**CS1555 Recitation 8 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)*

1. **Not Deferrable** (*default*): *every time a database modification statement is executed, the constraints are checked.*
2. **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*
3. **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\_2 DEFERRED;  
INSERT INTO defimm VALUES (1234);  
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.

CREATE OR REPLACE PROCEDURE *transfer\_funds*(from\_account varchar, to\_account varchar, amount integer)  
LANGUAGE plpgsql  
AS $$  
DECLARE  
 from\_account\_balance numeric(15, 3);  
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 'ERROR: balance is too low';  
 END IF;  
END;  
$$

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

BEGIN;  
SET CONSTRAINTS ALL DEFERRED;  
CALL *transfer\_funds*('124', '123', 100);  
COMMIT;

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

CREATE OR REPLACE FUNCTION *can\_pay\_loan*(customer\_ssn char(9))  
RETURNS BOOLEAN

AS $$  
DECLARE  
 can\_pay BOOLEAN := false;  
BEGIN  
 SELECT (a.ssn = $1) INTO can\_pay  
 FROM account a LEFT JOIN loan l ON a.ssn = l.ssn  
 WHERE a.ssn = $1 AND a.balance>l.amount OR l.ssn IS null;  
  
 RETURN can\_pay;  
END  
$$ LANGUAGE plpgsql;

4. Use the function created using the ssn 123456789.

SELECT can\_pay\_loan('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.

-- the function

CREATE OR REPLACE FUNCTION before\_insert\_on\_customer()  
RETURNS TRIGGER

AS $$  
BEGIN  
 new.name := upper(new.name);  
 RETURN new;  
END  
$$ LANGUAGE plpgsql;

-- the trigger

CREATE TRIGGER before\_insert\_on\_customer  
before insert on customer  
FOR EACH ROW EXECUTE PROCEDURE *before\_insert\_on\_customer*();

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

INSERT INTO customer VALUES

('123444444', 'foo bar', '123-123-1234', '0 nothing st', 1);