CS1555 / CS2550 Recitation 11

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

PART 0: Review of HW6 Questions

1.d) For each year, list the most read message ID(s).

2.d) Write a trigger called **CreateConversation** that will create an new conversation entry whenever a message is added with a conversation ID that does not yet exist.

PART 1: Constraint Evaluation Modes and Transactions

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

a) <u>**Not Deferrable**</u> (*default*): *every time a database modification statement is executed, the constraints are checked.*

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

c) <u>**Deferrable Initially Deferred**</u>: the constraints are check just BEFORE each transaction commits.

1. Use the create statement with the deferred statement mentioned below

→ NotDef (<u>ssn</u> number) with **Not Deferrable** constrain for the primary key.
→ DefImm (<u>ssn</u> number) with **Deferrable Initially Immediate** setting for the primary key constraint.
→ DefDef (<u>ssn</u> number) with **Deferrable Initially Deferred** setting for primary key constraint.

2. For each table created above, run the SQL statements and mention if and when you encounter an error.
a) insert value 1234
b) insert value 1234
c) commit;

3. Run: < set constraint *constraint_name* deferred > for the constraint set in table DefImm; Run the previous insert again. Do you see any difference?

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;

PART 2: Procedures and Functions

Before we start:
- Copy and run the file creating the Bank Accounts database using:

```
host cp ~panos/1555/recitation/bankdb.sql bankdb.sql
@bankdb
```

---

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

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