18.1-2 : Concurrency Control

Angie Yang

18 Concurrency Control

- Multiple transactions happening concurrently in the database
- System controls interactions through **concurrency-control schemes**

18.1 Lock-Based Protocols

- Requires that accessing data items are mutually exclusive
- When one transaction is accessing a data item, it holds a **lock** on the item, preventing other transactions from changing that data item
- Locks
  - **Shared** (S): transaction can read, but not write data item
  - **Exclusive** (X): transaction can read and write data item

Lock Compatibility

- Transaction **requests** a lock to concurrency-control manager
- If requested lock is compatible, it is **granted**
  - Else, transaction **waits** until incompatible transactions finish

	S	X
S	true	false
X	false	false

Figure 18.1 Lock-compatibility matrix comp.

```
T1: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    unlock(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(A).
```

Figure 18.2 Transaction T_1.

```
T2: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A + B).
```

Figure 18.3 Transaction T_2.

Example

T_1: transfer $50 from B to A

T_2: display sum of A, B

A=$100, B=$200

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)		

Figure 18.4 Schedule 1.

Example cont.

```
T3: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(B);
    unlock(A).
```

Figure 18.5 Transaction T_3 (transaction T_1 with unlocking delayed).

```
T4: lock-S(A);
    read(A);
    lock-S(B);
    read(B);
    display(A + B);
    unlock(A);
    unlock(B).
```

Figure 18.6 Transaction T_4 (transaction T_2 with unlocking delayed).

- Wait until entire transaction is finished to unlock, preventing inconsistent states
- Locking protocols to limit legal schedules

Two-Phase Locking Protocol

1. Growing phase: transaction can obtain, not release locks
2. Shrinking phase: transaction can release, not obtain locks

- Transactions are two-phase if all data items are locked and then unlocked
- Transaction obtains final lock, reaches **lock point** and begins shrinking stage
- Ensures conflict serializability

Lock Implementation

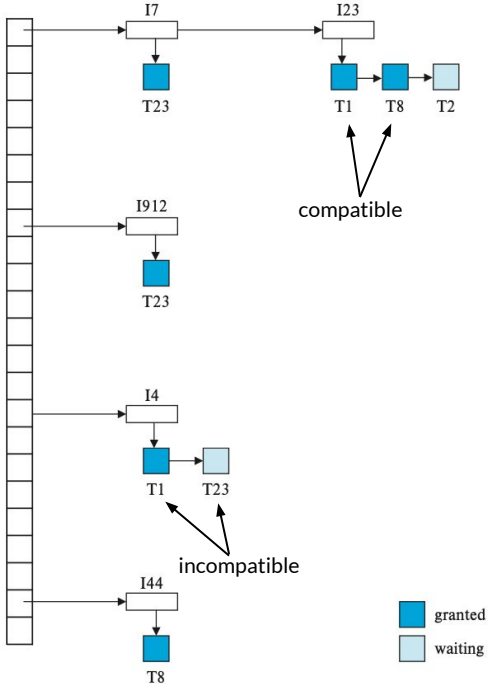

Figure 18.10 Lock table.

Lock Manager

- Linked list holding request records for each locked data item
- Hash table (**lock table**)
  - Key: data item
  - Value: linked list
- Receives requests, adds transaction to corresponding data items
- Checks compatibility before granting locks
- Deletes record when data item is unlocked

Graph-Based Protocols

- Partial ordering using a database graph
- Tree protocol for **exclusive** locks
  - First lock can be on any data item
  - After, data items can only be locked if parent is locked by same transaction
  - Data items can be unlocked at any time
  - Transactions cannot “relock” data items
- Conflict serializable

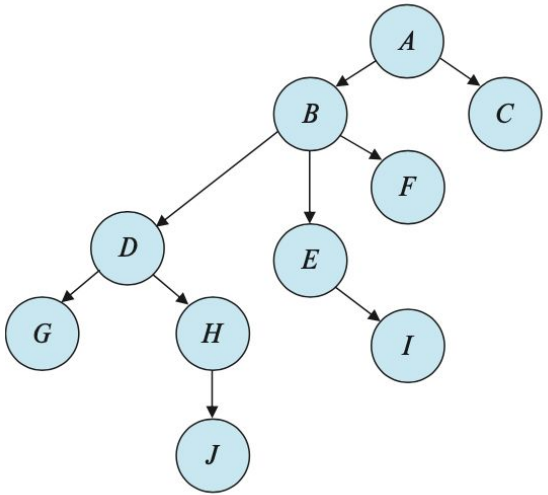

Figure 18.11 Tree-structured database graph.

T_{10}: lock-X(B); lock-X(E); lock-X(D); unlock(B); unlock(E); lock-X(G); unlock(D); unlock(G).
T_{11}: lock-X(D); lock-X(H); unlock(D); unlock(H).
T_{12}: lock-X(B); lock-X(E); unlock(E); unlock(B).
T_{13}: lock-X(D); lock-X(H); unlock(D); unlock(H).

18.2 Deadlock Handling

Deadlock: Every transaction in progress is waiting for a data item that another transaction is currently locking → no transaction is able to move forward

- Deadlock prevention or deadlock detection and recovery
  - Both can use transaction rollback

Deadlock Prevention

- Require all transactions lock data items before execution (Lock-Based)
  - Hard to predict which data items need to be locked, some may not be used
- Data ordering - require transactions to lock data items in the specified order (Graph-Based)
- Preemption and transaction rollbacks - timestamps
  - **Wait-die**: requesting transaction waits if it is older, if not, rollback
  - **Wound-wait**: requesting transaction waits if younger, rollback if not
- Lock timeouts: transactions timeout and restart

Deadlock Detection, Recovery

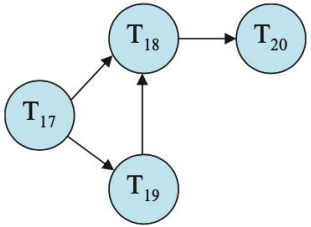

Figure 18.13 Wait-for graph with no cycle.

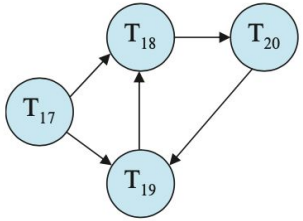

Figure 18.14 Wait-for graph with a cycle.

1. Maintain info about current allocation of data items to transactions, outstanding data item requests
2. Algorithm to detect deadlock state
3. Recovery from deadlock depending on algorithm
    a. Select transaction to rollback for minimum cost
        i. How long
        ii. How many data items
        iii. How many transactions
    b. Total or partial rollback to point of deadlock

The end