Chapter 7.
Transaction Management and Concurrency Control

Transaction Support
Properties of Transaction Support
Concurrency Control
Concurrency Control Techniques
Deadlock
Database Recovery
Recovery Facilities
Recovery Techniques

Introduction

- Reliability and consistency must be maintained in the presence of failures of both hardware and software components and when multiple users are accessing the database.
- All functions are dependent.
- Both concurrency control and recovery services are required to protect database from data inconsistencies and data loss.
- Many DBMS allow simultaneously operation on the database.
- If these operations are not controlled, the accesses will interfere with one another and the database become inconsistent.

- To overcome this problem, DBMS implements a concurrency control protocol that prevent databases accesses from interfering with one another.
- Database recovery is the process of restoring the database to a correct state following a failure.
- The failure caused by the system crash due to hardware or software errors in the application or media failure such as head crash.
- Whatever cause of failure, the DBMS must be able to recover from failure and restore the database to consistent state.

Transaction Support

Transaction

Action or series of actions, carried out by user or application, which reads or updates contents of database.

- Logical unit of work on the database, may be involve entire program or single SQL statement or any operation.
- Application program is series of transactions with non-database processing in between.
- Transforms database from one consistent state to another, although consistency may be violated during transaction.

Example Transaction

```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, new_salary)
```

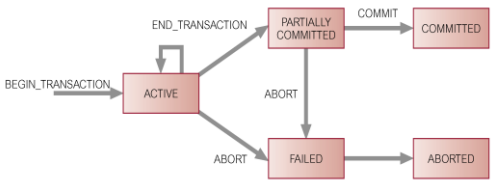
(a)

```
delete(staffNo = x)
for all PropertyForRent records, pno
begin
    read(propertyNo = pno, staffNo)
    if (staffNo = x) then
    begin
        staffNo = newStaffNo
        write(propertyNo = pno, staffNo)
    end
end
```

(b)

- Can have one of two outcomes:
  - **Success** - transaction *commits* and database reaches a new consistent state.
  - **Failure** - transaction *aborts* and database must be restored to consistent state before it started.
  - Such a transaction is *rolled back* or *undone*.
- Committed transaction cannot be aborted.
- If committed transaction is a mistake, then have to perform compensating transaction to reverse its effect.
- Aborted transaction that is rolled back can be restarted later.

State Transition Diagram for Transaction



Properties of Transactions

- Four basic *(ACID)* properties of a transaction are:
  - **Atomicity** -'All or nothing' property. A transaction is an indivisible unit that is either performed in its entirety or is not performed at all.
    - it is responsibility of the recovery subsystem of DBMS to ensure atomicity.
  - **Consistency -** Must transform database from one consistent state to another.
    - it is responsibility of both DBMS and the application developers to ensure consistency.

- **Isolation** - Partial effects of incomplete transactions should not be visible to other transactions.
  - responsibility of the concurrency control subsystem to ensure isolation.
- **Durability -** Effects of a committed transaction are permanent and must not be lost because of later failure.
  - responsibility of the recovery system to ensure durability.

Concurrency Control

Process of managing simultaneous operations on the database without having them interfere with one another.

- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

Need for Concurrency Control

- Three (3) potential problems caused by concurrency:
  - **Lost update problem.**
  - **Uncommitted dependency problem.**
  - **Inconsistent analysis problem.**

Lost Update Problem

- ***Successfully completed update operation by one user can be overridden by another user.***

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

- T_1 and T_2 start at nearly the same time and both read the balance as \$100, T_2 increases bal_x by \$100 to \$200 and stores the update in the database.

- Meanwhile transaction T_1 decrements its copy of bal_x by \$10 to \$90 and stores this value in the database, overwriting the previous update and thereby losing the \$100 previously added to the balance.
- Loss of T_2's update avoided by preventing T_1 from reading bal_x until after update.

Uncommitted Dependency Problem

- ***Occurs when one transaction can see intermediate results of another transaction before it has committed.***
- T_4 updates bal_x to \$200 but it aborts, so bal_x should be back at original value of \$100.
- T_3 has read new value of bal_x (\$200) and uses value as basis of \$10 reduction, giving a new balance of \$190, instead of \$90.

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

- Problem avoided by preventing T_3 from reading bal_x until after T_4 commits or aborts.

Inconsistent Analysis Problem

- ***Occurs when transaction reads several values but second transaction updates some of them during execution of first.***
- ***Sometimes referred to as dirty read or unrepeatable read.***
- T_6 is totaling balances of account x (\$100), account y (\$50) and account z (\$25).
- Meantime, T_5 has transferred \$10 from bal_x to bal_z, so T_6 now has wrong result (\$10 too high).

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

- Problem avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

Concurrency Control Techniques

- Two basic concurrency control techniques:
  - Locking,
  - Timestamping.
- Both are conservative approaches: delay transactions in case they conflict with other transactions.

Locking

Transaction uses locks to deny access to other transactions and to prevent incorrect updates.

- Generally, a transaction must claim a *shared* (*read*) or *exclusive* (*write*) lock on a data item before read or write.
- Lock prevents another transaction from modifying item or even reading it, in the case of a write lock.

Locking - Basic Rules

- If transaction has shared lock on item, can read but not update item.
- If transaction has exclusive lock on item, can both read and update item.
- Reads cannot conflict, so more than one transaction can hold shared locks simultaneously on same item.
- Exclusive lock gives transaction exclusive access to that item.
- Some systems allow transaction to upgrade read lock to an exclusive lock or downgrade exclusive lock to a shared lock.

Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the others to be released.

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(bal_x)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

- Only one way to break deadlock: abort one or more of the transactions.
- Deadlock should be transparent to user, so DBMS should restart transaction(s).
- Three general techniques for handling deadlock:
  - Timeouts.
  - Deadlock prevention.
  - Deadlock detection and recovery.

Timeouts

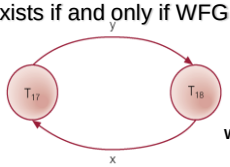
- Transaction that requests lock will only wait for a system-defined period of time.
- If lock has not been granted within this period, lock request times out.
- In this case, DBMS assumes transaction may be deadlocked, even though it may not be and it aborts and automatically restarts the transaction.

Deadlock Prevention

- DBMS looks ahead to see if transaction would cause deadlock and never allows deadlock to occur.
- Could order transactions using transaction timestamps:
  - Wait-Die - only an older transaction can wait for younger one, otherwise transaction is aborted (*dies*) and restarted with same timestamp.
  - Wound-Wait - only a younger transaction can wait for an older one. If older transaction requests lock held by younger one, younger one is aborted (*wounded*).

Deadlock Detection and Recovery

- DBMS allows deadlock to occur but recognizes it and breaks it.
- Usually handled by construction of Wait-For-Graph (WFG) showing transaction dependencies:
  - Create a node for each transaction.
  - Create edge T_i -> T_j, if T_i waiting to lock item locked by T_j.
- Deadlock exists if and only if WFG contains cycle.



Wait-For-Graph (WFG)

Timestamping

- Transactions ordered globally so that older transactions, transactions with *smaller* timestamps, get priority in the event of conflict.
- Conflict is resolved by rolling back and restarting transaction.
- No locks so no deadlock.

Timestamp

A unique identifier created by DBMS that indicates relative starting time of a transaction.

- Can be generated by using system clock at time transaction started or by incrementing a logical counter every time a new transaction starts.

Database Recovery

Process of restoring database to a correct state in the event of a failure.

- **Need for Recovery Control**
  - Two types of storage: volatile (main memory) and nonvolatile.
  - Volatile storage does not survive system crashes.
  - Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.

Types of Failures

- **System crashes** due to hardware or software errors, resulting in loss of main memory.
- **Media failures** such as head crashes or unreadable media, resulting in loss of parts of secondary storage.
- **Application software errors** such as logical errors in the program that is accessing the database which cause one or more transactions to fail.
- **Natural physical disasters** such as fires, floods, earthquakes or power failures.
- **Carelessness or unintentional destruction** of data or facilities by operators or users.
- **Sabotage** or intentional corruption or destruction of data, hardware or software facilities.

Recovery Facilities

- DBMS should provide following facilities to assist with recovery:
  - **Backup mechanism**, which makes periodic backup copies of database.
  - **Logging facilities**, which keep track of current state of transactions and database changes.
  - **Checkpoint facility**, which enables updates to database in progress to be made permanent.
  - **Recovery manager**, which allows DBMS to restore database to consistent state following a failure.

Recovery Techniques

Home

- If database has been damaged:
  - Need to restore last backup copy of database and reapply updates of committed transactions using log file.
- If database is only inconsistent:
  - Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
  - No need to backup but can restore database using before-and-after-images in the log file.