```
/*
    Name: Audrey Nguyen
    Data and Database Managment with SQL
    Assignment - Banking Database
*/

-------------------------------------------------------------------------------------
/*                              Banking DDL                                */
-------------------------------------------------------------------------------------

        CREATE TABLE branch (
                branch_name     VARCHAR(100),
                branch_city     VARCHAR(50) NOT NULL,
                assets          NUMERIC(14,2) CHECK (assets>=0.00),
                CONSTRAINT branch_pkey PRIMARY KEY (branch_name)
        );

        CREATE TABLE customer (
                cust_ID         VARCHAR(10),
                customer_name   VARCHAR(30) NOT NULL,
                customer_street VARCHAR(50) NOT NULL,
                customer_city   VARCHAR(50) NOT NULL,
                CONSTRAINT customer_pkey PRIMARY KEY (cust_ID)
        );

        CREATE TABLE account (
                account_number  VARCHAR(15),
                branch_name     VARCHAR(100),
                balance         NUMERIC(14,2) CHECK (balance>=0.00),
                CONSTRAINT account_pkey PRIMARY KEY (account_number),
                CONSTRAINT account_fkey FOREIGN KEY (branch_name) REFERENCES branch (branch_name)
                        ON DELETE SET NULL
                        ON UPDATE CASCADE
        );

        CREATE TABLE depositor (
                cust_ID         VARCHAR(10),
                account_number  VARCHAR(15) NOT NULL,
                CONSTRAINT depositor_pkey PRIMARY KEY (cust_ID, account_number),
                CONSTRAINT depositor_fkey FOREIGN KEY (cust_ID) REFERENCES customer (cust_ID)
                        ON DELETE CASCADE       -- Foreign key constraints only on cust_ID
                        ON UPDATE CASCADE       -- and not on account_number
        );

        CREATE TABLE loan (
                loan_number     VARCHAR(15),
                branch_name     VARCHAR(100),
                amount          NUMERIC(14,2) CHECK (amount>=0.00),
                CONSTRAINT loan_pkey PRIMARY KEY (loan_number),
                CONSTRAINT loan_fkey FOREIGN KEY (branch_name) REFERENCES branch (branch_name)
                        ON DELETE SET NULL
                        ON UPDATE CASCADE
        );

        CREATE TABLE borrower (
                cust_ID         VARCHAR(10),
                loan_number     VARCHAR(15) NOT NULL,
                CONSTRAINT borrower_pkey PRIMARY KEY (cust_ID, loan_number),
                CONSTRAINT borrower_fkey_1 FOREIGN KEY (cust_ID) REFERENCES customer (cust_ID)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
                CONSTRAINT borrower_fkey_2 FOREIGN KEY (loan_number) REFERENCES loan (loan_number)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
        );
```

```
-----------------------------------------------------------------------------------------------
/*      SQL function that accepts a principal mortgage amount (P), an annual percentage         */
/*      rate (APR), and the number of years a mortgage will be paid back over. Calculate the    */
/*      associated monthly mortgage payment (A).                                                */
-----------------------------------------------------------------------------------------------

        CREATE OR REPLACE FUNCTION Nguyen_27_monthlyPayment (P NUMERIC(9,2), APR NUMERIC(8,6), years NUMERIC(3,0))
        RETURNS NUMERIC
        LANGUAGE plpgsql
        AS
        $$
                DECLARE
                        i NUMERIC(8,7);
                        n NUMERIC(5,0);
                        temp1 NUMERIC(17,15);
                        A NUMERIC(8,2);

                BEGIN
                        -- Calculate interest rate 'i'
                        i := (Nguyen_27_monthlyPayment.APR/12);

                        -- Calculate number of mortgage payments 'n'
                        n := (Nguyen_27_monthlyPayment.years * 12);


                        -- Calculate the 'i' plus 1. Take that result
                        -- to the power of 'n'. Then 1 is subtracted from that total.
                        temp1 := (POWER((1+i),n)) - 1;

                        -- Calculate the monthly mortgage payment 'A'
                        A := Nguyen_27_monthlyPayment.P * (i + (i/temp1));

                        -- Return the montly mortgage payment 'A'
                        RETURN A;
                END;
        $$;


        /* Call function Nguyen_27_monthlyPayment() using the following SELECT statment: */
        SELECT Nguyen_27_monthlyPayment(250000,0.04125,30);



-----------------------------------------------------------------------------------------------
/*      A query to find the cust_ID and customer_name of each customer at the bank who          */
/*      only has a loan at the bank, and no account.                                            */
-----------------------------------------------------------------------------------------------

        SELECT cust_ID, customer_name
        FROM depositor NATURAL RIGHT OUTER JOIN borrower NATURAL FULL OUTER JOIN customer
        WHERE loan_number IS NOT NULL AND account_number IS NULL;


-----------------------------------------------------------------------------------------------
/*      A query to find the cust_ID and customer_name of each customer who lives on the */
/*      same street and in the same city as customer ‘12345’. Customer ‘12345’ included in      */
/*      query results.                                                                          */
-----------------------------------------------------------------------------------------------

        SELECT b.cust_ID, b.customer_name,
        FROM customer AS a, customer AS b
        WHERE a.cust_ID = '12345'
                AND a.customer_street = b.customer_street
                AND a.customer_city = b.customer_city;
```

```
-----------------------------------------------------------------------------------------------
/*      A query to find the name of each branch that has at least one customer who has an       */
/*      account in the bank and who lives in “Harrison”.                                        */
-----------------------------------------------------------------------------------------------

        SELECT branch_name
        FROM depositor NATURAL JOIN customer NATURAL JOIN account
        WHERE customer_city='Harrison'
        GROUP BY branch_name
        HAVING COUNT(branch_name) >= 1;


-----------------------------------------------------------------------------------------------
/*      Query to find each customer who has an account at every branch located in Brooklyn.     */
-----------------------------------------------------------------------------------------------

        -- SELECT values from customer_name attribute/column
        SELECT customer_name

        -- INNER JOIN customer, depositor, account, and branch alias 'b' relations/tables
        FROM customer NATURAL JOIN depositor NATURAL JOIN account NATURAL JOIN branch AS b

        -- WHERE values from branch_city attribute/column equals to 'Brooklyn'
        WHERE b.branch_city = 'Brooklyn'

        -- GROUP values in customer_name attribute/column
        GROUP BY customer_name

        -- HAVING number of branch located in 'Brooklyn' equals to every branch located in 'Brooklyn'
        HAVING
                COUNT(DISTINCT b.branch_name) = (SELECT COUNT(branch_name)
                        FROM branch
                                WHERE branch_city = 'Brooklyn');

-----------------------------------------------------------------------------------------------
/*      A SQL trigger to carry out the following action: If an account is deleted, a trigger    */
/*      delete the dependent tuple(s) from the depositor table for every owner of the deleted   */
/*      account. Note that there may be jointly-owned bank accounts. In other words, a trigger  */
/*      that performs the exact action of an ON DELETE CASCADE clause of a FOREIGN KEY          */
/*      CONSTRAINT. Below are both trigger function definition, and your trigger definition.    */
-----------------------------------------------------------------------------------------------

        -- Trigger function definition.
        CREATE OR REPLACE FUNCTION Nguyen_27_bankTriggerFunction()
        RETURNS TRIGGER
        LANGUAGE plpgsql
        AS
        $$
                BEGIN
                        DELETE FROM depositor
                        WHERE account_number = OLD.account_number;
                        RETURN NULL;
                END;
        $$;

        -- Trigger definition.
        CREATE TRIGGER Nguyen_27_bankTrigger
        AFTER DELETE ON account
        FOR EACH ROW
        EXECUTE PROCEDURE Nguyen_27_bankTriggerFunction();



        -- Invoke the trigger and trigger function using the following DELETE statement on account:
        -- The DELETE statement on account relation will trigger the same action that is accomplished
        -- by the ON DELETE CASCADE on foreign key constraint account_number in depositor relation.
        DELETE FROM account WHERE account_number = '774436581';
```

```
-----------------------------------------------------------------------------------------------
/*      A procedure that accepts an instructor ID as input, calculates the total number of      */
/*      course sections taught by that instructor, and adds a tuple to he instructor_course_nums*/
/*      table consisting of the instructors ID number, name, and total courses taught - call    */
/*      theseattributes: ID, name, and tot_courses. If the instructor already has an entry in   */
/*      the table,procedure makes sure the total number of courses taught in the                */
/*      instructor_course_nums table is up-to-date.                                          */
-----------------------------------------------------------------------------------------------

        CREATE TABLE instructor_course_nums
        (
                ID              VARCHAR(35),
                name            VARCHAR(50) NOT NULL,
                tot_courses     INTEGER,
                CONSTRAINT instructor_course_nums_pkey PRIMARY KEY (ID)
        );


        CREATE OR REPLACE PROCEDURE Nguyen_27_insCourseNumsProc(ID VARCHAR(35))
        LANGUAGE plpgsql
        AS
        $$
                DECLARE
                        numOfCoursesTaught INTEGER;
                        nameOfInstructor VARCHAR(50);

                BEGIN
                        -- Number of courses taught
                        SELECT COUNT(teaches.ID) INTO numOfCoursesTaught

                        -- from teaches relation
                        FROM teaches

                        -- specified by the input instructor ID
                        WHERE teaches.ID = Nguyen_27_insCourseNumsProc.ID;

                        -- Name of instructor
                        SELECT name INTO nameOfInstructor

                        -- from instructor relation
                        FROM instructor

                        -- specified by input instructor ID
                        WHERE instructor.ID = Nguyen_27_insCourseNumsProc.ID;

                        -- Insert into instructor_course_nums
                        INSERT INTO instructor_course_nums

                        -- the specified instructor ID, instructor's name,
                        -- and number of courses taught by that instructor
                        VALUES(Nguyen_27_insCourseNumsProc.ID, nameOfInstructor, numOfCoursesTaught)

                        -- If instructor already exist
                        ON CONFLICT ON CONSTRAINT instructor_course_nums_pkey

                        -- then update, instead of insert, the number of courses taught by the instructor

                        DO UPDATE SET tot_courses = numOfCoursesTaught;
                END;
        $$;

        -- Call procedure Nguyen_27_insCourseNumsProc() using the following CALL statement:
        -- '83821' is instructor_ID from teaches relation in university_small database provided by Dr. Morabito.
        CALL Nguyen_27_insCourseNumsProc('83821');
```