ACID Properties of Transactions

Assumed or required properties of transactions.

*Atomicity*
Each transaction is done completely or not at all.

*Consistency*
A transaction is assumed to leave the database in a consistent state.

*Isolation*
Concurrent transactions should not interfere with one another.

*Durability*
When a transaction finishes (commits), its actions are permanent.

Handled by locking and logging. Locks ensure isolation, while logging allows for rollbacks and crash recovery.

Declaring a Transaction

START TRANSACTION (also BEGIN WORK)
COMMIT
ROLLBACK

Classic Problems

*lost update*
two xacts read the same item, and both later write it - the first's update gets lost

*dirty read*
one xact T1 reads the output of T2 - if T2 has to rollback then so does T2

*phantom*
subtle problem caused by insertions which other xacts would not have locked
MySQL solves with next-key locks, and in general with some sort of index lock

*deadlock*
two xacts waiting for lock releases from each other

Goal: serializability
A scheduling of multiple xacts is called *serializable* if it is equivalent to some serial schedule (in which there would be no concurrency).

In general, serializability is achieved through **two-phase locking** (2PL) - locks are placed on data items, and no locks are set after any locks are released.

The dirty-read problem (and in general that of cascading rollback) is achieved through the use of **strict 2PL** - all locks are held until the commit point of the xact.

**Lock Types**

- *shared*  
  for reading
- *exclusive*  
  for writing
- *update*  
  shared lock that can be upgraded to exclusive

**Deadlock**

A big problem with locking. Suppose T1 locks A and T2 locks B. Then T1 asks for a lock on B (and has to wait for T2) and T2 asks for a lock on A (so is waiting for T1).

Generally the lock manager has the job of **deadlock detection** and of resolving the problem, typically by forcing some xact to rollback.

**Lock Granularity**

- database
- table  
  ex - LOCK TABLE student IN SHARED MODE
- page
- row (or record)
- index (or column)

(From MySQL manual) **InnoDB has several types of record-level locks:**

- **Record lock**: This is a lock on an index record.
- **Gap lock**: This is a lock on a gap between index records, or a lock on the gap before or after a range of records.
- **Next-key lock**: This is a combination of a record lock on the index record and a gap lock on the gap before the index record.
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;

SELECT * FROM child WHERE id > 100 FOR UPDATE;
sets a next-key lock - avoids the phantom problem

SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;

Multiple Granularity Locks

MySQL and many database providers support locks at various levels of granularity. Locks must be set on the coarsest items first, and intent locks are used. For example, an IX lock on a table R says that "I will be putting an X lock on some rows (but not necessarily all rows) of R"

lock names: S, X, IS, IX

Backing away from lock protocols

Using locks can reduce concurrency and slow throughput. You may wish to back off of the locking protocol, at some risk to accuracy.

SET TRANSACTION ISOLATION LEVEL
- READ UNCOMMITTED dirty read allowed
- READ COMMITTED default
- REPEATABLE READ
- SERIALIZABLE most safe but slowest

Logging

The binary log (MySQL’s name) contains all statements that update data or potentially could have updated it (for example, a DELETE which matched no rows).

Used for

rollbacks - a xact may time out or have entered deadlock

recovery - systems crash, need to be able to recover

Long Transactions

A big problem, since the log fills up. Log management a job for the DBA, to make sure the log
is frequently flushed to disk. Also, frequent commits of longer xacts are helpful.