# CS1555 Recitation 8

**Objective:** To understand how triggers work in PostgreSQL and to practice writing triggers.

Before we start, download and run the script **Bank_DB.sql** from the course website to setup the database. The database instance is shown below:

**Account**

| **acc_no** | **Ssn** | **Code** | **open_date** | **Balance** | **close_date** |
|---|---|---|---|---|---|
| 123 | 123456789 | 1234 | 2008-09-10 | 500 | null |
| 124 | 111222333 | 1234 | 2009-10-10 | 1000 | null |

**Loan**

| **Ssn** | **Code** | **open_date** | Amount | close_date |
|---|---|---|---|---|
| 111222333 | 1234 | 2010-09-15 | 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 |
|---|---|---|
|  |  |  |

## Notes:

- Triggers are defined on a single table in PostgreSQL.
- 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.
- The statements in the trigger function need to be properly ended with "**;**"
- In Oracle, in the trigger body, if you select or update the table that the trigger is being defined on, you would get an error saying "*table ... is mutating, trigger/function may not see it*". This is OK in PostgreSQL as long as you avoid indefinite recursion.
- PL/pgSQL is SQL enhanced with control statement like any high-level programming languages. Examples include: If-Then-Else, Loops, etc.

## Part 1: Triggers

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 function func_1() returns trigger as
$$
begin
  update customer
  set num_accounts = num_accounts + 1
  where ssn = new.ssn;
  return new;
end;
$$ language plpgsql;

drop trigger if exists trig_1 on account;
create trigger trig_1
  after insert
  on account
  for each row
execute procedure func_1();
```

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', '2010-10-10', 300, null).

3. Similarly, create a trigger that, upon deleting an account, updates the corresponding num_accounts. To test the trigger, delete from the account entries for ssn='123456789'. Then check the value of num_accounts.

```
create or replace function func_2() returns trigger as
$$
begin
  update customer
  set num_accounts = num_accounts - 1
  where ssn = old.ssn;
  return new;
end;
$$ language plpgsql;

drop trigger if exists trig_2 on account;
create trigger trig_2
  after delete
  on account
  for each row
execute procedure func_2();
```

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

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 function func_3() returns trigger as
$$
begin
  insert into loan
  values (new.ssn, new.code, current_date, abs(new.balance),
null);
  new.balance := 0;
  return new;
end;
$$ language plpgsql;

drop trigger if exists trig_3 on account;
create trigger trig_3
  before
    update of balance
  on account
  for each row
  when (new.balance < 0)
execute procedure func_3();
```

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

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 function func_4() returns trigger as
$$
declare
  totalBalance numeric(15, 2);
  totalLoan    numeric(15, 2);
begin
  select sum(balance) into totalBalance
  from account;
  select sum(amount) into totalLoan
  from loan;
  if totalBalance < totalLoan * 2 then
    insert into alert
    values (current_date, totalBalance, totalLoan);
  end if;
  return new;
end;
$$ language plpgsql;
```

```
drop trigger if exists trig_4_account on account;
create trigger trig_4_account
  after update or delete
  on account
execute procedure func_4();

drop trigger if exists trig_4_loan on loan;
create trigger trig_4_loan
  after insert or update
  on loan
execute procedure func_4();
```

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