Grant Clegg

Monty Saengsavang

Yu Jun Zhao

Group 4

CS 157A Project Final Report

Bank Management Application

Description: A banking system that allows the manager to monitor employees and customers accounts. It allows the manager to create, delete, and update accounts (employees and customers). The manager can use the system to help customers to make choices and to maximize the bank profits.

Requirements:

An online banking application with the following tables:

1. Customers (First, Last names, SSN, Addr): The information about customers will be their full names, their social security number, and their address. Customers may have three types of accounts: checking, savings, and credit. Customers may also have loans. These will be modeled through relationships.
2. Checking Accounts (Balance, Account ID, Purchase Limit, Routing Number): Checking accounts will have the Account ID, the balance, the single purchase limit, and a routing number for things such as direct deposit.
3. Savings Accounts (Balance, Account ID, Interest Rate, Withdrawal Limit, Minimum Balance, Routing Number): Savings accounts will also have an account ID and balance as well as an interest rate, withdrawal limit, routing number, and minimum balance.
4. Credit Card Account (Balance, Account ID, Due Date, Interest Rate, Minimum Payment, Limit, Last Statement, Last payment): Credit card accounts will have IDs, Balances, Payment Due Dates, Interest Rates, Minimum Payments, limits, and the Last Statement Balance.

5. Transaction History (Acct #, Transaction id, Transaction amount, Date): Transaction History will have the Acct# for the transaction, the transaction ID, the transaction amount, and the date the transaction was completed.
6. Loan(Loan #, total loan, interest rates, debt remaining, monthly payment, loan term, loan term remaining): Loans will contain information on the loan number, original amount of the loan, the remaining money owed for the loan, the monthly payment, and how many years the loan term was for.
7. Employees (First, Last Name, Employee ID, SSN, Job, Salary): Bank employee table information will be stored here with employee's full name, id, ,SSN, job type, and salary.

Transactions:

1. View all customers with savings accounts below the minimum balance.
2. Display all balances for all account types
3. Display all employees that have accounts at the bank
4. Display employees with above average salaries
5. Display employees below average salary
6. View all loans with a remaining debt above an input threshold
7. View all customers whose credit card account last payment was less than the minimum payment for the account.
8. View customers who made transactions from checking accounts above the purchase limit for that account
9. View customers whose loan is set to close within 1 year.

10. Calculate potential profit from credit card interest if only minimum payment is made on the balance
11. Calculate loss from savings account interest
12. Calculate expense for employee salaries
13. View current loans from customers with combined checking and savings account balance below monthly payment.
14. Calculate profit on loans based on interest rate and time remaining in loan
15. Insert/remove employee

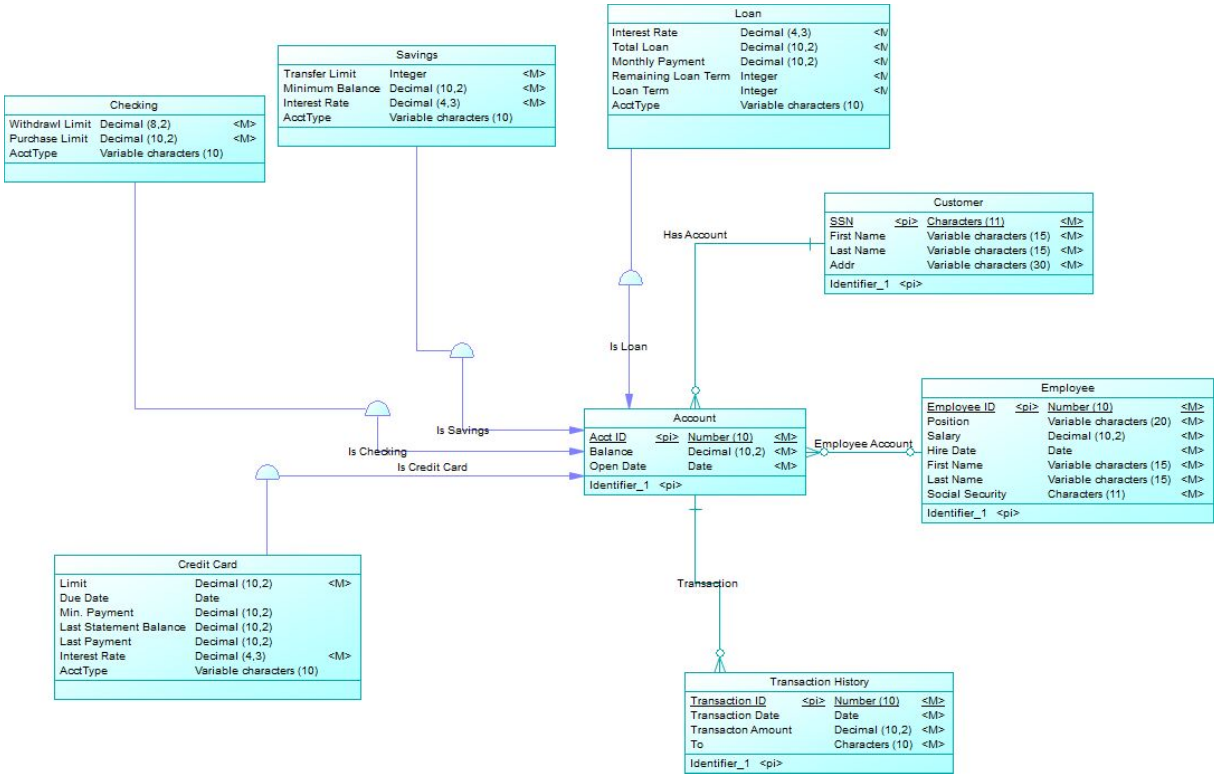
CS157A – Project-Milestone1

Database Design

1. Textual description of the project (Milestone 0)

A banking system that allows the manager to monitor employees and customers accounts. It allows the manager to create, delete, and update accounts (employees and customers). The manager can use the system to help customers to make choices and to maximize the bank profits. Note that this product is for a small, local bank that has one location.

2. E/R Diagram



Checking		
Withdrawl Limit	Decimal (8,2)	<M>
Purchase Limit	Decimal (10,2)	<M>
AcctType	Variable characters (10)	

Savings		
Transfer Limit	Integer	<M>
Minimum Balance	Decimal (10,2)	<M>
Interest Rate	Decimal (4,3)	<M>
AcctType	Variable characters (10)	

Loan		
Interest Rate	Decimal (4,3)	<M
Total Loan	Decimal (10,2)	<M
Monthly Payment	Decimal (10,2)	<M
Remaining Loan Term	Integer	<M
Loan Term	Integer	<M
AcctType	Variable characters (10)	

Customer			
SSN	<pi>	Characters (11)	<M>
First Name		Variable characters (15)	<M>
Last Name		Variable characters (15)	<M>
Addr		Variable characters (30)	<M>
Identifier_1	<pi>		

Account			
Acct ID	<pi>	Number (10)	<M>
Balance		Decimal (10,2)	<M>
Open Date		Date	<M>
Identifier_1	<pi>		

Employee			
Employee ID	<pi>	Number (10)	<M>
Position		Variable characters (20)	<M>
Salary		Decimal (10,2)	<M>
Hire Date		Date	<M>
First Name		Variable characters (15)	<M>
Last Name		Variable characters (15)	<M>
Social Security		Characters (11)	<M>
Identifier_1	<pi>		

Credit Card		
Limit	Decimal (10,2)	<M>
Due Date	Date	
Min. Payment	Decimal (10,2)	
Last Statement Balance	Decimal (10,2)	
Last Payment	Decimal (10,2)	
Interest Rate	Decimal (4,3)	<M>
AcctType	Variable characters (10)	

Transaction History			
Transaction ID	<pi>	Number (10)	<M>
Transaction Date		Date	<M>
Transacton Amount		Decimal (10,2)	<M>
To		Characters (10)	<M>
Identifier_1	<pi>		

3. Description of Entities.

Description of what each entity represents.

Entity name	Description
Customer	Represents the customers of this bank
Account	Represents a general account managed by this bank, superclass of Checking Account, Savings Account, Credit Card, and Loan
Checking Accounts	Represents the checking accounts managed by this bank, subclass of Account
Savings Accounts	Represents the savings accounts managed by this bank, subclass of Account
Credit Card	Represents the credit card accounts managed by this bank, subclass of Account
Transaction History	A list of all transactions processed by the bank
Loan	A list of all outstanding loans managed by the bank, subclass of Account
Bank Employees	Represents all employees working at this bank

4. Description of relationships on E/R Diagram.

Name	Entity1	Entity2	Entity 1 -> Entity 2 Role Cardinality	Entity 2 -> Entity 1 Role Cardinality	Description
Has Account	Customer	Account	1 to many	1 to 1	Each single customer can have more than one account, while each account can only have only 1 customer.
Employee Account	Employee	Account	1 to many	1 to 1	Each employee can have multiple accounts, but each account can only be associated to one employee.
Transaction	Account	Transaction History	1 to many	1 to 1	An account can have many transactions in the history, but each entry in the history may only be for one account.
Is Checking	Checking	Account	Is A	Parent	A checking account is an account.
Is Savings	Savings	Account	Is A	Parent	A savings account is an account.
Is Credit Card	Credit Card	Account	Is A	Parent	A credit card account is an account
Is Loan	Loan	Account	Is A	Parent	A loan account is an account

5. Description of Entity attributes.

Description of attributes in each entity:

Name	Used By	Used By An Identifier	Data Type	Description

Last Statement Balance	Credit Card		DECIMAL(10,2)	The amount in dollars, due from the credit card’s last statement (last month).
Last Payment	Credit Card		DECIMAL(10,2)	The amount, in dollars, that was last paid by the owner of the credit card
Limit	Credit Card		DECIMAL(10,2)	An total amount, in dollars, that credit card owners cannot exceed when making purchases
Interest Rate	Credit Card		DECIMAL(3,2)	A rate that is multiplied to the amount owed to calculate interest on a credit card.
Due Date	Credit Card		DATE	A set date every month that a credit card balance is due
Minimum Payment	Credit Card		DECIMAL(10,2)	A value that is required to be paid by the card owner to stay in good standing. 3% of balance.
AcctType	Credit Card		CHAR(10)	The type of the credit card. Ex) Travel, cash back.
SSN	Customer	Yes	CHAR(11)	Social security number that is used to uniquely identify each customer.
First Name	Customer		VARCHAR(15)	First name of a customer
Last Name	Customer		VARCHAR(15)	Last name of a customer
Address	Customer		VARCHAR(30)	Address of a customer
Purchase Limit	Checking		DECIMAL(10,2)	A limit, in dollars, that owners of

				the checking account cannot exceed when making a single puchase
Withdraw Limit	Checking		DECIMAL(8,2)	The amount limit a person can withdraw from an atm in one day
AcctType	Checking		Char(10)	The type of checking acct. Ex)Student, Standard
Transfer Limit	Savings		INT	The number of times a person can withdraw from a savings account in one month
Minimum Balance	Savings		DECIMAL(10,2)	The minimum amount a savings account can have in its balance before a fee is charged. This is in dollars.
Interest Rate	Savings		DECIMAL(3,2)	A rate that is used to calculate the interest.
AcctType	Savings		Char(10)	The type of the savings account ex)High interest, student
Interest Rate	Loan		DECIMAL(3,2)	The rate a loan will charge interest
Total Loan	Loan		DECIMAL(10,2)	The total amount being loaned
Monthly Payment	Loan		DECIMAL(10,2)	The monthly payment amount for the loan
Remaining Loan Term	Loan		INT	The number of months left until the loan is completely paid off

Loan Term	Loan		INT	The number of months the loan was agreed to be paid within
AcctType	Loan		Char(10)	The type of loan. Ex) Home, auto, personal, business
Employee ID	Bank Employee	Yes	NUMBER(10)	ID to specifically identify a Bank Employee. Ten Digit.
Hired Date	Bank Employee		DATE	Identify when the employee is hired.
Position	Bank Employee		VARCHAR(20)	Title of the bank employee's job
Salary	Bank Employee		DECIMAL(10,2)	The amount the bank employee will be paid yearly. This is in dollars.
Transaction Date	Transaction History		DATE	The date that shows when the transaction took place.
Transaction Amount	Transaction History		Decimal(10,2)	The amount that was transferred, withdrawed, or deposited for one account.
Transaction ID	Transaction History	Yes	NUMBER(10)	Unique ID to access a specific transaction. Ten Digit.
Account ID	Account	Yes	NUMBER(10)	Ten digit Id associated to each account.
Balance	Account		DECIMAL(10,2)	Amount in dollars held in this account
Open Date	Account		DATE	Date the account was opened

6. Analysis of functional and non-functional requirements.

Description of design assumptions and detailing the textual description of project.

Functional = what is the application supposed to do. Should contain requirements specified in Milestone 0.

Non-functional – examples: scalability, flexibility, extensibility, efficiency of storage, efficiency of processing. Describe how your application will meet these requirements.

Functional Requirements:

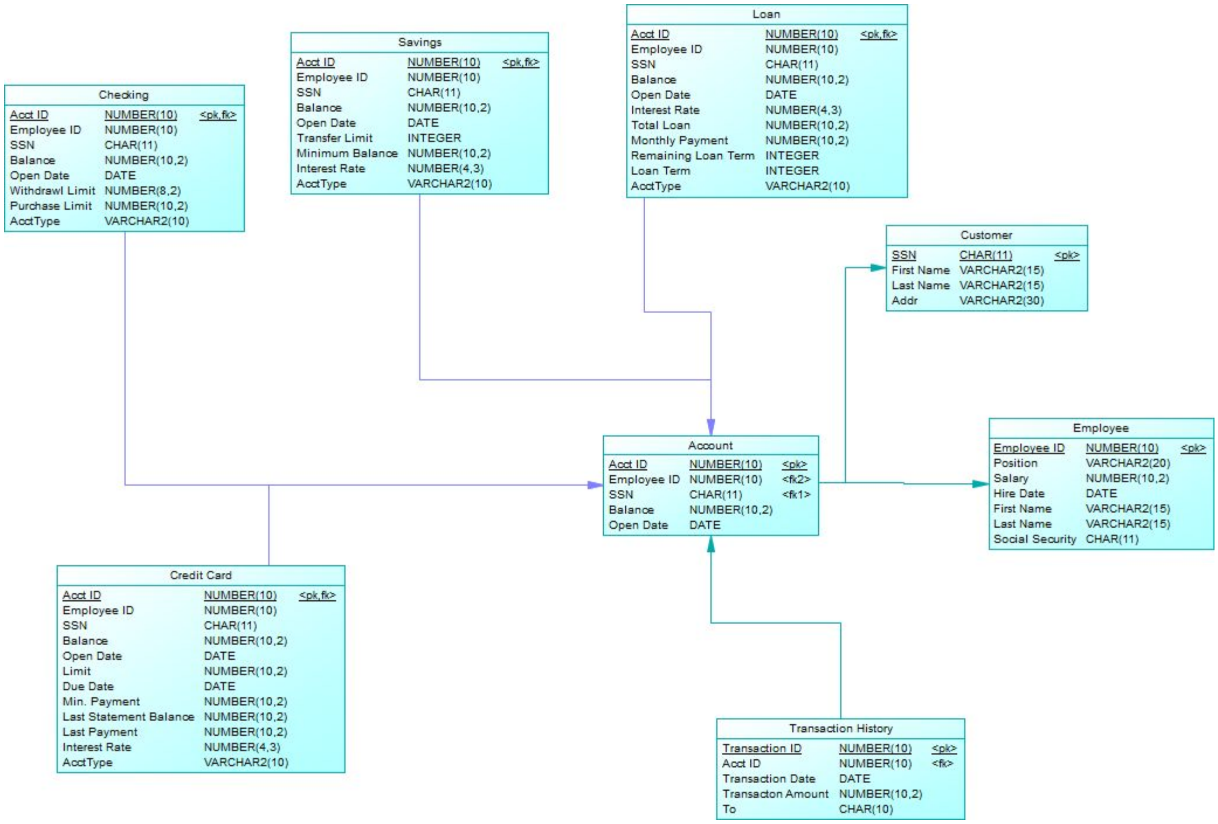
1. View all customers with savings accounts below the minimum balance.
2. Display all balances for all account types
3. Display all employees that have accounts at the bank
4. Display employees with above average salaries
5. Display employees below average salary
6. View all loans with a remaining debt above an input threshold
7. View all customers whose credit card account last payment was less than the minimum payment for the account.
8. View customers who made transactions from checking accounts above the purchase limit for that account
9. View customers whose loan is set to close within 1 year.
10. Calculate potential profit from credit card interest if only minimum payment is made on the balance
11. Calculate loss from savings account interest
12. Calculate expense for employee salaries
13. View current loans from customers with combined checking and savings account balance below monthly payment.
14. Calculate profit on loans based on interest rate and time remaining in loan
15. Insert/remove employee

Non-functional Requirements:

1. Retreive response from queries in less than 5 seconds
2. All account balances or amounts must be accurate to two decimal places
3. Data must be secure and unaccessible by unauthorized parties
4. Data retreived by the system must be accurate

In order to meet these requirements, we will have to organize our data efficiently in order to perform simple and quick queries. To keep our data safe, we will have to use a trusted DBMS. Lastly, to make sure data retreived by the system is accurate, we will perform tests to ensure data is valid.

7. Relational model (translating E/R into table model).



Checking		
Acct ID	NUMBER(10)	<pk,fk>
Employee ID	NUMBER(10)	
SSN	CHAR(11)	
Balance	NUMBER(10,2)	
Open Date	DATE	
Withdrawl Limit	NUMBER(8,2)	
Purchase Limit	NUMBER(10,2)	
AcctType	VARCHAR2(10)	

Savings		
Acct ID	NUMBER(10)	<pk,fk>
Employee ID	NUMBER(10)	
SSN	CHAR(11)	
Balance	NUMBER(10,2)	
Open Date	DATE	
Transfer Limit	INTEGER	
Minimum Balance	NUMBER(10,2)	
Interest Rate	NUMBER(4,3)	
AcctType	VARCHAR2(10)	

Loan		
Acct ID	NUMBER(10)	<pk,fk>
Employee ID	NUMBER(10)	
SSN	CHAR(11)	
Balance	NUMBER(10,2)	
Open Date	DATE	
Interest Rate	NUMBER(4,3)	
Total Loan	NUMBER(10,2)	
Monthly Payment	NUMBER(10,2)	
Remaining Loan Term	INTEGER	
Loan Term	INTEGER	
AcctType	VARCHAR2(10)	

Customer		
SSN	CHAR(11)	<pk>
First Name	VARCHAR2(15)	
Last Name	VARCHAR2(15)	
Addr	VARCHAR2(30)	

Account		
Acct ID	NUMBER(10)	<pk>
Employee ID	NUMBER(10)	<fk2>
SSN	CHAR(11)	<fk1>
Balance	NUMBER(10,2)	
Open Date	DATE	

Employee		
Employee ID	NUMBER(10)	<pk>
Position	VARCHAR2(20)	
Salary	NUMBER(10,2)	
Hire Date	DATE	
First Name	VARCHAR2(15)	
Last Name	VARCHAR2(15)	
Social Security	CHAR(11)	

Credit Card		
Acct ID	NUMBER(10)	<pk,fk>
Employee ID	NUMBER(10)	
SSN	CHAR(11)	
Balance	NUMBER(10,2)	
Open Date	DATE	
Limit	NUMBER(10,2)	
Due Date	DATE	
Min. Payment	NUMBER(10,2)	
Last Statement Balance	NUMBER(10,2)	
Last Payment	NUMBER(10,2)	
Interest Rate	NUMBER(4,3)	
AcctType	VARCHAR2(10)	

Transaction History		
Transaction ID	NUMBER(10)	<pk>
Acct ID	NUMBER(10)	<fk>
Transaction Date	DATE	
Transacton Amount	NUMBER(10,2)	
To	CHAR(10)	

8. Normalization – 3NF

Check that each table in your design is normalized. If not perform normalization.

Our table was created specifically to satisfy 3NF requirements. Therefore, it is normalized.

9. DDL Script that creates a database

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Created on:     11/30/2018 9:28:18 AM                        */
/*==============================================================*/

CREATE USER Client IDENTIFIED BY password;

GRANT CONNECT, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE TRIGGER,
CREATE SEQUENCE to Client;

GRANT UNLIMITED TABLESPACE TO Client;

CONNECT Client/password;

ALTER SESSION SET CURRENT_SCHEMA = Client;

alter table TRANSACTION_HISTORY
   drop constraint FK_TRANSACT_TRANSACTI_ACCOUNT;

drop index EMPLOYEE_ACCOUNT_FK;

drop index HAS_ACCOUNT_FK;

drop table ACCOUNT cascade constraints;

drop table CHECKING cascade constraints;

drop table CREDIT_CARD cascade constraints;

drop table CUSTOMER cascade constraints;

drop table EMPLOYEE cascade constraints;

drop table LOAN cascade constraints;

drop table SAVINGS cascade constraints;

drop index TRANSACTION_FK;

drop table TRANSACTION_HISTORY cascade constraints;
```

```
/*==============================================================*/
/* Table: CUSTOMER                                              */
/*==============================================================*/
create table CUSTOMER
(
   SSN                  CHAR(11)             not null,
   FIRST_NAME           VARCHAR2(15)         not null,
   LAST_NAME            VARCHAR2(15)         not null,
   ADDR                 VARCHAR2(50)         not null,
   constraint PK_CUSTOMER primary key (SSN)
);

/*==============================================================*/
/* Table: EMPLOYEE                                              */
/*==============================================================*/
create table EMPLOYEE
(
   EMPLOYEE_ID          NUMBER(10)           not null,
   POSITION             VARCHAR2(20)         not null,
   SALARY               NUMBER(10,2) not null CHECK(SALARY >= 31200),
   HIRE_DATE            DATE                 not null,
   FIRST_NAME           VARCHAR2(15)         not null,
   LAST_NAME            VARCHAR2(15)         not null,
   SOCIAL_SECURITY      CHAR(11)      not null unique,
   constraint PK_EMPLOYEE primary key (EMPLOYEE_ID)
);

/*==============================================================*/
/* Table: ACCOUNT                                               */
/*==============================================================*/
create table ACCOUNT
(
   ACCT_ID              NUMBER(10)           not null,
   EMPLOYEE_ID          NUMBER(10) REFERENCES EMPLOYEE(EMPLOYEE_ID) ON DELETE SET NULL,
   SSN                  CHAR(11) REFERENCES CUSTOMER(SSN) ON DELETE CASCADE,
   BALANCE              NUMBER(10,2) not null CHECK(BALANCE >= 0),
   OPEN_DATE            DATE                 not null,
   constraint PK_ACCOUNT primary key (ACCT_ID)
);

/*==============================================================*/
/* Index: HAS_ACCOUNT_FK                                        */
```

```
/*==============================================================*/
create index HAS_ACCOUNT_FK on ACCOUNT (
   SSN ASC
);

/*==============================================================*/
/* Index: EMPLOYEE_ACCOUNT_FK                                   */
/*==============================================================*/
create index EMPLOYEE_ACCOUNT_FK on ACCOUNT (
   EMPLOYEE_ID ASC
);

/*==============================================================*/
/* Table: CHECKING                                              */
/*==============================================================*/
create table CHECKING
(
   ACCT_ID              NUMBER(10) not null REFERENCES ACCOUNT(ACCT_ID) ON DELETE CASCADE,
   EMPLOYEE_ID          NUMBER(10),
   SSN                  CHAR(11),
   BALANCE              NUMBER(10,2) not null  CHECK(BALANCE >= 0),
   OPEN_DATE            DATE                 not null,
   WITHDRAWL_LIMIT      NUMBER(8,2)          not null,
   PURCHASE_LIMIT       NUMBER(10,2)         not null,
   ACCTTYPE             VARCHAR2(10),
   constraint PK_CHECKING primary key (ACCT_ID)
);

/*==============================================================*/
/* Table: CREDIT_CARD                                           */
/*==============================================================*/
create table CREDIT_CARD
(
   ACCT_ID              NUMBER(10) not null REFERENCES ACCOUNT(ACCT_ID) ON DELETE CASCADE,
   EMPLOYEE_ID          NUMBER(10),
   SSN                  CHAR(11),
   BALANCE              NUMBER(10,2) not null CHECK(BALANCE >= 0),
   OPEN_DATE            DATE                 not null,
   LIMIT                NUMBER(10,2)         not null,
   DUE_DATE             DATE,
   MIN__PAYMENT         NUMBER(10,2),
   LAST_STATEMENT_BALANCE NUMBER(10,2),
   LAST_PAYMENT         NUMBER(10,2),
   INTEREST_RATE        NUMBER(10,2)         not null,
   ACCTTYPE             VARCHAR2(15),
   constraint PK_CREDIT_CARD primary key (ACCT_ID)
);

/*==============================================================*/
/* Table: LOAN                                                  */
/*==============================================================*/
create table LOAN
(
   ACCT_ID              NUMBER(10) not null REFERENCES ACCOUNT(ACCT_ID) ON DELETE CASCADE,
   EMPLOYEE_ID          NUMBER(10),
   SSN                  CHAR(11),
   BALANCE              NUMBER(10,2) not null CHECK(BALANCE >=0),
   OPEN_DATE            DATE                 not null,
   INTEREST_RATE        NUMBER(4,3)          not null,
   TOTAL_LOAN           NUMBER(10,2)         not null,
   MONTHLY_PAYMENT      NUMBER(10,2)         not null,
   REMAINING_LOAN_TERM  INTEGER              not null,
   LOAN_TERM            INTEGER              not null,
   ACCTTYPE             VARCHAR2(15),
   constraint PK_LOAN primary key (ACCT_ID)
);

/*==============================================================*/
/* Index: REMAINING_LOAN_TERM                                   */
/*==============================================================*/
CREATE INDEX loan_indx ON Loan(remaining_loan_term ASC);

/*==============================================================*/
/* Table: SAVINGS                                               */
/*==============================================================*/
create table SAVINGS
(
   ACCT_ID              NUMBER(10) not null REFERENCES ACCOUNT(ACCT_ID) ON DELETE CASCADE,
   EMPLOYEE_ID          NUMBER(10),
   SSN                  CHAR(11),
   BALANCE              NUMBER(10,2) not null CHECK(BALANCE >=0),
   OPEN_DATE            DATE                 not null,
   TRANSFER_LIMIT       INTEGER              not null,
   MINIMUM_BALANCE      NUMBER(10,2)         not null,
   INTEREST_RATE        NUMBER(4,3)          not null,
   ACCTTYPE             VARCHAR2(20),
  constraint PK_SAVINGS primary key (ACCT_ID)
);

/*==============================================================*/
/* Table: TRANSACTION_HISTORY                                   */
/*==============================================================*/
create table TRANSACTION_HISTORY
(
  TRANSACTION_ID       NUMBER(10)           not null,
  ACCT_ID              NUMBER(10)           not null,
  TRANSACTION_DATE     DATE                 not null,
  TRANSACTON_AMOUNT    NUMBER(10,2)         not null,
  "TO"                 NUMBER(10)           not null,
  constraint PK_TRANSACTION_HISTORY primary key (TRANSACTION_ID)
);

/*==============================================================*/
/* Index: TRANSACTION_FK                                        */
/*==============================================================*/
create index TRANSACTION_FK on TRANSACTION_HISTORY (
  ACCT_ID ASC
);

alter table TRANSACTION_HISTORY
  add constraint FK_TRANSACT_TRANSACTI_ACCOUNT foreign key (ACCT_ID)
    references ACCOUNT (ACCT_ID);
```

10. Populating tables with sample data

Prepare text files with data and use one of the loading methods presented in the tutorials/lecture. Depending on the tables and problem, 7-30 rows per each table.

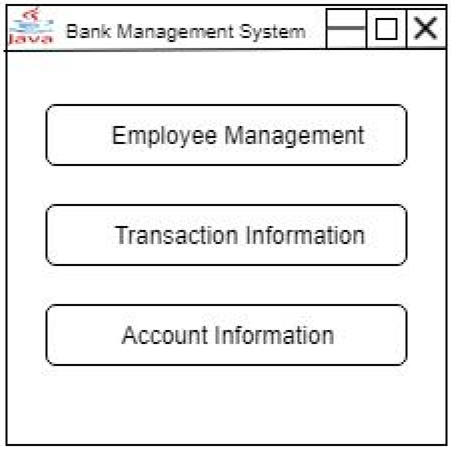
CS157A – Project-Milestone2

Application Design

In this milestone you will concentrate on proposing / designing / implementing the following aspects of your database application: GUI design, application architecture, performance, data integrity and concurrency management.

1. Design of the application menu

Propose and design a menu for your application (use 'mock' menu).



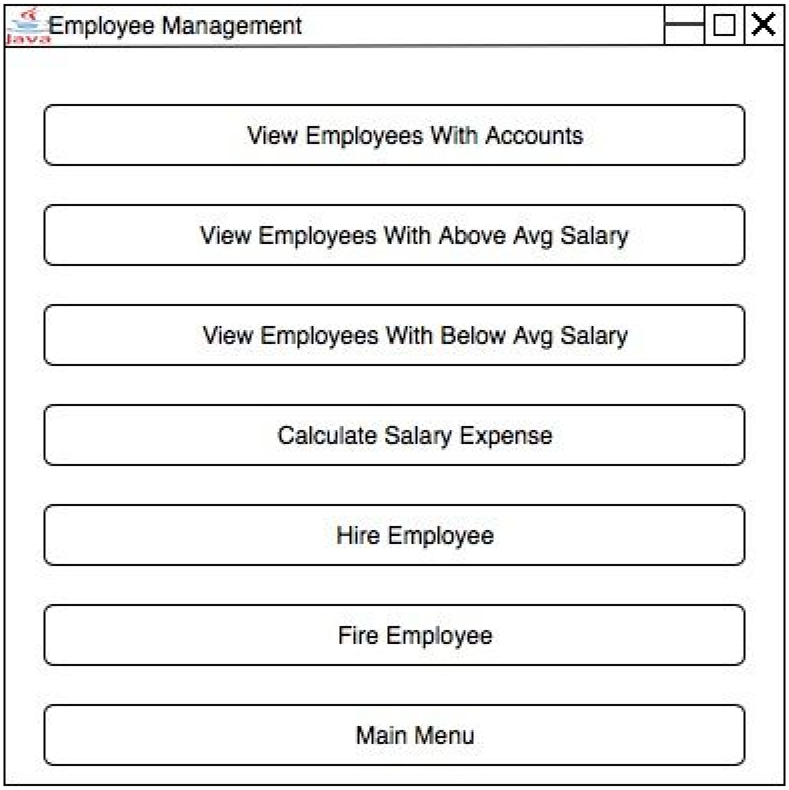
The Employee Management section will contain the following functions: view all employees with accounts, view employees with above average salaries, view employees with below average salaries, calculate payroll expenses, hire employee, and fire employee.

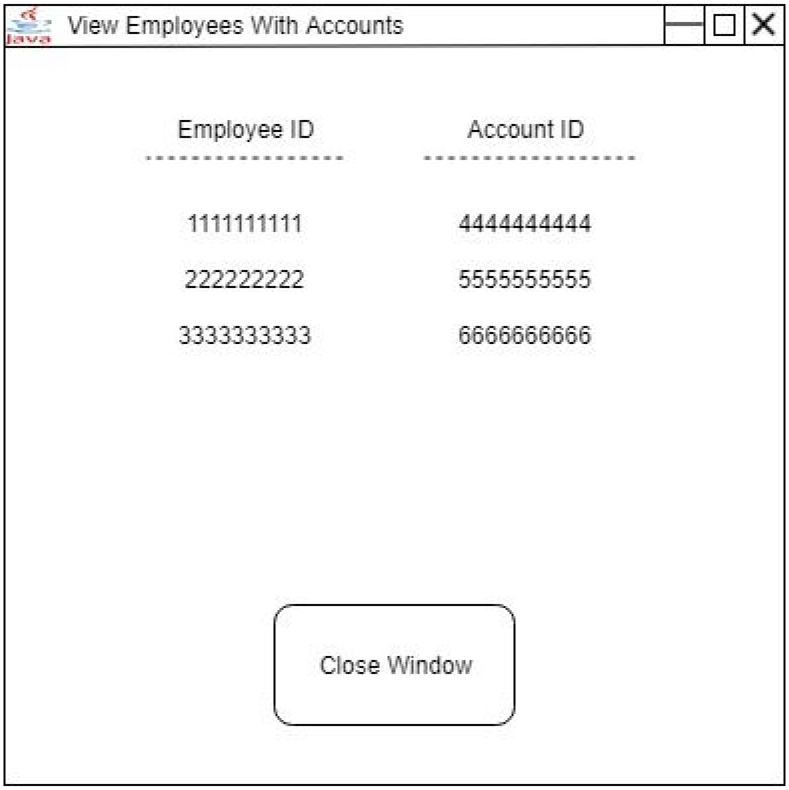
The Transaction Information section will contain the following functions: view customers who made purchases above the purchase limit of their checking account and view all customers who made less than the minimum payment on their credit card

The Account Information section will contain the following functions: display all balances for all account types, view all loans with remaining balance above a user defined threshold, view all customers whose loans will be closing within one year, calculate potential profit from credit card accounts if only minimum payments are made, and calculate profits on loans based on remaining amount owed, interest, and the time remaining in the loan.

2. GUI design for the chosen module

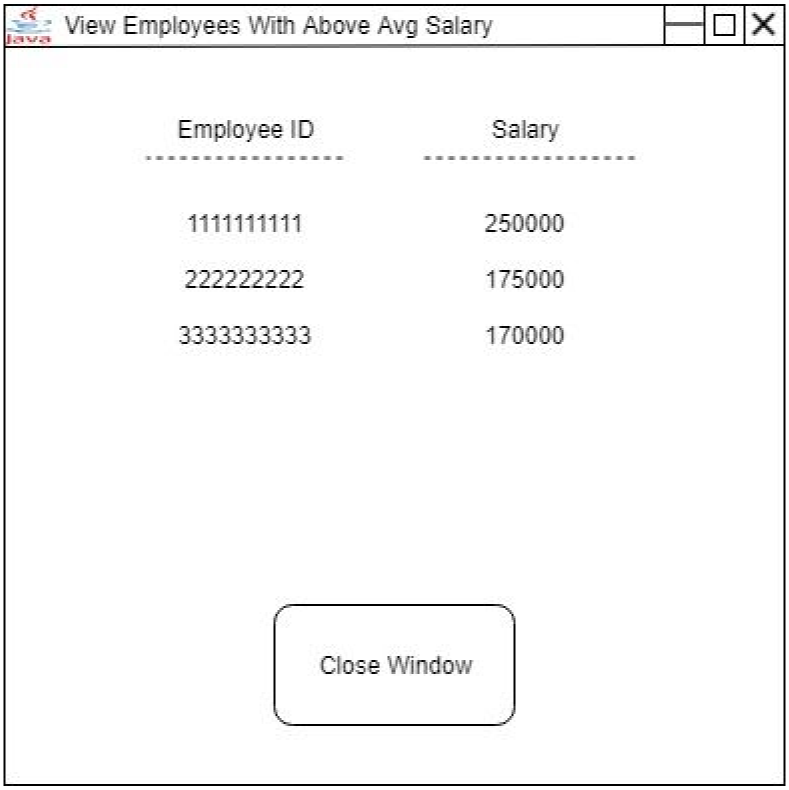
Propose and design a ‘mock’ interface for the chosen module of your application.


Java
Employee Management
View Employees With Accounts
View Employees With Above Avg Salary
View Employees With Below Avg Salary
Calculate Salary Expense
Hire Employee
Fire Employee
Main Menu



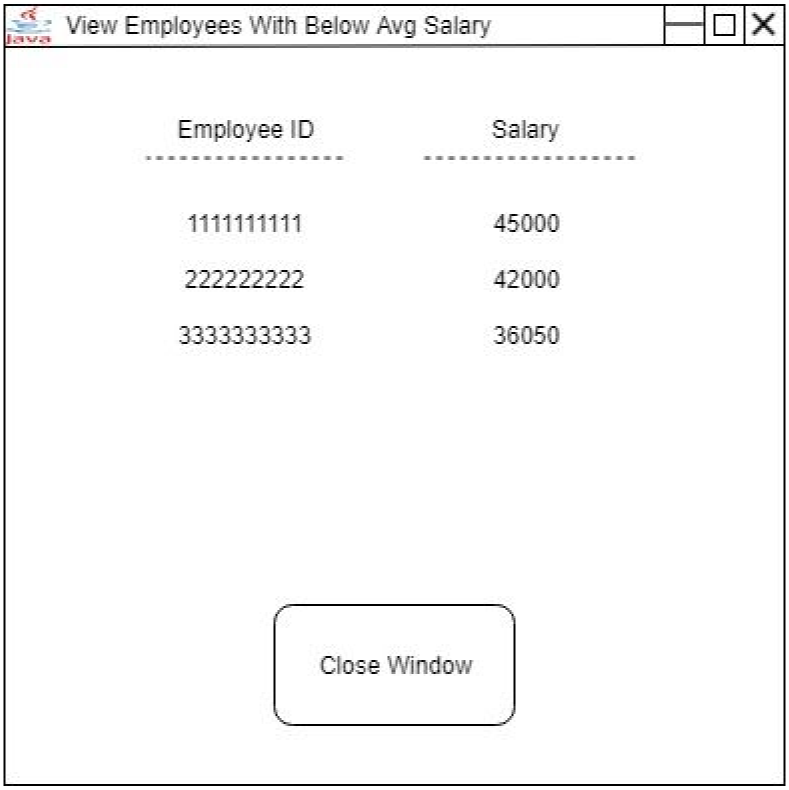
View Employees With Accounts

Employee ID	Account ID
1111111111	4444444444
222222222	5555555555
3333333333	6666666666



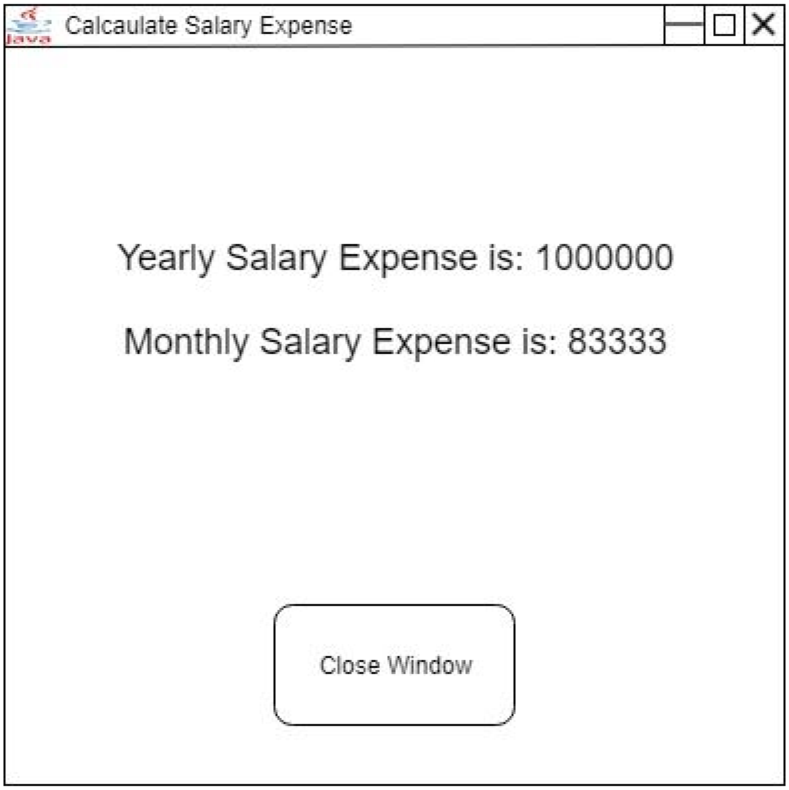
View Employees With Above Avg Salary

Employee ID	Salary
1111111111	250000
222222222	175000
3333333333	170000



View Employees With Below Avg Salary

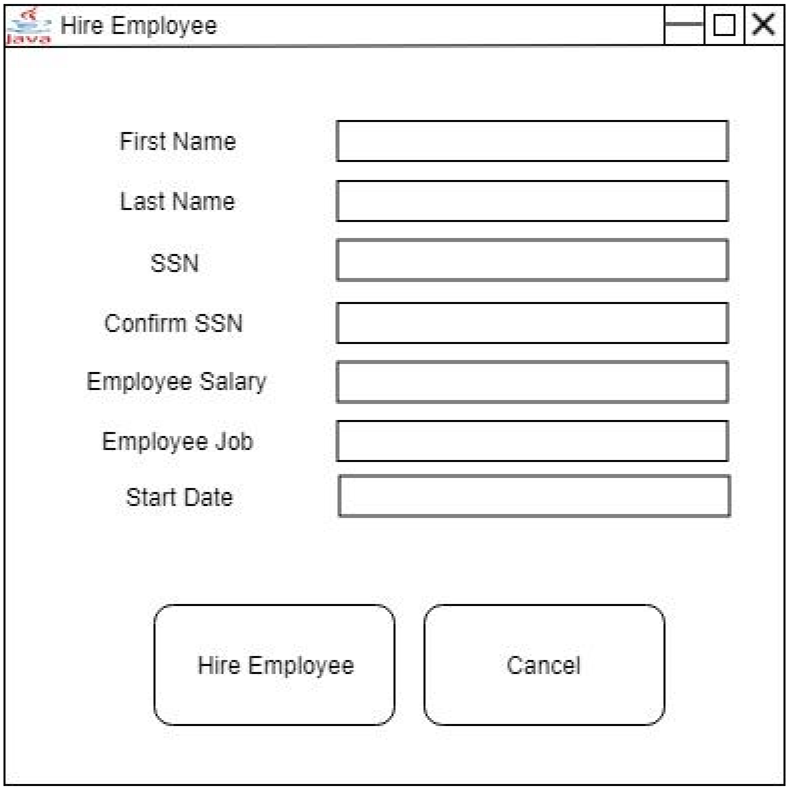
Employee ID	Salary
1111111111	45000
222222222	42000
3333333333	36050

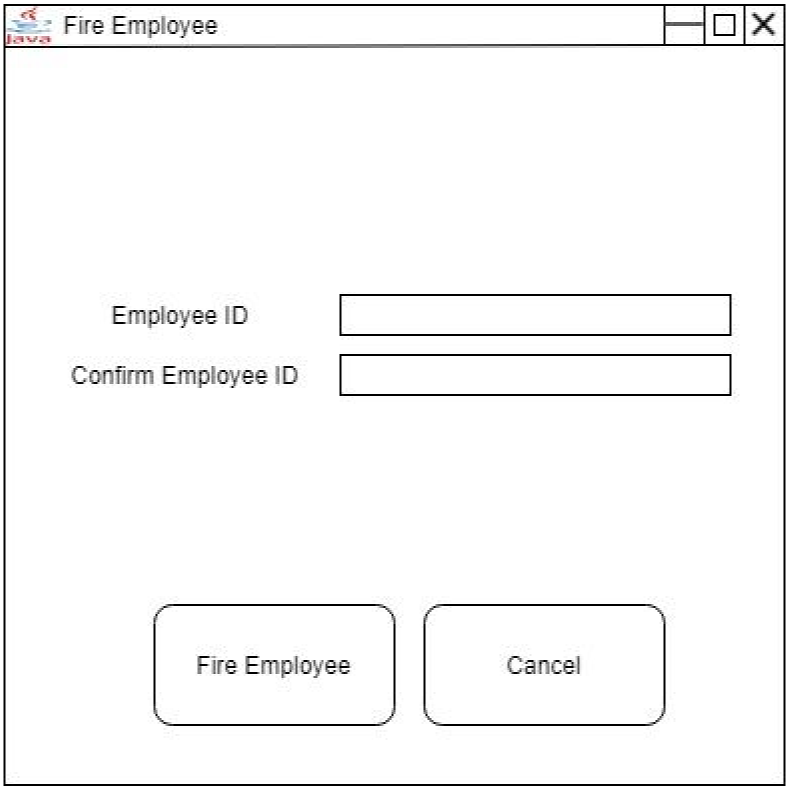


Calcaulate Salary Expense

Yearly Salary Expense is: 1000000

Monthly Salary Expense is: 83333


Hire Employee
First Name
Last Name
SSN
Confirm SSN
Employee Salary
Employee Job
Start Date
Hire Employee
Cancel


Fire Employee
Employee ID
Confirm Employee ID
Fire Employee
Cancel

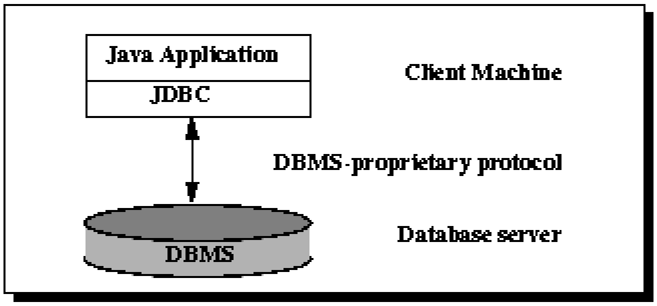
3. Application architecture

Modules + descriptions of the modules. Include diagram of the Client-Server architecture. The project must be implemented using Client-Server architecture and using JDBC.

Modules and Descriptions

1. Employee Management: Contains all functions related to the management of employees such as hiring and firing and salary management.
2. Transaction Information: Contains all functions related to the transaction history such as customers who made purchases above their purchase limit or customers who have not made the minimum payment for their credit card
3. Account Information: Contains all functions related to the accounts such expense for savings account interest, profit from credit card or loan interest, loans ending soon, etc. as described in milestone 1.

Client Server Architecture



4. Server-side and Client-side functionality.

Describe which functionalities will be implemented on the Client side and which will be implemented on the server side.

Server Side

1. Calculate potential profit from credit card interest if only minimum payment is made on the balance
2. Calculate salary expense
3. Calculate loss from savings account interest
4. Calculate expense for employee salaries
5. Calculate profit on loans based on interest rate and time remaining in loan

Client Side

1. Display all customers with savings accounts below the minimum balance.
2. Return to main menu
3. Insert/Remove employee
4. Display all balances for all account types
5. Display all employees that have accounts at the bank
6. Display employees with above average salaries
7. Display employees below average salary
8. View all loans with a remaining debt above an input threshold
9. View all customers whose credit card account last payment was less than the minimum payment for the account.
10. View customers who made transactions from checking accounts above the purchase limit for that account
11. View customers whose loan is set to close within 1 year.
12. View current loans from customers with combined checking and savings account balance below monthly payment.

5. Integrity: constraints and triggers

Describe how integrity constraints will be enforced in your database system application (in the table definition and programmatically with assertion/triggers). Include code for defining constraints and code for triggers.

We have two check constraints to ensure

Insufficient balance constraint

```
ALTER TABLE checking_account ADD CONSTRAINT CHECK (balance >= 0);
```

Prevent hiring below minimum wage constraint

```
ALTER TABLE employee ADD CONSTRAINT CHECK (salary >= 31200);
```

We have foreign key constraints and primary key constraints declared in Milestone 1

6. Performance: Indexes

Propose and design a set of indexes suitable for your systems, taking into account most common query transactions performed in your application. Give code for creating these indexes. Consider also designing materialized views to speed up certain queries.

Our DBMS automatically makes primary keys indexes, so all of our primary keys are indexes. See Milestone 1.

The remaining loan term of loan accounts could be indexed. By doing so, the server could quickly populate the customers whose loan is set to close within 1 year.

```
CREATE INDEX loan_indx ON Loan(remaining_loan_term ASC)

CREATE VIEW loan_view AS

SELECT account_id, interest_rate, remaining_loan_term FROM LOAN;
```

7. Concurrency: Transactions for the chosen module

Describe atomic transactions in the chosen module and how concurrency will be supported.

Hiring and firing employees will be atomic transactions since they are the only two functions in the employee management module that manipulate the data. Insertion and deletion of employees can be transactions on the serialized isolation levels. Users will be able to safely query data before or after an employee is hired or fired.

CS157A – Project-Milestone3

For Milestone 3, we implemented our views, stored procedure, and triggers required for the Employee Management module of our program. In total, there are three views, one stored provedure, and six triggers. The views are meant to make accessing the three selection functions faster. The stored procedure handles the calculations required for the salary expenditure function of the module, and the triggers are used to automatically generate the primary keys for the employee, transaction history, and account tables which are employee_id, transaction_id, and acct_id respectively.

Views:

These views are meant to make queries more secure by limiting what there is access to. The code for the view is as follows:

```
CREATE OR REPLACE VIEW remaining_loan_view AS
        SELECT acct_id as acct_id, interest_rate as intr_rate, remaining_loan_term as
remain_term FROM LOAN;

CREATE OR REPLACE VIEW all_emps AS
   SELECT EMPLOYEE_ID as emp_id, FIRST_NAME as f_name, LAST_NAME as l_name,
SOCIAL_SECURITY as ssn, POSITION as position, HIRE_DATE as hire_date, SALARY as sal FROM
EMPLOYEE ORDER BY SALARY DESC;

CREATE OR REPLACE VIEW emps_with_accts AS
   SELECT DISTINCT EMPLOYEE_ID as emp_id, FIRST_NAME as f_name, LAST_NAME as l_name
   FROM EMPLOYEE WHERE
      EMPLOYEE_ID IN(SELECT EMPLOYEE_ID FROM ACCOUNT
                WHERE EMPLOYEE_ID IS NOT NULL);

CREATE OR REPLACE VIEW emps_with_above_avg_sal AS
   SELECT EMPLOYEE_ID as emp_id, SALARY as sal, FIRST_NAME as f_name, LAST_NAME as
l_name
   FROM EMPLOYEE WHERE
      SALARY >= (SELECT AVG(SALARY) FROM EMPLOYEE);

CREATE OR REPLACE VIEW emps_with_below_avg_sal AS
   SELECT EMPLOYEE_ID as emp_id, SALARY as sal, FIRST_NAME as f_name, LAST_NAME as
l_name
   FROM EMPLOYEE WHERE
      SALARY <= (SELECT AVG(SALARY) FROM EMPLOYEE);
```

Stored Procedures:

The first stored procedure takes two output variables. The first variable contains the sum of the salaries which is the yearly expense the bank must spend on its employees. The monthly value divides the yearly value by 12 to calculate how much the bank must pay its employees each month. The code for the stored procedure is as follows:

```
CREATE OR REPLACE PROCEDURE calc_salary_exp(yearly out NUMBER, monthly out NUMBER) IS
  CURSOR emp_cur IS SELECT SALARY from EMPLOYEE;
  v_temp number(10,2);
  v_benefits number(10,2);
BEGIN
  yearly := 0;
  OPEN emp_cur;
LOOP
  FETCH emp_cur INTO v_temp;
  EXIT WHEN emp_cur%NOTFOUND;
  IF v_temp < 41629.00 THEN
    v_benefits := 1000.00;
    v_temp := v_temp + (v_temp * .06) + v_benefits;
  ELSIF v_temp < 52612.00 THEN
    v_benefits := 2000.00;
    v_temp := v_temp + (v_temp * .08) + v_benefits;
  ELSIF v_temp < 268750.00 THEN
    v_benefits := 3000.00;
    v_temp := v_temp + (v_temp * .093) + v_benefits;
  ELSE
    v_benefits := 4000.00;
    v_temp := v_temp + (v_temp * .103) + v_benefits;
  END IF;
  yearly := yearly + v_temp;
  dbms_output.put_line(yearly);
END LOOP;
monthly := yearly/12;
CLOSE emp_cur;
END calc_salary_exp;
/
```

The second stored procedure is used when the user tries to higher a new employee. It calls for an insert on the all_emps view which fires an INSTEAD OF trigger that will be discussed further in the Triggers section.

```
CREATE OR REPLACE PROCEDURE hire (position in varchar2, salary in number, hiredate in date,
first in varchar2, last in varchar2, social in char) IS
BEGIN
   INSERT INTO all_emps(position, sal, hire_date, f_name, l_name, ssn) VALUES (position, salary,
hiredate, first, last, social);
COMMIT;
END hire;
/
```

The third stored procedure runs a transaction for firing. It tries to delete from the all_emps view which then fires an instead of trigger to delete from the actual employee table.

```
CREATE OR REPLACE PROCEDURE fire (emp in number, test out number) IS
BEGIN
   SELECT count(emp_id) into test from all_emps where emp_id = emp;
   IF  test > 0 THEN
      DELETE FROM all_emps WHERE emp_id = emp;
      COMMIT;
      test := 1;
   ELSE
      test := 0;
   END IF;
END fire;
/
```

The last stored procedure calculates the net income of an employee. It deducts from the employees gross salary based on the California income tax brackets and counts the cash value of their employee benefits as income.

```
CREATE OR REPLACE PROCEDURE emp_net_sal(emp_id in NUMBER, netyearly out NUMBER,
netmonthly out NUMBER) IS
   v_benefits number(10,2);
BEGIN
   netyearly := 0;
   netmonthly := 0;

   SELECT SALARY INTO netyearly FROM EMPLOYEE WHERE EMPLOYEE_ID = emp_id;

   IF netyearly < 41629.00 THEN
      v_benefits :=1000.00;
      netyearly := netyearly - (netyearly * .06) + v_benefits;
   ELSIF netyearly < 52612.00 THEN
      v_benefits := 2000.00;
      netyearly := netyearly - (netyearly * .08) + v_benefits;
   ELSIF netyearly < 268750.00 THEN
      v_benefits := 3000.00;
      netyearly := netyearly - (netyearly * .093) + v_benefits;
   ELSE
      v_benefits := 4000.00;
      netyearly := netyearly - (netyearly * .103) + v_benefits;
   END IF;
   netmonthly := netyearly/12;
END emp_net_sal;
/
```

Triggers:

The code for these triggers required sequences to be created in order to generate the values. Since all the primary keys are ten digit numbers, all of our sequences start at the value of 1,000,000,000. In order to obtain the maximum number of keys, there is a separate sequence for the transaction history ids, employee ids, and account ids. The account types all share the same sequence because we don’t want accounts to have the same account id even if they are of different account types. Even though the employee module does not involve creating accounts or transactions, we felt it would be good to add them to the database anyways. The sequences and triggers are coded as follows:

```
DROP SEQUENCE emp_seq;

CREATE SEQUENCE emp_seq START WITH 1000000000;

DROP SEQUENCE acct_seq;

CREATE SEQUENCE acct_seq START WITH 1000000000;

DROP SEQUENCE trans_seq;

CREATE SEQUENCE trans_seq START WITH 1000000000;

CREATE OR REPLACE TRIGGER new_checking
```

```
BEFORE INSERT ON CHECKING
FOR EACH ROW
DECLARE
   v_next number(10);
   v_today date;
BEGIN
   v_next := acct_seq.NEXTVAL;
   v_today := sysdate;

   SELECT v_next
   INTO :new.ACCT_ID
   FROM dual;

   SELECT v_today
   INTO :new.OPEN_DATE
   FROM dual;

   INSERT INTO ACCOUNT(ACCT_ID, EMPLOYEE_ID, SSN, BALANCE, OPEN_DATE) VALUES
   (v_next, :new.EMPLOYEE_ID, :new.SSN, :new.BALANCE, v_today);
END;
/


CREATE OR REPLACE TRIGGER new_hire
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
   SELECT emp_seq.NEXTVAL
   INTO :new.EMPLOYEE_ID
   FROM dual;
END;
/

CREATE OR REPLACE TRIGGER new_loan
BEFORE INSERT ON LOAN
FOR EACH ROW
DECLARE
   v_next number(10);
   v_today date;
```

```
BEGIN
   v_next := acct_seq.NEXTVAL;
   v_today := sysdate;

   SELECT v_next
   INTO :new.ACCT_ID
   FROM dual;

   SELECT v_today
   INTO :new.OPEN_DATE
   FROM dual;

   INSERT INTO ACCOUNT(ACCT_ID, EMPLOYEE_ID, SSN, BALANCE, OPEN_DATE) VALUES
   (v_next, :new.EMPLOYEE_ID, :new.SSN, :new.BALANCE, v_today);
END;
/

CREATE OR REPLACE TRIGGER new_savings
BEFORE INSERT ON SAVINGS
FOR EACH ROW
DECLARE
   v_next number(10);
   v_today date;
BEGIN
   v_next := acct_seq.NEXTVAL;
   v_today := sysdate;

   SELECT v_next
   INTO :new.ACCT_ID
   FROM dual;

   SELECT v_today
   INTO :new.OPEN_DATE
   FROM dual;

   INSERT INTO ACCOUNT(ACCT_ID, EMPLOYEE_ID, SSN, BALANCE, OPEN_DATE) VALUES
   (v_next, :new.EMPLOYEE_ID, :new.SSN, :new.BALANCE, v_today);
END;
/
```

```
CREATE OR REPLACE TRIGGER new_trans
BEFORE INSERT ON TRANSACTION_HISTORY
FOR EACH ROW
BEGIN
   SELECT trans_seq.NEXTVAL
   INTO :new.TRANSACTION_ID
   FROM dual;
END;
/
```

This next trigger fires when the end user activates the hire employee functionality, as seen in the procedure section, this cause an insert to be attempted on the all_emps view, but this INSTEAD OF trigger diverts it to an insertion on the actual employee table

```
CREATE OR REPLACE TRIGGER new_hire_attempt
INSTEAD OF INSERT ON all_emps
FOR EACH ROW
BEGIN
INSERT INTO EMPLOYEE(POSITION, SALARY, HIRE_DATE, FIRST_NAME, LAST_NAME,
SOCIAL_SECURITY)
VALUES(:new.position, :new.sal, :new.hire_date, :new.f_name, :new.l_name, :new.ssn);
END;
/
```

The last trigger fires when the user tries to fire an employee. It is triggered by the fire procedure and it causes the delete to delete from the employee table rather than trying to delete from the all_emps view.

```
CREATE OR REPLACE TRIGGER fire_attempt
INSTEAD OF DELETE ON all_emps
FOR EACH ROW
BEGIN
DELETE FROM EMPLOYEE WHERE EMPLOYEE_ID = :old.emp_id;
END;
/
```