Coordinator: Pär Douhan, pdo@du.se

Lab assignment - Procedures, functions and triggers, the construction of a banking system to Gagnef Savings Bank

In this lab we 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)
```

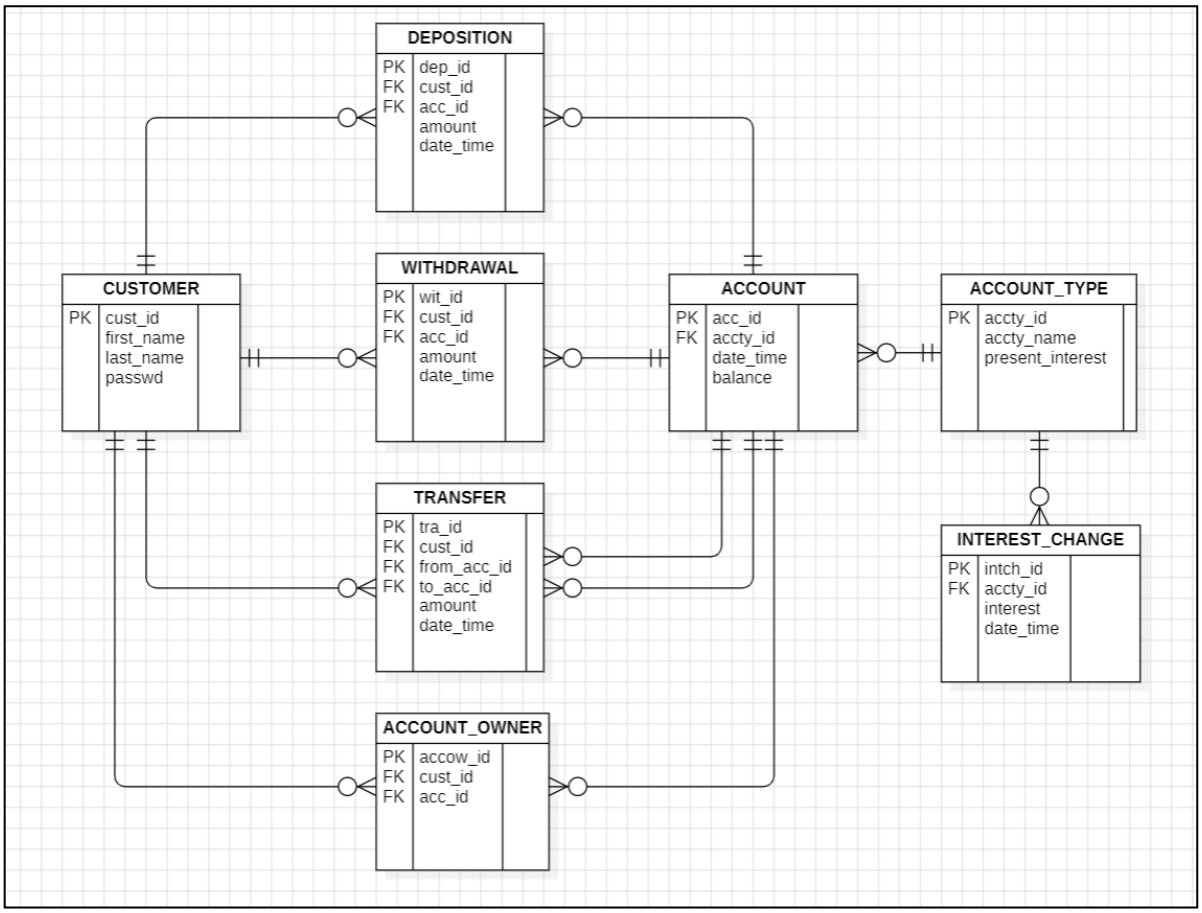
Coordinator: Pär Douhan, pdo@du.se

```
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
```

Coordinator: Pär Douhan, pdo@du.se

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*.



Coordinator: Pär Douhan, pdo@du.se

Task 2

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

```
/******** CUSTOMER TABLE ********/
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_custid_pk primary key(cust_id);


/******** ACCOUNT_TYPE TABLE ********/
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 accounttype_acctyid_pk primary key(accty_id);


/******** INTEREST_CHANGE TABLE ********/
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 interestchange_intchid_pk primary key(intch_id)
add constraint interestchange_acctyid_fk foreign key(accty_id) references
account_type(accty_id);


/******** ACCOUNT TABLE ********/
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_accid_pk primary key(acc_id)
add constraint account_acctyid_fk foreign key(accty_id) references
account_type(accty_id);
```

Coordinator: Pär Douhan, pdo@du.se

```
/******** ACCOUNT_OWNER TABLE ********/
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 accountowner_accowid_pk primary key(accow_id)
add constraint accountowner_custid_fk foreign key(cust_id) references
customer(cust_id)
add constraint accountowner_accid_fk foreign key(acc_id) references
account(acc_id);


/******** DEPOSITION TABLE ********/
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_depid_pk primary key(dep_id)
add constraint deposition_custid_fk foreign key(cust_id) references
customer(cust_id)
add constraint deposition_accid_fk foreign key(acc_id) references
account(acc_id);


/******** WITHDRAWAL TABLE ********/
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_witid_pk primary key(wit_id)
add constraint withdrawal_custid_fk foreign key(cust_id) references
customer(cust_id)
add constraint withdrawal_accid_fk foreign key(acc_id) references
account(acc_id);
```

Coordinator: Pär Douhan, pdo@du.se

```
/******** TRANSFER TABLE ********/
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_traid_pk primary key(tra_id)
add constraint transfer_custid_fk foreign key(cust_id) references
customer(cust_id)
add constraint transfer_fromaccid_fk foreign key(from_acc_id) references
account(acc_id)
add constraint transfer_toaccid_fk foreign key(to_acc_id) references
account(acc_id);
```

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.

```
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,'Password must contain 6 characters!');
end;
/
```

Task 4

Create a procedure called do_new_customer. 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.

```
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;
/
```

Coordinator: Pär Douhan, pdo@du.se

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:

```
ORA-20001: Password must contain 6 characters! ORA-06512: at
"SQL_KLXDRLKTEHZPQVKDMAISAPBIY.BIUFER_CUSTOMER", line 2 ORA-06512: at
"SQL_KLXDRLKTEHZPQVKDMAISAPBIY.DO_NEW_CUSTOMER", line 8 ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 172
```

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-----------------------------------
```

Task 6

Create a 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

```
create sequence pk_seq;

create or replace trigger bifer_accountowner_pk
before insert
on account_owner
for each row
begin
  select pk_seq.nextval
  into :new.accow_id
  from dual;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

```
create or replace trigger bifer_withdrawal_pk
before insert
on withdrawal
for each row
begin
  select pk_seq.nextval
  into :new.wit_id
  from dual;
end;
/

create or replace trigger bifer_deposition_pk
before insert
on deposition
for each row
begin
  select pk_seq.nextval
  into :new.dep_id
  from dual;
end;
/

create or replace trigger bifer_transfer_pk
before insert
on transfer
for each row
begin
  select pk_seq.nextval
  into :new.tra_id
  from dual;
end;
/

create or replace trigger bifer_interestchange_pk
before insert
on interest_change
for each row
begin
  select pk_seq.nextval
  into :new.intch_id
  from dual;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

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-----------------------------------
```

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.

```
create or replace function log_in(
p_cust_id in customer.cust_id%type,
p_passwd  in customer.passwd%type)
return number
as
v_result customer.cust_id%type;
begin
    select cust_id
    into v_result
    from customer
    where cust_id = p_cust_id
    and passwd = p_passwd;
    return 1;
exception
    when no_data_found then
    return 0;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

Test that the login function works by excute (with your own parameters) and show the result of the login test here:

```
select log_in('650707-1111','qwerTY') as log_in
from dual;
```

LOG_IN
1

```
select log_in('650707-1111','olle85') as log_in
from dual;
```

LOG_IN
0

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.

```
create or replace function get_balance(
p_acc_id in account.acc_id%type)
return account.balance%type
as
v_balance account.balance%type;
begin
    select balance
    into v_balance
    from account
    where acc_id = p_acc_id;
    return v_balance;
end;
/
```

Show the result of the function test here:

```
select get_balance(123) as current_balance
from dual;
```

CURRENT_BALANCE
0

Coordinator: Pär Douhan, pdo@du.se

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.*

```
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_result number(1);
begin
    select count(accow_id)
    into v_result
    from account_owner
    where cust_id = p_cust_id
    and acc_id = p_acc_id;
    return v_result;
end;
/
```

Show the result of the function test here:

```
select get_authority('650707-1111',123) as authorization
from dual;

select get_authority('650707-1111',5899) as authorization
from dual;
```

AUTHORIZATION
1

```
select get_authority('650707-1111',8896) as authorization
from dual;
```

AUTHORIZATION
0

Coordinator: Pär Douhan, pdo@du.se

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.

```
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;
/
```

Task 11

Create a trigger called bifer_withdrawal. The trigger must ensure that you cannot withdraw more money than 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.

```
create or replace trigger bifer_withdrawal
before insert
on withdrawal
for each row
begin
  if ( :new.amount > get_balance(:new.acc_id) ) then
  raise_application_error(-20001,'You do not have enough money in your
account!');
  end if;
end;
/
```

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.

```
create or replace trigger aifer_withdrawal
after insert
on withdrawal
for each row
begin
      update account
      set balance = balance - :new.amount
      where acc_id = :new.acc_id;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

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.

```
create or replace trigger bifer_transfer
before insert
on transfer
for each row
begin
  if ( :new.amount > get_balance(:new.from_acc_id) ) then
  raise_application_error(-20001,'You do not have enough money in your
account!');
  end if;
end;
/
```

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.

```
create or replace trigger aifer_transfer
after insert
on transfer
for each row
begin
      update account
      set balance = balance - :new.amount
      where acc_id = :new.from_acc_id;

      update account
      set balance = balance + :new.amount
      where acc_id = :new.to_acc_id;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

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.

```
create or replace procedure do_deposition(
p_cust_id in deposition.cust_id%type,
p_acc_id  in deposition.acc_id%type,
p_amount  in deposition.amount%type)
as
begin
     insert into deposition(cust_id, acc_id, amount, date_time)
     values(p_cust_id, p_acc_id,p_amount,sysdate);
     commit;
     dbms_output.put_line('The current balance of the account
'||p_acc_id||' is '||get_balance(p_acc_id)||' sek.');
end;
/
```

Task 16

Make sure that the procedure do_deposition works!
Show the result of the test here:

```
begin
     do_deposition('650707-1111',123,3240);
end;
/
```

```
Statement processed.
The current balance of the account 123 is 3240 sek.
```

```
select balance
from account
where acc_id = 123;
```

BALANCE
3240

Coordinator: Pär Douhan, pdo@du.se

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.

```
create or replace procedure do_withdrawal(
p_cust_id in withdrawal.cust_id%type,
p_acc_id  in withdrawal.acc_id%type,
p_amount  in withdrawal.amount%type)
as
begin
    if (get_authority(p_cust_id, p_acc_id)=0) then
        raise_application_error(-20001,'Unauthorized user!');
    else
        insert into withdrawal(cust_id, acc_id, amount, date_time)
        values(p_cust_id, p_acc_id,p_amount,sysdate);
        commit;
        dbms_output.put_line('The current balance of the account
'||p_acc_id||' is '||get_balance(p_acc_id)||' sek.');
    end if;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

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:

```
begin
   do_withdrawal('650707-1111',123,1000);
end;
/
```

```
Statement processed.
The current balance of the account 123 is 2240 sek.
```

```
select balance
from account
where acc_id = 123;
```

BALANCE
2240

```
begin
   do_withdrawal('650707-1111',123,3000);
end;
/
```

```
ORA-20001: You do not have enough money in your account! ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.BIFER_WITHDRAWAL", line 3
ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.DO_WITHDRAWAL", line 10
ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

```
begin
   do_withdrawal('861124-4478',123,100);
end;
/
```

```
ORA-20001: Unauthorized user! ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.DO_WITHDRAWAL", line 8
ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Coordinator: Pär Douhan, pdo@du.se

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.

```
create or replace procedure do_transfer(
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)
as
begin
    if (get_authority(p_cust_id, p_from_acc_id)=0) then
        raise_application_error(-20001,'Unauthorized user!');
    else
        insert into transfer(cust_id, from_acc_id, to_acc_id, amount, date_time)
        values(p_cust_id, p_from_acc_id, p_to_acc_id, p_amount, sysdate);
        commit;
        dbms_output.put_line('The current balance of the account
'||p_from_acc_id||' is '||get_balance(p_from_acc_id)||' sek.');
        dbms_output.put_line('The current balance of the account
'||p_to_acc_id  ||' is '||get_balance(p_to_acc_id)  ||' sek.');
    end if;
end;
/
```

Coordinator: Pär Douhan, pdo@du.se

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:

```
begin
  do_transfer('650707-1111',123,8896,1500);
end;
                                          /
```

```
Statement processed.
The current balance of the account 123 is 740 sek.
The current balance of the account 8896 is 1500 sek.
```

```
select balance
from account
where acc_id in (123,8896);
```

BALANCE
740
1500

```
begin
  do_transfer('861124-4478',123,8896,600);
end;
/
```

```
ORA-20001: Unauthorized user! ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.DO_TRANSFER", line 9
ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

```
begin
  do_transfer('650707-1111',123,8896,2000);
end;
/
```

```
ORA-20001: You do not have enough money in your account! ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.BIFER_TRANSFER", line 3
ORA-06512: at "SQL_DKPWOAHFQVBTAZSMMFCCQLGAF.DO_TRANSFER", line 11
ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```