Transaction Management

Hiểu và sử dụng transaction trong bài toán Concurrency

Ngày 20 tháng 8 năm 2022

Nội dung

Nội dung

Giới thiệu về Transactions

Nội dụng bao gồm:

- Khái niệm Transaction
- Tính ACID
- Xử lý đồng thời Transactions

Khái niệm Transaction

Bài toán Concurrency Control:

- Nhiều người cùng truy cập database tại một thời điểm.
- Một người cập nhật trong khi có người khác đọc cùng một dữ liệu?
- Cả hai người cùng cập nhật dữ liệu cùng một lúc?

Vậy làm thế nào để truy cập dữ liệu hiệu quả và an toàn?

Khái niệm Transaction

Transaction:

- Chuỗi các operations được executed như một *single*, *logical*, *atomic* unit
- Kết quả trung gian không ảnh hưởng đến transactions khác.
- Hoàn thành tất cả hoặc không gì cả - all-or-nothing.

Các thao tác với database sẽ được thực hiện thông quan transactions.

Tính ACID

4 tính chất của transaction:

- Atomic: hoặc là tất cả hoặc không.
- Consistency: đảm bảo tính toàn vẹn, trách nhiệm của người dùng.
- Isolation: độc lập, không ảnh bởi transaction khác.
- Durability: khi hoàn thành, đảm bảo thay đổi được lưu lại.

Consistency và Isolation

Chuyển tiền từ tài khoản A sang cho B:

- Trừ tiền A
- Cộng tiền B
- Tiêu chí toàn vẹn: A + B

Chương trình lỗi: cộng tiền B thiếu 1 dollar so với trừ A

Consistency và Isolation

Tính cô lập thể hiện:

- Đồng thời xử lý = Tuần tự
- Lập lịch xen kẽ các transactions = Tuần tự thực hiện

Atomicity và Durability

Transaction có thể bị dừng giữa chừng:

- Bị hủy bởi người dùng
- Hệ thống bị crash (mất điện)
- Dữ liệu bất thường hay không truy cập được ổ đĩa

Các thao tác giữa chừng cần phải undo?
DBMS sử dụng logs để lưu các thay đổi

- Có thể undo
- Hay khôi phục hệ thống nếu chưa kịp lưu vào ổ đĩa

Lập lịch xử lý transactions

Xử lý tuần tự vs interleaved (xen kẽ):

- Tận dụng CPU
- Tránh chờ I/O
- Đảm bảo thời gian response

Lập lịch xử lý transactions

Một transaction - một series các actions:

- Transaction T đọc object O: $R_T(O)$
- T ghi object O: $W_T(O)$
- T complete: $Commit_T$
- T hủy (rollback): $Abort_T$

Bỏ qua T nếu ngữ cảnh rõ ràng

Ví dụ lịch xử lý transactions

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	Commit
Commit	

Xung đột dữ liệu

Khi có nhiều transactions đồng thời, tồn tại các xung đột:

- Dirty read: đọc dữ liệu đã thay đổi bởi một transaction khác chưa commit
- Nonrepeatable read: đọc (sử dụng) cùng một dữ liệu lần 2 nhưng thấy đã bị thay đổi bởi transaction khác
- Phantom read: chạy lại cùng câu truy vấn với search condition và thấy tập dữ liệu trả về đã bị thay đổi bởi transaction khác

Transaction Isolation Level

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommited	X (PG)	X	X
Read commited	–	X	X
Repeatable read	–	–	X (PG)
Serializable	–	–	–

Bảng: Bốn mức isolation của transactions

PG: not in Postgresql

Serializable Level

Isolation level kiểm soát mức độ isolation của một transaction với các transaction đồng thời khác. Cao nhất là *Serializable*:

- T chỉ đọc dữ liệu commited (no dirty read)
- Ko có giá trị đang đọc hay ghi bởi T được thay đổi bởi transaction khác
- T đang đọc một tập search condition thì tập này không được thay đổi bởi transaction khác

REPEATABLE READ Level

T chỉ đọc dữ liệu commited và ko có giá trị đang đọc hay ghi bởi T được thay đổi bởi transaction khác. Nhưng T có thể gặp phantom read. Ví dụ T truy vấn tất cả Books còn bản copies, transaction khác lại thêm một Book mới. Vậy T bị miss Book mới.

READ COMMITED Level

T chỉ đọc dữ liệu commited và ko có giá trị đang ghi bởi T được thay đổi bởi transaction khác. Tuy nhiên, một giá trị T đọc có thể bị thay đổi bởi transaction khác, dẫn đến "nonrepeatable read".

READ UNCOMMITED Level

Ở level này transaction có thể đọc giá trị đang được update bởi transaction (uncommited). Có thể coi là transaction không cần isolation.

Strict Two-phase Locking - Strict 2PL

Một cách để thực hiện các mức isolation là dùng cơ chế locking - Strict 2PL:

- T muốn đọc (ghi) O thì phải lấy được shared lock (exclusive lock của O. Khi có một exclusive lock trên O, các transactions không thể lấy lock shared hay exclusive lock khác trên O mà phải chờ.
- Tất cả locks được release khi transaction kết thúc.

Strict 2PL sẽ đảm bảo mức isolation "serializable", kết quả thực hiện sẽ giống khi làm tuần tự các transactions. Tuy vậy vấn đề deadlock có thể xảy ra?

Nội dung

Transactions trong SQL

Isolation mặc định: READ COMMITED Level
Tạo transaction bằng command `BEGIN` và `COMMIT`:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
-- etc etc
COMMIT;
```

Transaction characters

```
START TRANSACTION [ transaction_mode [, ...] ]

where transaction_mode is one of:

    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED
    | READ UNCOMMITTED }
    READ WRITE | READ ONLY
```

https://www.postgresql.org/docs/current/sql-start-transaction.html
https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html

Transaction trong SQL

DBMS mặc định xử lý mỗi câu SQL (statement) trong một transaction. Bắt đầu bằng *BEGIN* và kết thúc *COMMIT* (nếu thành công).

Rollback transaction

Sử dụng command *ROLLBACK* để hủy transaction.

Rollback transaction

Sử dụng command *ROLLBACK* để hủy transaction.
Nếu transaction bị lỗi thì không thể *COMMIT*

Rollback transaction

Sử dụng command *ROLLBACK* để hủy transaction.
Nếu transaction bị lỗi thì không thể *COMMIT*

Exercise: make an error transaction

Add a constraint that doesn't allow any tags with length shorter than 2 characters.
Create a transaction that violates the above constraint.

Ví dụ sử dụng transaction

Khi nào sử dụng transaction (explicit)?

Ví dụ sử dụng transaction

Khi nào sử dụng transaction (explicit)?
Một luồng xử lý hoặc là thành công hoặc là thất bại.

Ví dụ sử dụng transaction

Khi nào sử dụng transaction (explicit)?
Một luồng xử lý hoặc là thành công hoặc là thất bại.

Exercise: lấy ví dụ sử dụng transaction

Chuyển tiền ngân hàng, trừ tài khoản A và cộng vào B.
Các thao tác cần được làm trong cùng một transaction.

Rollback with SAVEPOINT

Bỏ qua một phần và commit phần còn lại.

Rollback with SAVEPOINT

Bỏ qua một phần và commit phần còn lại.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
        WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE account SET balance = balance + 100.00
        WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE account SET balance = balance + 100.00
        WHERE name = 'Wally';
COMMIT;
```

Ví dụ về SAVEPOINT

Exercise: savepoint khi thêm tags

Tạo một transaction thêm nhiều tags.
Định nghĩa một savepoint.
Thực hiện rollback về savepoint.

Ví dụ về SAVEPOINT

Exercise: savepoint khi thêm tags

Tạo một transaction thêm nhiều tags.
Định nghĩa một savepoint.
Thực hiện rollback về savepoint.

More about savepoint

Tạo một transaction có nhiều savepoints.
Thực hiện rollback về các savepoints.
Kết quả sẽ ntn?.

Thời gian trong transaction

Transaction được coi là Atomicity.
Vậy thời gian trong transaction sẽ ra sao?

Deadlock trong SQL

Ví dụ tình huống xảy ra deadlock.
Khi đó DBMS sẽ xử lý ra sao?

Nội dung

Cơ chế Locking

Xin chào

Nội dung

Giới thiệu về Crash Recovery

Xin chào