Laboratorio di Basi di Dati 2
III esercitazione

Operazioni preliminari

Collegarsi al Server (IP `130.251.61.30 (webdev.disi.unige.it)`), utilizzando la login corrispondente al vostro utente: se il vostro utente è il numero `XX`, la login è `gruppoXX`, la password è `gruppoXX` (da cambiare al primo accesso). Accedete la base di dati di vostra proprietà ed accedere alla vostra base di dati (ad esempio: `db_bd2user_n`).

ATTENZIONE: la prima volta che vi connette al server dovete registrare il server stesso (seguite le istruzioni presenti nella guida allo svolgimento di questa esercitazione).

Punto 1: Creazione base di dati, caricamento dati, specifica ed esecuzione transazioni da `pgadmin III`.

1. *Creazione base di dati* Considerare la seguente semplice base di dati relativa ad una banca e costituita unicamente da due relazioni:

   `Account(number, branch`$_o{}^{Branch}$`, balance)`
   `Branch(branchID, bbalance)`

   La prima registra informazioni sui conti correnti, le relative filiali e saldi, la seconda memorizza il saldo di ogni filiale. Creare in uno schema opportuno della vostra base di dati le due tabelle, specificando opportuni vincoli di chiave e chiave esterna.

2. *Transazioni SQL.* Iniziamo ad inserire solo alcuni conti correnti. Come primo inserimento, inserire una tupla nella relazione con

   ```
   INSERT INTO Account VALUES (0,NULL,100);
   ```

   e una con

   ```
   INSERT INTO Account VALUES (5,NULL,0);.
   ```

   Considerare ora la seguente sequenza di inserimenti

   ```
   INSERT INTO Account VALUES (1,NULL,0);
   INSERT INTO Account VALUES (2,NULL,0);
   INSERT INTO Account VALUES (3,NULL,0);
   INSERT INTO Account VALUES (4,NULL,0);
   INSERT INTO Account VALUES (5,NULL,0);
   ```

   Osservare i messaggi restituiti nel caso in cui ciascun comando di inserimento venga eseguito singolarmente e nel caso in cui tutti i comandi vengano eseguiti in sequenza, nell'ambito della stessa esecuzione. Cosa cambia? Quali tuple sono state inserite nella relazione nei due casi? Perché? Riuscite a spiegare quali e quante transazioni avete eseguito nei due casi? Riscrivere comandi SQL utilizzando i comando espliciti di inizio transazione e commit (`BEGIN;` e `COMMIT;` in PostgreSQL). Provare a sostituire `COMMIT` con `ROLLBACK` e analizzare il risultato.

3. *Transazioni SQL e uso di funzioni* Cancellare tutti i dati inseriti nella relazione `Account`. Provare ora a utilizzare la funzione `inserisci`, che inserisce nella tabella `Account` `num` conti correnti, definita come segue

   ```
   CREATE FUNCTION inserisci(num integer) RETURNS void AS $$
   BEGIN
   FOR i IN 1..num LOOP
       BEGIN
       RAISE NOTICE 'Inserito conto %', i;  -- stampa i
       INSERT INTO Account VALUES (i,NULL,0);
       END;
   END LOOP;
   END;
   $$ LANGUAGE plpgsql;
   ```

Per invocare la funzione utilizzare `SELECT * FROM inserisci(...)`. Osservare i messaggi restituiti e le tuple effettivamente inserite nella relazione. Provare a fornire una giustificazione ai risultati ottenuti: quale comportamento transazionale viene adottato?

4. *Transazioni con savepoint* Cancellare i conti inseriti nella tabella `Account`. Provare a creare una transazione (senza uso di funzioni) che per prima cosa inserisce una tupla della tabella `Branch`, a cui assegnare tutti i conti. Quindi inserire i conti con `number` uguale a `1,2,3,4,5` nella tabella `Account`, assegnati alla filiale appena inserita. Utilizzando i savepoint, fare in modo che, prima dell'inserimento del conto con `number` uguale a `5`, vengano disfatti tutti gli inserimenti dei conti effettuati in precedenza (in Postgres `SAVEPOINT <label>, ROLLBACK TO <label>`). Commentare i risultati ottenuti.
Attenzione: in PostgreSQL, i savepoint non possono essere utilizzati all'interno del corpo delle funzioni e nei blocchi di gestione delle eccezioni (che di fatto vengono implementati con un meccanismo simile ai savepoint). L'esercizio é volutamente molto semplice per limitazioni dell'applicazione client `pgadmin III`.

5. *Transazioni concorrenti in* `pgadmin III`*: isolamento.* Simuliamo l'esecuzione di due transazioni concorrenti aprendo due finestre SQL da `pgadmin III`. Useremo ciascuna finestra per scrivere il codice relativo ad una certa transazione. Cancellare inizialmente tutte le tuple contenute nella tabella `Account` e aggiungere una tupla per i conti 1,2,3,4,5 con valori di `balance` pari a `10,20,30,40,50`, rispettivamente. L'obiettivo di questo esercizio é prendere confidenza con i livelli di isolamento. Per ogni punto seguente, descrivere il comportamento e motivarlo secondo le anomalie e i livelli di isolamento visti a lezione. Valutare se e come cambierebbe il comportamento del sistema variando il livello di isolamento (livello di default: `READ COMMITTED`).

   - *Conflitti letture/scritture*

```
T1                                          T2
start transaction;
insert into account
values (6,NULL, 90);
                                    start transaction;
                                    insert into account
                                    values (6,NULL, 110);
commit;
                                    commit;
```

   - *Letture irripetibili*

```
T1                                          T2
start transaction read only;
set transaction isolation level
read committed;
select * from account
where number = 1;
                                    start transaction;
                                    set transaction isolation level
                                    read committed;
                                    update account
                                    set balance = 90
                                    where number = 1;
                                    commit;
select * from account
where number = 1;
commit;
```

   - *Aggiornamento fantasma*

```
T1                                          T2
start transaction read only;
set transaction isolation level
read committed;
select balance from account
where number = 2;
select balance from account
where number = 3;
                                            start transaction;
                                            set transaction isolation level
                                            read committed;
                                            update account
                                            set balance = balance - 10
                                            where number = 1;
                                            update account
                                            set balance = balance + 10
                                            where number = 2;
                                            commit;
select balance from account
where number = 1; -- 20
commit;
```

- *Aggiornamento concorrente*

```
T1                                          T2

start transaction;
set transaction isolation level
repeatable read;
update account
set balance = balance - 10
where number = 1;
                                            start transaction;
                                            set transaction isolation level
                                            repeatable read;
                                            update account
                                            set balance = balance + 10
                                            where number = 1;
commit;
                                            commit;
```

- *Inserimento fantasma*

```
T1                                          T2

start transaction read only;
select sum(balance) from account;
                                            start transaction;
                                            insert into account values (10,NULL, 60);
select sum(balance) from account;
                                            commit;
select sum(balance) from account;
commit;
```