CS1555 Recitation 10 Solution

Objective: to understand how triggers work in Oracle and to practice writing triggers.

Before we start:
- Copy the file creating the Bank Accounts database using:
  host cp ~panos/1555/recitation/bankdb.sql bankdb.sql
- Setup the Oracle environment and login to Oracle.

Account

acc_no	**Ssn**	**Code**	**open_date**	**Balance**	**close_date**
123	123456789	1234	09/10/08	500	null
124	111222333	1234	10/10/09	1000	null

Loan

Ssn	**Code**	**open_date**	Amount	close_date
111222333	1234	09/15/10	100	null

Bank

Code	Name	Addr
1234	Pitt Bank	111 University St

Customer

Ssn	Name	Phone	Addr	num_accounts
123456789	John	555-535-5263	100 University St	1
111222333	Mary	555-535-3333	20 University St	1

Alert

Alert_date	Balance	Loan

1. Triggers

Notes:

- Triggers are defined on a single table in Oracle.
- With the "*for each row*" option, the trigger is row-level. In this mode, there are 2 special variables **new** and **old** to refer to new and old tuples, respectively.
- If "for each row" is not specified, then the trigger is a statement trigger- i.e., the trigger is fired only once, when the triggering event is met, if the optional trigger constraint is met.
- In the trigger body, **new** and **old** variables are used with a colon preceding it (i.e., **:new** and **:old**), while in the "*when...*" clause they are used **without** the colon.
- The "*referencing...*" clause can be used to assign aliases to the variables **old** and **new**.
- The statements in the trigger body need to be properly ended with "**;**"
- A slash (/) is needed in the new line following the trigger definition to activate the trigger.
- If there is an error in the trigger, when you compile it you only receive a very general, unhelpful error message. To view detailed error messages, type `show errors.`

- With Oracle, in the trigger body avoid select or update the table that the trigger is being defined on. You would get this error "*table ... is mutating, trigger/function may not see it*". (This might be OK in some other DBMSs as long as you avoid indefinite recursion).
- Oracle does not support sub-queries in the when clause.
- If you need to see what Triggers you have defined in your database, execute the following command: SELECT trigger_name FROM user_triggers;

Now do the following:

1. Create a trigger that, when a customer opens new account (s), updates the corresponding num_accounts, to reflect the total number of accounts this customer has.

```
create or replace trigger trig_1
after insert
on account
for each row
begin
update customer
set num_accounts = num_accounts+1
where ssn = :new.ssn;
end;
/
```

2. To test how the trigger works, insert a new account for customer '123456789', then display the num_accounts of that customer. An example tuple may be with values ('333', '123456789', '1234', '10-oct-2010', 300, null).

```
insert into account values('333','123456789','1234', '10-oct-2010', 300, null);
select * from customer where ssn = '123456789';
```

3. Similarly, create a trigger that, upon deleting an account, updates the corresponding num_accounts.

```
create or replace trigger trig_2
after delete
on account
for each row
begin
update customer
set num_accounts = num_accounts -1
where ssn = :old.ssn;
end;
/
```

4. To test the trigger, delete from the account entries for ssn='123456789'. Then check the value of num_accounts.

```
delete from account where ssn='123456789';
select * from customer where ssn = '123456789';
```

SQL/PL is SQL enhanced with control statement like any high level programming languages. Examples include: If-Then-Else, Loops, etc. The assignment operator for values is := , where for tuples is INTO.

5. Create a trigger that upon updating an account's balance, if the new balance is negative then sets the balance to 0 and create a new loan for the negative amount (for this database, assume that this can happen only once per day).

```
create or replace trigger trig_3
before update of balance
on account
for each row
when (new.balance < 0)
begin
insert into loan values (:new.ssn, :new.code, current_date, abs(:new.balance), null);
:new.balance := 0;
end ;
/
```

OR

```
create or replace trigger trig_3
before update of balance
on account
for each row
when (new.balance < 0)
begin
insert into loan values (:new.ssn, :new.code, current_date, abs(:new.balance), null);
select 0 into :new.balance from dual;
end ;
/
```

Note:

a. The DUAL table from the above trigger is a placeholder when no table is needed in the from clause. For more information, please refer to: https://en.wikipedia.org/wiki/DUAL_table
b. You cannot use UPDATE Balance table in the place of ":new.balance := 0;" or "select 0 into :new.balance from dual;" above, because it will cause a mutation trigger error.

6. To test how the trigger works, update the balance of the account '124' to -50, then check the data in the Loan table.

```
update account set balance = -50 where acc_no = 124;
select * from account where acc_no = 124;
select * from loan;
```

7. Create two triggers for Account and Loan tables that upon any changes in the two tables, if the sum of balance amount over all accounts is less than double the sum of loan amount over all loans, create a new alert with current date, total balance amount and total loan amount (for this database, assume that this can happen only once per day).

```
create or replace trigger trig_4_account
after insert or update or delete on account
declare
   totalBalance NUMBER;
   totalLoan NUMBER;
begin
   select sum(balance) into totalBalance from account;
   select sum(amount)  into totalLoan from loan;
   if totalBalance < totalLoan * 2 then
      insert into alert values (current_date, totalBalance, totalLoan);
   end if;
end;
/

create or replace trigger trig_4_loan
after insert or update on loan
declare
   totalBalance NUMBER;
   totalLoan NUMBER;
begin
   select sum(balance) into totalBalance from account;
   select sum(amount)  into totalLoan from loan;
   if totalBalance < totalLoan * 2 then
      insert into alert values (current_date, totalBalance, totalLoan);
   end if;
end;
/
```

8. To test the trigger, update the balance of the account '124' to 50, then check the data in the Alert table.

```
update account set balance = 50 where acc_no = 124;
select * from alert;
```