Lecture #16: Concurrency Control Theory

15-445/645 Database Systems (Fall 2019)
https://15445.courses.cs.cmu.edu/fall2019/
Carnegie Mellon University
Prof. Andy Pavlo

1 Transactions

A *transaction* is the execution of a sequence of one or more operations (e.g., SQL queries) on a shared database to perform some higher level function. They are the basic unit of change in a DBMS. Partial transactions are not allowed.

Example: Move $100 from Andy's bank account to his bookie's account

1. Check whether Andy has $100.
2. Deduct $100 from his account.
3. Add $100 to his bookie's account.

Executing concurrent transactions in a DBMS is challenging. It is difficult to ensure correctness while also executing transactions quickly. We need formal correctness criteria:

- Temporary inconsistency is allowed.
- Permanent inconsistency is bad.

The scope of a transaction is only inside the database. It cannot make changes to the outside world because it cannot roll those back.

2 Definitions

A database is a set of named data objects $(A, B, C, ..)$. A transaction is a sequence of read and write operations $(R(A), W(B))$.

The outcome of a transaction is either `COMMIT` or `ABORT`.

- If `COMMIT`, all of the transaction's modifications are saved to the database.
- If `ABORT`, all of the transaction's changes are undone so that it is like the transaction never happened. Aborts can be either self-inflicted or caused by the DBMS.

Correctness Criteria: **ACID**

- **A**tomicity: All actions in the transaction happen, or none happen.
  "All or Nothing"
- **C**onsistency: If each transaction is consistent and the database is consistent at the beginning of the transaction, then the database is guaranteed to be consistent when the transaction completes.
  "It looks correct to me..."
- **I**solation: The execution of one transaction is isolated from that of other transactions.
  "As if alone"
- **D**urability: If a transaction commits, then its effects on the database persist.
  "The transaction's changes can survive failures..."

3 ACID: Atomicity

The DBMS guarantees that transactions are **atomic**. The transaction either executes all its actions or none of them.

Approach #1: Shadow Paging

- DBMS makes copies of pages and transactions make changes to those copies. Only when the transaction commits is the page made visible to others.
- Originally from System R but abandoned in the early 1980s. Few systems do this today (CouchDB, LMDB).

Approach #1: Logging

- DBMS logs all actions so that it can undo the actions of aborted transactions.
- Think of this like the black box in airplanes.
- Logging is used by all modern systems for audit and efficiency reasons.

4 ACID: Consistency

The "world" represented by the database is **consistent** (e.g., correct). All questions (i.e., queries) that the application asks about the data will return correct results.

Database Consistency:

- The database accurately represents the real world entity it is modeling and follows integrity constraints.
- Transactions in the future see the effects of transactions committed in the past inside of the database.

Transaction Consistency:

- If the database is consistent before the transaction starts, it will also be consistent after.
- Ensuring transaction consistency is the application's responsibility.

5 ACID: Isolation

The DBMS provides transactions the illusion that they are running alone in the system. They do not see the effects of concurrent transactions. This is equivalent to a system where transactions are executed in serial order (i.e., one at a time). But in order to get better performance, the DBMS has to interleave the operations of concurrent transactions.

Concurrency Control

A *concurrency control protocol* is how the DBMS decides the proper interleaving of operations from multiple transactions.

There are two categories of concurrency control protocols:

1. **Pessimistic:** The DBMS assumes that transactions will conflict, so it doesn't let problems arise in the first place.
2. **Optimistic:** The DBMS assumes that conflicts between transactions are rare, so it chooses to deal with conflicts when they happen.

The order in which the DBMS executes operations is called an *execution schedule*. The goal of a concurrency control protocol is to generate an execution schedule that is is equivalent to some serial execution:

- **Serial Schedule:** A schedule that does not interleave the actions of different transactions.
- **Equivalent Schedules:** For any database state, the effect of execution the first schedule is identical to the effect of executing the second schedule.
- **Serializable Schedule**: A schedule that is equivalent to some serial execution of the transactions.

When the DBMS interleaves the operations of concurrent transactions, it can create anomalies:

- **Read-Write Conflicts ("Unrepeatable Reads"):** A transaction is not able to get the same value when reading the same object multiple times.
- **Write-Read Conflicts ("Dirty Reads"):** A transaction sees the write effects of a different transaction before that transaction committed its changes.
- **Write-Write conflict ("Lost Updates"):** One transaction overwrites the uncommitted data of another concurrent transaction.

There are actually two types for serializability: (1) conflict and (2) view. Neither definition allows all schedules that you would consider serializable. In practice, DBMSs support conflict serializability because it can be enforced efficiently. To allow more concurrency, some special schedules are handled at the application level.

Conflict Serializability

Schedules are equivalent to some serial schedule. This is what (almost) every DBMS supports when you ask for the `SERIALIZABLE` isolation level.

Schedule S is conflict serializable if you are able to transform S into a serial schedule by swapping consecutive non-conflicting operations of different transactions.

Verify using either the swapping method or dependency graphs.

Dependency Graphs (aka "precedence graph"):

- One node per transaction.
- Edge from T_i to T_j if an operation O_i of T_i conflicts with an operation O_j of T_j and O_i appears earlier in the schedule than O_j.
- A schedule is conflict serializable if and only if its dependency graph is acyclic.

View Serializability

Allows for all schedules that are conflict serializable and "blind writes". Thus allows for slightly more schedules than Conflict serializability, but difficult to enforce efficiently. This is because the DBMS does not now how the application will "interpret" values.

6 ACID: Durability

All of the changes of committed transactions must be **durable** (i.e., persistent) after a crash or restart. The DBMS can either use logging or shadow paging to ensure that all changes are durable.