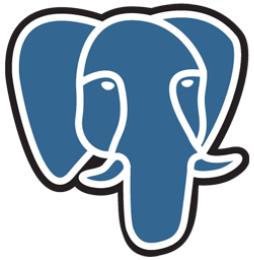

PostgreSQL

PGDay.IT 2011
Monash University Prato Centre
Venerdì 25 Novembre 2011

Serializable Snapshot Isolation (SSI) in PostgreSQL 9.1

Marco Nenciarini
Italian PostgreSQL Users Group

www.itpug.org
www.postgresql.org

Chi sono?

- DBA, sviluppatore e sysadmin presso 2ndQuadrant
  - Database OLTP business critical
  - Data warehousing
- Membro della comunità di PostgreSQL
  - Co-Fondatore di ITPUG
- Debian Developer

Sommario

- Le transazioni e i livelli di isolamento
- A cosa servono le transazioni serializzabili?
- Transazioni serializzabili prima della 9.1
- Anomalie
- Un nuovo approccio: Serializable Snapshot Isolation
- Conclusioni

Transazioni

- Sequenza di operazioni eseguita in maniera atomica
- Proprietà **ACID**
  - **Atomicità**
    Indivisibile, o tutto o niente
  - **Coerenza**
    Integrità dei dati
  - **Isolamento**
    Concorrenza con altre transazioni
  - **Durabilità**
    Persistenza dei dati

Isolamento

- Sono previsti 4 livelli di isolamento
  - **Read uncommitted**

    nessun isolamento, può leggere da transazioni che saranno annullate (dirty read)
  - **Read committed**

    letture successive della stessa riga possono ritornare valori differenti (non repeatable read)
  - **Repeatable read**

    letture successive della stessa riga sono coerenti, ma la stessa query eseguita due volte potrebbe ritornare dati diversi (phantom read)
  - **Serializable**

Transazioni Serializzabili

Se due o più transazioni vengono eseguite in maniera concorrente deve esistere una sequenza di esecuzione che porti allo stesso risultato

- Il programmatore si deve assicurare solo che la transazione esegua le operazioni corrette
- Il database garantisce che non ci saranno problemi di concorrenza

In pratica nei database

- Strong Strict Two Phase Locking (SS2PL)
  - Lock fino al commit su ogni oggetto coinvolto
  - Costoso
  - Deadlock
  - Database proprietari, no in PostgreSQL
- Snapshot Isolation (SI)
  - Non garantisce la serializzabilità
  - Nella maggior parte dei casi è sufficiente
  - PostgreSQL, Oracle, Microsoft SQL Server

Snapshot Isolation

- Accede a un'istantanea (snapshot) del database
  - MultiVersion Concurrency Control (MVCC)
- Veloce e poco costoso
- Non è permesso modificare alcun valore che sia stato cambiato in un'altra transazione concorrente
  - Chi primo arriva...

Vantaggi e svantaggi di Snapshot Isolation

- Letture non attendono mai, non bloccano le scritture
- Evita tutti i tipi di anomalie viste fino ad ora
  - Dirty read
  - Non repetable read
  - Phantom read
- Facilmente comprensibile
  - Transazioni concorrenti non si vedono fra loro
- Svantaggio: Anomalie
  - Alcune esecuzioni non possono essere serializzate

Anomalie in Snapshot Isolation

- Ci sono insiemi di transazioni per cui qualsiasi esecuzione è serializzabile
  - Esempio: transazioni eseguite dal benchmark TPC-C (OLTP)
- Snaphot Isolation può dare luogo a anomalie quando esiste un sottoinsieme di transazioni in cui ognuna modifica qualcosa nell'insieme di dati letti dalle altre
  - Vincoli di integrità non dichiarati esplicitamente nel database possono essere violati (write skew)
  - Abbastanza raro nella pratica

Esempio: Bianco e Nero

```
-- Crea una tabella che contiene punti colorati
CREATE TABLE punti (id int PRIMARY KEY, colore text);

-- 10 punti. Bianchi i pari, neri i dispari
INSERT INTO punti
  WITH x(id) AS (SELECT generate_series(1,10))
  SELECT id, CASE WHEN id % 2 = 0 THEN 'bianco'
    ELSE 'nero' END FROM x;
```

Esempio: Bianco e Nero (pre 9.1)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
UPDATE punti SET colore = 'nero'
  WHERE colore = 'bianco';




COMMIT;
```

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
UPDATE punti SET colore = 'bianco'
  WHERE colore = 'nero';
COMMIT;
```

- Adesso i punti pari sono neri e i punti dispari sono bianchi
  - **Impossibile da serializzare**

Esempio: Dipendenza incrociata

```
-- Tabella contenente dei generici dati
CREATE TABLE dati (
    classe int NOT NULL,
    valore int NOT NULL
);


-- Dati su cui dobbiamo effettuare un'analisi
INSERT INTO dati VALUES
    (1, 10),
    (1, 20),
    (2, 100),
    (2, 200);
```

Esempio: Dipendenza incrociata (pre 9.1)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT SUM(valore) FROM dati
  WHERE class = 1;
-- Risultato: 30


INSERT INTO dati VALUES (2, 30);


COMMIT;
```

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT SUM(valore) FROM dati
  WHERE class = 2;
-- Risultato: 300


INSERT INTO dati VALUES (1, 300);


COMMIT;
```

- Ogni transazione ha modificato i dati letti dall'altra
  - **Impossibile da serializzare**

Esempio: Protezione scoperto

```
-- Simuliamo dei conti in banca
CREATE TABLE conto (
    proprietario text NOT NULL,
    tipo text NOT NULL,
    saldo money NOT NULL DEFAULT '0.00'::money,
    PRIMARY KEY (nome, tipo)
);


-- Inseriamo due conti appartenenti a 'Carlo'
INSERT INTO conto VALUES
    ('Carlo', 'cassa', 500),
    ('Carlo', 'deposito', 500);
```

Esempio: Protezione scoperto (pre 9.1)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT tipo, saldo FROM conto
  WHERE nome = 'Carlo';
-- Totale 1000, posso prelevare 900



UPDATE conto SET saldo = saldo - 900
  WHERE nome = 'Carlo'
  AND tipo = 'deposito';


COMMIT;
```

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT tipo, saldo FROM conto
     WHERE nome = 'Carlo';
-- Totale 1000, posso prelevare 900


UPDATE conto SET saldo = saldo - 900
  WHERE nome = 'Carlo'
  AND tipo = 'cassa';

COMMIT;
```

- Saldo finale: -800
  - **Vincolo violato**

Serializable Snapshot Isolation

- A partire da PostgreSQL 9.1
  - È il primo (al momento l'unico)
- Il vecchio livello **serializable** diventa **repeatable read**
  - Più di quanto chiede lo standard (No Phantom Read)
- Costo aggiuntivo trascurabile rispetto a SI
  - Molto più veloce del Two Phase Locking
- Elevata concorrenza
  - Stessi vantaggi di SI
  - Nessun locking aggiuntivo

Come funziona SSI

- Come Snapshot Isolation
  - Usa MVCC per leggere da uno snaphot
- Mantiene traccia dell'insieme di lettura e scrittura di ogni transazione
- Identifica i conflitti di letura/scrittura al commit
- La transazione fallisce quando viene identificata una possibile anomalia
  - Il programma deve eseguire nuovamente la transazione
- Falsi positivi

Esempio: Dipendenza logica (9.1)

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT tipo, saldo FROM conto
  WHERE nome = 'Carlo';
-- Totale 1000, posso prelevare 900



UPDATE conto SET saldo = saldo - 900
  WHERE nome = 'Carlo'
  AND tipo = 'deposito';

COMMIT;
```

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT tipo, saldo FROM conto
  WHERE nome = 'Carlo';
-- Totale 1000, posso prelevare 900


UPDATE conto SET saldo = saldo - 900
  WHERE nome = 'Carlo'
  AND tipo = 'cassa';

COMMIT;
-- ERROR: could not serialize access
```

- Il database si è accorto dell'anomalia
  - La transazione fallita può essere eseguita nuovamente

Conclusioni

- Alte prestazioni in ambienti ad elevata concorrenza
  - Riduce i conflitti
  - No locking
- Permette di spostare la gestione della concorrenza all'interno del database
  - Applicazioni più semplici
  - Meno errori
  - Meno debugging
- PostgreSQL all'avanguardia nell'industria dei database

Domande?

- E-Mail: marco.nenciarini@2ndquadrant.it
- URL: www.2ndquadrant.it
- Blog: blog.2ndquadrant.it

Licenza Creative Commons

Attribuzione

Non commerciale

Condividi allo stesso modo

2.5 Italia

http://creativecommons.org/licenses/by-nc-sa/2.5/it/