

Concurrency Control in RDBMS

Arjen Lentz (arjen@mysql.com)
Community Relations Manager
MySQL AB



Defining Concurrency Control

- The mechanisms required to manage multiple users accessing the same groups of resources (tables, rows)
- with the least possible overhead
- either optimized for a specific task, or general purpose.

Note!

- Lots of variations & combinations possible in implementation
- no architecture is best in all or even most scenarios
- so we're just exploring what's out there
- not playing a “mine is better than yours”.

“Lost Update”

Connection A

- ...
- SELECT row1
- ...
- ...
- ...
- UPDATE row1
- ...

Connection B

- ...
- ...
- ...
- SELECT row1
- ...
- ...
- ...
- UPDATE row1

“Uncommitted Dependency” (1)

Connection A

- ...
- ...
- ...
- **SELECT row1**
- ...
- ...

Connection B

- ...
- **UPDATE row1**
- ...
- ...
- ...
- **ROLLBACK**

“Uncommitted Dependency” (2)

Connection A

- ...
- ...
- ...
- UPDATE row1
- ...
- ...

Connection B

- ...
- UPDATE row1
- ...
- ...
- ...
- ROLLBACK

“Inconsistent Analysis”

Connection A

- SELECT SUM(accounts)
- acc[1]=40 ... sum=40
- acc[2]=50 ... sum=90
- ...
- ...
- ...
- ...
- ...
- ...
- acct[3]=20 ... sum=110
(should be 120!)

Connection B

- acc[1]=40
- acc[2]=50
- acc[3]=30
- ...
- UPDATE acc3=acc3 - 10
(30 -> 20)
- UPDATE acc1=acc1 +10
(40 -> 50)
- COMMIT

Locking Granularity

- Database – Table – Page – Row
- **Database**
 - SQLite
- **Table**
 - MySQL/MyISAM, MySQL/MEMORY
- **Page**
 - BerkeleyDB, Ingres
- **Row**
 - Oracle, DB2, MSSQL, Informix, PostgreSQL, Firebird, MySQL/InnoDB, MySQL/NDB Cluster, MySQL/Falcon

Locking Types

- Shared (read) – *may co-exist with other shared locks*
 - Others may read.
 - Others may NOT write.
 - MySQL/MyISAM also has a special **READ LOCAL** lock
- Update – *“upgrade” to exclusive before an actual update*
 - Others may read.
 - No other update lock allowed.
 - May co-exist with shared locks.
- Exclusive (write) – no other locks can co-exist
 - Others may NOT read.
 - Others may NOT write.

What is a Transaction?

- **START TRANSACTION**
- ... one or more operations ...
- **COMMIT / ROLLBACK**

- In **AUTOCOMMIT=1** mode, each command is a transaction.

Transactions: The ACID Rules

- *Atomicity*
 - One or more operations regarded as a single unit of work.
 - Indivisible: either all or nothing.
- *Consistency*
 - The database must always move from one consistent state to another.
 - Inconsistencies may exist during a transaction.
- *Isolation*
 - Others must not see volatile changes.
- *Durability*
 - Once committed, data must persist somewhere/somehow.

“Lost Update” revisited

Connection A

- ...
- SELECT row1
- ...
- ...
- ...
- UPDATE row1
- ...

Connection B

- ...
- ...
- ...
- SELECT row1
- ...
- ...
- ...
- UPDATE row1

Isolation Levels (1/4)

- **Read Uncommitted**
 - The query sees the result of all transactions, both ongoing and committed.
 - Writes need to use locking.
- Prevents “*Lost Update*”
- “*Dirty Read*” possible

“Dirty Read”

Connection A

- START TRANSACTION
- SELECT row1
- UPDATE row1
- ...
- ROLLBACK

Connection B

- START TRANSACTION
- ...
- ...
- **SELECT row1**
- ...

Isolation Levels (2/4)

- **Read Committed**
 - Read queries sees the result of any transactions committed before that point of time.
 - Effects of uncommitted transactions are not seen.
- Prevents “*Dirty Read*”
- “*Non-Repeatable Read*” possible

“Non-Repeatable Read”

Connection A

- START TRANSACTION
- ...
- SELECT row1
- ...
- ...
- ...
- **SELECT row1**
- ...

Connection B

- START TRANSACTION
- SELECT row1
- ...
- UPDATE row1
- COMMIT

Isolation Levels (3/4)

- **Repeatable Read**
 - The transaction sees the changes made by the transactions that were committed before the transaction was started.
 - No changes made later by committed or uncommitted transactions are seen.
 - Each read within the transaction always gives the same result (except for changes made inside the transaction).
- Prevents “*Non-Repeatable Read*”
- “*Phantoms*” possible
- *MySQL/InnoDB uses next-key locking: no phantoms*

“Phantoms”

Connection A

- START TRANSACTION
- SELECT rows (id < 10)
- (retrieved id 1...7)
- ...
- ...
- ...
- SELECT rows (id < 10)
- (retrieved id 1...8)
- ...

Connection B

- START TRANSACTION
- ...
- ...
- ...
- INSERT row (id=8)
- COMMIT

Isolation Levels (4/4)

- **Serializable**
 - Transactions are completely isolated from each other.
 - All **SELECTS** acquire shared locks.
- Prevents “*Phantoms*”

Isolation Consequences

- Higher isolation level:
 - increased “correctness” of operations
 - reduced concurrency
- An implementation *should* escalate an unsupported isolation level to a *higher* one that it does implement.
- Some shift down instead
(REPEATABLE READ -> READ COMMITTED)

Lock Escalation

- Replacing multiple locks with one higher level lock
Rows -> Pages -> Tables
- Reduces memory consumption and management overhead
- Impacts concurrency
- Upto RDBMS to decide when and how
- MySQL/InnoDB needs no escalation as its row-level locking information uses only very little memory

Pessimistic or Optimistic Locking

- **Do we assume...**
 - a) that we probably will encounter access conflicts (pessimistic), or
 - b) that we can complete without hindrance? (optimistic)
- **Pessimistic:** *lock everything we might read or write*
Users wait for locks to clear.
- **Optimistic:** *Take action only when conflict occurs*
Users get rejected and need to try operation again.

Other Relevant Concepts

- **Two-Phase Locking Protocol**
 - Before operating on any object, a lock on that object must be acquired.
 - After releasing a lock, no more locks may be acquired.
- **Buzzword MVCC = Multi-Versioned Concurrency Control**
 - **SELECTS** don't need any locks! (invented by Jim Starkey)
 - Supported by Oracle, PostgreSQL, Firebird, MySQL/InnoDB, MySQL/Falcon.
- **Deadlocks**
 - Access conflict which cannot be resolved by waiting.
 - One (or all) transactions involved need to be rolled back.

Explicit Locking

- **Row level or escalated**
 - **SELECT . . . LOCK IN SHARE MODE** (shared)
 - **SELECT . . . FOR UPDATE** (exclusive)
- **Table level**
 - **LOCK TABLE . . . READ** (shared)
 - **LOCK TABLE . . . WRITE** (exclusive)
- Explicit locking is *not* defined/supported by SQL standards!

So now you know! Use wisely.

Questions?

Thank you!

Arjen Lentz
arjen@mysql.com