Esercitazione 07

Transazioni

Si considerino le tabelle create nell'esercitazione 1. Per ogni esercizio scrivere il dettaglio delle transazioni richieste e commentare il livello d'isolamento scelto per garantire che le operazioni nelle transazioni siano eseguite in modo corretto.

Esercizio 1

Si assume che la tabella Museo possa essere aggiornata da applicazioni diverse, non sincronizzate fra loro. Scrivere una transazione che aggiunga un museo e dimostrare cosa succede se due applicazioni aggiungono lo stesso museo nello stesso istante usando lo schema della transazione proposta.

Soluzione

L'inserimento di un museo con dati non determinati da altre tabelle è un semplice INSERT senza dipendenze interne. Quindi, in linea generale, non è necessario attivare nessuna transazione esplicita. Infatti, scrivere, ad esempio,

```
BEGIN ;
INSERT INTO Museo( id, nome, citta, prezzo, prezzoRidotto ) VALUES
(5, 'Museo di Verona', 'Verona', 40.00 , 15.00);
COMMIT;
```

è equivalente a scrivere

```
INSERT INTO Museo( id, nome, citta, prezzo, prezzoRidotto ) VALUES
(5, 'Museo di Verona', 'Verona', 40.00 , 15.00);
```

Se due istruzioni INSERT dello stesso museo sono date "contemporaneamente", in realtà una delle due viene eseguita prima dell'altra. Di conseguenza, la seconda INSERT fallirebbe perché il valore della chiave è duplicato:

```
INSERT INTO Museo( id, nome, citta, prezzo, prezzoRidotto ) VALUES
(5, 'Museo di Verona', 'Verona', 50.00, 15.00) ;

ERRORE : un valore chiave duplicato viola il vincolo univoco
"musei_pkey"
DETTAGLI: La chiave (id) =(5) esiste già.
```

Esercizio 2

Si assuma che una transazione deve visualizzare i prezzi dei musei di Verona che hanno parte decimale diversa da 0 e, poi, aggiornare tali prezzi del 10% arrotondando alla seconda cifra decimale. L'altra transazione (concorrente) deve aggiornare il prezzo dei musei di Verona aumentandoli del 10% e arrotondando alla seconda cifra decimale.

Soluzione

Usando il livello d'isolamento di default di PostgreSQL (READ COMMITTED) la prima transazione potrebbe aggiornare prezzi che non ha visualizzato perché, nel frattempo, l'altra transazione ha fatto COMMIT. È necessario richiedere un livello d'isolamento REPEATABLE READ. Se si verifica il caso che il COMMIT della seconda transazione avviene tra il SELECT e UPDATE della prima transazione, la prima transazione termina all'esecuzione dell'UPDATE con l'errore: ERRORE: serializzazione dell'accesso fallita a causa di modifiche concorrenti.

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT * FROM Museo
WHERE prezzo <> ceil ( prezzo ) AND citta ILIKE 'Verona';

UPDATE Museo SET prezzo = round( prezzo * 1.10, 2 )
WHERE prezzo <> ceil ( prezzo ) AND citta ILIKE 'Verona' ;

COMMIT;
```

```
BEGIN;
UPDATE Museo SET prezzo = round ( prezzo * 1.10, 2 )
WHERE citta ILIKE 'Verona';
COMMIT;
```

Esercizio 3

In una transazione si deve inserire una nuova mostra al museo di Castelvecchio di Verona con prezzo d'ingresso a 40 euro e prezzo ridotto a 20. Nell'altra transazione (concorrente) si deve calcolare il prezzo medio delle mostre di Verona prima considerando solo i prezzi ordinari e, in un'interrogazione separata, considerando solo i prezzi ridotti.

Soluzione

Se viene utilizzato il livello di default le SELECT della seconda transazione possono selezionare due insiemi di mostre diversi tra loro. È necessario quindi usare isolation level SERIALIZABLE:

```
INSERT INTO Mostra( titolo, inizio, fine, museo,
                    citta, prezzoIntero, prezzoRidotto )
VALUES( 'Mostra Scaligera', '2017-02-12', '2017-11-30',
        'CastelVecchio', 'Verona', 40.00 , 20.00) ;

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT AVG ( prezzoIntero ) FROM Mostra WHERE citta ILIKE 'Verona';
SELECT AVG ( prezzoRidotto ) FROM Mostra WHERE citta ILIKE 'Verona';
COMMIT;
```

Esercizio 4

In una transazione si deve aumentare il prezzo intero di tutte le mostre di Verona del 10% mentre, nell'altra, si devono ridurre i prezzi ridotti di tutte le mostre del 5%. In entrambi i casi, l'importo finale si deve arrotondare alla seconda cifra decimale.

Soluzione

Banalmente, si possono eseguire i due update senza bisogno di transazioni particolare. L'esito dei due update è indipendende dall'ordine con cui vengono eseguiti.

```
UPDATE Mostra SET prezzoIntero = round ( prezzoIntero * 1.10 , 2)
WHERE citta ILIKE 'Verona';

UPDATE Mostra SET prezzoRidotto = round ( prezzoRidotto * 0.95 , 2)
WHERE citta ILIKE 'Verona';
```

Esercizio 5

In una transazione, calcolare la media dei prezzi dei musei di Vicenza ed aggiungere un nuovo museo a Verona ('Museo moderno') con prezzo uguale alla media calcolata. In un'altra transazione calcolare la media dei prezzi dei musei di Verona e aggiungere un nuovo museo a Vicenza con prezzo uguale alla media calcolata sui musei di Verona.

Si segnala che:

1. Con SELECT si possono anche creare colonne con valori costanti.

Esempio: SELECT 'Museo Moderno', 'Verona', ecc FROM ...

2. INSERT accetta di inserire anche risultati ottenuti da SELECT interne.

Esempio: INSERT INTO Museo (nome, citta, ...) SELECT 'Museo Moderno', 'Verona', ... FROM...

Soluzione

È necessario il SERIALIZABLE perché un insert aggiunge una riga che deve essere usata dall'altro insert. Quindi il risultato finale dipende dall'ordine con cui vengono inseriti. Se si ponesse REPEATABLE READ, questa dipendenza si perderebbe e quindi il risultato finale sarebbe errato.

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO Museo( nome, citta, prezzo )
    SELECT 'Museo Preistorico di Verona', 'Verona', AVG( prezzo )
    FROM Museo WHERE citta ILIKE 'Vicenza';
COMMIT;


BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO Musei( nome, citta, prezzo )
     SELECT 'Museo provinciale di Vicenza', 'Vicenza', AVG ( prezzo )
     FROM Musei WHERE citta ILIKE 'Verona';
COMMIT;
```