Basi di Dati
Modulo Laboratorio

Lezione 8: Introduzione al controllo di concorrenza in SQL

DR. SARA MIGLIORINI

Controllo concorrenza – Transazioni

- Una transazione SQL è una sequenza di istruzioni SQL che può essere eseguita in concorrenza con altre transazioni in modo isolato.
- Se si implementasse un isolamento completo, il grado di parallelismo sarebbe limitato.
- Si accettano quindi diversi livelli di isolamento per favorire un maggior grado di parallelismo.
- Nel modulo di teoria è stato spiegato che diversi livelli di isolamento possono determinare delle anomalie di esecuzioni:
  - perdita di aggiornamento (lost update)
  - lettura sporca (dirty read)
  - letture inconsistenti (non-repeatable read)
  - aggiornamento fantasma (ghost update)
  - inserimento fantasma (phantom update)

Controllo concorrenza – Transazioni

- In PostgreSQL è prevista un'altra anomalia che nel testo di teoria non è considerata: mancata serializzazione (serialization anomaly) :
  - Il risultato di un gruppo di transazioni (nessun abort) è inconsistente con tutti gli ordini di esecuzioni seriali delle stesse.
- In questa lezione si introducono i livelli di isolamento implementati in PostgreSQL e come si possono attivare.

Controllo di concorrenza – Transazioni

- In PostgreSQL una transazione inizia con il comando BEGIN e termina o con il comando COMMIT (per confermare tutte le istruzioni della transazione) o con il comando ROLLBACK (per annullare tutte le operazioni).

Transazione passaggio importo da un conto ad un altro

```
BEGIN;
UPDATE accounts SET balance = balance -100.0 WHERE name ='Alice';
UPDATE accounts SET balance = balance +100.0 WHERE name ='Bob';
COMMIT;
```

Controllo di concorrenza – Transazioni

Transazione passaggio importo da un conto ad un altro

```
BEGIN;
UPDATE accounts SET balance = balance -100.0 WHERE name ='Alice';
UPDATE accounts SET balance = balance +100.0 WHERE name ='Bob';
ABORT;
```

Alice ha ancora l'importo originale nel suo conto!

Controllo concorrenza – Introduzione

- PostgreSQL mantiene la consistenza dei dati usando un modello multi-versione (Multiversion Concurrency Control (MVCC)):
- MVCC è una metodologia più versatile del locking a due fasi (stretto) tipica dei DBMS tradizionali.
- In generale, le transizioni in PostgreSQL sono gestite come:
  - Ciascuna transazione vede un'istantanea della base di dati.
  - Le letture su questa istantanea sono sempre possibili e non sono mai bloccate anche se ci sono altre transazioni che stanno modificando la base di dati.
  - Le scritture possono essere sospese. Una scrittura è sospesa quando, in parallelo, un'altra transazione (non ancora chiusa) ha modificato la sorgente dei dati che si vuole aggiornare.
  - Al COMMIT, il sistema registra sempre l'istantanea aggiornata come nuova base di dati.
- In questo modo ci sono meno conflitti e una performance più ragionevole in ambito multiutente.

Controllo di concorrenza – Introduzione

Esempio di sessioni concorrenti

Si consideri la tabella Studente con il seguente contenuto:

SELECT matricola, cognome, città FROM Studente;

```
Matricola   | Cognome  | Città
------------+----------+----------
IN0001      | Rossi    | Verona
IN0002      | Paoli    | Padova
IN0003      | Bianchi  | VERONA
IN0004      | Rossi    | Verona
IN0005      | Verdi    | Va
IN0006      | Nocasa   |
```

Controllo di concorrenza – Introduzione

Esempio di sessioni concorrenti			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	SELECT * FROM STUDENTE; ... *-- Tabella iniziale*		
3	UPDATE Studente SET città = 'Vicenza' WHERE matricola = 'IN0006'; UPDATE 1	SELECT città FROM STUDENTE 3 WHERE matricola = 'IN0006'; città --------	3
4	SELECT matricola, cognome, città FROM STUDENTE WHERE matricola = 'IN0006'; matricola | cognome | città ----------+---------+------- IN0006 | Nocasa | Vicenza	UPDATE Studente SET città = 'VI' WHERE matricola = 'IN0006'; *-- esecuzione sospesa. La tabella è bloccata dall'istante 3 in T1*	4
5	COMMIT;		5
6	continua...	UPDATE 1 *-- il COMMIT di T1 ha sbloccato!*	6

I numeri ai lati rappresentano gli istanti in cui si eseguono le istruzioni.

Controllo di concorrenza – Introduzione

Esempio di sessioni concorrenti			
Transazione 1		Transazione 2	
7	SELECT matricola, cognome,città FROM STUDENTE WHERE matricola = 'IN0006'; matricola | cognome | città ----------+---------+--------- IN0006 | Nocasa | Vicenza		7
8		COMMIT;	8
9	SELECT matricola, cognome, città FROM STUDENTE WHERE matricola = 'IN0006'; matricola | cognome | città ----------+---------+------- IN0006 | Nocasa | VI	SELECT matricola, cognome, città FROM STUDENTE WHERE matricola = 'IN0006'; matricola | cognome | città ----------+---------+------- IN0006 | Nocasa | VI	9

I numeri ai lati rappresentano gli istanti in cui si eseguono le istruzioni.

Livelli di isolamento

- I 4 livelli di isolamento offerti da PostgreSQL 11.3 sono, in ordine decrescente di isolamento:

1. Serializable: Garantisce che un'intera transazione è eseguita in un qualche ordine sequenziale rispetto ad altre transazioni: completo isolamento da transazioni concorrenti.
   - Anomalie possibili: nessuna!
2. Repeatable Read: Garantisce che i dati letti durante la transazione non cambieranno a causa di altre transazioni: rifacendo la lettura dei medesimi dati, si ottengono sempre gli stessi.
   - Questo livello è più restrittivo del livello standard 'Repeatable Read'.
   - Anomalie possibili: mancata serializzazione (rispetto a quanto visto a teoria questo livello di isolamento risolve anche l'anomalia di inserimento fantasma).

Livelli di isolamento

3. Read Committed: Garantisce che qualsiasi SELECT di una transazione veda solo i dati confermati (COMMITTED) **prima che la SELECT inizi**.
   - È il livello di isolamento di default usato da PostgreSQL.
   - Anomalie possibili: lettura inconsistente, aggiornamento fantasma, inserimento fantasma e mancata serializzazione.
4. Read Uncommitted: In PostgreSQL (già da 11.3) è implementato come Read Committed. Quindi NON esiste questo livello di isolamento.

- Il cambio di livello di isolamento si effettua con SET TRANSACTION appena dopo il BEGIN della transazione.

Controllo concorrenza - SET TRANSACTION

Sintassi

```
SET TRANSACTION transaction_mode;
dove transaction_mode è uno tra:
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED |
                  READ UNCOMMITTED }
    | READ WRITE | READ ONLY
```

Modo standard

```
BEGIN ;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
...
COMMIT;
```

Modo compatto

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
...
COMMIT ;
```

Livello di isolamento Read Committed

- È il livello di default, quindi è sufficiente BEGIN;
- SELECT vede solo i dati registrati (COMMITTED) in altre transazioni e quelli modificati da eventuali comandi precedenti nella medesima transazione.
- UPDATE, DELETE vedono i dati come SELECT. Inoltre, se i dati che devono essere aggiornati sono stati modificati ma non registrati in transazioni concorrenti, il comando deve:

  1. attendere il COMMIT o ROLLBACK della transazione dove è stato fatto il cambio e
  2. riesaminare le righe selezionate per verificare che soddisfano ancora i criteri del comando.

- Con 'Read Committed' possono verificarsi le anomalie: lettura inconsistente , aggiornamento fantasma , inserimento fantasma e mancata serializzazione.

Livello di isolamento Read Committed

Caso Lettura inconsistente			
Si assuma che esista la tabella W:			
Si considerino le seguenti transazioni concorrenti			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	SELECT hit FROM W WHERE hit <=9; 9		2
3		UPDATE W SET hit =10 WHERE hit = 9;	3
4		COMMIT ;	
5	SELECT hit FROM W WHERE hit <=9; __		
6	COMMIT;		

id (serial)	hit (integer)
1	9
2	10

Livello di isolamento Read Committed

Caso Lettura inconsistente			
Si assuma che esista la tabella W:			
Si considerino le seguenti transazioni concorrenti			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	SELECT hit FROM W WHERE hit <=9; 9		2
3			3
4		COMMIT ;	
5	SELECT hit FROM W WHERE hit <=9; __		
6	COMMIT;		

T1 ha due letture diverse della stessa tabella senza averla modificata.

Livello di isolamento Read Committed

id (serial)	hit (integer)
1	9
2	10

Caso Aggiornamento Fantasma			
Si assuma la tabella W e che esista un vincolo di programma SUM(hit)<=20.			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	DO $$ DECLARE n INTEGER;		2
3	BEGIN		3
4	n := (SELECT hit FROM W WHERE id =1); *-- n=9*		4
5		UPDATE W SET hit = 8 WHERE id =1;	5
6		UPDATE W SET hit = 12 WHERE id =2;	6
7		COMMIT;	7
8	n := n + (SELECT hit FROM W WHERE id =2); *-- n=12*		8

9	IF n > 20 THEN
	RAISE 'La somma è %', n;
	END IF;
			9

Livello di isolamento Read Committed

Caso Aggiornamento Fantasma			
Si assuma la tabella W e che esista un vincolo di programma SUM(hit)<=20.			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	DO $$ DECLARE n INTEGER;		2
3	BEGIN		3
4	n := (SELECT hit FROM W WHERE id =1); *-- n=9*		4
5			5
6		=2;	6
7		COMMIT;	7
8	n := n + (SELECT hit FROM W WHERE id =2); *-- n=12*		8

9	IF n > 20 THEN
	RAISE 'La somma è %', n;
	END IF;
			9

La base di dati soddisfa sempre il vincolo, ma per T1 no. T1 ha perso un aggiornamento!

Livello di isolamento Read Committed

id (serial)	hit (integer)
1	9
2	10

Read Committed: caso più articolato			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	…		2
3	UPDATE W SET hit=hit +1; UPDATE 2		3
4	…	DELETE FROM W WHERE hit =10; *-- DELETE in attesa di T1*	4
5	…		5
6	COMMIT ;	DELETE 0;	6
7		COMMIT;	7

Livello di isolamento Read Committed

Read Committed: caso più articolato			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN;	1
2	…		2
3	UPD UPD		3
4	…		4
5	…		5
6	COMMIT ;	DELETE 0;	6
7		COMMIT;	7

DELETE non cancella la tupla id=2 selezionata al passo 4!
La tupla, infatti, viene aggiornata da UPDATE; UPDATE blocca DELETE fino al COMMIT.
DELETE, quindi, deve riesaminare (**solo**) la tupla id=2.
Al riesame, la clausola WHERE hit=10 non vale più. Tupla non cancellata.

Livello di isolamento Repeatable Read

- Differisce da 'Read Committed' per il fatto che i comandi di una transazione vedono sempre gli stessi dati. PostgreSQL associa alla transizione una istantanea della base di dati all'esecuzione del suo primo comando.
- 'Repeatable Read' in PostgreSQL è più stringente di quanto richiesto dallo standard SQL.
- Due SELECT identiche successive vedono sempre gli stessi dati.
- UPDATE e DELETE vedono i dati come SELECT. Se i dati che devono essere aggiornati sono stati modificati ma non registrati in transazioni concorrenti, il comando deve attendere il COMMIT/ROLLBACK della transazione dove è stato fatto il cambio e riesaminare le righe selezionate.
- In caso di ROLLBACK, UPDATE e DELETE possono procedere. In caso di COMMIT, i dati sono cambiati, quindi UPDATE e DELETE vengono bloccati con l'errore ERROR: «could NOT serialize access due to concurrent UPDATE».

Livello di isolamento Repeatable Read

'Repeatable Read': più restrittivo di 'Committed Read'			
Si assuma che esista la tabella W:			
Si considerino le seguenti transazioni concorrenti			
Transazione 1		Transazione 2	
1	BEGIN;	BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;	1
2	UPDATE W SET hit=hit +1; UPDATE 2		2
3		DELETE FROM W WHERE hit =10;	3
4		COMMIT ;	4
5	COMMIT;	ERROR : could NOT serialize access due to concurrent UPDATE	5

id (serial)	hit (integer)
1	9
2	10

Livello di isolamento Repeatable Read

- Cattura tutte anomalie tranne mancata serializzazione.
- Se si usa questo livello, si deve prevedere la possibilità di transazioni abortite per aggiornamenti concorrenti.

Livello di isolamento Repeatable Read

Repeatable Read: anomalia 'mancata serializzazione'			
Repeatable Read ammette l'anomalia mancata serializzazione.			
Transazione 1		Transazione 2	
1	BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;	BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;	1
2	INSERT INTO W(hit) SELECT MAX (hit)+1 FROM W; INSERT 0 1		2
3		INSERT INTO W(hit) SELECT MAX (hit)+1 FROM W; INSERT 0 1	3
4	COMMIT;	COMMIT;	4

id (serial)	hit (integer)
1	9
2	10

La tabella contiene due valori 11 e non 11 e 12. Il risultato ottenuto non è consistente con nessun ordine di esecuzione delle due transizioni.

Livello di isolamento Serializable

- È il più restrittivo: nessuna anomalia è permessa.
- Se si usa questo livello, si deve prevedere la possibilità di transazioni abortite per aggiornamenti concorrenti (come nel caso di ‘Repeatable Read’).

Livello di isolamento Serializable

Serializable: anomalie di serializzazione non sono possibili!			
Transazione 1		Transazione 2	
1	BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;	BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;	1
2	INSERT INTO W(hit) SELECT MAX (hit)+1 FROM W; INSERT 0 1		2
3	COMMIT;	INSERT INTO W(hit) SELECT MAX (hit)+1 FROM W; INSERT 0 1	3
4		COMMIT; ERROR : could NOT serialize access due to READ/WRITE dependencies among Transactions DETAIL : Reason code : Canceled ON identification AS a pivot, during COMMIT attempt. HINT: The TRANSACTION might succeed if retried.	4

Livello di isolamento Serializable

- Un metodo pratico per decidere se è necessario il livello serializable è: verificare se la selezione delle righe di un UPDATE/INSERT/DELETE in una transizione può essere modificata in una transizione concorrente (che può aggiungere/togliere o modificare le righe di interesse).

Livello di isolamento Serializable

Serializable: avvertenze

- L'uso di transazioni 'Serializable' rende più semplice lo sviluppo di programmi SQL: è sufficiente dimostrare che una transazione, da sola, determina sempre uno stato corretto per avere la garanzia che essa continuerà ad essere corretta anche in ambiente concorrente dichiarandola 'Serializable'.
- Dall'altra parte però dichiarare tutte le transazioni 'Serializable' determina, in generale, una limitazione alle prestazioni (throughput) di un DMBS.

Controllo della Concorrenza

Cosa fare in caso di failure:

- È fondamentale prevedere e gestire gli errori di concorrenza che possono occorrere con livello di isolamento 'Repeatable Read' o 'Serializable'.
- Se una transizione fallisce, il codice di fallimento è SQLSTATE = '40001'.
- SQLSTATE è una variabile di ambiente che si può leggere sia da programmi interni sia da programmi esterni.
- In caso di errore, si deve semplicemente ritentare l'esecuzione della transazione.
- Il costo computazionale di rieseguire una transazione è solitamente meno oneroso di quello che si avrebbe se si gestissero le transazioni usando i lock espliciti come, ad esempio, SELECT FOR UPDATE e SELECT FOR SHARE.

Lock Espliciti

- PostgreSQL permette di attivare lock espliciti di tabelle e anche di righe di tabelle.
- Lock espliciti dovrebbero essere gestiti a livello di applicazione quando il modello MVCC non garantisce il comportamento richiesto (soprattutto a livello di prestazioni).
- In questo corso i lock espliciti non sono considerati.
- Si rimanda al capitolo 13.3 del manuale di PostgreSQL per chi volesse approfondire l'argomento.