Database Systems - Concurrency Control Techniques -

Agnès Voisard
Institute of Computer Science,
Databases and Information Systems Group
and Fraunhofer FOKUS

2023
v3

Notes

Notes

Contents

Motivation

- Ensures non-interference or isolation of concurrently executing transactions
- Ensures serializability of schedules using protocols
- Set of protocols can use:
    - Techniques of locking data items
    - Transaction timestamps
      (unique identifier for each transaction and timestamp ordering to ensure serializability)
    - Multiversions of data items

Locking Techniques

Lock: Variable associated with a data item X in the DB.
Describes the status of that item with respect to possible operations that can be applied to it.
Notation: Lock(X)

Guarantees exclusive use of a data item to a current transaction.

DBMS has a **lock manager** subsystem: Keeps track of locks and controls access to locks.

Locking Techniques (cont'd)

Binary locks: 2 states for Lock(X) (value of the lock):

- Locked (1): X cannot be accessed by a DB operation that requires X. Transaction has to wait. Lock(X) := 1.
- Or unlocked (0): X can be accessed when requested. Lock(X) := 0.

2 operations included in the transactions:
`lock-item(X)` and `unlock-item(X)`

If X is locked, transactions waiting to access it are put on a waiting queue for X until it is unlocked.

Binary Locks

Every transaction T must obey the rules:

- T issues `lock-item(X)` before any read-item(X) or write-item(X)
- T issues `unlock-item(X)` after all read-item(X) or write-item(X) operations are completed in T
- T will not issue a `lock-item` if it already **holds the lock** on X
- T will not issue an `unlock-item` unless it already holds the lock on X

At most 1 transaction can hold the lock on a particular item.

Lock table:
Records `[data-item, LOCK, T]` and a queue for waiting transactions.

Shared and Exclusive Locks

Previous solution too restrictive.
Several transactions should be able to access X for reading purposes (only, i.e., still exclusive access to X when T writes an item X).

Multiple-mode lock. 3 locking operations:
`read-lock(X)`; "read-locked" item is also called share-locked (shared): Other transactions are allowed to read it.

`write-lock(X)`; "write-locked" item is also called exclusive-locked:
A single transaction holds the lock. Used when potential for conflict exists.

`unlock(X)`

Shared and exclusive locks (cont'd)

Rules enforced by the system:

1. T must issue `read-lock(X)` or `write-lock(X)` before any read-item(X).
2. T must issue `write-lock(X)` before any write-item(X).
3. T must issue `unlock(X)` after all read-item(X) and write-item(X).
4. T will not issue a `read-lock(X)` or a `write-lock(X)` if it already holds a a read lock or a write lock on X.
5. T will not issue an `unlock(X)` unless it already holds a read lock or a write lock on X.

Locking

Remarks:
Locks prevent serious data inconsistencies, but
Schedules may create deadlocks
(2 transactions wait for each other to unlock data)

Using locks (binary or multiple-mode) does not guarantee serializability of schedules in which the transactions participate.

2-Phase locking to guarantee serializability

(does not prevent deadlocks)
Lock upgrade: From shared (read) to exclusive (write)
Lock downgrade: From exclusive to shared
2-Phase locking protocol: Defines the way transactions acquire and release locks.
2 phases:

- **Growing phase**: Transaction acquires all the required locks without unlocking any data.
  Once all locks are acquired: Transaction is in its **locked point**.
- **Schrinking phase**: Transaction releases all locks and cannot obtain new locks.

Picture on black board.

Variations of 2-phase locking

◇ Previous method: **Basic 2PL**

◇ **Conservative 2PL** (static 2PL):
Transaction must *lock all the items it accesses before* it begins execution.
(predeclaration of items reads and writes)
Deadlock-free protocol.

◇ Most popular version of 2PL: **Strict 2PL**
A transaction T *does not release any of its locks until after it commits or aborts*.
I.e., no other transaction can read or write an item that is written by T unless T has committed.

2-Phase locking to guarantee serializability (cont'd)

Theorem: If every transaction in a schedule S follows the conservative 2-phase locking protocol, S is serializable.

Proof: A conservative 2PL schedule is equivalent to the serial schedule in which each transaction runs instantaneously at the time that it commits.

Limits the concurrency in a schedule.

Deadlocks

When?

Each of 2 transactions is waiting for the other to release the lock on an item.

Deadlocks (cont')

Example: Partial schedule in the state of a deadlock

```
      T1                         T2

1. read-lock(Y);
2. read-item(Y);
3.                             read-lock(X);
4.                             read-item(X);
5. write-lock(X);
6.                             write-lock(Y);
```

Line 5: T1 waits for T2 to release X. Line 6: T2 waits for T1 to release Y.

Deadlocks (cont'd)

3 basic techniques to control deadlock:

- DEADLOCK AVOIDANCE
  A transaction T requesting a new lock is aborted if there is a possibility of deadlock.
  If T aborted, changes made by it are rolled back and locks obtained by T are released, and T is
  rescheduled for execution.
  Works because it avoids the conditions that lead to deadlocking.
- DEADLOCK DETECTION
  DBMS periodically tests the DB for deadlocks. If one is found, a victim is aborted (rolled back + restarted) and the other ones continues.

Deadlocks (cont'd)

- DEADLOCK PREVENTION
  The transaction must obtain all the locks it needs before it can be executed.
  Avoids rollback of conflicting transactions.
  But the serial lock assignment increases action-response times. Limits concurrency (conservative 2PL).

Livelocks

Transaction in a state of **livelock** if it cannot proceed for an indefinite period of time while other transactions continue. Occurs when the waiting scheme is unfair: priority of some transactions over others.

Standard solution: First-come-first-serve queue,
or a transaction gets a high priority the longer it waits.

Starvation: When dealing with deadlocks, the victim is always the same.

Granularity of Data Items

Database item:

- Database record
- Field value of a DB record
- Disk block
- Whole file
- Whole database

Size of data items: **Data item granularity**
Best size depends on the types of transactions involved.
E.g., if a typical transaction accesses a small number of records, granularity = one record.
If many records of the same file are considered, better to have block or file granularity.

Granularity of Data Items (cont'd)

The larger the data item size is, the lower is the degree of concurrency permitted ("too much is blocked").

Example:
Size is a disk block.
T1 needs to lock a record A.
T1 will lock the whole disk block X that contains A.
T2 needs to lock B that is also in X. It must wait until T1 releases X.
If the data item size was a single record, T2 would lock a different item (i.e., record B) than T1 (i.e., record A).

Granularity of Data Items (cont'd)

But the smaller the data item size is, the more items will exist in the DB.
System will have a larger number of locks handled by the lock manager.
More storage place for the lock table.

Summary

- Locks
- 2PL
- Deadlocks
- Granularity of data items

What will come next?

1. Welcome to Database Systems
2. Introduction to Database Systems
3. Entity Relationship Design Diagram (ERM)
4. Relational Model
5. Relational Algebra
6. Structured Query Language (SQL)
7. Relational Database Design - Functional Dependencies
8. Relational Database Design - Normalization
9. Online Analytical Processing + Embedded SQL
10. Data Mining
11. Physical Representation - Storage and File Structure
12. Physical Representation - Indexing and Hashing
13. Transactions
14. Concurrency Control Techniques
15. Recovery Techniques
16. Query Processing and Optimization