Chapter 15  Concurrency Control

- Validation phase
- Write phase
- Validation test
- Multiversion timestamp ordering
- Multiversion two-phase locking
  - Read-only transactions
  - Update transactions
- Snapshot isolation
  - Lost update
  - First committer wins
  - First updater wins
  - Write skew
  - Select for update
- Insert and delete operations
- Phantom phenomenon
- Index-locking protocol
- Predicate locking
- Weak levels of consistency
  - Degree-two consistency
  - Cursor stability
- Optimistic concurrency control without read validation
- Conversations
- Concurrency in indices
  - Crabbing
  - B-link trees
  - B-link-tree locking protocol
  - Next-key locking

Practice Exercises

15.1  Show that the two-phase locking protocol ensures conflict serializability, and that transactions can be serialized according to their lock points.

15.2  Consider the following two transactions:

\[ T_{34}: \text{read}(A); \]
\[ \quad \text{read}(B); \]
\[ \quad \text{if } A = 0 \text{ then } B := B + 1; \]
\[ \quad \text{write}(B). \]

\[ T_{35}: \text{read}(B); \]
\[ \quad \text{read}(A); \]
\[ \quad \text{if } B = 0 \text{ then } A := A + 1; \]
\[ \quad \text{write}(A). \]

Add lock and unlock instructions to transactions \( T_{31} \) and \( T_{32} \), so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?

15.3  What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking?
15.4 Consider a database organized in the form of a rooted tree. Suppose that we insert a dummy vertex between each pair of vertices. Show that, if we follow the tree protocol on the new tree, we get better concurrency than if we follow the tree protocol on the original tree.

15.5 Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol, and vice versa.

15.6 Consider the following extension to the tree-locking protocol, which allows both shared and exclusive locks:

- A transaction can be either a read-only transaction, in which case it can request only shared locks, or an update transaction, in which case it can request only exclusive locks.

- Each transaction must follow the rules of the tree protocol. Read-only transactions may lock any data item first, whereas update transactions must lock the root first.

Show that the protocol ensures serializability and deadlock freedom.

15.7 Consider the following graph-based locking protocol, which allows only exclusive lock modes, and which operates on data graphs that are in the form of a rooted directed acyclic graph.

- A transaction can lock any vertex first.

- To lock any other vertex, the transaction must be holding a lock on the majority of the parents of that vertex.

Show that the protocol ensures serializability and deadlock freedom.

15.8 Consider the following graph-based locking protocol, which allows only exclusive lock modes and which operates on data graphs that are in the form of a rooted directed acyclic graph.

- A transaction can lock any vertex first.

- To lock any other vertex, the transaction must have visited all the parents of that vertex and must be holding a lock on one of the parents of the vertex.

Show that the protocol ensures serializability and deadlock freedom.

15.9 Locking is not done explicitly in persistent programming languages. Rather, objects (or the corresponding pages) must be locked when the objects are accessed. Most modern operating systems allow the user to set access protections (no access, read, write) on pages, and memory access that violate the access protections result in a protection violation (see the Unix mprotect command, for example). Describe how the access-protection mechanism can be used for page-level locking in a persistent programming language.
Consider a database system that includes an atomic increment operation, in addition to the read and write operations. Let $V$ be the value of data item $X$. The operation

\[
\text{increment}(X) \text{ by } C
\]

sets the value of $X$ to $V + C$ in an atomic step. The value of $X$ is not available to the transaction unless the latter executes a read($X$). Figure 15.23 shows a lock-compatibility matrix for three lock modes: share mode, exclusive mode, and incrementation mode.

### Figure 15.23 Lock-compatibility matrix.

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

15.10. Show that, if all transactions lock the data that they access in the corresponding mode, then two-phase locking ensures serializability.

15.11. In timestamp ordering, W-timestamp($Q$) denotes the largest timestamp of any transaction that executed write($Q$) successfully. Suppose that, instead, we defined it to be the timestamp of the most recent transaction to execute write($Q$) successfully. Would this change in wording make any difference? Explain your answer.

15.12. Use of multiple-granularity locking may require more or fewer locks than an equivalent system with a single lock granularity. Provide examples of both situations, and compare the relative amount of concurrency allowed.

15.13. Consider the validation-based concurrency-control scheme of Section 15.5. Show that by choosing Validation($T_i$), rather than Start($T_i$), as the timestamp of transaction $T_i$, we can expect better response time, provided that conflict rates among transactions are indeed low.

15.14. For each of the following protocols, describe aspects of practical applications that would lead you to suggest using the protocol, and aspects that would suggest not using the protocol:

- Two-phase locking.
- Two-phase locking with multiple-granularity locking.
Practice Exercises

• The tree protocol.
• Timestamp ordering.
• Validation.
• Multiversion timestamp ordering.
• Multiversion two-phase locking.

15.15 Explain why the following technique for transaction execution may provide better performance than just using strict two-phase locking: First execute the transaction without acquiring any locks and without performing any writes to the database as in the validation-based techniques, but unlike the validation techniques do not perform either validation or writes on the database. Instead, rerun the transaction using strict two-phase locking. (Hint: Consider waits for disk I/O.)

15.16 Consider the timestamp-ordering protocol, and two transactions, one that writes two data items $p$ and $q$, and another that reads the same two data items. Give a schedule whereby the timestamp test for a write operation fails and causes the first transaction to be restarted, in turn causing a cascading abort of the other transaction. Show how this could result in starvation of both transactions. (Such a situation, where two or more processes carry out actions, but are unable to complete their task because of interaction with the other processes, is called a livelock.)

15.17 Devise a timestamp-based protocol that avoids the phantom phenomenon.

15.18 Suppose that we use the tree protocol of Section 15.1.5 to manage concurrent access to a B+ -tree. Since a split may occur on an insert that affects the root, it appears that an insert operation cannot release any locks until it has completed the entire operation. Under what circumstances is it possible to release a lock earlier?

15.19 The snapshot isolation protocol uses a validation step which, before performing a write of a data item by transaction $T$, checks if a transaction concurrent with $T$ has already written the data item.

   a. A straightforward implementation uses a start timestamp and a commit timestamp for each transaction, in addition to an update set, that is the set of data items updated by the transaction. Explain how to perform validation for the first-committer-wins scheme by using the transaction timestamps along with the update sets. You may assume that validation and other commit processing steps are executed serially, that is for one transaction at a time,

   b. Explain how the validation step can be implemented as part of commit processing for the first-committer-wins scheme, using a modification of the above scheme, where instead of using update sets, each data item has a write timestamp associated with it. Again, you
may assume that validation and other commit processing steps are executed serially.

c. The first-updater-wins scheme can be implemented using timestamps as described above, except that validation is done immediately after acquiring an exclusive lock, instead of being done at commit time.
   i. Explain how to assign write timestamps to data items to implement the first-updater-wins scheme.
   ii. Show that as a result of locking, if the validation is repeated at commit time the result would not change.
   iii. Explain why there is no need to perform validation and other commit processing steps serially in this case.

Exercises

15.20 What benefit does strict two-phase locking provide? What disadvantages result?

15.21 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.

15.22 Consider a variant of the tree protocol called the forest protocol. The database is organized as a forest of rooted trees. Each transaction $T_i$ must follow the following rules:
   • The first lock in each tree may be on any data item.
   • The second, and all subsequent, locks in a tree may be requested only if the parent of the requested node is currently locked.
   • Data items may be unlocked at any time.
   • A data item may not be relocked by $T_i$ after it has been unlocked by $T_i$.

Show that the forest protocol does not ensure serializability.

15.23 Under what conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them?

15.24 If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain your answer.

15.25 In multiple-granularity locking, what is the difference between implicit and explicit locking?

15.26 Although SIX mode is useful in multiple-granularity locking, an exclusive and intention-shared (XIS) mode is of no use. Why is it useless?
15.27 The multiple-granularity protocol rules specify that a transaction \( T_i \) can
lock a node \( Q \) in S or IS mode only if \( T_i \) currently has the parent of \( Q \)
locked in either IX or IS mode. Given that SIX and S locks are stronger than
IX or IS locks, why does the protocol not allow locking a node in S or IS
mode if the parent is locked in either SIX or S mode?

15.28 When a transaction is rolled back under timestamp ordering, it is assigned
a new timestamp. Why can it not simply keep its old timestamp?

15.29 Show that there are schedules that are possible under the two-phase lock-
ing protocol, but are not possible under the timestamp protocol, and vice
versa.

15.30 Under a modified version of the timestamp protocol, we require that a
commit bit be tested to see whether a read request must wait. Explain how
the commit bit can prevent cascading abort. Why is this test not necessary
for write requests?

15.31 As discussed in Exercise 15.19, snapshot isolation can be implemented
using a form of timestamp validation. However, unlike the multiversion
timestamp-ordering scheme, which guarantees serializability, snapshot
isolation does not guarantee serializability. Explain what is the key differ-
ence between the protocols that results in this difference.

15.32 Outline the key similarities and differences between the timestamp based
implementation of the first-committer-wins version of snapshot isola-
tion, described in Exercise 15.19, and the optimistic-concurrency-control-
without-read-validation scheme, described in Section 15.9.3.

15.33 Explain the phantom phenomenon. Why may this phenomenon lead to
an incorrect concurrent execution despite the use of the two-phase locking
protocol?

15.34 Explain the reason for the use of degree-two consistency. What disadvan-
tages does this approach have?

15.35 Give example schedules to show that with key-value locking, if any of
lookup, insert, or delete do not lock the next-key value, the phantom
phenomenon could go undetected.

15.36 Many transactions update a common item (e.g., the cash balance at a
branch), and private items (e.g., individual account balances). Explain
how you can increase concurrency (and throughput) by ordering the op-
erations of the transaction.

15.37 Consider the following locking protocol: All items are numbered, and
once an item is unlocked, only higher-numbered items may be locked.
Locks may be released at any time. Only X-locks are used. Show by an
example that this protocol does not guarantee serializability.
Bibliographical Notes

Gray and Reuter [1993] provides detailed textbook coverage of transaction-processing concepts, including concurrency-control concepts and implementation details. Bernstein and Newcomer [1997] provides textbook coverage of various aspects of transaction processing including concurrency control.

The two-phase locking protocol was introduced by Eswaran et al. [1976]. The tree-locking protocol is from Silberschatz and Kedem [1980]. Other non-two-phase locking protocols that operate on more general graphs are described in Yannakakis et al. [1979], Kedem and Silberschatz [1983], and Buckley and Silberschatz [1985]. Korth [1983] explores various lock modes that can be obtained from the basic shared and exclusive lock modes.

Practice Exercise 15.4 is from Buckley and Silberschatz [1984]. Practice Exercise 15.6 is from Kedem and Silberschatz [1983]. Practice Exercise 15.7 is from Kedem and Silberschatz [1979]. Practice Exercise 15.8 is from Yannakakis et al. [1979]. Practice Exercise 15.10 is from Korth [1983].

The locking protocol for multiple-granularity data items is from Gray et al. [1975]. A detailed description is presented by Gray et al. [1976]. Kedem and Silberschatz [1983] formalizes multiple-granularity locking for an arbitrary collection of lock modes (allowing for more semantics than simply read and write). This approach includes a class of lock modes called update modes to deal with lock conversion. Carey [1983] extends the multiple-granularity idea to timestamp-based concurrency control. An extension of the protocol to ensure deadlock freedom is presented by Korth [1982].

The timestamp-based concurrency-control scheme is from Reed [1983]. A timestamp algorithm that does not require any rollback to ensure serializability is presented by Buckley and Silberschatz [1983]. The validation concurrency-control scheme is from Kung and Robinson [1981].

Multiversion timestamp order was introduced in Reed [1983]. A multiversion tree-locking algorithm appears in Silberschatz [1982].

Degree-two consistency was introduced in Gray et al. [1975]. The levels of consistency—or isolation—offered in SQL are explained and critiqued in Berenson et al. [1995]. Many commercial database systems use version-based approaches in combination with locking. PostgreSQL, Oracle, and SQL Server all support forms of the snapshot isolation protocol mentioned in Section 15.6.2. Details can be found in Chapters 27, 28, and 30, respectively.

It should be noted that on PostgreSQL (as of version 8.1.4) and Oracle (as of version 10g), setting the isolation level to serializable results in the use of snapshot isolation, which does not guarantee serializability. Fekete et al. [2005] describes how to ensure serializable executions under snapshot isolation, by rewriting certain transactions to introduce conflicts; these conflicts ensure that the transactions cannot run concurrently under snapshot isolation; Jorwekar et al. [2007] describes an approach, that given a set of (parametrized) transactions running under snapshot isolation, can check if the transactions are vulnerability to nonserializability.
Concurrency in B⁺-trees was studied by Bayer and Schkolnick [1977] and Johnson and Shasha [1993]. The techniques presented in Section 15.10 are based on Kung and Lehman [1980] and Lehman and Yao [1981]. The technique of key-value locking used in ARIES provides for very high concurrency on B⁺-tree access and is described in Mohan [1990a] and Mohan and Narang [1992]. Ellis [1987] presents a concurrency-control technique for linear hashing.