Isolation levels

Topic 6 Lesson 9
Providing different levels of the isolation property

Adapted from online MySQL documentation

MySQL documentation
https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-locking-reads.html
https://mysqlserverteam.com/mysql-connection-handling-and-scaling

Locks effect on database performance

- Locks force transactions to wait
  - Abort and restart of transactions due to deadlock wastes the work done by the aborted transaction
  - Locks limit database access to transactions and this typically leads to reduced DB performance
  - In practice, lock contention is low, and deadlocks are rare, so the DBMS does a lot of lock management work and reduces the DB performance for issues that typically do not occur
  - Waiting for locks becomes the bigger problem as more transactions execute concurrently

Thrashing due to too many transactions

- Allowing more concurrent transactions initially increases throughput, but at some point, the increase in active transactions can result in thrashing
- Thrashing occurs because too many transactions compete for memory resources or buffer frames in the DBMS buffer
- The buffer manager is rapidly exchanging data pages for data on hard disk due to the active transaction's data requests.
- The database server is only completing this page swapping and no other actions are being completed, so the transactions do not progress. The swapping causes a very high rate of disk access.

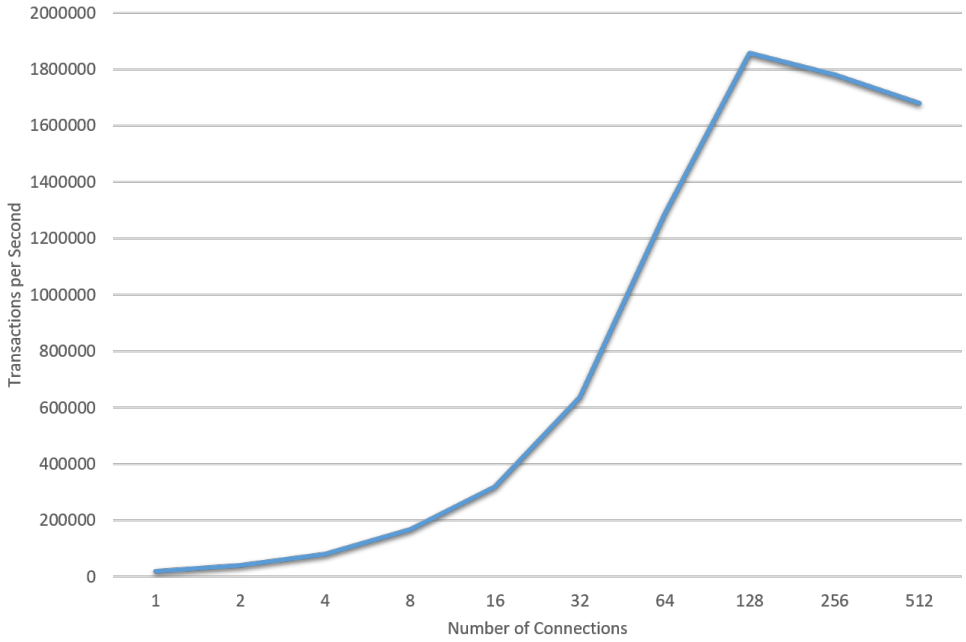
Locking can also lead to thrashing

When too many transactions request locks, a large number of transactions suddenly become blocked, and few transactions can make progress, so we see a decline in transaction throughput. This is known as lock thrashing.

Too many transactions

The initial relationship between number of transactions and number of transactions processed per second is close to linear

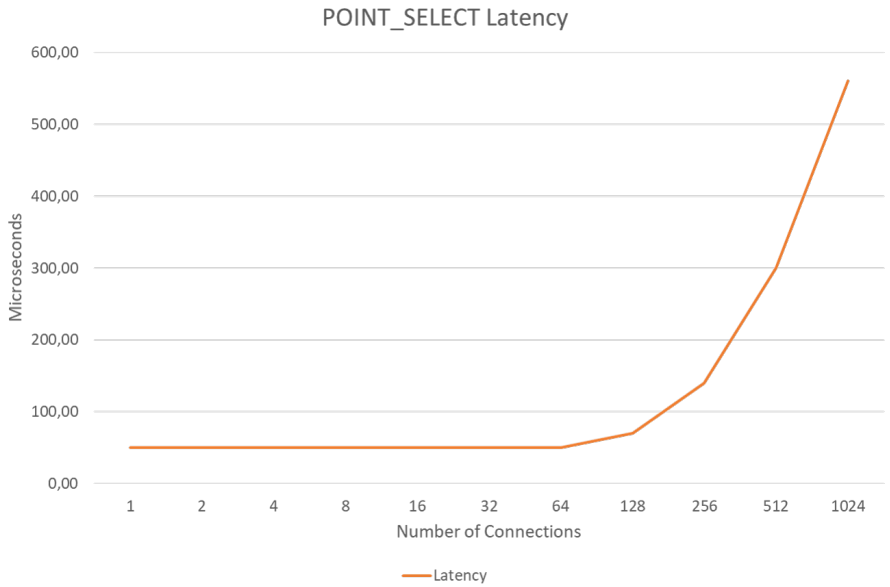
However we hit a decline in transactions processed per second when there are more than 128 connections



https://mysqlserverteam.com/mysql-connection-handling-and-scaling/

Thrashing effect on the transactions

- This graph displays the amount of time a transaction needs to wait to receive a read of a data object
- We see constant latency time until we hit the 128-connection limit, then there is exponential growth in the wait time



https://mysqlserverteam.com/mysql-connection-handling-and-scaling/

Thrashing solutions

- Increase the size of the buffer pool to accommodate the needs of the active transactions
- Limit the maximum number of concurrent transactions to prevent thrashing
  - MySQL has a system wide variable, innodb_thread_concurrency that limits the number of threads (clients) that the server will handle
    - Once the maximum is reached, it places certain clients to sleep for a duration, in order to lower the active connections
    - This value needs to be managed and optimized by the DBA
- Minimize lock contention by reducing the time a transaction holds locks or by avoiding hotspots in the database
  - A hot spot is a database object that is frequently accessed and modified and causes a lot of blocking delays. Hot spots can significantly affect performance.

Minimizing lock contention

1. Lower the number of locks
   - Declaring a transaction as “READ ONLY” increases concurrency since the transaction is not requiring locks on objects
   - SYNTAX: START TRANSACTION READ ONLY;
   - This relaxes the isolation property and ensure this transactions reads the most up to date version of an DB object
   - Addresses “hot spots” in the database
2. Lower the amount of time a lock is held by a transaction

Isolation levels

1. Lowering the amount of time a lock is held by a transaction
   - The isolation level construct allows a transaction to trade off concurrency against exposure of a transaction to other transaction's uncommitted changes
   - Isolation levels relaxes the isolation property by providing different isolation levels for read operations
   - Determines when the changes made by one transaction become visible to other transactions

SQL syntax for declaring isolation level

- The isolation level can be set at the transaction, session or at the global level.
- SET **TRANSACTION** ISOLATION LEVEL l*evels*; -- where level can be
  - SERIALIZABLE
  - REPEATABLE READ
  - READ COMMITTED
  - READ UNCOMMITTED
- EXAMPLE: SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
- Default is that the command affects the next transaction
- SYNTAX for setting at the global and session level
  - SET [GLOBAL|SESSION] TRANSACTION ISOLATION LEVEL l*evels*;
  - **GLOBAL** applies to all subsequent sessions. Existing sessions are unaffected.
  - **SESSION** applies to all subsequent transactions within the current session

SERIALIZABLE isolation level

- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- Transaction obtains locks on (sets of) accessed objects and holds both read and write locks until the end of the transaction.
- Serializability requires locks (typically intention locks) on higher level objects
- It will guarantee serializability and a recoverable schedule
- Provides the definition of the isolation property in ACID

REPEATABLE READ isolation level

- SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
- Transaction obtains locks on (sets of) accessed objects and holds both read and write locks until the end of the transaction.
  - Holds lock for the same duration as serializable isolation level, but does not lock sets of objects at higher level (no intention locks)
- Transactions may experience the phantom read phenomenon
- Default isolation level for MySQL

READ COMMITTED isolation level

- SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
- Obtains exclusive locks before writing a data object to the database and holds the exclusive lock until the end of the transaction; obtains shared locks before reading, but releases them immediately after reading
- No intention locks
- Since read locks are not held to the end of the transaction, transactions could possibly experience the nonrepeatable read phenomenon, if another transaction modified and committed a change to an object read by original transaction

READ UNCOMMITTED isolation level

- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
- Transaction does not obtain shared locks for reading
- Transaction has access to uncommitted updates made by active transactions
- Transaction is not allowed to perform any writes
  - Does not request any locks ever
- Equivalent to setting a transaction to READ ONLY

SQL Standard: Isolation levels

Creator of the transaction determines how much isolation is necessary for the current transaction

Isolation level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

InnoDB and transactions

- All user activity occurs inside a transaction
- If autocommit mode is enabled, each SQL statement forms a single transaction on its own.
- Perform a multiple-statement transaction by starting it with an explicit START TRANSACTION;
- The autocommit mode is disabled within a session with SET autocommit = 0
  - The session will have a transaction open until it is explicitly closed
  - Issue COMMIT or ROLLBACK to close the transaction
- Default InnoDB Isolation level is **REPEATABLE READ**
- InnoDB performs row level locking
  - Only if two transactions try to modify the same row does one of the transactions wait for the other to complete

InnoDB and locks

- InnoDB implements standard row-level locking where there are two types of locks
  - (S) shared locks
    - permits the transaction that holds the lock to read a row.
  - (X) exclusive locks
    - permits the transaction that holds the lock to update or delete a row.
- InnoDB supports *multiple granularity locking* which permits coexistence of record locks and locks on entire tables.
  - Intention locks are table locks in InnoDB that indicate which type of lock a transaction will require later for a row in that table.
  - Intention shared (IS) Transaction T intends to set *S* locks on individual rows in table t. (SELECT … LOCK IN SHARE MODE)
  - Intention exclusive(IX) Transaction T intends to set *X* locks on individual rows in table t (SELECT … LOCK FOR UPDATE)

InnoDB's deadlock protocols

- In Innodb, you can choose either deadlock detection or a wait timeout.
- If you choose deadlock detection, then InnoDB will automatically rollback a transaction when the deadlock occurs. It detects deadlocks by creating a wait-for graph.
- The global variable **innodb_deadlock_detect** is set to 1, for deadlock detection.
- When **innodb_deadlock_detect** is set to 0 then the lock manager assumes a transaction is deadlocked after it has waited the number of seconds equal to the value of the global variable **innodb_lock_wait_timeout.** Its default value is 50 seconds.
- To report on any deadlock conditions, use the command:
  - SHOW ENGINE INNODB STATUS;

Summary

- The InnoDB storage engine supports transactions
- MySQL allows a transaction creator, a user session or a database administrator to determine the level of Isolation for transactions
- Isolation levels controls the types of database changes an active transaction sees written by other active transactions
- MySQL implements intent lock at the table level
- MySQL provides 2 different protocols for deadlocks
- MySQL provides commands that allow you to list the transactions' locks currently active in the system