Unit-9. Concurrency Control Techniques

- Ankit Pangeni

DATE

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each other. It is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases. Therefore, Concurrency control is the most important element for the proper functioning of a multi user system.

* Concurrency Control protocols.

Different Concurrency control protocols offer different benefits between the amount of concurrency they allow & the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Two-Phase Locking Technique

DATE

* Concept of two-phase locking

Some of the main techniques for concurrency control of transactions are based on the concept of locking data items.

A lock is a mechanism to control concurrent access to a data item. Generally, there is one lock for each data item in the database. Any transaction cannot read or write data until it acquires an appropriate lock on it.

→ Locking is an operation which secures: permission to read, or write on a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock. To guarantee serializability, we use two-phase locking protocol. A transaction is said to follow the Two-phase Locking protocol if Locking & Unlocking can be done in two phases

- Growing phase: Here a new lock on the data item may be acquired by the transaction, but none can be released.
- Shrinking Phase: Here, existing lock held by the transaction may be released, but no new locks can be acquired

PAGE

classmate

DATE

The two main modes in which a data item may be locked are:

- Exclusive (X) mode → Data item can be read as well as written.
- Shared (S) mode → Data item can only be read.

Request / Held	S	X
S	True	False
X	False	False

fig: Lock Compatibility matrix.

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.

Types of locks and system locks table

There are various ~~mode~~ kinds of locks.

1. Binary Locks

In binary lock, a lock on a data item can be in two states: it is either locked or unlocked. (or 1 and 0, for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1, item X can't be accessed by a database operation that requests the item. Otherwise, if it is 0, then the item can be accessed when requested & the lock value is changed to 1. We refer to the current state of the lock associated with item X as lock(X). Two operations, lock_item(X) & unlock_item(X) are used with binary locking.

PAGE

DATE

2. Shared-Exclusive or (Read/write) Lock.

This type of locking mechanism differentiates the locks based on their uses. If lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed

DATE

* Lock Conversion (Upgrading, Downgrading)

The mechanism of conversion from shared lock to exclusive lock is known as upgrade of lock. Whereas the mechanism of conversion from exclusive lock to shared lock is downgrade of lock. When upgrading and downgrading of locks is used, the lock table must include transaction identifiers in the record structure for each lock to store the information on which the transactions hold locks on the item.

* Disadvantages of two-phase locking or Shared-Exclusive

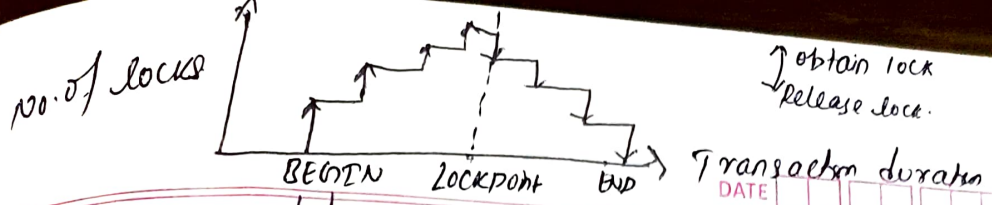
- May not be free from irrecoverability
- May not be free from deadlock
- May not be free from starvation
- May not be free from cascading rollback

* Guaranteeing Serializability by Two-Phase locking

A transaction is said to follow the 2P locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation. Such a transaction can be divided into two phases :-
an expanding phase: during which new locks on items can be acquired but none can be released;
and a shrinking phase: during which existing locks can be released but no new locks can be acquired. If lock conversion is allowed, then upgrading of locks must be done during the expanding phase & downgrading must be done in shrinking phase.

classmate

PAGE



DATE

We can determine the serializability on the basis of its lock point. Here lock point ensures that T_1 has occured first which is followed by T_2. There is no any loop. Thus it ensures serializability. This is one of the main advantage of 2PL.

	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	-----	-----
5	unlock(A)	
6		Lock-X(C)
7	unlock(B)	
8		unlock(A)
9		unlock(C)
10	-----	-----

T_1 Growing phase

Lock point of T_1

T_1 Shrinking phase

T_2 Growing phase

Lock point of T_2

T_2 Shrinking phase

S => Shared lock
X => Exclusive lock

Fig: 2PL

* Types of two-phase locking

There are four types of two-phase locking

1. Basic 2PL → we just studied
2. Conservative 2PL
3. Strict 2PL
4. Rigorous 2PL

} Imp.

PAGE

DATE

- **Conservative 2PL:-**

According to this scheme, a transaction locks all data items before beginning its operations and unlocks those data items only after executing its last operation ie. pre-declare all the locks on all variables before its execution. It solves the problem of deadlock, ~~irrecoverability, starvation and Cascading Roolback~~ but it often degrades the system performance by unnecessary delaying other transaction which want to access some of the data item locked by the transaction. It is very difficult to use in practice.

- **Strict 2PL**

According to this protocol, a transaction does not release the exclusive locks ie. write locks until it commits or aborts. It solves the irrecoverability problem & Problem of Cascading roolback but not the problem of deadlock & starvation.

Note: It should satisfy the basic 2PL

T_1	T_2	T_3
X(A)		
R(A)		
W(A)		
Unlock(A)	S(A)	
-	R(A)	
-		S(A)
-		R(A)

roolback, roolback, cascade fail

fig: cascading roolback Problem

T_1	T_2	T_3
X(A)		
R(A)		
W(A)		
	S(A)	
	R(A)	
		S(A)
		R(A)
Commit		
Unlock(A)		

fig: Solving roolback problem

PAGE

DATE

Note:- Cascading roubaek problem :- If the faiwre of one transaction causes several other dependent transactions to roubaek or abort, Pt is called crp.

Irrecovability:- The faiwre of a transaction has cause another transaction to roubaek. But the another transaction has already committed, so Pt is Irrecovable

T_1	T_2
X(A)	
R(A)	
W(A)	
UNLOCK(A)	
--	S(A)
--	R(A)
--	UNLOCK(A)
--	Commit
--	
fail	

roll back

not possible of rollback

T_1	T_2
X(A)	
R(A)	
--	
Commit	
UNLOCK(A)	
	S(A)
	R(A)
	UNLOCK(A)
	Commit

fig: Problem of Irrecovability

fig: Irrecoverable problem ~~solved~~

- Rigorous 2PL

According to this protocol, a transaction does not release any of exclusive as well as shared locks until it commits or aborts. It also solves the Problem of Cascading roubaek & irrecovability but the problem of deadlock & starvation still exists.

PAGE

classmate

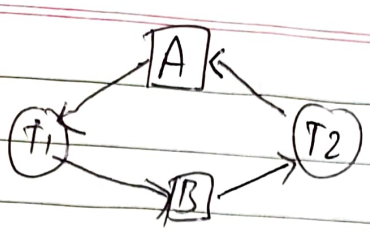
DATE

* Dealing with deadlock & Starvation

Deadlock	Starvation.
1) A deadlock is a condition in which two or more transaction are waiting for each other.	1) Starvation happens if same transaction is always chosen as victim.
2) It occurs when two or more transactions are unable to proceed because each is waiting for one of the other to do something.	2) It occurs if the waiting scheme for locked items is unfair, giving priority to some transactions over other.
3) It is also known as circular waiting.	3) It is also known as livelock
Avoidance: → Acquire locks at once before starting → Acquire locks with predefined order.	Avoidance. → Switch priorities so that every thread has a chance to have high priority → Use FIFO order among competing requests.
6) Resources are blocked by the processes	6) Resources are continuously utilized by high priority processes

DATE

Deadlock



Two transaction T1 & T2 need two resource A & B for completing the task. T1 requests resource A & is granted and holds it. T2 requests resource B & is also granted & holds it. Now, if T1 request for B held by T2, then request is denied until T2 release it. Unfortunately, instead of releasing B, T2 asks for A held by T1. At this state, both transaction are blocked & will remain so forever. This situation is an example of deadlock

* Necessary Conditions for deadlock.
  - Mutual exclusion → Only one transaction at a time Can use a resource
  - Hold & wait → Transaction holding a resource while waiting to acquire another resource held by another transaction
  - No preemption → Resources are only released when a transaction is executed
  - Circular wait → One transaction T_1 waiting for resource held by another transaction T_2 while T_2 is waiting for resource hold by T3 & T3 is waiting for resource hold by T_1.

* Deadlock detection

Maintain a wait for graph ie. resource allocation graph where node represent processes and if process i is waiting for a resource held by process j, then there exists an edge from i to j. If there is a cycle in the graph, deadlock exists.

ex:

PAGE

classmate

DATE

* Deadlock Recovery

When a deadlock algorithm determines that a deadlock exists, the system must recover from deadlock. There are two approaches

- Simply kill one or more processes to break circular wait
- Preempt some resources from one or more processes involved in deadlock.

Dealing with Starvation

One solution for starvation is to have a fair waiting scheme, such as using a first-come-first-served queue. where transactions are enabled to lock an item in order in which they originally requested to lock. Another scheme allows some transactions to have higher priority over others. The wait-die and wound-wait avoid starvation, because they restart a transaction that have been aborted

(Timestamp Ordering)

DATE

* Timestamp ordering Concurrency Control concept

A timestamp is a unique identifier created by the DBMS to identify a transaction.

The timestamp ordering is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation. The priority of the older transaction is high so it executes first. To determine the timestamp of a transaction, the system ~~times~~ time is used ~~by the~~ by the protocol.

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times & T2 has entered at 009. T1 has the higher priority, so it executes 1st as it entered the system 1st. The time stamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on data.

Timestamp values are assigned in order in which the transactions are submitted to the system, so it can be thought of as the transaction start time. We denote time stamp transaction of transaction T as TS(T).

Concurrency control techniques based on timestamp ordering don't use locks, hence, deadlocks can't occur.

- Read time stamp (R_TS) :- Latest transaction number which performed Read successfully
- Write time stamp (W_TS) :- Latest transaction number which performed write successfully

Note: R_TS & RTS both same

DATE

* Basic Time Stamp ordering:

Rules:

1) Transaction T_i issues a Read(A) operation

a) If $WTS(A) > TS(T_i)$, Roleback T_i (operation rejected)
b) Otherwise execute R(A) operation
Set $RTS(A) = MAX\{ RTS(A), TS(T_i) \}$

2) Transaction T_i issues Write(A) operation.

a) If $RTS(A) > TS(T_i)$ then Roleback T_i (operation rejected)
b) If $WTS(A) > TS(T_i)$ then Roleback T_i (" ")
c) Otherwise execute Write(A) operation
Set $WTS(A) = TS(T_i)$

* Strict Time Stamp ordering

A variation of basic Time Stamp ordering called Strict TO (Time) ensure that the schedules are both Strict (for easy recoverability) & (conflict) serializable

1. Transaction T_i issues a write_item(A) operation:
   - If $TS(T) > RTS(A)$, then delay T until the transaction T' that wrote or read A has committed or aborted
2. (Read_item(A)) - If $TS(T) > WTS(A)$, then delay T until the transaction T' that wrote or read A has committed or aborted.

classmate

PAGE

DATE

Thomas's Write rule

A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(A) operation as follows.

- If RTS(A) > TS(T), then abort & roll back T & reject the operation
- If WTS(A) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes
- If the conditions given in above two do not occur, then execute write_item(A) of T & set WTS(A) to TS(T)

PAGE

DATE

(Multiversion Concurrency Control)

* Concept of multiversion concurrency control technique.

→ It provides concurrent access to database without locking the data.
→ This technique keeps the record of old values of a data item when the data item is updated
→ These are known as multiversion concurrency control, because several version (values) of an item are maintained.
→ When a transaction requires access to an item an appropriate version is chosen to maintain the serializability of the current executing schedule, if possible.
→ It improves the performance of database applications in a multiuser environment

* Multiversion technique based on timestamp ordering

Assume $X_1, X_2, \ldots, X_n$ are the version of a data item X created by a write operation of transactions. For each version, the value of version X_i, a RTS & WTS are associated.

- RTS :- The read timestamp is the largest of all the timestamps of transactions that have successfully read version
- W-TS :- The write timestamp is the timestamp of the transaction that wrote the value of version

DATE

To ensure Serializability, the following two rules are used

- If transaction T issues a write_item(X) operation, & version of X has highest WTS() of all versions of X that is also less than or equal to TS(T), & RTS() > TS(T), then abort & roll back transaction T; otherwise create a new version of X with RTS() = WTS() = TS(T)

- If transaction T issues a Read(X) operation, find version of X that has highest WTS() of all versions of X that is also less than or equal to TS(T); then return the value of to transaction T, & set the value of RTS() to the larger of TS(T) & the current RTS()

Multiversion locking using certify locks (2 phase locking)

In this scheme, there are three locking modes for an item; read, write & certify. Hence, the state of LOCK(X) for an item X can be one of read-locked, write-locked, certify-locked or unlocked. The idea behind it is to allow other transactions T_2 to read an item X while a single transaction T, holds a write lock on X. This is accomplished by allowing two versions for each item X; one, the committed version, must always have been written by some committed transaction. and the second local version X' can be created when a transaction T acquires a write lock on X.

Once T is ready to commit, it must obtain a certify lock on all items that it currently holds

write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction may have to delay its commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks are acquired, the committed version X of the data item is set to the value of version X, version X is discarded, & the certify locks are then released.

Validation (optimistic) Techniques & Snapshot Isolation (Concurrency Control)

Concept of validation (optimistic) Techniques

The locked based & time stamp ordering protocols were pessimistic techniques. Because they always think negative that the transactions can go in conflict.

Validation phase is a optimistic control technique. Here, the transaction is executed in 3 phases

- Read phase: Here, the transaction T is read & executed. It is used to read the value of various data item & stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.

DATE

- Validation phase :- Here, temporary variable value will be validated against the actual data to see if it violates the serializability.

- Write phase :- If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.
(ie. the changes are permanently applied to the DB)

Note: Each phase has following timestamps.
→ Start(Ti):- time when Ti started execution
→ Validation(Ti):- time when Ti finishes its read phase & starts its validation phase.
→ Finish(Ti):- time when Ti finishes its write phase

* Snapshot Isolation Concurrency control.

→ Here, each transaction operates on database snapshot.
→ The snapshot is taken once the transaction starts.
→ When transaction concludes, it will successfully commit only if the values update by the transaction haven't been changed externally since snapshot was taken
→ In practice, snapshot isolation is implemented within multi version concurrency control (MVCC), where generational values of each data item (versions) are maintained
→ It is widely used in practice
→ The main reason for its adoption is that it allows better performance than serializability, yet still avoids most of the concurrency anomalies that serializability avoids (but not always all).

classmate
PAGE

Questions asked from this Chapter

Q. Explain deadlock and starvation. Explain Time stamp based protocol for concurrency control. (2078 - 10 marks (2075 - 5 marks)

Q. What is concurrency control? Name various methods of controlling the concurrency control. Differentiate between Binary lock and Shared/Exclusive lock (2076 - 10 marks) (2073 - 6 marks)

Q. What is Granularity of data items? How does it effect in concurrency control? (2076 - 5 marks) (2069 - 5 marks) (2067 - 5 marks)

Q. Explain 2 phase locking technique in brief. (2076 - 5 marks (2072 - 6 marks) (2067 - 5 marks) (2068 - 5 marks)

Q. Discuss the problems of deadlock & starvation, and the different approaches to dealing with these problems. (2067 - 5 marks) (2073 - 5 marks) (2075 - 6 ma