## The construction of a banking system to Gagnef Savings Bank

**In this you will create a small banking system for Gagnef Savings Bank**. The former potato farmer Thomas Kvist, who incidentally is very indolent, has taken up the post of bank manager at the new bank. Thomas has been through a tough academic analysis, and worked out something that resembles a system specification. Below you will find an overview of the tables you need to create for the task at hand.

SQL> desc customer

Name Null? Type

----------------------------------------- -------- ------------

**CUST\_ID** NOT NULL VARCHAR2(11)

FIRST\_NAME NOT NULL VARCHAR2(25)

LAST\_NAME NOT NULL VARCHAR2(25)

PASSWD NOT NULL VARCHAR2(6)

SQL> desc account\_type

Name Null? Type

----------------------------------------- -------- ------------

**ACCTY\_ID** NOT NULL NUMBER(6)

ACCTY\_NAME NOT NULL VARCHAR2(20)

PRESENT\_INTEREST NOT NULL NUMBER(5,2)

SQL> desc interest\_change

Name Null? Type

----------------------------------------- -------- ------------

**INTCH\_ID** NOT NULL NUMBER(6)

ACCTY\_ID NOT NULL NUMBER(6)

INTEREST NOT NULL NUMBER(5,2)

DATE\_TIME NOT NULL DATE

SQL> desc account

Name Null? Type

----------------------------------------- -------- ------------

**ACC\_ID** NOT NULL NUMBER(8)

ACCTY\_ID NOT NULL NUMBER(6)

DATE\_TIME NOT NULL DATE

BALANCE NOT NULL NUMBER(10,2)

SQL> desc account\_owner

Name Null? Type

----------------------------------------- -------- ------------

**ACCOW\_ID** NOT NULL NUMBER(9)

CUST\_ID NOT NULL VARCHAR2(11)

ACC\_ID NOT NULL NUMBER(8)

SQL> desc withdrawal

Name Null? Type

----------------------------------------- -------- ------------

**WIT\_ID** NOT NULL NUMBER(9)

CUST\_ID NOT NULL VARCHAR2(11)

ACC\_ID NOT NULL NUMBER(8)

AMOUNT NOT NULL NUMBER(10,2)

DATE\_TIME NOT NULL DATE

SQL> desc deposition

Name Null? Type

----------------------------------------- -------- ------------

**DEP\_ID** NOT NULL NUMBER(9)

CUST\_ID NOT NULL VARCHAR2(11)

ACC\_ID NOT NULL NUMBER(8)

AMOUNT NOT NULL NUMBER(10,2)

DATE\_TIME NOT NULL DATE

SQL> desc transfer

Name Null? Type

----------------------------------------- -------- ------------

**TRA\_ID** NOT NULL NUMBER(9)

CUST\_ID NOT NULL VARCHAR2(11)

FROM\_ACC\_ID NOT NULL NUMBER(8)

TO\_ACC\_ID NOT NULL NUMBER(8)

AMOUNT NOT NULL NUMBER(10,2)

DATE\_TIME NOT NULL DATE

**Task 1**

Draw a data model of the table descriptions above. The column at the top in each table is the primary key column. Referential integrity is apparent through the naming of the columns except in one case, the transfer table's two columns: from\_acc\_id and to\_acc\_id both refer to account(acc\_id). It is clear that the bank manager used *action-modeling* as a method. This is when the actions to **withdraw** money, **deposit** money and **transfer** money between accounts are represented by separate tables with meaningful names. The column cust\_id in the customer table is a *person identification number*.

A diagram of a data flow

Description automatically generated

**Task 2**

Use SQL DDL to implement the data model above, in the data base.

**PL/SQL (Procedural Language/SQL) Script:**

--Table 1

**CREATE** **TABLE** customer(

CUST\_ID **VARCHAR2**(11)**NOT** **NULL**,

FIRST\_NAME **VARCHAR2**(25) **NOT** **NULL**,

LAST\_NAME **VARCHAR2**(25) **NOT** **NULL**,

PASSWD **VARCHAR2**(6) **NOT** **NULL**);

**ALTER** **TABLE** customer

**ADD** **CONSTRAINT** customer\_CUST\_ID\_pk **PRIMARY** **KEY** (CUST\_ID);

--Table 2

**CREATE** **TABLE** account\_type(

ACCTY\_ID **NUMBER**(6)**NOT** **NULL**,

ACCTY\_NAME **VARCHAR2**(20) **NOT** **NULL**,

PRESENT\_INTEREST **NUMBER**(5,2) **NOT** **NULL**);

**ALTER** **TABLE** account\_type

**ADD** **CONSTRAINT** account\_type\_ACCTY\_ID\_pk **PRIMARY** **KEY** (ACCTY\_ID);

--Table 3

**CREATE** **TABLE** interest\_change(

INTCH\_ID **NUMBER**(6)**NOT** **NULL**,

ACCTY\_ID **NUMBER**(6) **NOT** **NULL**,

INTEREST **NUMBER**(5,2) **NOT** **NULL**,

DATE\_TIME **DATE** **NOT** **NULL**);

**ALTER** **TABLE** interest\_change

**ADD** **CONSTRAINT** interest\_change\_INTCH\_ID\_pk **PRIMARY** **KEY** (INTCH\_ID)

**ADD** **CONSTRAINT** interest\_change\_ACCTY\_ID\_fk **FOREIGN** **KEY**(ACCTY\_ID) **REFERENCES**

account\_type (ACCTY\_ID);

--Table 4

**CREATE** **TABLE** **account**(

ACC\_ID **NUMBER**(8)**NOT** **NULL**,

ACCTY\_ID **NUMBER**(6) **NOT** **NULL**,

DATE\_TIME **DATE** **NOT** **NULL**,

BALANCE **NUMBER**(10,2)**NOT** **NULL**);

**ALTER** **TABLE** **account**

**ADD** **CONSTRAINT** account\_ACC\_ID\_pk **PRIMARY** **KEY** (ACC\_ID)

**ADD** **CONSTRAINT** account\_ACCTY\_ID\_fk **FOREIGN** **KEY**(ACCTY\_ID) **REFERENCES**

account\_type (ACCTY\_ID);

--Table 5

**CREATE** **TABLE** account\_owner(

ACCOW\_ID **NUMBER**(9)**NOT** **NULL**,

CUST\_ID **VARCHAR2**(11) **NOT** **NULL**,

ACC\_ID **NUMBER**(8) **NOT** **NULL**);

**ALTER** **TABLE** account\_owner

**ADD** **CONSTRAINT** account\_owner\_ACCOW\_ID\_pk **PRIMARY** **KEY** (ACCOW\_ID)

**ADD** **CONSTRAINT** account\_owner\_CUST\_ID\_fk **FOREIGN** **KEY**(CUST\_ID) **REFERENCES**

customer (CUST\_ID)

**ADD** **CONSTRAINT** account\_owner\_ACC\_ID\_fk **FOREIGN** **KEY**(ACC\_ID) **REFERENCES** account (ACC\_ID);

--Table 6

**CREATE** **TABLE** withdrawal(

WIT\_ID **NUMBER**(9)**NOT** **NULL**,

CUST\_ID **VARCHAR2**(11) **NOT** **NULL**,

ACC\_ID **NUMBER**(8) **NOT** **NULL**,

AMOUNT **NUMBER**(10,2)**NOT** **NULL**,

DATE\_TIME **DATE** **NOT** **NULL**);

**ALTER** **TABLE** withdrawal

**ADD** **CONSTRAINT** withdrawal\_WIT\_ID\_pk **PRIMARY** **KEY** (WIT\_ID)

**ADD** **CONSTRAINT** withdrawal\_CUST\_ID\_fk **FOREIGN** **KEY**(CUST\_ID) **REFERENCES** customer (CUST\_ID)

**ADD** **CONSTRAINT** withdrawal\_ACC\_ID\_fk **FOREIGN** **KEY**(ACC\_ID) **REFERENCES** account (ACC\_ID);

--Table 7

**CREATE** **TABLE** deposition(

DEP\_ID **NUMBER**(9)**NOT** **NULL**,

CUST\_ID **VARCHAR2**(11) **NOT** **NULL**,

ACC\_ID **NUMBER**(8) **NOT** **NULL**,

AMOUNT **NUMBER**(10,2)**NOT** **NULL**,

DATE\_TIME **DATE** **NOT** **NULL**);

**ALTER** **TABLE** deposition

**ADD** **CONSTRAINT** deposition\_DEP\_ID\_pk **PRIMARY** **KEY** (DEP\_ID)

**ADD** **CONSTRAINT** deposition\_CUST\_ID\_fk **FOREIGN** **KEY**(CUST\_ID) **REFERENCES**

customer (CUST\_ID)

**ADD** **CONSTRAINT** deposition\_ACC\_ID\_fk **FOREIGN** **KEY**(ACC\_ID) **REFERENCES**

account (ACC\_ID);

--Table 8

**CREATE** **TABLE** transfer(

TRA\_ID **NUMBER**(9)**NOT** **NULL**,

CUST\_ID **VARCHAR2**(11) **NOT** **NULL**,

FROM\_ACC\_ID **NUMBER**(8) **NOT** **NULL**,

TO\_ACC\_ID **NUMBER**(8) **NOT** **NULL**,

AMOUNT **NUMBER**(10,2)**NOT** **NULL**,

DATE\_TIME **DATE** **NOT** **NULL**);

**ALTER** **TABLE** transfer

**ADD** **CONSTRAINT** transfer\_TRA\_ID\_pk **PRIMARY** **KEY** (TRA\_ID)

**ADD** **CONSTRAINT** transfer\_CUST\_ID\_fk **FOREIGN** **KEY**(CUST\_ID) **REFERENCES**

customer (CUST\_ID)

**ADD** **CONSTRAINT** transfer\_FROM\_ACC\_ID\_fk **FOREIGN** **KEY**(FROM\_ACC\_ID) **REFERENCES**

account(ACC\_ID) **ON** **DELETE** **CASCADE**

**ADD** **CONSTRAINT** transfer\_TO\_ACC\_ID\_fk **FOREIGN** **KEY**(TO\_ACC\_ID) **REFERENCES**

account(ACC\_ID) **ON** **DELETE** **CASCADE**;

Data model is implemented.

**Task 3**

Create a trigger called biufer\_customer that starts **before** insert or update of the column **passwd** in the **customer table**. The trigger shall verify that the password is exactly six characters long, no more, no less. Unless this requirement is fulfilled, the trigger shall stop the transaction and confirm that this error occurred.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** biufer\_customer

**BEFORE** **INSERT** **OR** **UPDATE**

**OF** passwd

**ON** customer

**FOR** **EACH** **ROW**

**WHEN** (**LENGTH**(**NEW**.passwd) <> 6)

**BEGIN**

**RAISE\_APPLICATION\_ERROR**(-20001, 'Check the password');

**END**;

Trigger called biufer\_customer is created

**Task 4**

Create a procedure called do\_new\_cutomer. The procedure shall be used to add new rows to the customer table (i.e. add new customers). **Create the input parameters in the following order:** cust\_id, first\_name, last\_name, passwd.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_new\_customer(

p\_cust\_id **IN** customer.cust\_id%**TYPE**,

p\_first\_name **IN** customer.first\_name%**TYPE**,

p\_last\_name **IN** customer.last\_name%**TYPE**,

p\_passwd **IN** customer.passwd%**TYPE**)

**AS**

**BEGIN**

**INSERT** **INTO** customer(cust\_id, first\_name, last\_name, passwd)

**VALUES** (p\_cust\_id, p\_first\_name, p\_last\_name, p\_passwd);

**COMMIT**;

**END**;

procedure called do\_new\_cutomer is created.

**Task 5**

Add 4 rows to the customer table. Do that by using the procedure **do\_new\_customer**. Test if the trigger biufer\_customer works. Do that by adding a password that has the wrong format.

Start with the trigger test below, before you call the procedure in the anonymous block.

**Trigger test**:

begin

do\_new\_customer('861124-4478','Raul','Ortiz','qwe');

end;

**Show the result of the trigger test here:**

A computer screen shot of a computer error

Description automatically generated

1. **Add 4 customers:**

-------------------------Start copy and paste---------------------------------

BEGIN

do\_new\_customer('650707-1111','Tito','Ortiz','qwerTY');

do\_new\_customer('560126-1148','Margreth','Andersson','olle85');

do\_new\_customer('840317-1457','Mary','Smith','asdfgh');

do\_new\_customer('861124-4478','Vincent','Ortiz','qwe123');

COMMIT;

END;

/

-------------------------End copy and paste---------------------------------

4 customers are added using the do\_new\_cutomer procedure.

**Task 6**

Createa sequence called pk\_seq. Use the sequence to create primary key values for the following table's primary key columns:

* Transfer
* Deposition
* Withdrawal
* Account\_owner
* Interest\_change

**PL/SQL Script:**

**CREATE** **SEQUENCE** pk\_seq

**START** **WITH** 1

**INCREMENT** **BY**

1

A sequence called pk\_seq is created.

Now it is time to put some data into the tables account\_type, account and account\_owner.

-------------------------Start copy and paste---------------------------------

INSERT INTO account\_type(accty\_id,accty\_name,present\_interest)

VALUES(1,'farmer account',2.4);

INSERT INTO account\_type (accty\_id,accty\_name,present\_interest)

VALUES(2,'potato account',3.4);

INSERT INTO account\_type (accty\_id,accty\_name,present\_interest)

VALUES(3,'hog account',4.4);

COMMIT;

INSERT INTO account(acc\_id,accty\_id,date\_time,balance)

VALUES(123,1,SYSDATE - 321,0);

INSERT INTO account(acc\_id,accty\_id,date\_time,balance)

VALUES(5899,2,SYSDATE - 2546,0);

INSERT INTO account(acc\_id,accty\_id,date\_time,balance)

VALUES(5587,3,SYSDATE - 10,0);

INSERT INTO account(acc\_id,accty\_id,date\_time,balance)

VALUES(8896,1,SYSDATE - 45,0);

COMMIT;

INSERT INTO account\_owner(accow\_id,cust\_id,acc\_id)

VALUES(pk\_seq.NEXTVAL,'650707-1111',123);

INSERT INTO account\_owner(accow\_id,cust\_id,acc\_id)

VALUES(pk\_seq.NEXTVAL,'560126-1148',123);

INSERT INTO account\_owner(accow\_id,cust\_id,acc\_id)

VALUES(pk\_seq.NEXTVAL,'650707-1111',5899);

INSERT INTO account\_owner(accow\_id,cust\_id,acc\_id)

VALUES(pk\_seq.NEXTVAL,'861124-4478',8896);

COMMIT;

-------------------------End copy and paste---------------------------------

Data is added into the tables account\_type, account and account\_owner.

**Task 7**

Create a function named log\_in. This should return **0**(zero) if the login failed, or **1**(one) if the login was successful. To be able to login a customer has to provide two parameters: cust\_id and passwd.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **FUNCTION** log\_in(p\_cust\_id **IN** customer.cust\_id%**TYPE**,

p\_passwd **IN** customer.passwd%**TYPE**)

**RETURN** **NUMBER**

**AS**

v\_count **NUMBER**;

**BEGIN**

**SELECT** **COUNT**(\*)

**INTO** v\_count

**FROM** customer

**WHERE** cust\_id = p\_cust\_id **AND** passwd = p\_passwd;

**IF** v\_count = 1 **THEN**

**RETURN** 1; -- Login successful

**ELSE**

**RETURN** 0; -- Login failed

**END IF**;

**END**;

A function named log\_in is created.

**Test that the login function works by excute (with your own parameters):**

SELECT log\_in('a\_cust\_id','the\_password')

FROM dual;

**Show the result of the login test here:**

**PL/SQL Script and Result:**

When logging in with fake details,

**SELECT** log\_in('650007-1111','abcdef')

**FROM** dual;

A screenshot of a computer code

Description automatically generated

When logging in with original details,

**SELECT** log\_in('650707-1111','qwerTY')

**FROM** dual;

A screenshot of a phone number

Description automatically generated

**Task 8**

Create a function called get\_balance. This should return the current balance for the account whose account number (acc\_id) is sent to the function**.**

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **FUNCTION** get\_balance(p\_acc\_id **IN** **account**.acc\_id%**TYPE**)

**RETURN** **NUMBER**

**AS**

v\_balance **NUMBER**;

**BEGIN**

**SELECT** balance

**INTO** v\_balance

**FROM** **account**

**WHERE** acc\_id = p\_acc\_id;

**RETURN** v\_balance;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

**RETURN** -1;

**END**;

A function called get\_balance is created.

**Make sure that the function works!**

**Show the result of the function test here:**

**PL/SQL Script and Result:**

Got the results for acc\_id ‘8896’ – original acc\_id,

**SELECT** get\_balance('8896') **FROM** dual;

A screenshot of a computer

Description automatically generated

Results for fake acc\_id,

**SELECT** get\_balance('8966') **FROM** dual;

A screenshot of a computer

Description automatically generated

**Task 9**

Create a function called get\_authority. This function takes two parameters (found in the **account\_owner** table): cust\_id and acc\_id, and returns **1** (one), if the customer has the right to make withdrawals from the account, or **0** (zero), if the customer doesn't have any authority to the account.

*When a customer transfers money from one account to another, use the function to check if the customer has authority to the account represented by the column* ***from\_acc\_id*** *in the transfer table.*

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **FUNCTION** get\_authority(p\_cust\_id **IN** account\_owner.cust\_id%**TYPE**,

p\_acc\_id **IN** account\_owner.acc\_id%**TYPE**)

**RETURN** **NUMBER**

**AS**

v\_count **NUMBER**;

**BEGIN**

**SELECT** **COUNT**(\*)

**INTO** v\_count

**FROM** account\_owner

**WHERE** cust\_id = p\_cust\_id **AND** acc\_id = p\_acc\_id;

**IF** v\_count > 0 **THEN**

**RETURN** 1; -- Customer has authority

**ELSE**

**RETURN** 0; -- Customer does not have authority

**END IF**;

**END**;

A function called get\_authority is created

**Make sure that the function works!**

**Show the result of the function test here:**

**PL/SQL Script and Result:**

**SELECT** get\_authority('650707-1111',123)

**FROM** dual;

A screenshot of a phone

Description automatically generated

**SELECT** get\_authority('650607-1111',132)

**FROM** dual;

A screenshot of a phone number

Description automatically generated

**Task 10**

Create a trigger called aifer\_deposition. The trigger shall ensure that the balance is right after deposition to an account.

The trigger should start **after insert** on the deposition table.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_deposition

**AFTER** **INSERT** **ON** deposition

**FOR** **EACH** **ROW**

**BEGIN**

**UPDATE** **account**

**SET** balance = balance + :NEW.amount

**WHERE** acc\_id = :NEW.acc\_id;

**END**;

A trigger called aifer\_deposition is created.

**Task 11**

Create a trigger called bifer\_withdrawal. The trigger must ensure that you cannot withdraw more money than there is available in the account.

The trigger should start before insert into the withdrawal table.

**NOTE!** The trigger should use the function get\_balance for this job.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** bifer\_withdrawal

**BEFORE** **INSERT** **ON** withdrawal

**FOR** **EACH** **ROW**

**DECLARE**

v\_balance **NUMBER**;

**BEGIN**

-- Getting the current balance of the account

v\_balance := get\_balance(:NEW.acc\_id);

-- Checking if the withdrawal amount is greater than the balance

**IF** :NEW.amount > v\_balance **THEN**

-- If the amount is greater, then raise an error

**RAISE\_APPLICATION\_ERROR**(-20001, 'Insufficient funds for withdrawal');

**END IF**;

**END**;

A trigger called bifer\_withdrawal is created.

**Task 12**

Create a trigger called aifer\_withdrawal. The trigger shall ensure that the balance is correct after withdrawal on an account.

The trigger should start **after insert** on the withdrawal table.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_withdrawal

**AFTER** **INSERT** **ON** withdrawal

**FOR** **EACH** **ROW**

**BEGIN**

-- Checking if the account has enough balance

**IF** get\_balance(:NEW.acc\_id) >= :NEW.amount **THEN**

-- Updating the account balance

**UPDATE** **account**

**SET** balance = balance - :NEW.amount

**WHERE** acc\_id = :NEW.acc\_id;

**ELSE**

-- Raising an error if the fund is insufficient

**RAISE\_APPLICATION\_ERROR**(-20002, 'Insufficient funds for withdrawal');

**END IF**;

**END**;

A trigger called aifer\_withdrawal is created.

**Task 13**

Create an additional trigger, this time with the name bifer\_transfer. The trigger must ensure that you cannot take out more money than there is available in the account that you are moving money from, when you transfer money from one account to another.

**NOTE!** The trigger should use the function get\_balance for this job.

*This is a good example of how we can reuse our SQL code!*

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** bifer\_transfer

**BEFORE** **INSERT** **ON** transfer

**FOR** **EACH** **ROW**

**BEGIN**

-- Checking if the account has enough balance

**IF** get\_balance(:NEW.from\_acc\_id) < :NEW.amount **THEN**

-- If the balance is insufficient, raise an error

**RAISE\_APPLICATION\_ERROR**(-20003, 'Insufficient funds for transfer');

**END IF**;

**END**;

A trigger called bifer\_transfer is created.

**Task 14**

Now it is time to create the last trigger in this lab. The trigger should have the name aifer\_transfer. It must ensure that the balance is correct on the accounts after the transaction is completed.

The trigger should start **after insert** on the transfer table.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_transfer

**AFTER** **INSERT** **ON** transfer

**FOR** **EACH** **ROW**

**BEGIN**

-- Subtracting the transfer amount from the source account

**UPDATE** **account**

**SET** balance = balance - :NEW.amount

**WHERE** acc\_id = :NEW.from\_acc\_id;

-- Adding the transfer amount to the destination account

**UPDATE** **account**

**SET** balance = balance + :NEW.amount

**WHERE** acc\_id = :NEW.to\_acc\_id;

**END**;

A trigger called aifer\_transfer is created.

**Task 15**

Create a procedure called do\_deposition. The procedure creates a row in the table **deposition**. After the transaction has committed, a message containing the balance of the account, after the deposit, should be printed out.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_deposition(

p\_dep\_id **IN** deposition.dep\_id%**TYPE**,

p\_cust\_id **IN** deposition.cust\_id%**TYPE**,

p\_acc\_id **IN** deposition.acc\_id%**TYPE**,

p\_amount **IN** deposition.amount%**TYPE**,

p\_date\_time **IN** deposition.date\_time%**TYPE**)

**AS**

v\_balance **NUMBER**;

**BEGIN**

-- Inserting the deposition

**INSERT** **INTO** deposition (dep\_id, cust\_id, acc\_id, amount, date\_time)

**VALUES** (p\_dep\_id, p\_cust\_id, p\_acc\_id, p\_amount, p\_date\_time);

-- Commit the transaction

**COMMIT**;

-- Retrieve and display the updated balance

v\_balance := get\_balance(p\_acc\_id);

**DBMS\_OUTPUT**.**PUT\_LINE**('Account balance after deposit: ' || v\_balance);

**END**;

A procedure called do\_deposition is created.

**Task 16**

Make sure that the procedure do\_deposition works!

**Show the result of the test here:**

**PL/SQL Script and Result:**

Balance before executing the procedure,

**select**\***from** **account**

A screenshot of a computer

Description automatically generated

**BEGIN**

do\_deposition(p\_dep\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_acc\_id => '123',

p\_amount => 7200,

p\_date\_time => **SYSDATE**);

**END**;

do\_deposition procedure is executed.

A close-up of a white background

Description automatically generated

Balance after executing the procedure,

**select**\***from** **account**

A screenshot of a computer

Description automatically generated

**Task 17**

Create a procedure called do\_withdrawal. The procedure creates a row in the table **withdrawal**. Declare an exception called unauthorized. Raise *unauthorized* when the customer doesn't have any authority to the account. Use the function get\_authority to perform the verification.

If the customer is **unauthorized**:

1. Stop the transaction!
2. Print out a message: **"Unauthorized user!"**

If the customer is **authorized**:

1. Print out a message with the balance of the account after the transaction.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_withdrawal(

p\_wit\_id **IN** withdrawal.wit\_id%**TYPE**,

p\_cust\_id **IN** withdrawal.cust\_id%**TYPE**,

p\_acc\_id **IN** withdrawal.acc\_id%**TYPE**,

p\_amount **IN** withdrawal.amount%**TYPE**,

p\_date\_time **IN** withdrawal.date\_time%**TYPE**)

**AS**

-- Declare the unauthorized exception

unauthorized **EXCEPTION**;

v\_authority **NUMBER**;

v\_balance **NUMBER**;

**BEGIN**

-- Checking if the customer is authorized to withdraw

v\_authority := get\_authority(p\_cust\_id, p\_acc\_id);

**IF** v\_authority = 0 **THEN**

-- If unauthorized, raise the unauthorized exception

**RAISE** unauthorized;

**ELSE**

-- Inserting the withdrawal record

**INSERT** **INTO** withdrawal (wit\_id, cust\_id, acc\_id, amount, date\_time)

**VALUES** (p\_wit\_id, p\_cust\_id, p\_acc\_id, p\_amount, p\_date\_time);

-- Commit the transaction

**COMMIT**;

-- Retrieve and display the updated balance

v\_balance := get\_balance(p\_acc\_id);

**DBMS\_OUTPUT**.**PUT\_LINE**('Account balance after withdrawal: ' || v\_balance);

**END IF**;

**EXCEPTION**

**WHEN** unauthorized **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**('Unauthorized user!');

**ROLLBACK**; -- Stop the transaction

**END**;

A procedure called do\_withdrawal is created.

**Task 18**

Make sure that the procedure do\_withdrawal works!

**Now** it's also an opportunity to see if the triggers do their jobs. Check if you can withdraw more money than what is available on your account. Make sure the balance is updated as it should.

**Show the results of the tests here:**

**PL/SQL Script and Result:**

Executing do\_deposition procedure with valid credentials.

**BEGIN**

do\_withdrawal(p\_wit\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_acc\_id => '123',

p\_amount => 1000,

p\_date\_time => **SYSDATE**);

**END**;

A close-up of a white background

Description automatically generated

Executing do\_deposition procedure with invalid credentials.

**BEGIN**

do\_withdrawal(p\_wit\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650101-1111',

p\_acc\_id => '123',

p\_amount => 1000,

p\_date\_time => **SYSDATE**);

**END**;

A close-up of a text

Description automatically generated

1. **Checking the current balance.**

**SELECT** balance **FROM** **account** **WHERE** acc\_id = '123';

A screenshot of a computer

Description automatically generated

1. **Attempting a withdrawal exceeding the balance**

**BEGIN**

do\_withdrawal(p\_wit\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_acc\_id => '123',

p\_amount => 100000,

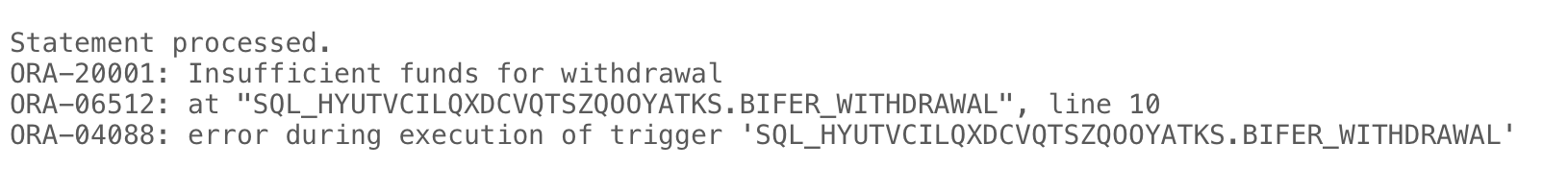
p\_date\_time => **SYSDATE**);

**EXCEPTION**

**WHEN** OTHERS **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**(**SQLERRM**); -- Print error message

**END**;



1. **Attempting a valid withdrawl**

**BEGIN**

do\_withdrawal(p\_wit\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_acc\_id => '123',

p\_amount => 500,

p\_date\_time => **SYSDATE**);

**EXCEPTION**

**WHEN** OTHERS **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**(**SQLERRM**);

**END**;

A close-up of a white background

Description automatically generated

1. **Checking the current balance**

**SELECT** balance **FROM** **account** **WHERE** acc\_id = '123';

A screenshot of a computer

Description automatically generated

**Task 19**

Create a procedure called do\_transfer. The procedure creates a row in the table **transfer**. Declare an exception called unauthorized. Raise *unauthorized* when the customer doesn't have any authority to the account from which money is withdrawn. Use the function get\_authority to perform the verification.

If the customer is **unauthorized**:

1. Stop the transaction!
2. Print out a message: **"Unauthorized user!"**

If the customer is **authorized**:

1. Print out a message with the balance of **both** the accounts after the transaction.

**PL/SQL Script:**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_transfer(

p\_tra\_id **IN** transfer.tra\_id%**TYPE**,

p\_cust\_id **IN** transfer.cust\_id%**TYPE**,

p\_from\_acc\_id **IN** transfer.from\_acc\_id%**TYPE**,

p\_to\_acc\_id **IN** transfer.to\_acc\_id%**TYPE**,

p\_amount **IN** transfer.amount%**TYPE**,

p\_date\_time **IN** transfer.date\_time%**TYPE**)

**AS**

-- Declaring the unauthorized exception

unauthorized **EXCEPTION**;

v\_authority **NUMBER**;

v\_balance\_from **NUMBER**;

v\_balance\_to **NUMBER**;

**BEGIN**

-- Checking if the customer is authorized to withdraw from the source account

v\_authority := get\_authority(p\_cust\_id, p\_from\_acc\_id);

**IF** v\_authority = 0 **THEN**

-- If unauthorized, unauthorized exception is raised

**RAISE** unauthorized;

**ELSE**

-- Inserting the transfer record

**INSERT** **INTO** transfer (tra\_id, cust\_id, from\_acc\_id, to\_acc\_id, amount, date\_time)

**VALUES** (p\_tra\_id, p\_cust\_id, p\_from\_acc\_id, p\_to\_acc\_id, p\_amount, p\_date\_time);

-- Commit the transaction

**COMMIT**;

-- Retrieve and display the updated balances

v\_balance\_from := get\_balance(p\_from\_acc\_id);

v\_balance\_to := get\_balance(p\_to\_acc\_id);

**DBMS\_OUTPUT**.**PUT\_LINE**('Balance in source account after transfer: ' || v\_balance\_from);

**DBMS\_OUTPUT**.**PUT\_LINE**('Balance in destination account after transfer: ' || v\_balance\_to);

**END IF**;

**EXCEPTION**

**WHEN** unauthorized **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**('Unauthorized user!');

**ROLLBACK**; -- Stop the transaction

**END**;

A procedure called do\_transfer is created.

**Task 20**

Try to move money between the accounts, using the procedure do\_transfer.

**Make sure the triggers work!**

**Show the results of the tests here:**

**PL/SQL Script and Result:**

1. **Balance before doing the transfer,**

**SELECT** balance **FROM** **account** **WHERE** acc\_id **IN** ('123', '8896');

A screenshot of a phone

Description automatically generated

1. **Testing the unauthorized transfer**

**BEGIN**

do\_transfer(p\_tra\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650101-1111',

p\_from\_acc\_id => '123',

p\_to\_acc\_id => '8896',

p\_amount => 300,

p\_date\_time => **SYSDATE**);

**EXCEPTION**

**WHEN** OTHERS **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**(**SQLERRM**); -- Print error message

**END**;

A close-up of a white background

Description automatically generated

1. **Testing the authorized transfer with sufficient funds**

**BEGIN**

do\_transfer(p\_tra\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_from\_acc\_id => '123',

p\_to\_acc\_id => '8896',

p\_amount => 300,

p\_date\_time => **SYSDATE**);

**EXCEPTION**

**WHEN** OTHERS **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**(**SQLERRM**);

**END**;

**A close-up of a white background

Description automatically generatedSELECT** balance **FROM** **account** **WHERE** acc\_id **IN** ('123', '8896');

A screenshot of a phone

Description automatically generated

1. **Testing the authorized transfer with insufficient funds**

**BEGIN**

do\_transfer(p\_tra\_id => pk\_seq.**NEXTVAL**,

p\_cust\_id => '650707-1111',

p\_from\_acc\_id => '123',

p\_to\_acc\_id => '8896',

p\_amount => 10000,

p\_date\_time => **SYSDATE**);

**EXCEPTION**

**WHEN** OTHERS **THEN**

**DBMS\_OUTPUT**.**PUT\_LINE**(**SQLERRM**);

**END**;

A close up of a text

Description automatically generated