Lesson 2 – Transaction Processing & Concurrency Control (Part 3)

By
NWKDVP Opatha (BSc(Hons),MCTS,MCPD,MIEEE)

Overview

- Purpose of Concurrency Control
- Types of Locks
  - Binary Locks
  - Shared/exclusive Locks
- Lock Conversion

1. Purpose of Concurrency Control

- Concurrency control techniques are required to ensure the isolation property of concurrently executing transactions.
- Most of the techniques ensure the serializability of schedules.
- But also these concurrency control techniques have several other purposes.

1. Purpose of Concurrency Control

- These protocols help,
  - To preserve database consistency through consistency preserving execution of transactions.
  - To resolve read-write and write-write conflicts.

Example: In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

2. Types of Concurrency Control Protocols

- There are different concurrency control protocols.
  - Locking protocols
  - Timestamp
  - Multi-version
  - Validation & Certification

2.1 Locking Protocols

- **Lock** is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Several types of locks are used in concurrency control.
  - **Binary Locks**
  - **Read/Write(Shared/Exclusive) Locks**

2.1.1 Binary Lock

- A binary lock can have two states or values: locked and unlocked. Two operations are lock_item and unlock_item, are used with binary locking.
- It's implementation is simple. But too restrictive for database concurrency control purposes, and so are not used in practice.

2.1.1 Binary Lock

- Locking is an operation which secures,
  - permission to Read or
  - permission to Write a data item for a transaction.

  Example: Lock (X). Data item X is locked in behalf of the requesting transaction.
- Unlocking is an operation which removes these permissions from the data item

  Example: Unlock (X). Data item X is made available to all other transactions

2.1.1 Binary Lock

- **Lock Table**
  - Each lock can be a record with three fields: <Data_item_name,LOCK,Locking_transaction>
  - The system needs to maintain only these records for the items that are currently locked in a lock table.
  - Items not in the lock table are considered to be unlocked.

2.1.1 Binary Lock

The DBMS has a **lock manager subsystem** to keep track of and control access to locks

2.1.2 Shared/exclusive locks

- The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item.
- We should allow several transactions to access the same item X if they all access X for reading purposes only.
- This is because read operations on the same item by different transactions are not conflicting.

2.1.2 Shared/exclusive locks

- However, if a transaction is to write an item *X*, it must have exclusive access to X.
- There are three locking operations.
  - Read_lock(X)
  - Write_lock(X)
  - Unlock(X)

2.1.2 Shared/exclusive locks

- A lock associated with an item *X* may have three possible states,
  - Read-locked / Shared-locked
  - Write-locked/ Exclusive-locked
  - Unlocked

2.1.2 Shared/exclusive locks

- When we use the shared/exclusive locking scheme, the system must enforce the following rules,
  - A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.
  - A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.
  - A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.

2.1.2 Shared/exclusive locks

  - A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.

- **Lock Conversion**

  Sometimes, transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another.

2.1.2 Shared/exclusive locks

- Examples:
  - A transaction T to issue a read_lock(X) and then later to upgrade the lock by issuing a write_lock(X) operation.
    - If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded.
    - Otherwise, the transaction must wait.
  - A transaction T to issue a write_lock(X) and then later to downgrade the lock by issuing a read_lock(X) operation.

2.1.3. Two-Phase Locking

- Using binary locks or read/write locks in transactions, as described earlier, does not always guarantee serializability of schedules.

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);
write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);
	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

Initial values: *X*=20, *Y*=30

Result serial schedule T_1 followed by T_2: *X*=50, *Y*=80

Result of serial schedule T_2 followed by T_1: *X*=70, *Y*=50

2.1.3. Two-Phase Locking

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);	

	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);	

Result of schedule *S*: *X*=50, *Y*=50 (nonserializable)

2.1.3. Two-Phase Locking

- To guarantee serializability, it is necessary to follow an additional protocol concerning the positioning of locking and unlocking operations in every transaction.
- The best-known protocol, two-phase locking.

2.1.3. Two-Phase Locking

- A transaction is said to follow the **two-phase locking protocol** if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction.
- Such a transaction can be divided into two phases,
  - **Expanding Phase**
  - **Shrinking Phase**

2.1.3. Two-Phase Locking

- **Expanding Phase**
  - Duration which new locks on items can be acquired but none can be released.
- **Shrinking Phase**
  - Duration which existing locks can be released but no new locks can be acquired.

2.1.3. Two-Phase Locking

- If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase.
- Downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.

2.1.3. Two-Phase Locking

T_1'	T_2'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
X := X + Y;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	write_lock(Y);
	unlock(X)
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);

2.1.3. Two-Phase Locking

- Although the two-phase locking protocol guarantees serializability (that is, every schedule that is permitted is serializable), it does not permit all possible serializable schedules (that is, some serializable schedules will be prohibited by the protocol).

2.1.4 Deadlock & Starvation

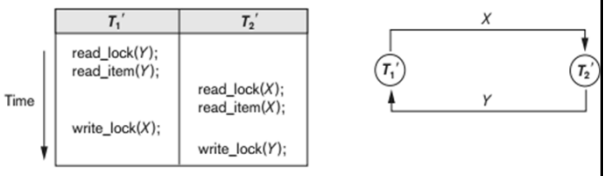
- **Deadlock** occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.
- To prevent from deadlocks it is possible to use **Deadlock Prevention Protocol.**
- One possible method is, requires that every transaction lock all the items it needs in advance (which is generally not a practical assumption).

2.1.4 Deadlock & Starvation

- If any of the items cannot be obtained, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items it needs.
- A more practical approach to dealing with deadlock is **deadlock detection**, where the system checks if a state of deadlock actually exists.

2.1.4 Deadlock & Starvation

- A simple way to detect a state of deadlock is for the system to construct and maintain a **wait-for graph**.



2.1.4 Deadlock & Starvation

- **Starvation** occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
- This may occur if the waiting scheme for locked items is unfair, giving priority to some transactions over others.

2.1.4 Deadlock & Starvation

- One solution for starvation is to have a fair waiting scheme, such as using a **first-come-first-served** queue.
- In this method, transactions are enabled to lock an item in the order in which they originally requested the lock.

Your Task..

- Find out,
  - What are the other available methods for deadlock prevention?
  - What are the variations of two phase locking protocol.

References

- Elmasri, R , Navathe, SB 2011 , Fundamentals of Database Systems, Addison-Wesley, United States of America , pp 778 – 785

Thank You.