Database System Concepts

Lock-Based Protocols

Deadlock Handling

Insert and Delete

Database System Concepts

Chapter 16: Concurrency Control

Departamento de Engenharia Informática
Instituto Superior Técnico

1st Semester
2009/2010

Slides (fortemente) baseados nos slides oficiais do livro "Database System Concepts"

Outline

1. Lock-Based Protocols
   - Two-Phase Locking
   - Graph-Based Protocols
   - Timestamp-Based Protocols
2. Deadlock Handling
   - Deadlock Prevention
   - Deadlock Detection
   - Deadlock Recovery
3. Insert and Delete

Outline

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols
Deadlock Handling
Insert and Delete

1. Lock-Based Protocols
   - Two-Phase Locking
   - Graph-Based Protocols
   - Timestamp-Based Protocols
2. Deadlock Handling
3. Insert and Delete

Lock-Based Protocols

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

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

Lock Compatibility

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- 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

Lock-compatibility matrix

	S	**X**
S	true	false
X	false	false

Locking and Transactions

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

Example

	T_2
1.	lock-S(A)
2.	read(A)
3.	unlock(A)
4.	lock-S(B)
5.	read(B)
6.	unlock(B)
7.	display($A + B$)

- Locking is not sufficient to guarantee serializability
- We need to use a locking protocol— a set of rules followed by all transactions for requesting and releasing locks

Pitfalls of Lock-Based Protocols

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- Consider the schedule:

	T_3	T_4
1.	lock-X(B)	
2.	read(B)	
3.	$B := B - 50$	
4.	write(B)	
5.		lock-S(A)
6.		read(A)
7.		lock-S(B)
8.	lock-X(A)	
9.	...	...

- Neither T_3 nor T_4 can make progress!
- 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

Pitfalls of Lock-Based Protocols (cont.)

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- 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 Two-Phase Locking Protocol

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- Transactions have two phases:
  - **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
- This is a protocol which *ensures conflict-serializable schedules*
  - It can be proved that the transactions can be serialized in the order of their lock points— the point where a transaction acquired its final lock
  - There can be conflict serializable schedules that cannot be obtained if two-phase locking is used

An Example

Database System Concepts

Lock-Based Protocols

Two-Phase Locking

Graph-Based Protocols

Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

	T_5	T_6	T_7
1.	lock-X(A)		
2.	read(A)		
3.	lock-S(B)		
4.	read(B)		
5.	write(A)		
6.	unlock(A)		
7.		lock-X(A)	
8.		read(A)	
9.		write(A)	
10.		unlock(A)	
11.			lock-S(A)
12.			read(A)
13.	unlock(B)		
14.			unlock(A)

Variations of Two-Phase Locking

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

Lock Conversions

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- 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

Graph-Based Protocols

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols
Deadlock Handling
Insert and Delete

- Graph-based protocols are an alternative to two-phase locking
- Impose a partial ordering ($\rightarrow$) on the set $D = d_1, d_2, \ldots, d_h$ of all data items
- If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j
- Implies that the set D may now be viewed as a directed acyclic graph, called a database graph
- An example: the tree-protocol

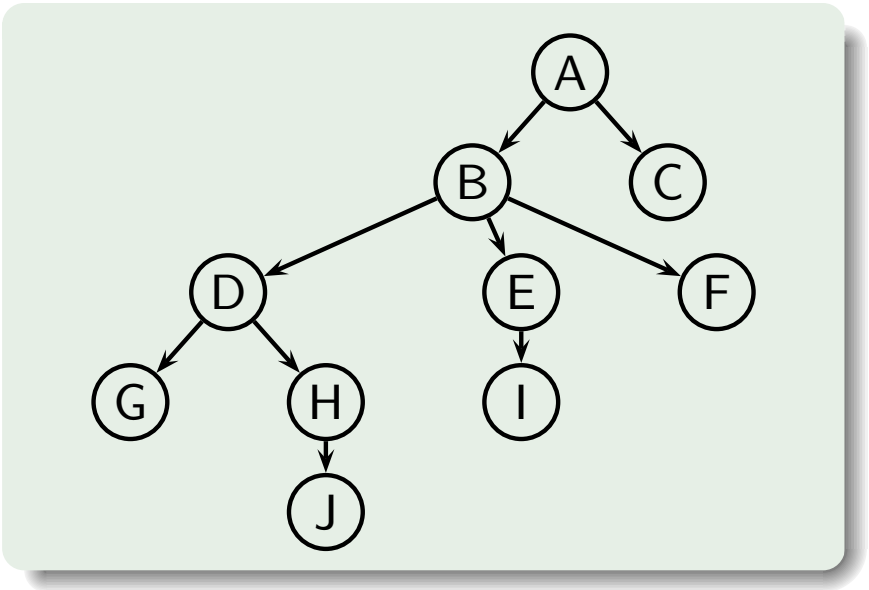
The Tree Protocol

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete



- The database graph is a rooted tree
- Only exclusive locks are allowed
- The protocol enforces the following rules:
  1. The first lock by T_i may be on any data item
  2. Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i
  3. Data items may be unlocked at any time
  4. A data item that was unlocked by T_i may not be locked by T_i again

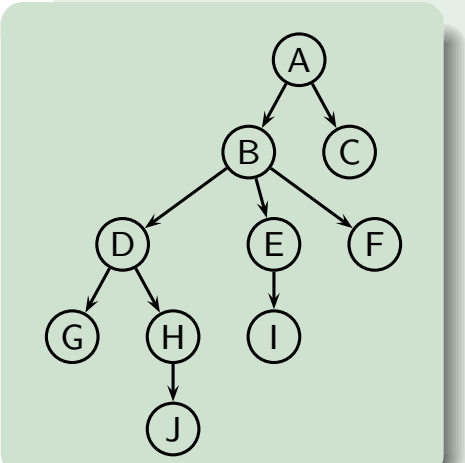
An Example

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete



	T_{10}	T_{11}	T_{12}	T_{13}
1.	lock-X(B)			
2.		lock-X(D)		
3.		lock-X(H)		
4.		unlock(D)		
5.	lock-X(E)			
6.	lock-X(D)			
7.	unlock(B)			
8.	unlock(E)			
9.			lock-X(B)	
10.			lock-X(E)	
11.		unlock(H)		
12.	lock-X(G)			
13.	unlock(D)			
14.				lock-X(D)
15.				lock-X(H)
16.				unlock(D)
17.				unlock(H)
18.			unlock(E)	
19.			unlock(B)	
20.	unlock(G)			

Properties of the Tree-Based Protocol

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- The tree protocol *ensures conflict serializability as well as freedom from deadlock*
- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol
  - shorter waiting times, and increase in concurrency
  - protocol is deadlock-free, no rollbacks are required
- Drawbacks
  - Protocol does not guarantee recoverability or cascade freedom
    - Need to introduce commit dependencies to ensure recoverability
  - Transactions may have to lock data items that they do not access
    - increased locking overhead, and additional waiting time
    - potential decrease in concurrency
- Schedules not possible under two-phase locking are possible under tree protocol, and vice versa

Timestamp-Based Protocols

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- Each transaction is issued a timestamp when it enters the system
- The protocol manages concurrent execution such that the time-stamps determine the serializability order
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
  - W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully
  - R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully
- The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order

Read and Write Rules

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- Suppose a transaction T_i issues a read(Q)
  - If TS(T_i) $<$ W-timestamp(Q), then T_i needs to read a value of Q that was overwritten by a "younger" transaction, hence the read is rejected and T_i is rolled back
  - Otherwise, the read is performed and R-timestamp(Q) is updated
- Suppose that transaction T_i issues write(Q)
  - If TS(T_i) $<$ R-timestamp(Q), then the value of Q that T_i is producing was needed was needed by a "younger" transaction, hence, the write operation is rejected, and T_i is rolled back
  - If TS(T_i) $<$ W-timestamp(Q), then T_i is attempting to write a value of Q that was already written by a "younger" transaction, hence, this write operation is rejected, and T_i is rolled back
  - Otherwise, the write is performed and W-timestamp(Q) is updated

An Example

Database System Concepts

Lock-Based Protocols

Two-Phase Locking

Graph-Based Protocols

Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

	T_1	T_2	T_3	T_4
1.				read(X)
2.		read(Y)		
3.	read(Y)			
4.			write(Y)	
5.			write(Z)	
6.				read(Z)
7.		read(Z)		
8.		rollback		
9.	read(X)			
10.			write(Z)	
11.			rollback	
12.				write(Y)
13.				write(Z)

Properties of the Timestamp-Based Protocol

Database System Concepts

Lock-Based Protocols
Two-Phase Locking
Graph-Based Protocols
Timestamp-Based Protocols

Deadlock Handling

Insert and Delete

- The timestamp-ordering protocol guarantees serializability
- Timestamp protocol ensures freedom from deadlock
- But the schedule may not be cascade-free, and may not even be recoverable
- To ensure cascadeless and/or recoverable schedules:
  - A transaction is structured such that its writes are all performed at the end of its processing; all writes of a transaction form an atomic action; no transaction may execute while a transaction is being written
  - Limited form of locking: wait for data to be committed before reading it
  - Use commit dependencies to ensure recoverability
- Can be improved using Thomas' write rule
  - Modified version of the timestamp-ordering protocol in which obsolete write operations may be ignored under certain circumstances

Outline

1 Lock-Based Protocols

2 Deadlock Handling
- Deadlock Prevention
- Deadlock Detection
- Deadlock Recovery

3 Insert and Delete

Deadlock Prevention

Database System Concepts

Lock-Based Protocols

Deadlock Handling

Deadlock Prevention

Deadlock Detection

Deadlock Recovery

Insert and Delete

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set
- Deadlock prevention protocols ensure that the system will never enter into a deadlock state.
- Some prevention strategies :
  - Require that each transaction locks all its data items before it begins execution (predeclaration)
  - Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol)

More Deadlock Prevention Strategies

Database System Concepts

Lock-Based Protocols

Deadlock Handling

Deadlock Prevention

Deadlock Detection

Deadlock Recovery

Insert and Delete

Following schemes use transaction timestamps for the sake of deadlock prevention alone

- wait-die scheme (non-preemptive)
  - older transaction may wait for younger one to release data item
  - younger transactions never wait for older ones; they are rolled back instead
  - a transaction may die several times before acquiring needed data item
- wound-wait scheme (preemptive)
  - older transaction wounds (forces rollback) of younger transaction instead of waiting for it
  - younger transactions may wait for older ones
  - may be fewer rollbacks than wait-die scheme
- Transactions are *not* assigned new timestamps
- Avoid starvation, but may cause unnecessary rollbacks

Timeout-Based Schemes

Database System Concepts

Lock-Based Protocols

Deadlock Handling

Deadlock Prevention

Deadlock Detection

Deadlock Recovery

Insert and Delete

- A transaction waits for a lock only for a specified amount of time
- After that, the wait times out and the transaction is rolled back
- Thus deadlocks are not possible
- Simple to implement, but starvation is possible
- Also difficult to determine good value of the timeout interval

Deadlock Detection

Database System Concepts

Lock-Based Protocols

Deadlock Handling

Deadlock Prevention

Deadlock Detection

Deadlock Recovery

Insert and Delete

- Deadlocks can be described as a wait-for graph, which consists of a pair $G = (V, E)$,
  - V is a set of vertices (all the transactions in the system)
  - E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$
- If $T_i \rightarrow T_j$ is in E, then T_i is waiting for T_j to release a data item
  - When T_i requests a data item currently being held by T_j, the edge $T_i \rightarrow Tj$ is inserted in the wait-for graph.
  - This edge is removed only when T_j is no longer holding a data item needed by T_i
- The system is in a deadlock state if and only if the wait-for graph has a cycle
  - Must invoke a deadlock-detection algorithm periodically to look for cycles.

Wait-for Graphs

Database System Concepts

Lock-Based Protocols

Deadlock Handling

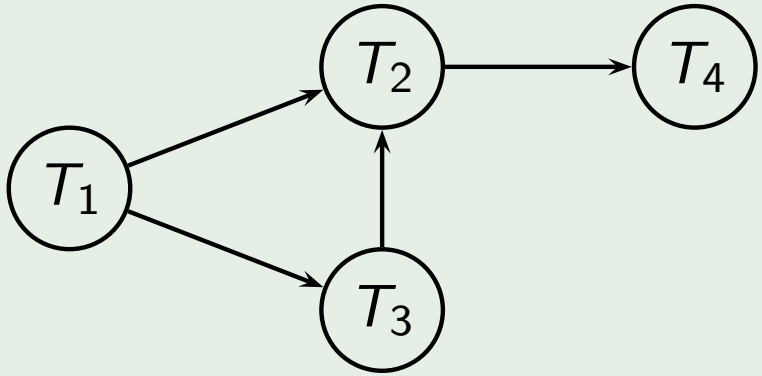
Deadlock Prevention

Deadlock Detection

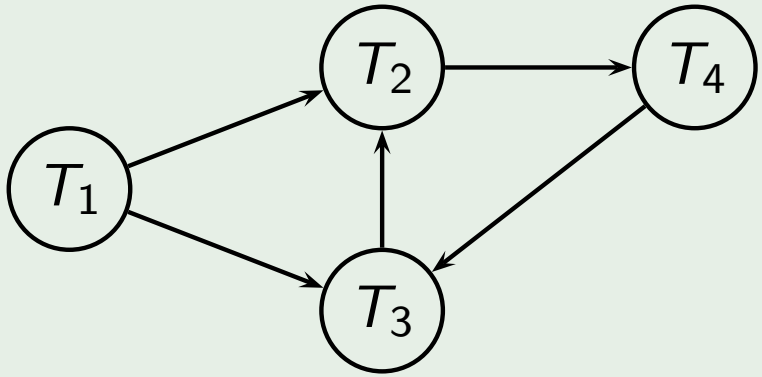
Deadlock Recovery

Insert and Delete

Wait-for graph without cycles



Wait-for graph with cycles



Deadlock Recovery

Database System Concepts

Lock-Based Protocols

Deadlock Handling

Deadlock Prevention

Deadlock Detection

Deadlock Recovery

Insert and Delete

When deadlock is detected:

- Some transaction will have to be rolled back (made a victim) to break deadlock
  - Select that transaction as victim that will incur minimum cost
    - performed less computations
    - used less data items
    - will cause less cascading rollbacks
- Rollback — determine how far to roll back transaction
  - Total rollback: abort the transaction and then restart it
  - More effective to rollback transaction only as far as necessary to break deadlock
- Starvation happens if same transaction is always chosen as victim
  - Include the number of rollbacks in the cost factor to avoid starvation

Outline

1 Lock-Based Protocols

2 Deadlock Handling

3 Insert and Delete

Insert and Delete Operations

- New conflicts arise between insertion/deletions and read/writes
- If two-phase locking is used :
  - A delete operation may be performed only if the transaction deleting the tuple has an exclusive lock on the tuple to be deleted
  - A transaction that inserts a new tuple into the database is given an X-mode lock on the tuple
- Insertions and deletions can lead to the phantom phenomenon
  - A transaction that scans a relation (e.g., find all accounts in Perryridge) and a transaction that inserts a tuple in the relation (e.g., insert a new account at Perryridge) may conflict in spite of not accessing any tuple in common
  - If only tuple locks are used, non-serializable schedules can result

The Phantom Phenomenon

- The transaction scanning the relation is reading information that indicates what tuples the relation contains, while a transaction inserting a tuple updates the same information
  - The information should be locked.
- One solution:
  - Associate a data item with the relation, to represent the information about what tuples the relation contains
  - Transactions scanning the relation acquire a shared lock in the data item
  - Transactions inserting or deleting a tuple acquire an exclusive lock on the data item
    - Note: locks on the data item do not conflict with locks on individual tuples
- Above protocol provides very low concurrency for insertions/deletions.
  - Index locking protocols provide higher concurrency by requiring locks on certain index buckets

End of Chapter 16