Database System Concepts 5th Ed.
© Silberschatz, Korth and Sudarshan, 2005
See www.db-book.com for conditions on re-use

Chapter 16: Concurrency Control

Lock-based Concurrency Control

- A DBMS must ensure
  - Only **serializable** (and **recoverable**) schedules are allowed, and
  - no actions of committed transaction is lost (even while undoing aborting transactions)

⇒ Usually uses a **locking protocol**

A **locking protocol** is a set of rules to be followed by transactions, enforced by DBMS, to guarantee **serializability** and **recoverability** (**safe interleaving**).

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes:
  1. **exclusive** (X) mode. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  2. **shared** (S) mode. Data item can only be read. S-lock is requested using **lock-S** instruction.
- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Lock-Based Protocols (Cont.)

- **Lock-compatibility matrix**

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>X</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
  - but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.
**Strict Two-Phase Locking**

- The most widely used locking protocol

**Strict 2PL:**

1. If T wants to read A, needs shared lock A.
2. If T wants to write A, needs exclusive lock A.
3. All T’s locks are released when T is completed.
   - i.e. committed or aborted

- T is blocked until the DBMS is able to grant it the appropriate lock
- If T has X(A), does not need S(A)
- If T has X(A), T’ can not be granted S(A) or X(A)

---

**A Non Strict-2PL Example**

\[ T_2 : \text{lock-S}(A); \]
\[ \quad \text{read}(A); \]
\[ \quad \text{unlock}(A); \]
\[ \quad \text{lock-S}(B); \]
\[ \quad \text{read}(B); \]
\[ \quad \text{unlock}(B); \]
\[ \quad \text{display}(A+B) \]

- Locking as above is not sufficient to guarantee serializability — if A and B get updated in-between the read of A and B, the displayed sum would be wrong.
- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

---

**Strict 2PL & Safe Interleaving**

- Strict 2PL can avoid anomalies. For instance:

\[
\begin{array}{c|c|c|c}
T_1 & T_2 & T_1 & T_2 \\
\hline
R(A) & W(A) & X(A) & R(A), W(A) \\
R(A) & W(A) & R(B) & R(B), W(B) \\
R(B) & W(B) & C & C \\
\end{array}
\]

---

**Pitfalls of Lock-Based Protocols**

- Consider the partial schedule

\[
\begin{array}{c|c}
T_3 & T_4 \\
\hline
\text{lock-X}(B) & \text{lock-S}(A) \\
\text{read}(B) & \text{read}(A) \\
B \rightarrow B = 50 & \text{lock-X}(A) \\
\text{write}(B) & \text{lock-S}(B) \\
\end{array}
\]

- Neither T_3 nor T_4 can make progress — executing lock-S(B) causes T_4 to wait for T_3 to release its lock on B, while executing lock-X(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a deadlock.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.
Deadlocks

- Blocking (suspending) transactions can result in a deadlock.
- General example:
  - T₁ is granted X(A)
  - T₂ is granted X(B)
  - T₁ needs X(B), therefore blocked.
  - T₂ needs X(A), therefore, blocked.
- DBMS must either
  - prevent such deadlocks, or
  - detect and resolve it.

A deadlock can be resolved by only aborting (and restarting) one of the transactions in the cycle.

Starvation

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- Starvation is also possible if concurrency control manager is badly designed. For example:
  - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

The 2PL Protocol (Revisit)

- This is a protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their lock points (i.e. the point where a transaction acquired its final lock).

The 2PL Protocol (Cont.)

- Two-phase locking does not ensure freedom from deadlocks
- Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called strict two-phase locking. Here a transaction must hold all its exclusive locks till it commits/aborts.
- Rigorous two-phase locking is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.
The 2PL Protocol (Cont.)

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used.
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
  Given a transaction $T_i$ that does not follow two-phase locking, we can find a transaction $T_j$ that uses two-phase locking, and a schedule for $T_i$ and $T_j$ that is not conflict serializable.

Lock Conversions

- Two-phase locking with lock conversions:
  - First Phase:
    * can acquire a lock-S on item
    * can acquire a lock-X on item
    * can convert a lock-S to a lock-X (upgrade)
  - Second Phase:
    * can release a lock-S
    * can release a lock-X
    * can convert a lock-X to a lock-S (downgrade)
  - This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Automatic Acquisition of Locks

- A transaction $T_i$ issues the standard read/write instruction, without explicit locking calls.
- The operation read($D$) is processed as:
  - if $T_i$ has a lock on $D$
    - then read($D$)
  - else begin
    - if necessary wait until no other transaction has a lock-X on $D$
      - grant $T_i$ a lock-S on $D$;
      - read($D$)
    - end

- write($D$) is processed as:
  - if $T_i$ has a lock-X on $D$
    - then write($D$)
  - else begin
    - if necessary wait until no other trans. has any lock on $D$,
      - if $T_i$ has a lock-S on $D$
        - then upgrade lock on $D$ to lock-X
      - else
        - grant $T_i$ a lock-X on $D$
        - write($D$)
    - end;
  - All locks are released after commit or abort.
Locking and Performance

- Lock-based protocols are based on
  - Blocking and Aborting
  ⇒ both involve performance penalty

- **Throughput** can be increased by
  - Locking the smallest sized objects possible
  - Reducing the holding lock time
  - Reducing hot spots

SQL Isolation Levels (revisit)

- SQL defines four choices of isolation
  - To relax serializability
  - It can be specified for each transaction

**Four Isolation Levels**

0: **READ UNCOMMITTED**
  - T may see uncommitted writes

1: **READ COMMITTED**
  - T only sees committed writes, but unrepeatable reads may exist

2: **REPEATABLE READ**
  - Reads are possible to repeat, but T may experience **phantoms**

3: **SERIALIZABILITY**
  - None of the anomalies are possible

Phantoms

- **Application 1:**
  INSERT INTO food
  VALUES (‘Sushi’, ‘Japanese’);

- **Application 2:**
  SELECT * FROM food;
  SELECT * FROM food
  WHERE cuisine=’Japanese’;

**Problem.** Second query of Application 2 may see sushi, even though its first query does not.

Isolation Levels & Locks

- **T, SERIALIZABLE**
  - T obtains S(A) and X(A), as well as locks for set of objects
  - Holds the locks until the end

- **T, REPEATABLE READ**
  - T obtains S(A) and X(A)
  - Holds the locks until the end

- **T, READ COMMITTED**
  - T obtains S(A) and X(A)
  - Release S(A) immediately, but holds X(A) until the end

- **T, READ UNCOMMITTED**
  - Does not obtain S(A)
  - T’s access mode is **READ ONLY**, no writes are allowed then