Application design

Mullins chapter 5

Bjarte Kileng

HVL

January 16, 2018

Application performance

Application performance

Poorly constructed and formulated application code accounts for the majority of relational database performance problems.

SQL execution time

Different, but logically equivalent SQL can have different execution time.

SQL – *Structured Query Language*

- SQL is declarative – We write what we want, not how to retrieve it.
- SQL versions:
  - (1986∼87) ANSI/ISO-standard
  - (1989) SQL-89 = "SQL1"
  - (1992) SQL-92 = "SQL2"
  - (1999) SQL:1999 = "SQL3"
  - (2003) SQL:2003
  - (2006) SQL:2006
  - (2008) SQL:2008
  - (2011) SQL:2011
- In general, DBMSs follow SQL standards.
- But, all commercial implementations are different.
- MySQL – See MySQL Standards Compliance

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

Access Path

- The DBMS analyzes each SQL and find data-navigational instructions – *access paths*.
- An *access path* is the "path" the system uses to retrieve and store data in a table.
- There can be more than one access path available for the same data.
- Choice of access path has no effect on the semantics (or meaning) of a statement.
- Choice of access path can have a major effect on the execution time of a statement.

Set-at-a-Time processing

- SQL quieries and data manipulations are performed on sets of data:
  - A single query return columns and rows – a table.
  - One data manipulation can modify multiple rows.
- Most programming languages act on one data record at a time.

Impedance mismatch

Programs expect data to be returned one row at a time, but SQL returns a data set at a time.

Cursor

- Most DBMSs provide a *cursor*.
- Takes input from SQL request.
- Gives a mechanism to fetch individual rows of a result set.

Embedding SQL in a program

- Programming languages need an interface for issuing SQL.
- Some programming languages support *Embedded SQL*, but most need a API (*connector*).
- Database abstraction layer:
  - Connector that can talk to many different DBMSs.
  - Provides rutines to allocate resources, execute SQL, control DBMS connections, handle transactions, etc.
- Object-relational mapping (*ORM*, *O/RM*, *O/R mapping*):
  - Connector that map database data with objects in the program.
  - E.g. Java Persistence API (*JPA*), Hibernate.

Some database abstraction layers

- ODBC – Open Database Connectivity, originally from Microsoft.
  - Available for many programming languages and DMBSs.
- JDBC – Java Database Connectivity.
- MDB2 – PHP.
- ADO – ActiveX Data Objects.

Database abstraction layers and drivers

- A database abstraction layer needs drivers to connect to a DBMS.
- Optimised for a specific DBMS.
- MDB2 – Drivers for Firebird, Frontbase, Interbase, MS SQL, MySQL, Oracle, PostgreSQL, Querysim, SQLite.

Embedded SQL

- SQL statements are written inline with the program.
- Pro*C (Oracle, Sybase) and ECPG (PostgreSQL) example:

```
{
    int a;
    /* ... */
    EXEC SQL SELECT salary INTO :a
        FROM Employee
        WHERE SSN=876543210;
    /* ... */
    printf("The salary is %d\n", a);
    /* ... */
}
```

- jOOQ embeds a SQL-like language into Java.

Connectors and DBMSs

- DBMSs vendors usually provide connectors to their DMBSs:
  - MySQL, see MySQL::MySQL Connectors.
- Connectors also for specific programming languages:
  - *libdbi* – Database abstraction layer for C.
  - PHP – Include connectors to most DMBSs.
  - Also database abstraction layers for Perl, Python, Java (JDBC).
  - Popular APIs are often copied to other programming languages:
    - *Creole* for PHP – Based on JDBC.
    - *ADOdb* for PHP – ADO for PHP.
- Client to server communication for Sybase and MS SQL requires FreeTDS on Linux.

Object orientation and SQL

- Database – Normalisation and shared data.
- OO – Encapsulation of data.
- OO and RDBMS are not inherently compatible.

Objects and relational database

- Serialization of object – Storing data using a flat file representation.
  - Also named marshalling.
- XML representation of data – Supported by many DBMSs.
- Object-relational mapping (ORM) – JPA, Hibernate.
- Object-relational database (ORD) – Objects, classes and inheritance are directly supported in schemas and in the query language.
  - Also named object-relational database management system (ORDBMS)

Types of SQL

- Planned or unplanned.
- Embedded or standalone.
- Dynamic or static.

Types of SQL – Planned or unplanned

Planned: SQL designed and tested for accuracy en efficiency before run in production.

- Typically embedded into application.

Unplanned: Created "on the fly" by end users during course of business.

- Also called *ad hoc SQL*.
- Significant source of inefficiency.

Types of SQL – Embedded or stand-alone

Embedded: Embedded within an application program.

Stand-alone: SQL run by itself within a query or reporting tool.

Types of SQL – Dynamic or static

Dynamic: Optimized at run time.

- The SQL depend on input data and the SQL is created at run time.
- Prepared statements with input parameters are dynamic. Optimization is done only once, but at run time.

Static: Optimized prior to execution.

- The full SQL statement is known at compilation.
- Can not change without reprogramming and recompilation.
- Performance of static SQL is generally better than dynamic SQL.

Questions

- Can we have static SQL with JDBC?

  No. There is no compile-time checking of SQL syntax in Java. Solutions exist though for combining static SQL with Java, e.g. pureQuery for DB2.

- Do you see any possibilities for combining static SQL with JDBC?

  We can use stored procedures. SQL in stored procedures may be optimized when stored in the DBMS before beeing run.

SQL Coding for Performance

SQL coding for performance

Let the SQL, rather than the application program do the work.

- Filter out data using the *WHERE* clause of SQL rather than copying data to the application for filtering by the application logic.
- The more filtering that is done by the DBMS, the less data has to be moved to the application.
- More later (chapter 12).

XML, XQuery and SQL/XML

- Major DBMSs support storing and managing XML data.
- XQuery – Querying and transforming XML data.
- XPath – Quering XML data.
- XSLT – Transforming XML data.
- SQL/XML – Extension to SQL standard.
  - XML data type.
  - Functions and routines for accessing and manipulating XML data in SQL databases.

Transactions

- An atomic unit of work for the DBMS:
  - Several SQL sentences can be put together and run as a unit.
  - No other concurrent connection will make the data inconsistent.
- When all the instructions have been accomplished, *COMMIT*:
  - All steps since last commit are externalised to the database.
- *ROLLBACK* – Will move transaction back to state as before the transaction was started.
- Can save an intermediate state of the transaction using *SAVEPOINT*.
- Can rollback transaction to a savepoint.
- MySQL and MariaDB – The XtraDB, InnoDB, NDB, solidDB, IBMDB2I, PrimeBase XT and Falcon engines support transactions.

ACID (Atomicity, consistency, isolation, durability)

Atomicity: All or none of the instructions of the transaction happen. The instructions is done as a single unit.

Consistency: The consistency of the state of data is preserved:

- Database go from one valid state to another.

Isolation: No transaction sees the other transactions.

- For a transaction it looks like it have the database alone.
- Many transactions can run at the same time.
- Requires a locking mechanism.

Durability: A committed transaction should withstand any kind of system failure.

More on ACID

- ACID transactions:
  - Will lock shared resources.
  - Should be short in duration.
  - Should never wait for user input when processing a transaction.
- ACID transactions and locking:
  - Due to the "isolation" requirement.
  - "Isolation" requirement is the most often relaxed ACID requirement.
- MySQL – Full ACID with the engines XtraDB, InnoDB, solidDB, IBMDB2I, PrimeBase XT, Falcon (MySQL 6).

Transaction Processing Systems

Transaction server

- Transaction control on level above DBMSs:
  - Still also transaction control in DBMSs.
- Why:
  - Environment with multiple, heterogeneous databases.

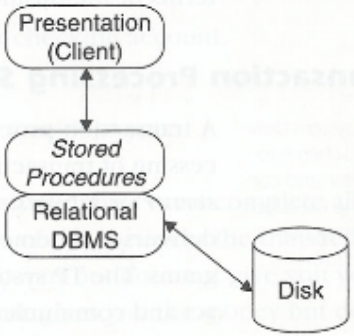

Figure 5-3 *Using a database server*

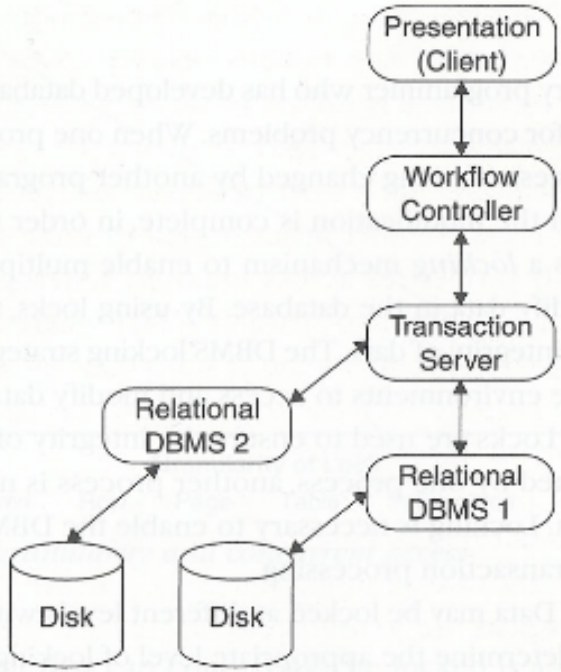

Figure 5-4 *Using a transaction server*

Transaction Processing Systems

Application server

- Include functionality to build, manage and distribute database applications.
- Usually includes features of a transaction server.
- Examples:
  - Java – *WebSphere Application Server* (IBM), *Apache Geronimo*, *Glassfish Application Server*, *JBoss* (Red Hat), *WebLogic Server* (Oracle).
  - PHP – Zend.
  - .Net – Base4, Spring Framework (Spring Framework also for Java).
  - Python – Zope.

Locking

Locking

When a user updates data and several concurrent users access the same data, locking is required in order to ensure data integrity.

- Can lock at different levels:
  - Lock granularity.
  - Column, row, page (block), table, tablespace or database locks.
- The finer the lock granularity:
  - The more concurrent access will be allowed.
  - The more resources are used by the DBMS to handle the locks.
- Observe that also gaps might be locked:
  - In XtraDB and InnoDB, if a row is locked, also the gap in indexes till the next row will normally be locked.

Locking granularity

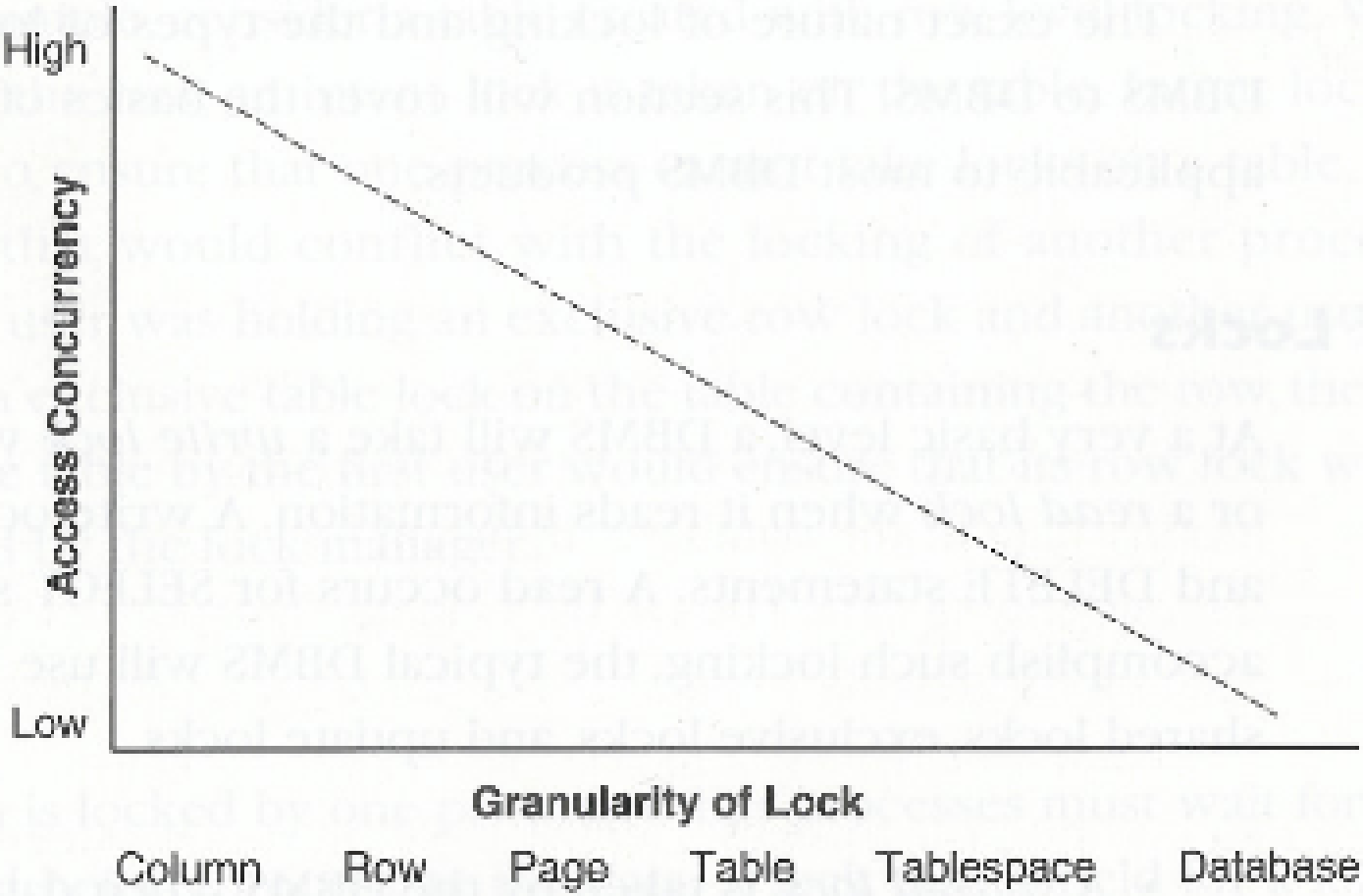

Figure 5-5 *Lock granularity and concurrent access*

MySQL and MariaDB

- All engines have the "`LOCK TABLE`".
- Of the "common" engines, only XtradB, InnoDB and NDB have row-level locking.
  - The engines solidDB, IBMDB2I, PrimeBase XT and Falcon are not in my "common" group.
- For InnoDB, see the MySQL manual "The InnoDB Transaction Model and Locking".

Types of locks

- Shared lock – Concurrent reads. No updates are allowed.
- Exclusive lock – Data is modified. No other client is granted access.
- Update lock:
  - Data must be read before it is changed or deleted.
  - Indicates that data may be changed in the future.
  - Legal – One update + multiple shared locks
  - Illegal – Update + (update or exclusive).
  - Concurrent users can read, but not change data.
  - When data is changed, the update lock is changed to an exclusive lock.
- MySQL/MariaDB – A shared lock can be updated to an exclusive lock if there are no conflicting locks from concurrent processes.

Intent locks

- Lock on table (or part of table).
- Needed when locks at different levels can cause conflicting locks:
  - One user locks some rows in a table. An intention lock is then put on the table as whole.
  - Another client tries to lock the whole table. This is refused due to the table intent lock.
- MySQL, see the manual "InnoDB Lock Modes".

Lock timeouts

- Data is locked for update by one client, then other clients must wait for the lock to be released.
- Clients will not wait forever:
  - The lock timeout value determines how long the client will wait.
  - When timeout occurs, transaction fails.
  - Application should then retry the transaction.
- MySQL – Parameter "innodb_lock_wait_timeout".

Deadlocks

- Known from DAT103.
- Circular wait.
- DBMS will chose one process to abort and rollback.
- If deadlock, application should retry the transaction.

How to avoid deadlocks

- Change the lock granularity.
- Access tables and rows in a given order (avoiding circular wait).
- Use a different isolation level (more on this next).

Deadlocks in MariaDB and MySQL

- XtraDB and InnoDB has deadlock detection. See section MySQL "Deadlock Detection and Rollback".
- A deadlock involving InnoDB or XtraDB tables toghether with tables from other engines may not be detected.
- Cause of deadlock can be found using:

```
SHOW ENGINE INNODB STATUS
```

- The above command requires the *SUPER* privilege.

Minimise Locking Problems

- Standardise the sequence of updates:
  - Will avoid deadlocks.
  - The deadlock condition *Circular wait* will then not occur.
- Do all data modification requests at the end of the transaction:
  - Can reduce the lock duration.

Isolation Levels

- ACID only with isolation level *SERIALIZABLE*.
  - The other levels break the isolation requirement.
- The levels are, from lowest to highest isolation:
  1. *READ UNCOMMITTED*,
  2. *READ COMMITTED*,
  3. *REPEATABLE READ*,
  4. *SERIALIZABLE*.

Extract from the Oracle documentation

From Introduction to Data Concurrency and Consistency.

In short, real-world considerations usually require a compromise between perfect transaction isolation and performance.

Setting the isolation level

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
  {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}
```

MySQL engines and isolation levels

- XtraDB, InnoDB and Falcon (MySQL 6) have all levels.
- Default isolation level in InnoDB is *REPEATABLE READ*.
  - Other DBMSs often has *READ COMMITTED* as the default level.
- NDB support only the *READ COMMITTED*.
- The other frequently used engines lack support for transactions.
- solidDB, IBMDB2I, PrimeBase XT and Falcon are full ACID, but I do not know the details of these engines.

READ UNCOMMITTED

- Also named *dirty read*.
- Has the highest level of availability and concurrency.
- Worst degree of data integrity.
- Will read data changed by other transactions still not committed.
- Only to be used when integrity problems can be tolerated:
  - Finding averages, making statistics, data warehouse.

READ COMMITTED

- Also called *cursor stability*.
- Default level for many DBMSs.
- A transaction can read data committed by another transaction.

REPEATABLE READ

- Data read in the transaction will not change during the transaction.
- Can be implemented using shared locks.
- No range locks.
- Phantom reads are possible.

Phantom reads

- Two identical reads in a transaction can return different rows.
- Transaction T_1 read a range of rows and acquire shared locks on the rows, but no range locks.
- Transaction T_2 insert a new row inside the range already read by T_1.
- Transaction T_1 issues the first read once more.
- Now, the resultset has changed even though the two reads by T_1 were identical. The new row inserted by T_2 will now appear in the new resultset.

REPEATABLE READ with MySQL and MariaDB

- The default isolation level.
- MySQL uses a kind of snapshot isolation.
- The first read of data creates a snapshot of the read data.
- Following reads of the same data will read from the snapshot.
- The MySQL and MariaDB implementation do probably not conform to the ANSI/ISO SQL standard for *REPEATABLE READ*.
- *REPEATABLE READ* closer to the ANSI/ISO SQL standard is probably obtained through *SERIALIZABLE* if using:

```
innodb_locks_unsafe_for_binlog=1
```

See the MySQL manual for "innodb_locks_unsafe_for_binlog".

SERIALIZABLE

- Highest isolation.
- Removes the possibilities for phantoms.
- Require locks on data and data ranges.
- Will guarantee that one transaction will finish without any other transaction changing the data used by the transaction.

SERIALIZABLE with InnoDB and XtraDB

- All SELECTs are transformed to:

```
SELECT ... LOCK IN SHARE MODE
```

- MariaDB and InnoDB lock implementation will acquire the necessary range locks to avoid phantom reads.
- See the MySQL manual, "InnoDB Record, Gap, and Next-Key Locks".

SERIALIZABLE in Oracle and PostgreSQL

- Uses snapshot isolation.
- Has similarities with the snapshot model of MySQL and MariaDB for *REPEATABLE READ*.
- The transaction will successfully commit only if the updates do not conflict with updates from other transactions on the same data
- Not truly serializable.
- Can give inconsistencis between concurrent transactions, e.g. the *Write Skew Anomaly*.
  - PostgreSQL is protected against the simple *Write Skew Anomaly* (ref) but with more than two concurrent transactions, problems can still occure (ref).
  - See also the Oracle documentation on *Data Concurrency and Consistency*.

Locks and resources

Locks and resources

With many locks, the DBMS might spend much resources (storage) to handle the locks.

Using lock escalation

- Increase the lock granularity:
  - E.g. locks on rows are replaced with one lock on the whole table.
- Will save space since fewer locks have to be stored.
- Will lock more data:
  - Concurrent access will suffer.

Implementation in DBMSs

- Lock escalation is used by some DBMSs:
  - DB2, Microsoft SQL Server.
  - System parameters determine how lock escalation is done.
- MariaDB, MySQL and Oracle – No automatic lock escalation:
  - Can be done by the application.
  - InnoDB – Lock escalation is not needed due to efficient storage of locks, see "The InnoDB Transaction Model and Locking".

Batch processing

- No online interaction.
- Do *COMMIT* periodically:
  - Will release locks and increase concurrency availability.
  - If batch program fails, can restart from the last *COMMIT*.
- Must be easy to restart the program from the last *COMMIT*:
  - The program must track its own progress.
- Schedule the batch jobs to run during off-peak online processing hours.