TRANSACTION MANAGEMENT

Part I

Sanit Sirisawatvatana and Sunisa Sathapornvajana

Chapter Objectives

- Function and importance of transactions.
- Properties of transactions.
- Concurrency Control
  - Locking
  - Deadlock
  - Timestamping
- Database Recovery
  - Transaction log
  - Checkpointing

Review

- Types of SQL statements
  - **DML**: Data Manipulation Language
    - SELECT, INSERT, UPDATE, DELETE statements, etc.
  - **DDL**: Data Definition Language
    - CREATE, ALTER, DROP, TRUNCATE statements, etc.
  - **DCL**: Data Control Language
    - GRANT and REVOKE statements
  - **TCL**: **Transaction Control Language**
    - COMMIT, ROLLBACK and SAVEPOINT statements

Transaction

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

- Transaction
  - Is a logical unit of work on the database
  - May be an entire (or a part of) program or a single statement that involve any number of operations on the database
  - Transforms database from one consistent state to another

- Database operations
  - Read/Write

Example: Transaction

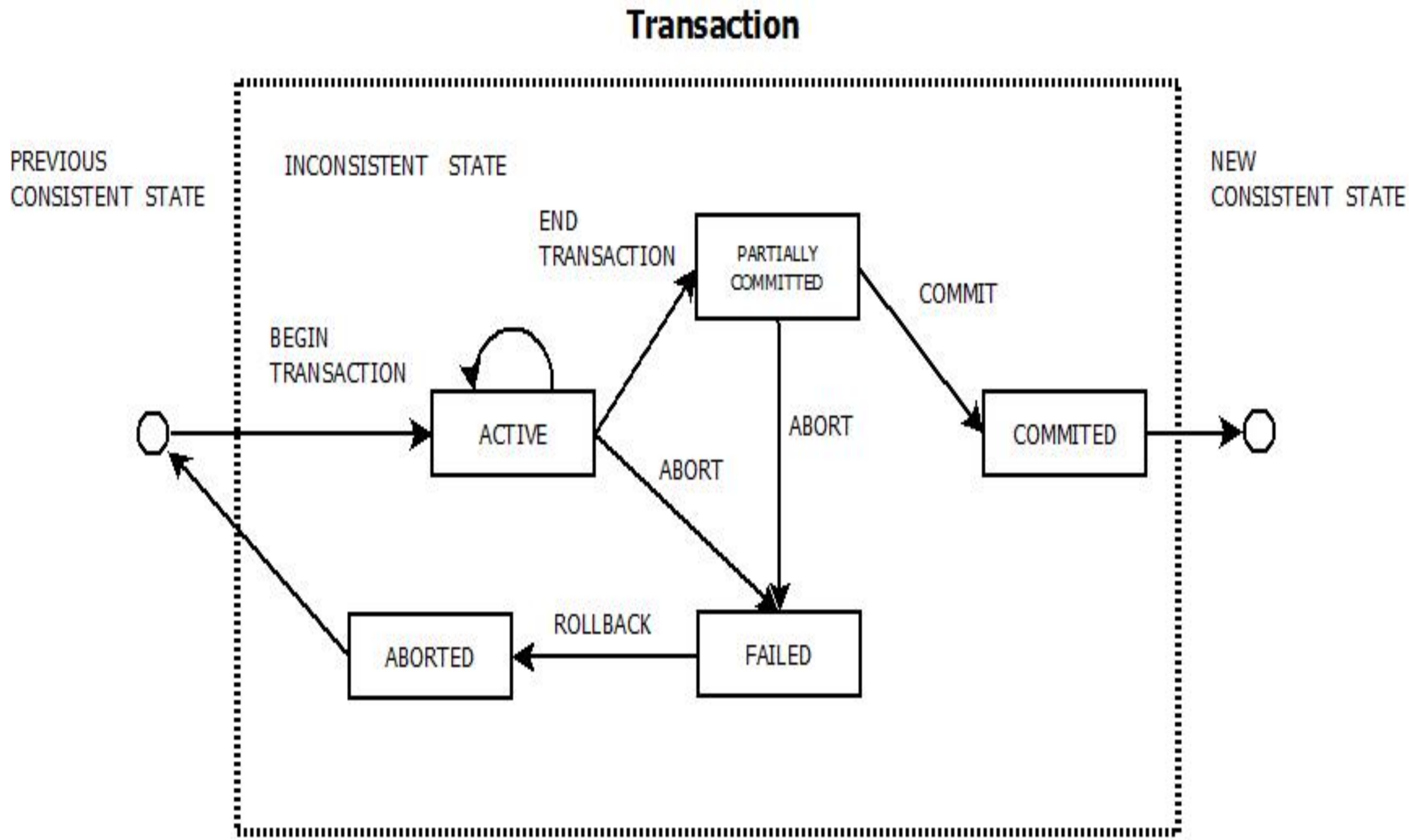
```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, new_salary)
```

(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)

State Transition Diagram for a Transaction



States of a Transaction

- ACTIVE
  - Executes database and non-database operations
- COMMITED
- ABORTED
- PARTIALL COMMITED
  - Occurs after the final statement has been executed
  - It may be found that transaction has violated an integrity constraint
- FAILED
  - Occurs when transaction cannot be committed or transaction is aborted
  - The user might abort the transaction
  - The concurrency control protocol might abort the transaction to ensure serializability

Outcomes of a Transaction

Two outcomes:

- COMMITED
  - Transaction completes successfully
  - The database reaches a new consistent state
- ABORTED
  - Transaction does not execute successfully
  - The database must be restored to the consistent state (previous consistent state) before it started (Rolled back or undone)

- A committed transaction cannot be aborted.
- An aborted transaction that is rolled back can be restarted later and may execute successfully and commit in later time.

Properties of Transactions (ACID)

All transactions should have basic four properties:

- Atomicity
  - An Indivisible unit
  - All or Nothing
- Consistency
  - Database transforms from one consistent state to another consistent state (consistent before and after transaction)
- Isolation
  - Executes independently of one another (not interfere with each other).
- Durability
  - The completed transaction is permanently recorded in database.
  - The completed transaction must not be lost from failure.

Example: Transaction

A banking application is the classic example of why transactions are necessary. Imagine a bank's database with two tables: checking and savings. To move $200 from Jane's checking account to her savings account, you need to perform at least three steps:

1. Make sure her checking account balance is greater than $200.
2. Subtract $200 from her checking account balance.
3. Add $200 to her savings account balance.

The entire operation should be wrapped in a transaction so that if any one of the steps fails, any completed steps can be rolled back.You start a transaction with the START TRANSACTION statement and then either make its changes permanent with COMMIT or discard the changes with ROLLBACK. So, the SQL for our sample transaction might look like this:

```
START TRANSACTION;
SELECT balance FROM checking WHERE customer_id = 10233276;
UPDATE checking SET balance = balance - 200.00 WHERE customer_id = 10233276;
UPDATE savings  SET balance = balance + 200.00 WHERE customer_id = 10233276;
COMMIT;
```

ACID Transaction

A — Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.

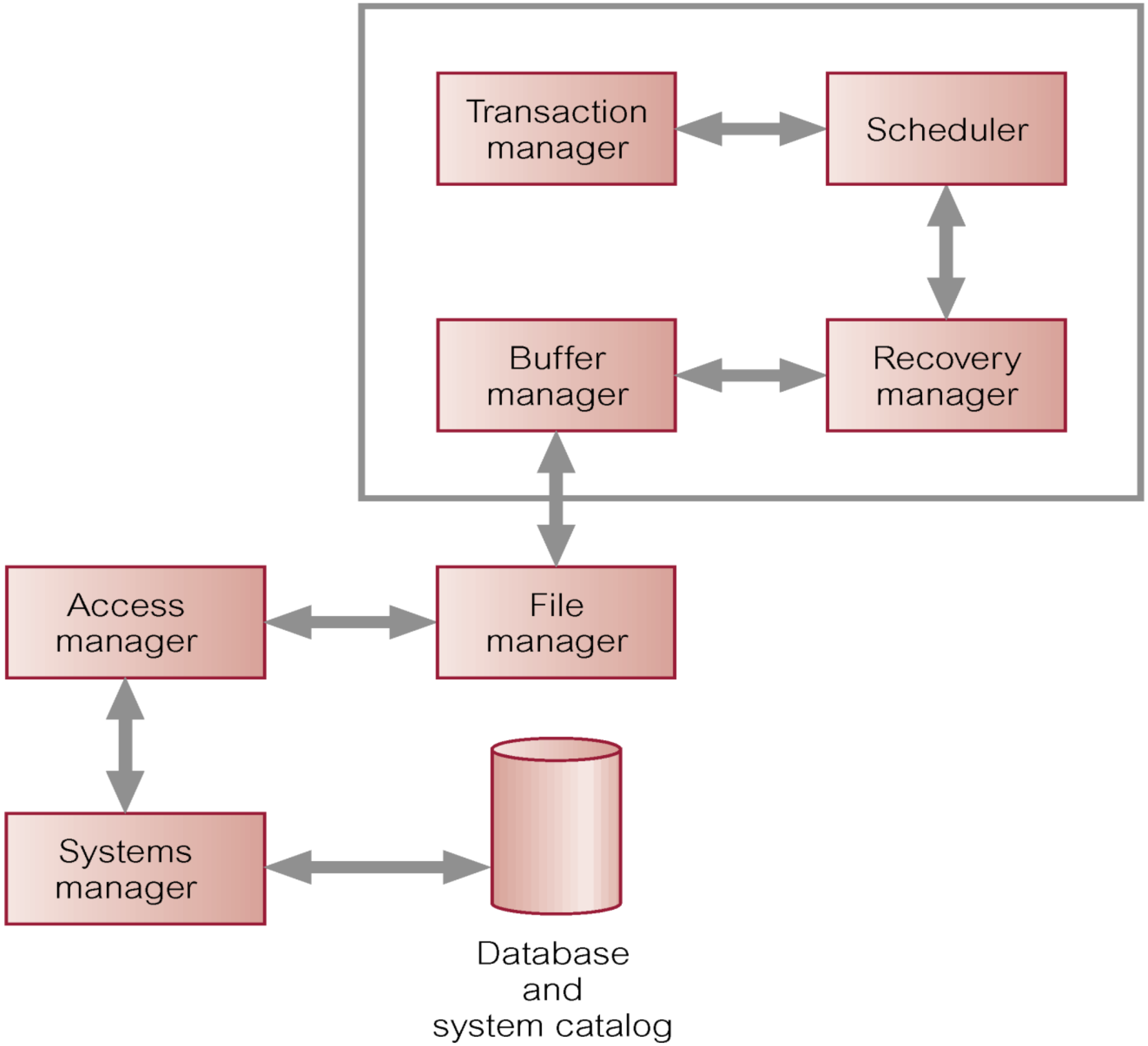
C — The consistency property ensures that any transaction will bring the database from one valid state to another.

I — The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other.

D — The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

https://www.openxcell.com/blog/cockroachdb-more-than-just-an-invincible-database/

DBMS Transaction Subsystem



DBMS Transaction Subsystem

- Concurrency Control
  - To maximize concurrency without allowing concurrently executing transactions to interfere with each other

- Database Recovery
  - To ensure that database is consistent when a failure occurs during the transaction

Concurrency Control

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

- when two or more users are accessing database simultaneously
  - All users are only reading
    - => No way to interfere one another
  - At least one is updating data
    - => May be interfere one another
    - => Can result in inconsistencies

Problems in Concurrency

Potential problems caused by concurrency:

- Lost update problem
- Uncommitted dependency (dirty read)
- Inconsistent analysis problem

The Lost Update Problem

- Successfully completed update is 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)	$\boxed{200}$
t_5	write(bal_x)	commit	90
t_6	commit		90

- At t4 T2 Loss the updated bal_x 200
- This can avoid by preventing T1 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.

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

- T4 updates bal_x to £200 but it aborts
- T3 has read new value of bal_x (£200) giving a new balance of £190, instead of £90.
- This problem can be avoided by preventing T3 from reading bal_x until after T4 commits or aborts.

Inconsistent Analysis Problem

- Occurs when transaction reads several values but second transaction updates some of them during execution of first.

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

- The problem can be avoided by preventing T6 from reading bal_x and bal_z until after T5 completed updates.

Inconsistent Retrieval Problems

- Interference causes inconsistency among multiple retrievals of a subset of data
  - Incorrect summary (Inconsistent analysis problem)
  - Non-repeatable (or fuzzy) read
  - Phantom read

Serializability and Recoverability

- When multiple transactions run concurrently, there is a possible that the database may be left in an inconsistent state.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
- Schedule
  - A sequence of the operations by a set of concurrent transactions that prevents the order of the operations in each of the individual transactions.

Serial Schedule vs. Non-serial Schedule

- Serial schedule
  - Is always a serializable schedule.
  - A schedule where operations of each transaction are executed consecutive without any interleaved operations from other transactions.
  - A transaction only starts when the other transaction finished executed.
- Non-serial schedule
  - A schedule where the operations from a set of current transactions are interleaved.
  - Is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions.

Serializability

- The objective of serializability
  - Is to find non-serial schedules that allow transactions to execute concurrently without interfering with one another, and thereby produce a database state that could be produced by a serial execution.

- In serializability, the ordering of read/writes is important:
  - If two transactions only read a data item, they do not conflict and order is not important.
  - If two transactions either read or write completely separate data items, they do not conflict and order is not important.
  - If one transaction writes a data item and another reads or writes same data item, order of execution is important.

Recoverability

- Serializability identifies schedules that maintain the consistency of database (none of the transactions in the schedule fails)
- Recoverability of transactions within a schedule:
  - If a transaction fails, the atomicity property requires that we undo the effects of transactions.
  - The durability property states that one a transaction commits, its changes cannot be undone (without running another, compensating, transaction).

Unrecoverable Schedule

- G is unrecoverable schedule, because T2 read the value of A written by T1, and committed.
- T1 later aborted, therefore the value read by T2 is wrong, but since T2 committed.

$$G = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ & Com. \\ Abort & \\ & \end{bmatrix}$$

Recoverable Schedule

- A schedule where, for each pair of transactions T_i and T_j, if T_j reads a data item previously written by T_i, then the commit operation of T_i precedes the commit operation of T_j.

$$F = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ Com. & \\ & Com. \\ & \end{bmatrix} F2 = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ Abort & \\ & Abort \\ & \end{bmatrix}$$

- F is recoverable because T1 commits before T2, that makes the value read by T2 corrects, Then T2 can commit itself.
- In F2, if T1 aborted, T2 has to abort because the value of A it read is incorrect

Concurrency Control Techniques

- Two concurrency control techniques
  - Locking
  - Timestamping
- Both are conservative (or pessimistic) approaches
  - They cause transaction to be delayed when they conflict with other transactions
- Optimistic approaches based on:
  - Transaction conflict is rare. So they allow transactions to run unsynchronized and check for conflicts only when the transaction commits at the end

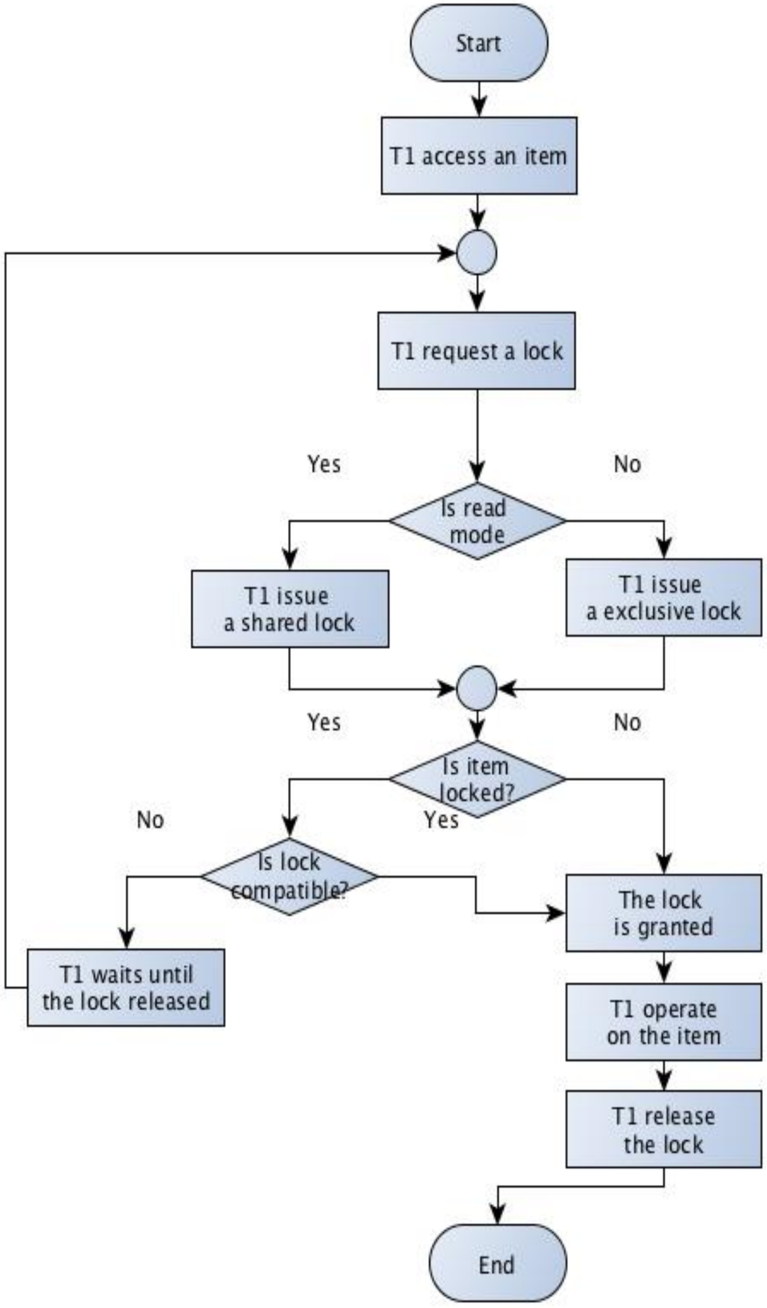
Locking Methods

- Definition:
  - A procedure used to control concurrency access to data. Transaction uses locks to deny access to other transactions to prevent incorrect results.
- Locking methods are the most widely used approach to ensure serializability of concurrent transactions.
- Transaction must request a lock on a data item before it read or write the data item.
- The lock prevents another transaction from modifying or reading the item.
- The other transaction must wait until the lock is released.

Types of a lock

- Two types:
  - Shared lock (read lock)
    - Is used for read-only mode (reading purpose)
    - Transaction requests shared lock on a data item in order to read its content.
    - Updating the data is not allowed
  - Exclusive lock (write lock)
    - Is used for write mode
    - Only the transaction that requests a lock can read and update the data item
    - The other transactions can not read , write or lock the data and must wait until the lock is released

Flowchart of locking



Lock Matrix

	T1 issue a lock at t = 0	
T2 issue a lock at t = 10	Shared	Exclusive
Shared	Both are granted shared locks Both only Read	Only T1 is granted an exclusive lock T2 wait for a lock T1 Read/Write
Exclusive	Only T1 is granted a shared lock T2 wait for a lock T1 only Read	Only T1 is granted an exclusive lock T2 wait for a lock T1 Read/Write

Some systems allow transaction to upgrade shared (read) lock to an exclusive lock, or downgrade exclusive lock to a shared lock.

Locking Granularity