 1~4 to select an action.
Create Bank Account (1) | Delete Bank Account (2) | Show Accounts at a Given Bank (3) | Check Account Balance (4) | Calcualte Your Net Worth (5)
4
(1) BANK OF AMERICA
(2) CHASE
(3) WELLS FARGO

Please input the number of the bank where you would like to check an account balance.
2
(1) Checking
(2) Savings

Please input the number of the account whos balance you would like to check.
2
Balance of CHASE Savings account: $575.75
```

8. Calculate Net Worth

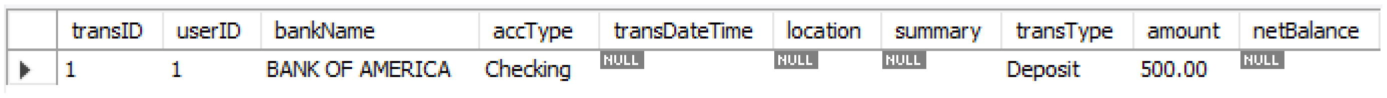


```
Please input a number from 1~4 to select an action.
Create Bank Account (1) | Delete Bank Account (2) | Show Accounts at a Given Bank (3) | Check Account Balance (4) | Calcualte Your Net Worth (5)
5
Your net worth across your accounts: $69675.75
```

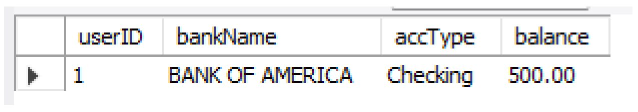
9. Deposit



```
Please input a number from 1~2 to select an action.
Deposit (1) | Withdraw (2)
1
(1) BANK OF AMERICA
(2) CHASE
(3) CITI
(4) WELLS FARGO

Please input the number of the bank where you would like to deposit.
1
(1) Checking

Please input the number of the account you would like to deposit into.
1

Please input the amount you would like to deposit.
500
```

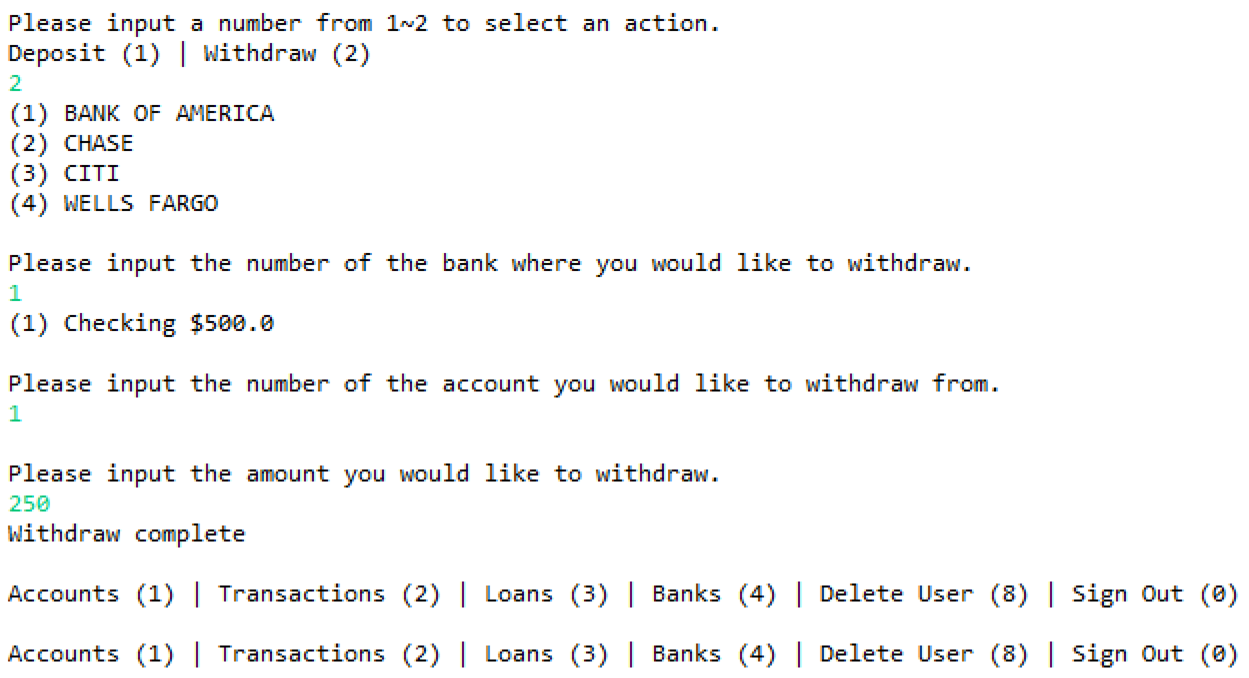


	transID	userID	bankName	accType	transDateTime	location	summary	transType	amount	netBalance
▶	1	1	BANK OF AMERICA	Checking	NULL	NULL	NULL	Deposit	500.00	NULL

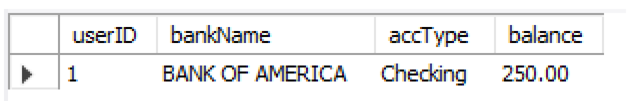


	userID	bankName	accType	balance
▶	1	BANK OF AMERICA	Checking	500.00

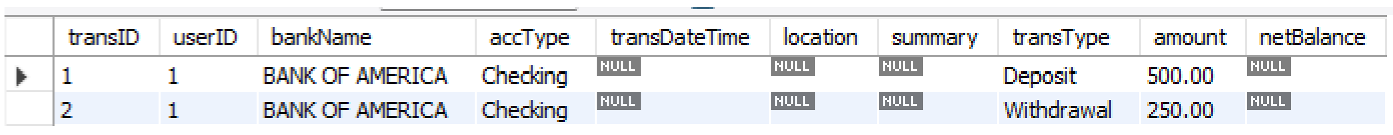
10. Withdraw



```
Please input a number from 1~2 to select an action.
Deposit (1) | Withdraw (2)
2
(1) BANK OF AMERICA
(2) CHASE
(3) CITI
(4) WELLS FARGO

Please input the number of the bank where you would like to withdraw.
1
(1) Checking $500.0

Please input the number of the account you would like to withdraw from.
1

Please input the amount you would like to withdraw.
250
Withdraw complete

Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)

Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
```

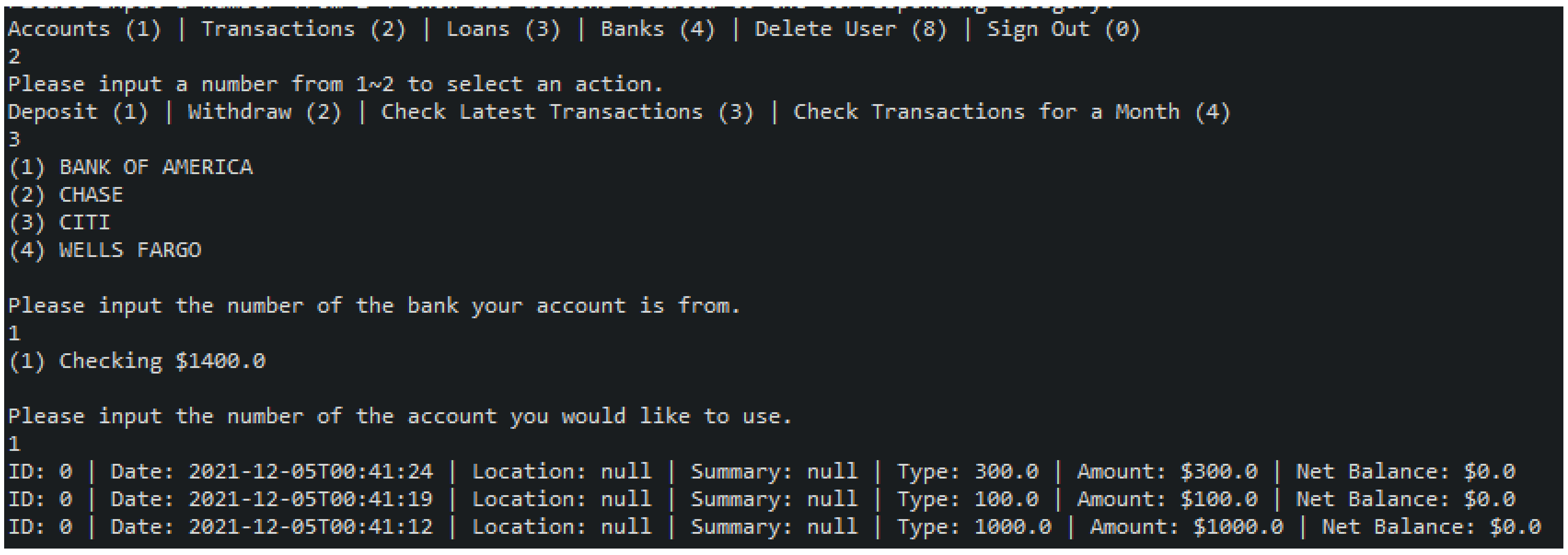


	userID	bankName	accType	balance
▶	1	BANK OF AMERICA	Checking	250.00

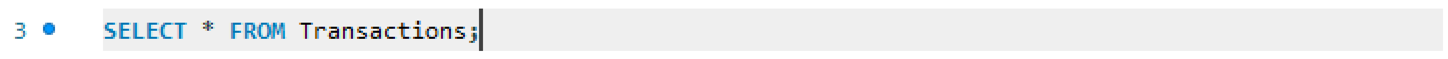


	transID	userID	bankName	accType	transDateTime	location	summary	transType	amount	netBalance
▶	1	1	BANK OF AMERICA	Checking	NULL	NULL	NULL	Deposit	500.00	NULL
	2	1	BANK OF AMERICA	Checking	NULL	NULL	NULL	Withdrawal	250.00	NULL

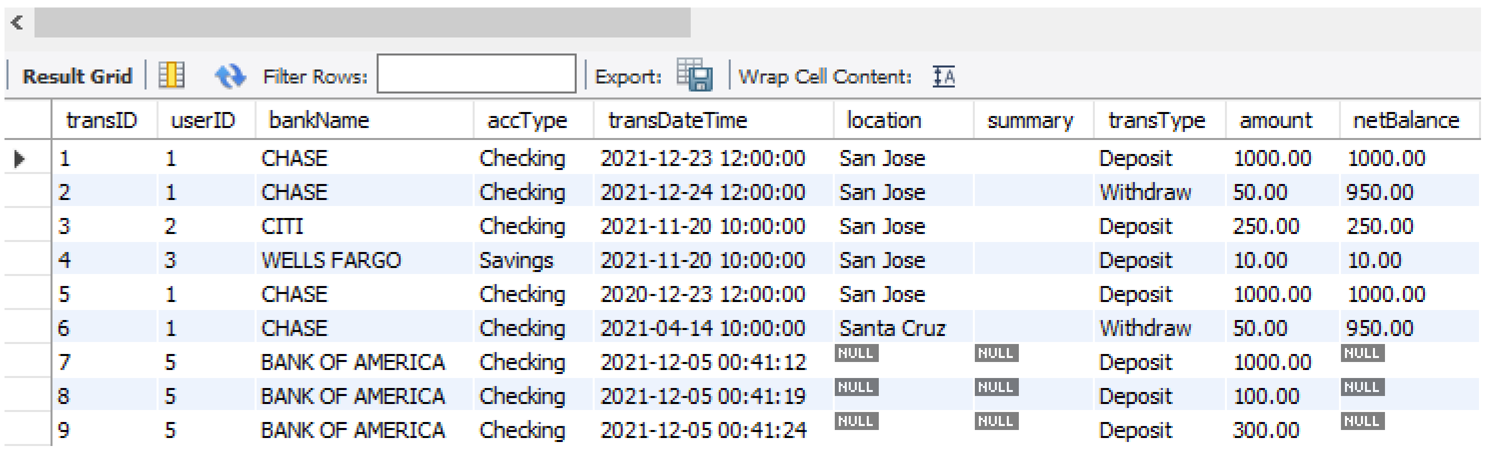
11. Get Recent Transactions



```
Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
2
Please input a number from 1~2 to select an action.
Deposit (1) | Withdraw (2) | Check Latest Transactions (3) | Check Transactions for a Month (4)
3
(1) BANK OF AMERICA
(2) CHASE
(3) CITI
(4) WELLS FARGO

Please input the number of the bank your account is from.
1
(1) Checking $1400.0

Please input the number of the account you would like to use.
1
ID: 0 | Date: 2021-12-05T00:41:24 | Location: null | Summary: null | Type: 300.0 | Amount: $300.0 | Net Balance: $0.0
ID: 0 | Date: 2021-12-05T00:41:19 | Location: null | Summary: null | Type: 100.0 | Amount: $100.0 | Net Balance: $0.0
ID: 0 | Date: 2021-12-05T00:41:12 | Location: null | Summary: null | Type: 1000.0 | Amount: $1000.0 | Net Balance: $0.0
```

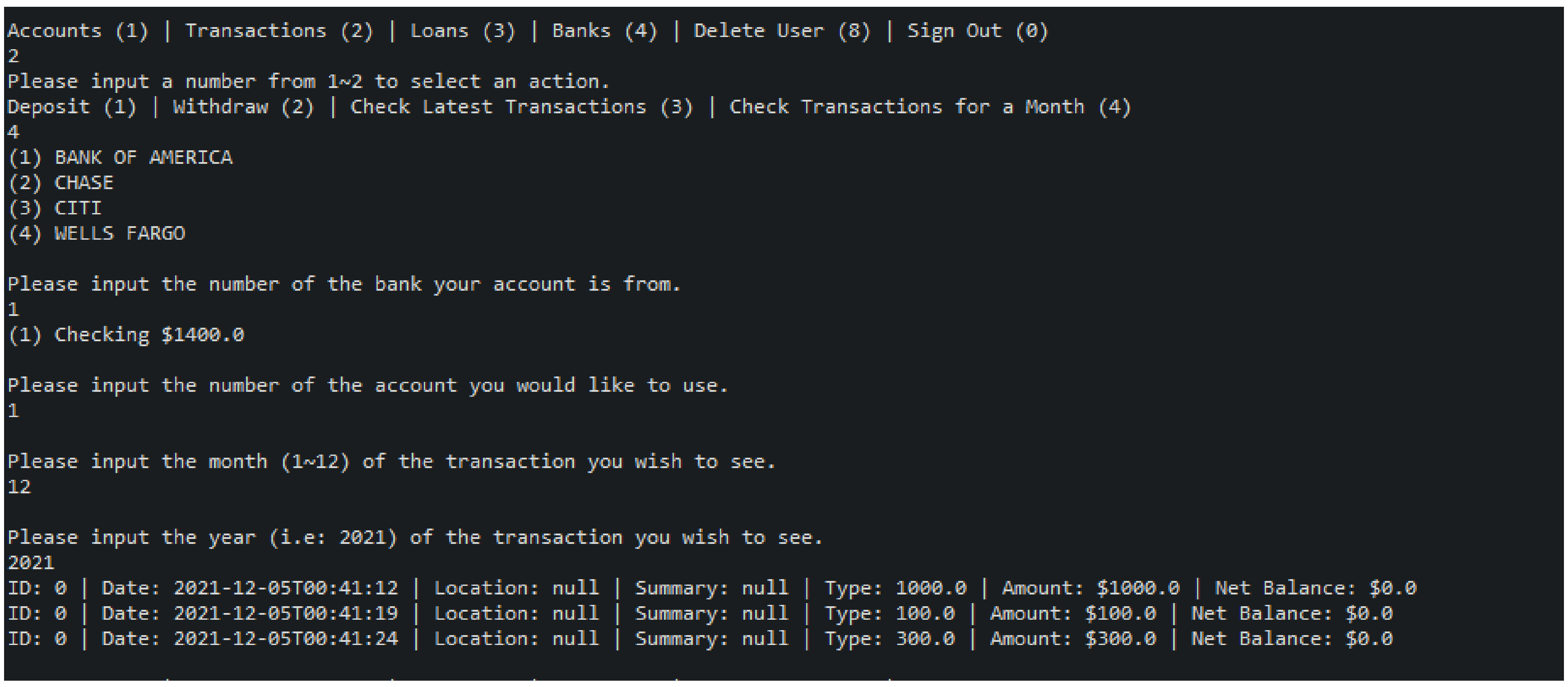


```sql
SELECT * FROM Transactions;
```

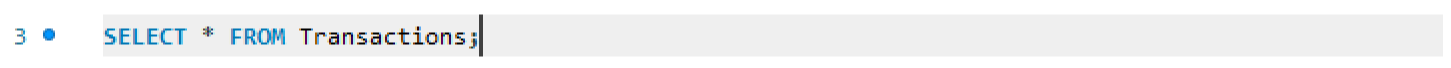


transID	userID	bankName	accType	transDateTime	location	summary	transType	amount	netBalance
1	1	CHASE	Checking	2021-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
2	1	CHASE	Checking	2021-12-24 12:00:00	San Jose		Withdraw	50.00	950.00
3	2	CITI	Checking	2021-11-20 10:00:00	San Jose		Deposit	250.00	250.00
4	3	WELLS FARGO	Savings	2021-11-20 10:00:00	San Jose		Deposit	10.00	10.00
5	1	CHASE	Checking	2020-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
6	1	CHASE	Checking	2021-04-14 10:00:00	Santa Cruz		Withdraw	50.00	950.00
7	5	BANK OF AMERICA	Checking	2021-12-05 00:41:12	NULL	NULL	Deposit	1000.00	NULL
8	5	BANK OF AMERICA	Checking	2021-12-05 00:41:19	NULL	NULL	Deposit	100.00	NULL
9	5	BANK OF AMERICA	Checking	2021-12-05 00:41:24	NULL	NULL	Deposit	300.00	NULL

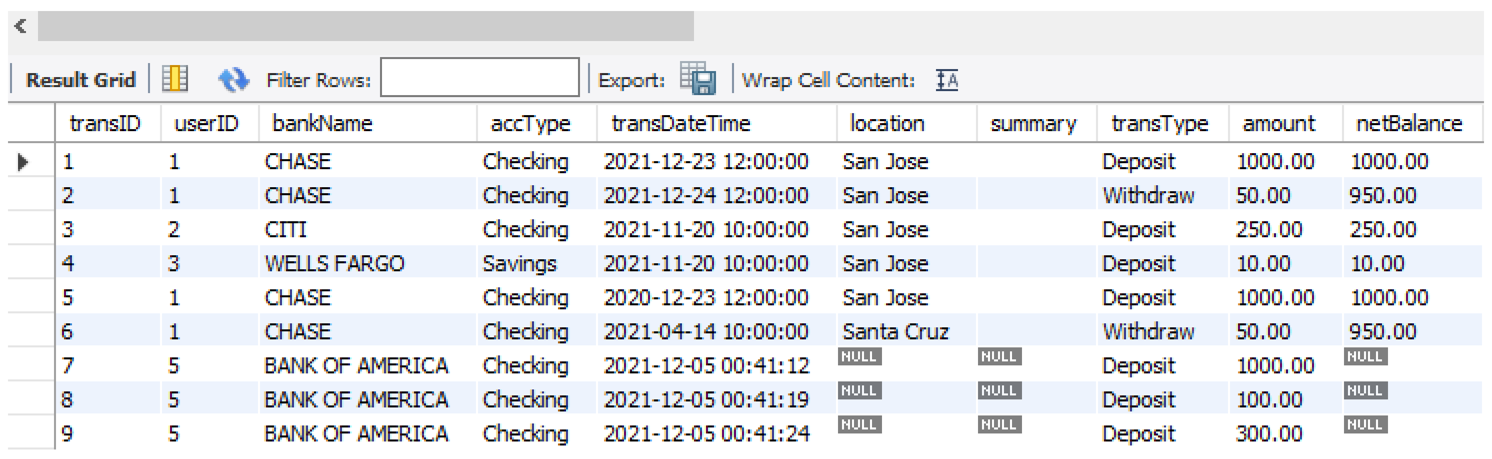
12. Get Monthly Transactions



```
Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
2
Please input a number from 1~2 to select an action.
Deposit (1) | Withdraw (2) | Check Latest Transactions (3) | Check Transactions for a Month (4)
4
(1) BANK OF AMERICA
(2) CHASE
(3) CITI
(4) WELLS FARGO

Please input the number of the bank your account is from.
1
(1) Checking $1400.0

Please input the number of the account you would like to use.
1

Please input the month (1~12) of the transaction you wish to see.
12

Please input the year (i.e: 2021) of the transaction you wish to see.
2021
ID: 0 | Date: 2021-12-05T00:41:12 | Location: null | Summary: null | Type: 1000.0 | Amount: $1000.0 | Net Balance: $0.0
ID: 0 | Date: 2021-12-05T00:41:19 | Location: null | Summary: null | Type: 100.0 | Amount: $100.0 | Net Balance: $0.0
ID: 0 | Date: 2021-12-05T00:41:24 | Location: null | Summary: null | Type: 300.0 | Amount: $300.0 | Net Balance: $0.0
```



```sql
SELECT * FROM Transactions;
```

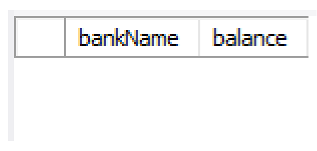


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

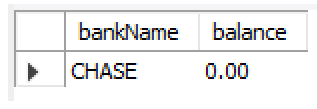
transID	userID	bankName	accType	transDateTime	location	summary	transType	amount	netBalance
1	1	CHASE	Checking	2021-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
2	1	CHASE	Checking	2021-12-24 12:00:00	San Jose		Withdraw	50.00	950.00
3	2	CITI	Checking	2021-11-20 10:00:00	San Jose		Deposit	250.00	250.00
4	3	WELLS FARGO	Savings	2021-11-20 10:00:00	San Jose		Deposit	10.00	10.00
5	1	CHASE	Checking	2020-12-23 12:00:00	San Jose		Deposit	1000.00	1000.00
6	1	CHASE	Checking	2021-04-14 10:00:00	Santa Cruz		Withdraw	50.00	950.00
7	5	BANK OF AMERICA	Checking	2021-12-05 00:41:12	NULL	NULL	Deposit	1000.00	NULL
8	5	BANK OF AMERICA	Checking	2021-12-05 00:41:19	NULL	NULL	Deposit	100.00	NULL
9	5	BANK OF AMERICA	Checking	2021-12-05 00:41:24	NULL	NULL	Deposit	300.00	NULL

13. Create a Bank

```
Please input a number from 1~4 show all actions related to the corresponding category.
Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
4
Please input a number from 1~2 to select an action.
Check Bank Balance (1) | Create Bank (2)
2

Please input the name of the bank where you would like to open an account.
chase
```



bankName	balance



bankName	balance
CHASE	0.00

Constraint Violation

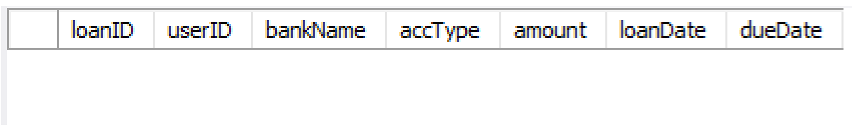
```
Please input the name of the bank where you would like to open an account.
chase
java.sql.SQLIntegrityConstraintViolationException: Duplicate entry 'CHASE' for key 'banks.PRIMARY'
This bank already exists

Accounts (1) | Transactions (2) | Loans (3) | Banks (4) | Delete User (8) | Sign Out (0)
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:117)
        at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:953)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1098)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1046)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPreparedStatement.java:1371)
        at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedStatement.java:1031)
        at sjsu.cs157a.bankingsystem.Database.createBank(Database.java:339)
        at sjsu.cs157a.bankingsystem.Bank.createBank(Bank.java:40)
        at sjsu.cs157a.bankingsystem.App.main(App.java:220)
```

14. Retrieve Bank Balance

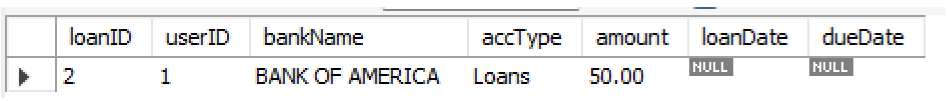
```
Please input a number from 1~2 to select an action.
Check Bank Balance (1) | Create Bank (2)
1
(1) CHASE

Please input the number of the bank whos balance you would like to check.
1
The balance of CHASE is $0.0
```

15. New Loan

```
Please input a number from 1~3 to select an action.
Show Loans (1) | Open New Loan (2)
2
Available banks.
(1) BANK OF AMERICA

Please input the number of the bank where you would like to open a loan.
1

Please input the amount of the loan.
50
Loan successfully taken out from BANK OF AMERICA for $50.0
```



	loanID	userID	bankName	accType	amount	loanDate	dueDate



	loanID	userID	bankName	accType	amount	loanDate	dueDate
▶	2	1	BANK OF AMERICA	Loans	50.00	NULL	NULL

16. Get All User Loans

```
Show Loans (1) | Open New Loan (2) | Make Loan Payment (3)
1
(1) BANK OF AMERICA $5000.0
```