CS1555 Recitation 8

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. Use the create statement with the deferred statement mentioned below

```
CREATE TABLE notdef (
     ssn integer,
     CONSTRAINT pk_ssn_1 PRIMARY KEY(ssn)
);

CREATE TABLE defimm (
     ssn integer,
     CONSTRAINT pk_ssn_2 PRIMARY KEY(ssn) DEFERRABLE INITIALLY
IMMEDIATE
);

CREATE TABLE defdef (
     ssn integer,
     CONSTRAINT pk_ssn_3 PRIMARY KEY(ssn) DEFERRABLE INITIALLY
DEFERRED
);
```

2. For each table created above, run the SQL statements and mention if and when you encounter an error.

```
INSERT INTO notdef VALUES (1234);
INSERT INTO notdef VALUES (1234);-- primary key constraint violation. The values should be unique.
```

3. Now, add <`SET CONSTRAINTS` *`<constraint_name>`* `DEFERRED`> for the constraint set in table defimm; Run the previous insert again. Do you see any difference?

NOTE: remember that we already have value 1234 in the table because of the previous insert statements.

```
BEGIN;
SET CONSTRAINTS pk_ssn_1 DEFERRED;
INSERT INTO defimm VALUES (1234);-- primary key constraint violation. The values should be unique.
COMMIT; -- primary key constraint violation. The values should be unique.
```

4. For each table created above, run the SQL statements and show the table content after the inserts.

a) set constraints all deferred
b) insert value 1235
c) insert value 1235
d) commit;

Notdef:

```
BEGIN;
SET CONSTRAINTS ALL DEFERRED;
INSERT INTO notdef VALUES (1235);
INSERT INTO notdef VALUES (1235);
COMMIT; -- No rows inserted. Error in second insert and transaction is rolled back.
```

Defimm:

```
BEGIN;
SET CONSTRAINTS ALL DEFERRED;
INSERT INTO defimm VALUES (1235);
INSERT INTO defimm VALUES (1235);
COMMIT; -- No rows inserted. Error at commit and transaction is rolled back.
```

Defdef:

```
BEGIN;
SET CONSTRAINTS ALL DEFERRED;
INSERT INTO defdef VALUES (1235);
INSERT INTO defdef VALUES (1235);
COMMIT;-- No row was inserted. Same reason as for the defimm table.
```

PART 2: Procedures and Functions

Before we start:

- Download the SQL script bank_db.sql from the course website, in the recitation page.

---

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.

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

3. Create a function that returns true if a customer can pay their loan or false when their balance is less than their loan.

4. Use the function created using the ssn 123456789.

5. Create a function that returns a trigger upon inserting a tuple into the table customer, it makes sure that the name is in upper cases.

6. Insert the following tuple, and then check the value after insertion.