CS1555 / CS2550 Recitation 9 Solution

Objective: To practice Evaluation Modes, Transactions, Procedures and Functions

PART 1: Constraint Evaluation Modes and Transactions

DEFERRED: Withheld for or until a stated time (COMMIT)

a) **Not Deferrable** (default): Every time a database modification statement is executed, the constraints are checked.

b) **Deferrable Initially Immediate**: Every time a database modification statement is executed, the constraints are checked IMMEDIATE. BUT the constraints can be deferred on demand, when needed

c) **Deferrable Initially Deferred**: The constraints are check just BEFORE each transaction commits.

1. Create the below tables with the specified evaluation modes:

→ NotDef (ssn numeric) with **Not Deferrable** setting for the primary key constraint.
→ DefImm (ssn numeric) with **Deferrable Initially Immediate** setting for the primary key constraint.
→ DefDef (ssn numeric) with **Deferrable Initially Deferred** setting for primary key constraint.

```
drop table if exists NOTDEF;
create table NOTDEF
(
  ssn numeric,
  constraint pk_ssn_1 PRIMARY KEY (ssn)
);

drop table if exists DEFIMM;
create table DEFIMM
(
  ssn numeric,
  constraint pk_ssn_2 PRIMARY KEY (ssn) Deferrable Initially Immediate
);

drop table if exists DEFDEF;
create table DEFDEF
(
  ssn numeric,
  constraint pk_ssn_3 PRIMARY KEY (ssn) Deferrable Initially Deferred
);
```

2. For each table created in Question 1, run the SQL statements below and pay attention to if and when you encounter an error.

a) start transaction read write;
b) insert value 1234
c) insert value 1234
d) commit;

Note: Unless START TRANSACTION is issued, PostgreSQL implicitly issues a COMMIT after each SQL statement.

For Table **NotDef**, the error occurred after statement c), because the primary key constraint is not deferrable.

For Table **DefImm**, the error occurred after statement c), because although the primary key constraint is deferrable, but it was not deferred during the transaction.

For Table **DefDef**, the error occurred after statement d), because the primary key constraint was deferred until the commit time.

3. Reset the database. Then for each table created in Question 1, run the SQL statements below. Do you see any difference in the error reporting compared to Question 2?

a) start transaction read write;
b) set constraints all deferred;
c) insert value 1234
d) insert value 1234
e) commit;

For Table **NotDef**, the error occurred after statement c), because the primary key constraint is not deferrable.

For Table **DefImm**, the error occurred after statement d), because the primary key constraint is deferrable and was deferred at the beginning of the transaction.

For Table **DefDef**, the error occurred after statement d), because the primary key constraint is deferrable and was deferred at the beginning of the transaction.

PART 2: Procedures and Functions

Before we start, run **Bank_DB.sql** downloadable from the course website to setup the database.

1. Create a stored procedure **transfer_fund** that, given a from_account, a to_account, and an amount, transfer the specified amount from from_account to to_account if the balance of the from_account is sufficient.

```
-- Before PostgreSQL 11, procedures are realized using functions.
create or replace function transfer_fund(from_account varchar,
                                         to_account varchar,
                                         amount numeric)
  returns void
as
$$
declare
  from_account_balance numeric;
begin
  select balance into from_account_balance
  from account
  where acc_no = from_account;
  if from_account_balance > amount then
    update account
    set balance = balance - amount
    where acc_no = from_account;
    update account
    set balance = balance + amount
    where acc_no = to_account;
  else
    raise notice 'balance is too low';
  end if;
end;
$$ language plpgsql;
```

2. Call the stored procedure to transfer $100 from account 124 to 123.

```
-- There are 2 ways to call a procedure:
-- a) Directly outside a PL/pgSQL block
select transfer_fund('124', '123', 100);

-- b) Inside a PL/pgSQL block (begin ... end;),
-- including in the body of a trigger /stored procedure/ function
do $$
  begin
    perform transfer_fund('124', '123', 100);
  end $$;
```

3. Create a function **compute_balance** that, given a specific ssn, calculate the total balance of the customer (the sum of total account balances less the loan amounts)

```
create or replace function compute_balance(customer_ssn varchar)
  returns numeric
as
$$
declare
  final_balance        numeric;
  total_account_balance numeric;
  total_loan           numeric;
begin
  select coalesce(sum(balance), 0) into total_account_balance
  from account
  where ssn = customer_ssn;
  select coalesce(sum(amount), 0) into total_loan
  from loan
  where ssn = customer_ssn;
  final_balance := total_account_balance - total_loan;
  return
    final_balance;
end;
$$ language plpgsql;

-- NOTE: because a customer might have no account or no loan,
-- resulting in sum(balance) is null or sum(amount) is null and
consequently,
-- the final balance is also null. To avoid this, we use the coalesce
function.
-- The coalesce function returns the first argument that is not null.
```

4. Use the function created, write a query to print the list of customers together with their total balance.

```
select ssn, compute_balance(ssn)
from customer;
```