PostgreSQL

PostgreSQL come sistema transazionale

Michele Finelli
m@biodec.com
BioDec

Indice

Indice

Punti di forza di PostgreSQL

PostgreSQL è un database server con le seguenti salienti caratteristiche:

1. sistema **transazionale**,
2. funzionalità di *high availability* e di replica,
3. sistema *enterprise*,
4. *free software*,
5. gratuità.

Cosa vuol dire essere un sistema transazionale ?

1. Un sistema transazionale è un sistema in cui ogni volta che vengono modificati dei dati, tali modifiche avvengono all'interno di una *cosa* chiamata **transazione**.
2. Una transazione garantisce delle **proprietà di persistenza e integrità**, che sono riassunte nell'acronimo *ACID*.

ACID

1. *ACID* significa *Atomicity*, *Consistency*, *Isolation* e *Durability*.
2. In un database ACID tali proprietà valgono sempre.
3. Se esistono delle situazioni in cui anche una sola delle proprietà cessa di valere, il sistema è *ipso facto* **non transazionale**.

Non essere ACID

- Si noti che non essere ACID non è di per sé un problema: esistono numerosi componenti di un sistema che **non sono transazionali**: ad esempio un *file system*.
- Il problema è che garantire le proprietà di integrità e di persistenza richieste è **costoso**, in termini di risorse computazionali — e per certi versi impossibile se trattiamo sistemi distribuiti.
- La quasi totalità dei sistemi NoSQL è in realtà **NoTransazionale** (il fatto che il *query language* non sia SQL è spesso un effetto collaterale).

Atomicity

- È la proprietà che garantisce che all'interno di una transazione **o tutte le operazioni hanno successo, oppure** — anche se una sola fallisce — **falliscono tutte**.
- In caso di fallimento, il sistema opera un *roll back*.
- In realtà forse sarebbe più corretto chiamarla *abortability*.

Consistency

- È la proprietà che garantisce che il database sia sempre in uno stato consistente.
- Il database passa, nel tempo, **da uno stato consistente ad un altro**.
- Non ha nulla a che fare con la *consistency* del *CAP theorem*.

Isolation

- È una proprietà che può essere definita in diversi modi: determina **quali dati sono visibili**, all'interno di una transazione, in presenza di accessi concorrenti che li possano modificare.
- È la proprietà più critica e spesso più sottovalutata: può dare origine a *bug* software molto subdoli.

Durability

- È la proprietà che garantisce che le modifiche causate dalle transazioni avvenute **siano salvate su uno *storage* persistente**.
- È una proprietà trasversale alle prime tre, ed è la più legata ad aspetti legati all'*hardware*.

ACID: an ill defined concept

1. *Atomicity* . . . non è proprio *atomicità*,
2. *Consistency* . . . lo è a modo suo,
3. di *Isolation levels* ce ne sono una pletora . . .
4. se vogliamo, anche *Durability* di solito si chiama *Persistence*.

> *ACID: more mnemonic than precise – Eric Brewer, 2012*

Isolamento e persistenza

- Nel seguito della presentazione ci focalizzeremo su due aspetti soli dell'essere ACID: l'isolamento e la persistenza.
- Vedremo prima cosa essi comportano da un punto di vista teorico.
- E successivamente come tali questioni sono affrontate in PostgreSQL.

Indice

Livelli di isolamento

- L'obiettivo è di dare l'impressione che, all'interno di una transazione, tutte le operazioni che si eseguono siano **isolate** dalle operazione che stanno compiendo altre transazioni concorrenti.
- Quando si effettuano delle **letture** o delle **scritture**, queste devono avvenire all'interno di un'immagine (consistente) del database, che non viene modificata finché la transazione non si è conclusa.

Anomalie

- Tutte le volte che all'interno di una transazione si "percepisce" quello che sta accadendo "fuori", significa che l'isolamento **non è perfetto**.
- L'isolamento perfetto è detto *serializzabile*, ma ha costi computazionali molto alti, e porta ad un degrado delle prestazioni: ecco perché si usano anche **altri livelli di isolamento**.
- A seconda di quello che si percepisce in questi altri livelli, è possibile che si presentino delle **anomalie**.
- La gravità di queste anomalie **dipende dall'applicazione**.

Tipi di anomalie

Anomalia	Descrizione	Bruttezza
Dirty read	Una transazione accede ad un dato scritto da una transazione concorrente, la quale **non sia stata ancora committata**.	**TOTALE**
Non-repeatable read	Una transazione **ripetendo una lettura** trova un dato che è stato modificato da una transazione che è terminata nel mentre.	**Marcata**
Phantom read	Come sopra, ma **ripetendo una query** e trovando un diverso insieme di righe.	*mmmh . . .*

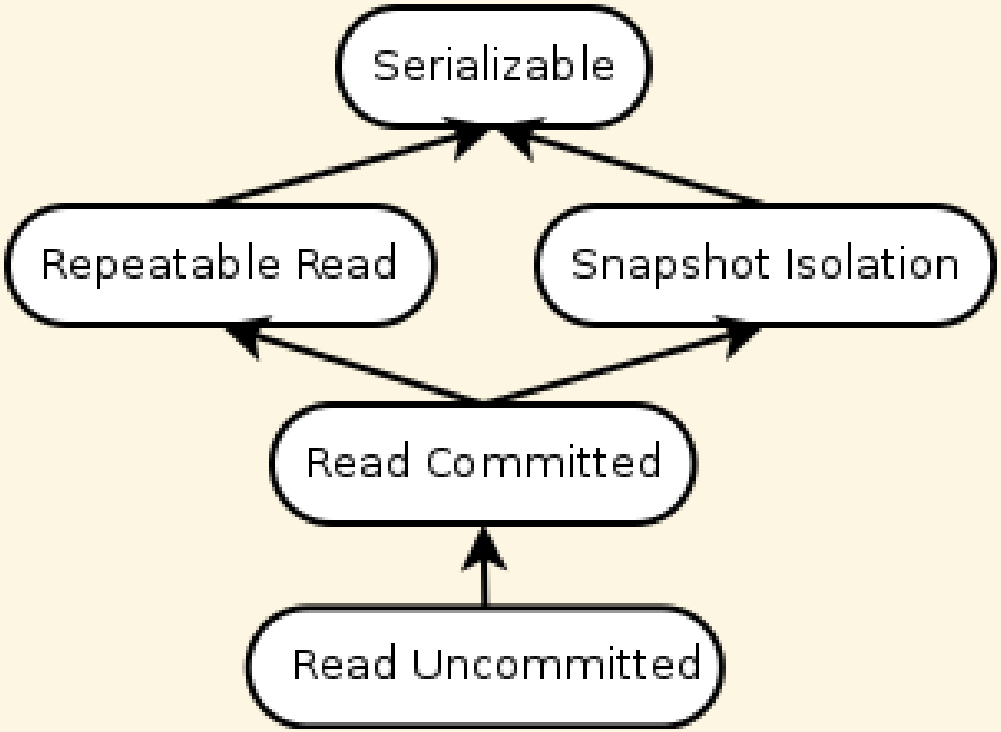
Anomalie consentite = livello di isolamento

- Lo standard SQL definisce quattro livelli di isolamento delle transazioni, così chiamati:
  - *Read Uncommitted*,
  - *Read Committed*,
  - *Repeatable Read*,
  - *Serializable*.
- Ogni livello permette o meno il presentarsi di una o più delle anomalie appena viste:
  - Dirty read,
  - Non-repeatable read,
  - Phantom read.

Livelli di isolamento

Livello di isolamento	Anomalia consentita		
	Dirty read	Non-repeatable read	Phantom read
Read uncommitted	Si	Si	Si
Read committed	No	Si	Si
Repeatable read	No	No	Si
Serializable	No	No	No

Relazioni tra i livelli



La freccia è la relazione "essere più debole di".

Serializable Snapshot Isolation

- POSTGRESQL permette diversi livelli, il default è *Read Committed*.
- In particolare POSTGRESQL realizza un livello di isolamento detto *Serializable Snapshot Isolation* (Ports - Gritten, Int. Conf. on VLDB 2012), che offre quasi le stesse garanzie del livello *Serializable* ma con minor impatto sulle prestazioni.
- La *reference* di POSTGRESQL su questo punto è alquanto sibillina.

From the source

In PostgreSQL, you can request any of the four standard transaction isolation levels. But internally, ***there are only three distinct isolation levels****, which correspond to the levels Read Committed, Repeatable Read, and Serializable.*

When you select the level Read Uncommitted you really get Read Committed, and ***phantom reads are not possible in the PostgreSQL implementation of Repeatable Read****, so the actual isolation level might be stricter than what you select.*

Indice

Persistenza

- L'aspetto della persistenza (o *Durability*, secondo l'acronimo ACID) è più semplice da esprimere e non soggetto a diversi tipi di anomalie.
- In pratica si tratta di garantire che un dato sia considerato modificato dopo che è avvenuta una *fsync()*.

fsync()

> *The fsync() function is intended to force a physical write of data from the buffer cache, and to assure that after a system crash or other failure that all data up to the time of the fsync() call is recorded on the disk. — POSIX*

È un'operazione molto dispendiosa, perché forza un I/O altrimenti differibile.

fsync e wal_sync_method

- Le possibili opzioni di wal_sync_method sono:

  open_datasync Scrive il WAL con l'opzione O_DSYNC di `open()`.

  fdatasync Invoca `fdatasync()`.

  fsync Invoca `fsync()`.

  fsync_writethrough Invoca `fsync()` ad ogni commit, forzando il *write-through* dell'eventuale *cache*.

  open_sync Scrive il WAL con l'opzione O_SYNC di `open()`.

- Il comando pg_test_fsync può aiutare a determinare il metodo migliore per la piattaforma in uso.

Indice

ACID in PostgreSQL: MVCC + WAL

- Poiché PostgreSQL è un sistema transazionale, deve implementare tutte le proprietà ACID.
- La parte **ACI** di **ACI**D è garantita da un meccanismo noto come *MVCC* (*Multi Version Concurrency Control*).
- La parte **D** di AC**ID** è garantita da un meccanismo diverso, noto come **WAL** (*Write Ahead Log*).

MVCC

1. Quando una transazione inizia un'operazione di scrittura viene definita una *transaction id*, o *XID*.
2. La XID è un valore a 32 bit.

MVCC

3. La XID determina la visibilità della transazione: tutte le transazioni con XID minore di una XID t sono nel **passato di** t, tutte quelle con XID maggiore sono nel **futuro di** t.
4. Le transazioni nel passato sono **visibili**, quelle nel futuro sono **invisibili**.

MVCC

5. I controlli dell'MVCC sono fatti al livello della singola tupla — per semplificare: l'unità di storage minima — con due campi `xmin` e `xmax`.
6. Quando una tupla viene creata `xmin` è posto pari all'XID della transazione che l'ha creata.
7. Quando una tupla viene cancellata, `xmax` è posto pari all'XID della transazione che l'ha cancellata.

MVCC

8. Se una tupla non ha un valore di `xmax`, significa che è *live*.
9. Si osservi che, alla cancellazione, una tupla non viene concretamente rimossa dalla *page* in cui si trova.
10. Si osservi anche che non vi è un concetto primitivo di *update*, ma questo viene modellato con una *insert* succeduto da una *delete*.

MVCC

11. La XID di una UPDATE va pertanto a valorizzare contemporaneamente sia il valore `xmin` della nuova tupla che il valore `xmax` della tupla vecchia.
12. Visto che il sistema non prevede mai di cancellare alcunché, esiste un meccanismo, realizzato dal comando *VACUUM*, che elimina le tuple non più richieste dalle transazioni in corso.

WAL

La parte **D** di AC**ID** è garantita da un meccanismo diverso, noto come *WAL* (Write Ahead Log).

1. In pratica, quando una *data page* è aggiornata, la modifica è immediatamente salvata sullo storage, nel WAL appunto.
2. Il data file corrispondente è modificato in seguito.
3. In caso di *crash* del database, il WAL viene letto e in caso di inconsistenza viene applicato nuovamente sui data file.

Database block lifecycle

La sequenza normale con la quale viene acceduto **un blocco di dati** è di solito il seguente:

1. Viene richiesta una pagina per memorizzare i dati nella memoria condivisa: se essi sono già presenti, non è necessario accedere al disco, altrimenti il sistema operativo carica la pagina in memoria.
2. In entrambi i casi le statistiche di `pgstat_bgwriter.buffers_alloc` vengono aggiornate — per esempio il *counter* sul numero di usi della pagina sarà incrementato.
3. Se si effettuano delle modifiche al blocco di dati esso è detto *dirty*.

Database block lifecycle

4. La transazione scrive sul WAL la modifica. Da questo momento che il blocco dirty sia poi scritto sullo storage è secondario al fine di ripristinare, ad esempio in caso di *failure*, l'informazione che il blocco sia stato cambiato.
5. Le statistiche in `pg_stat_user_tables` sono aggiornate in modo asincrono.
6. Sempre in modo asincrono, altri processi si preoccupano di scrivere il blocco dirty anche sul disco, e a quel punto l'area di memoria è marcata *clean*.

Point in time recovery

- L'esistenza del WAL permette di recuperare un'istanza di database al momento in cui il sistema passa da un WAL che è stato applicato ad uno nuovo.
- La copia dei data file fisici e del WAL già applicato permette di ripristinare il database ad uno stato coerente: anche uno stato relativo ad un istante nel passato.
- Questa possibilità è detta *Point in time recovery* (*PITR*).

Indice

Replica basata sul log shipping

- Le soluzioni di replica fra più istanze di POSTGRESQL utilizzano una tecnica, detta *Transaction Log Shipping*.
- Essa consiste nel mantenere dei server *in standby* aggiornati con il master attraverso la copia del WAL.
- Nonostante la tecnica sia unica, vi sono più soluzioni possibili, che si distinguono per essere basate sulla copia fisica del WAL file, o sulla copia continua (detta anche *streaming replication*) o una combinazione di entrambe le modalità.

Server standby e HA

- Un *server standby* è un server che riceve il WAL archiviato appena questo è disponibile e lo applica.
- Il server in standby può essere configurato anche come *hot standby* per ricevere le *query* di lettura.

Streaming replication

- Se non c'è attività sul server principale, l'operazione di archivio del WAL può essere differita, e quindi portare a ritardi nella sincronizzazione fra il server principale e quello in standby.
- Esiste un parametro l'`archive_timeout` che può forzare uno *switch* del WAL dopo un certo periodo.
- È però preferibile un'altra modalità di replica, detta appunto *streaming replication*, in cui il WAL viene copiato sul server in standby man mano che questi si crea.

Indice

Conclusioni

- PostgreSQL è un sistema transazionale **vero**.
- Esso ottiene tale obiettivo con un'implementazione **MVCC** che ha delle particolarità, e con un meccanismo di **WAL** per quanto riguarda la *Durability*.
- Avendo un meccanismo di WAL, una tecnica di *shipping* del medesimo è alla base della funzionalità di **replica** fra più istanze di PostgreSQL.
- Avere diverse repliche permette di realizzare sistemi *fault tolerant*, scalabili, eccetera.

Domande

Next

- La licenza è della presentazione è CC-BY-SA4.0 a.k.a. “Bàn xa vút da la vétta”; il PDF è reperibile all’URL `https://github.com/finelli/slideware`
- Contatti: `m@biodec.com`
- Ci vediamo all’Incontro DevOps Italia 2016, 1 Aprile 2016 a.k.a. “Non è uno scherzo”, a Bologna — maggiori informazioni su `http://www.incontrodevops.it/`