UNIT 4 TRANSACTIONS

Transaction Concepts –ACID Properties – Schedules – Serializability – **Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlocks** – Transaction Recovery – Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery

Recap

- Serialization

Concurrency Control

- **Concurrency Control** is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.
- **Concurrent Execution -** Multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.

Problems with Concurrent Execution

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

Problems with Concurrent Execution

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

Problems with Concurrent Execution

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

Concurrency Control Protocols

- The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions.
  - Lock Based Concurrency Control Protocol
  - Time Stamp Concurrency Control Protocol
  - Validation Based Concurrency Control Protocol

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**

	S	X
S	true	false
X	false	false

- 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.**

Lock-Based Protocols (Cont.)

Example of a transaction performing locking:

T_2: **lock-S**(*A*);
read (*A*);
unlock(*A*);

lock-S(*B*);
read (*B*);
unlock(*B*);
display(*A*+*B*)

Schedule With Lock Grants

- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce Serializability by restricting the set of possible schedules.

T_1	T_2	concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B)		
$B := B - 50$		
write(B)		
unlock(B)		
	lock-S(A)	
		grant-S(A, T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display($A + B$)	
lock-X(A)		
		grant-X(A, T_1)
read(A)		
$A := A + 50$		
write(A)		
unlock(A)		

Deadlock

- Consider the partial schedule

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

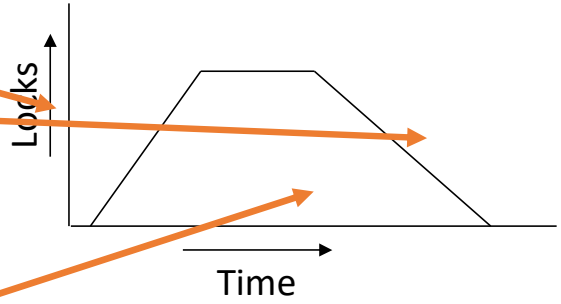
- Neither T_3 nor T_4 can make progress
- Executing **lock-S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of **T_3 or T_4 must be rolled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

Deadlock (Cont.)

- **Starvation** is also possible if concurrency control manager is badly designed.
- 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

- 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 Two-Phase Locking Protocol (Cont.)

- **Strict two-phase locking:** a transaction must hold all its exclusive locks till it commits/aborts.
  - Ensures recoverability and avoids cascading roll-backs
- **Rigorous two-phase locking**: a transaction must hold *all* locks till commit/abort.
  - Transactions can be serialized in the order in which they commit.

Lock Conversions

Two-phase locking protocol with lock conversions:

- Growing 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)**
- Shrinking Phase:
  - can release a lock-S
  - can release a lock-X
  - can convert a lock-X to a lock-S **(downgrade)**

T_1	T_2
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)
	lock-S(B)
	read(B)
	unlock(B)
	display($A + B$)
lock-X(A)	
read(A)	
$A := A + 50$	
write(A)	
unlock(A)	

Automatic Acquisition of Locks

operation read(D)

```
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
```

operation write(D)

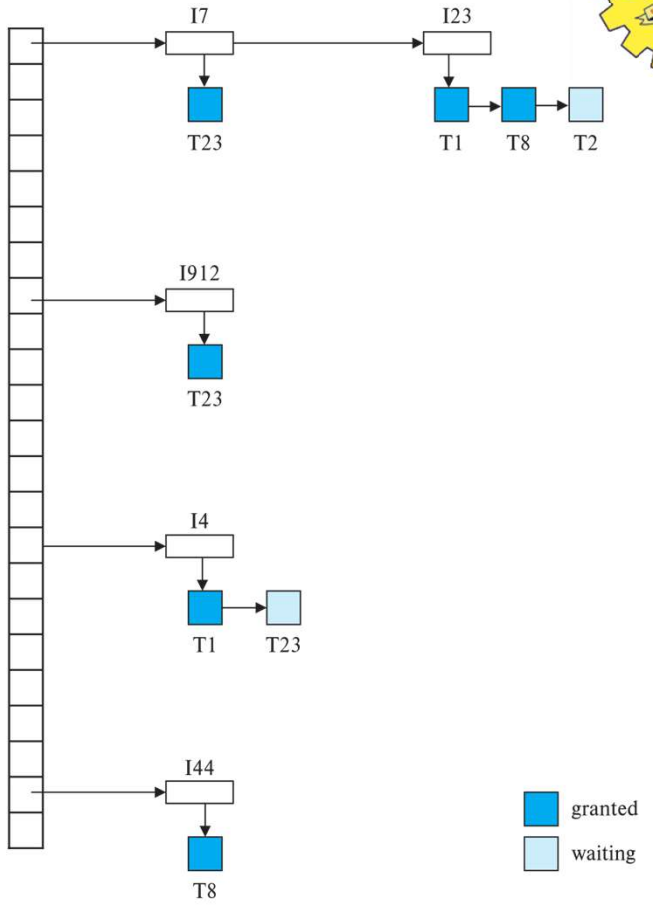
```
if T_i has a lock-X on D
  then
    write(D)
 else begin
    if necessary wait until no other trans. has any lock on D,
    if T_i has a lock-S on D
       then
         upgrade lock on D to lock-X
       else
         grant T_i a lock-X on D
      write(D)
 end;
```

Implementation of Locking

- **Lock manager** can be implemented as a separate process
- Transactions can send lock and unlock requests as messages
- The lock manager replies to a lock request by sending a lock grant messages
  - The requesting transaction waits until its request is answered
- The lock manager maintains an in-memory data-structure called a **lock table** to record granted locks and pending requests

Lock Table

- Dark rectangles indicate granted locks, light colored ones indicate waiting requests
- Lock table also records the type of lock granted or requested
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted
- If transaction aborts, all waiting or granted requests of the transaction are deleted
  - lock manager may keep a list of locks held by each transaction, to implement this efficiently




Thank
you