Transactions and Isolation

Tom Kelliher, CS 318

Apr. 29, 2002

1 Administrivia

Announcements

Normal form analyses due Wednesday. Toolboxes and projects due Friday.

Review for final on Friday. Course evaluation on Friday.

Assignment

Read 15.2–3.

From Last Time

Access path considerations for various DB operations.

Outline

1. Why transactions?

2. PostgreSQL transaction features.

3. Isolated transactions, serializability, lock granularity

Coming Up

Atomicity, durability, and distributed transactions.

2 Overview — Why Transactions?

1. Consistency need not be maintained during a transaction.

   Example: Holdings and Balance tables. For consistency, Balance should reflect all Holdings.

2. ACID properties:

   (a) Atomicity — All or nothing.

   (b) Consistency — If consistent before, consistent after.

   (c) Isolation — Concurrent transactions behave as is they were performed serially.

   (d) Durability — Once committed, remembered.

3 PostgreSQL Transaction Features

1. `BEGIN` — Begin a transaction in chained mode.

   Ordinarily, each SQL statement is considered its own transaction (unchained mode).

2. `COMMIT` — End a transaction, applying all modifications.

3. `ROLLBACK` — End a transaction, rolling back all modifications.

4. `SET TRANSACTION ISOLATION LEVEL READ COMMITTED`,
   `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` — Set isolation level. Default is read committed. Should be set immediately following `BEGIN`.

5. `LOCK` — Explicitly lock a table or row. Several options available. PostgreSQL always locks at the least restrictive level. `LOCK` allows the designer to override this behavior. Should be given immediately following `BEGIN`. Refer to online documentation.

4 Isolated Transactions

1. Consider two transactions in a banking system.

   (a) T1 is a deposit of $100.

   Operations: R1(x), W1(x).

   (b) T2 is a withdrawal of $50 from the same account.

   Operations: R2(x), w2(x).

2. Two correct serial execution schedules:

   (a) R1(x), W1(x), R2(x), W2(x).

   (b) R2(x), W2(x), R1(x), W1(x).

3. An incorrect concurrent schedule: R1(x), R2(x), W1(x), W2(x).

   Why incorrect?

   Note that R2(x) and W1(x) were commuted, relative to the first serial schedule.

4. Which operations are safe to commute?

5. Serializable schedule — a concurrent schedule which corresponds to some serial schedule.

   Examples: R1(y), R2(x), R1(x), R2(y). R1(x), R2(y), W1(x), W2(y).

   Two reads always commute. A write and another operation commute if they reference different objects.

6. Concurrency control: The part of the transaction processing system that enforces isolation.

4.1 Implementing Serializability

1. Use *strict* two-phase locking protocol:

    (a) Phase 1: Obtain locks.

    (b) Phase 2: Release locks.

2. Strict: all locks are held until transaction commits or rollbacks.

    Nonstrict: some locks are released before others.

3. Types of locks:

    (a) Read lock: Item may be read. May be shared with other read locks.

    A read lock request will wait for the release of a write lock on the same item.

    (b) Write lock: Item may be read or written. Not shareable with other locks.

    A write lock request will wait for the release of a write lock or all read locks on the same item.

4. Possible problems with nonstrict protocols: dirty read, nonrepeatable read, lost update.

    Example of a dirty read: W1(x), Rel1(x), R2(x), Rollb1(x).

    A nonrepeatable read: R1(x) Rel1(x), W2(x), Commit2(x), R1(x).

    Banking example illustrated lost update.

5. **Deadlock!!!**

4.2 Lock Granularity

1. Table locks: coarse.

2. Row or field locks: fine.

3. Impact upon concurrency?

4. DB isolation levels:

   (a) Read uncommitted: Read without obtaining a read lock. Dirty reads possible.

   (b) Read committed: Short term read lock acquired. Nonrepeatable reads possible.

   (c) Repeatable read: Long term tuple read locks acquired. Phantoms possible.

   (d) Serializable: Long term table read locks acquired. Concurrency reduced.

   Write locks are always of *long* duration.

5. Phantoms: an anomaly of row locking.

   (a) T1 read locks all tuples satisfying

   ```
   SELECT *
   FROM Transcript T
   WHERE T.Studid = '123456';
   ```

   (b) Subsequently, T2 inserts a new Transcript record for 123456, before T1 finishes.

   (c) The new record won't be seen by T1 — it is a phantom.

6. *Intention locks* necessary in presence of various lock granularities.