Transaction Management

Advanced Database Systems

When *Events* occur...

Hardware and Software failure

Multiple users are accessing the DB

DBMS should ensure that…

A Database

- ➔ *remains Reliable*
- ➔ *In a Consistent State*

DBMS Functions

- Transaction support
- Concurrency control
- Recovery techniques

Silberschatz, A., Korth, H. & Sudarshan, S. (2010). *Database System Concepts, 6th Edition*. McGraw-Hill Book Co.

Outline

- Transactions
  - Definition
  - Basic Operations – read and write
  - Properties – A.C.I.D.
  - Transaction Operations
  - States and Outcomes
  - Isolation Levels
- Transaction Definition in SQL

Transaction

- A logical unit of processing
  - Entire program
  - Part of a program
  - A single command
- Carried out by
  - A single user
  - An application program
- To access the database
  - Write: Insert, Update, Delete
  - Read: Retrieve

- A sequence of one or more SQL operations treated as a unit

Transaction Boundaries

- Begin transaction
- End transaction

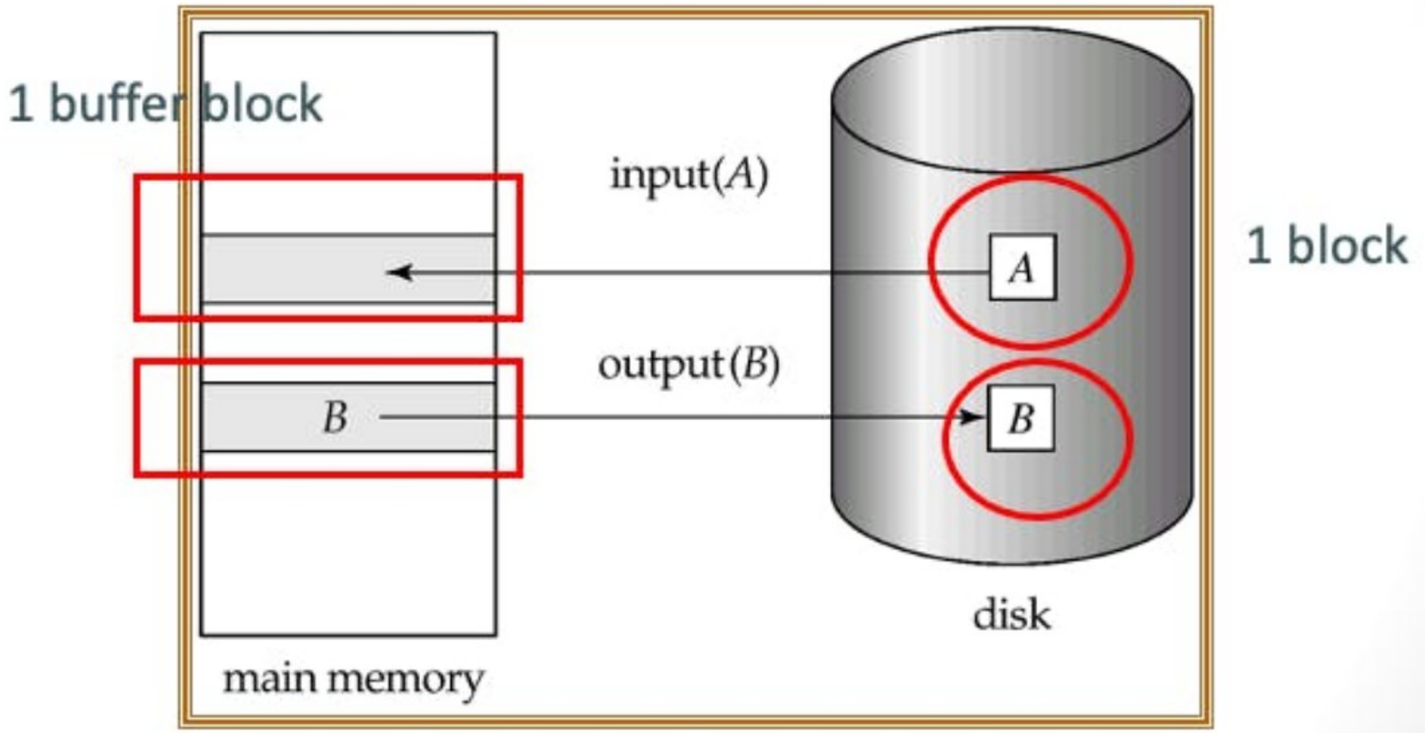
T_1:Withdrawal transaction

Time	T_1	bal_x
t_1	begin _transaction	100
t_2	read(bal_x)	100
t_3	bal_x=bal_x - 10	100
t_4	write(bal_x)	90
t_5	end_transaction	90
t_6	commit	90

Simple Model of a Database

- Database
  - A collection of named data items
  - Permanently resides on a disk, but some portion of it is temporarily residing in memory
- Granularity of data (what is read or written)
  - Field (Column)
  - Record (Tuple)
  - Whole disk block

Block Storage Operations

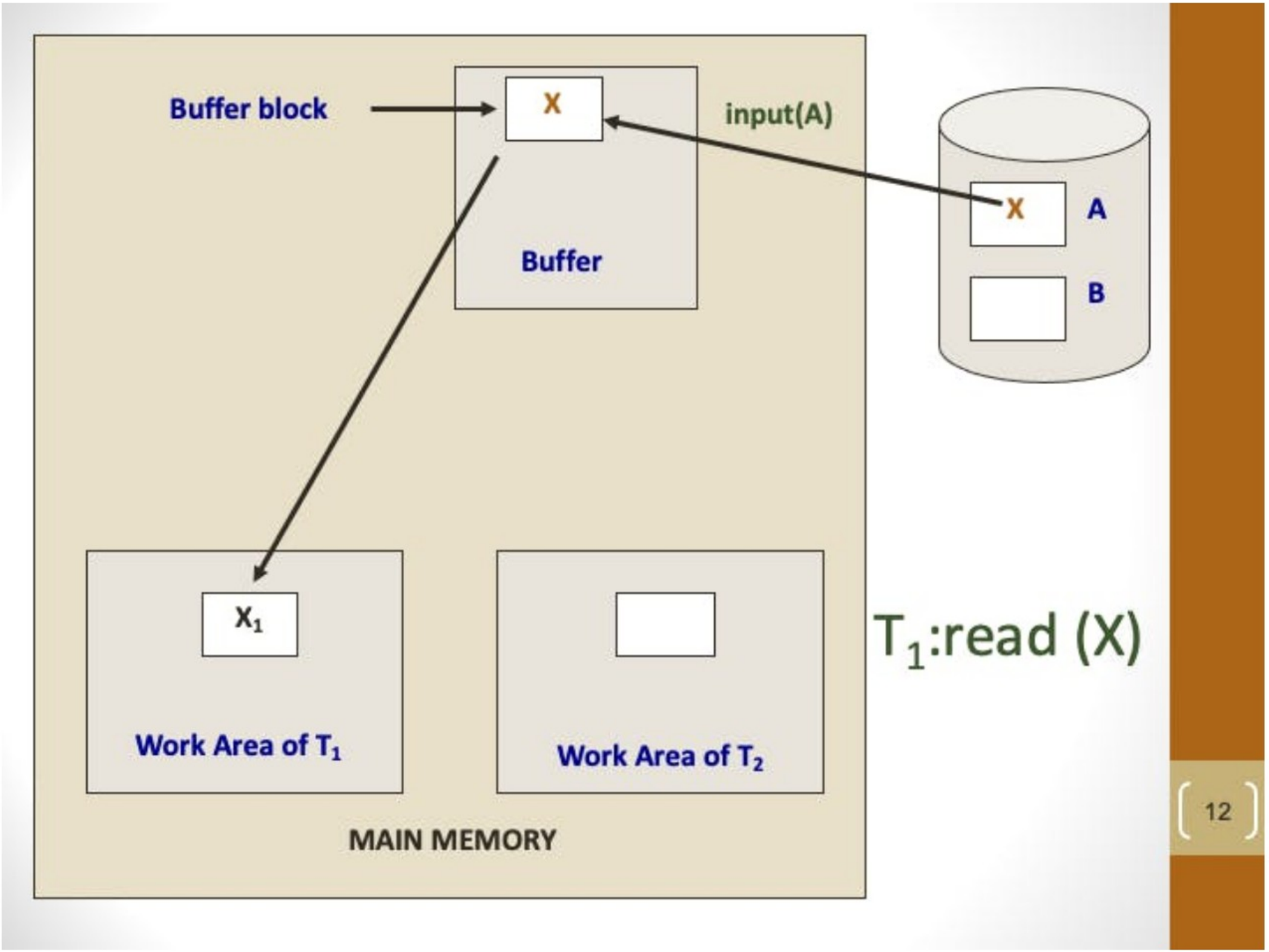


Basic Transaction Operations

- **read(X)**
  - Reads a database item named X into a program variable (local buffer)
- **write(X)**
  - Writes the value of program variable (local buffer) into the database item named X

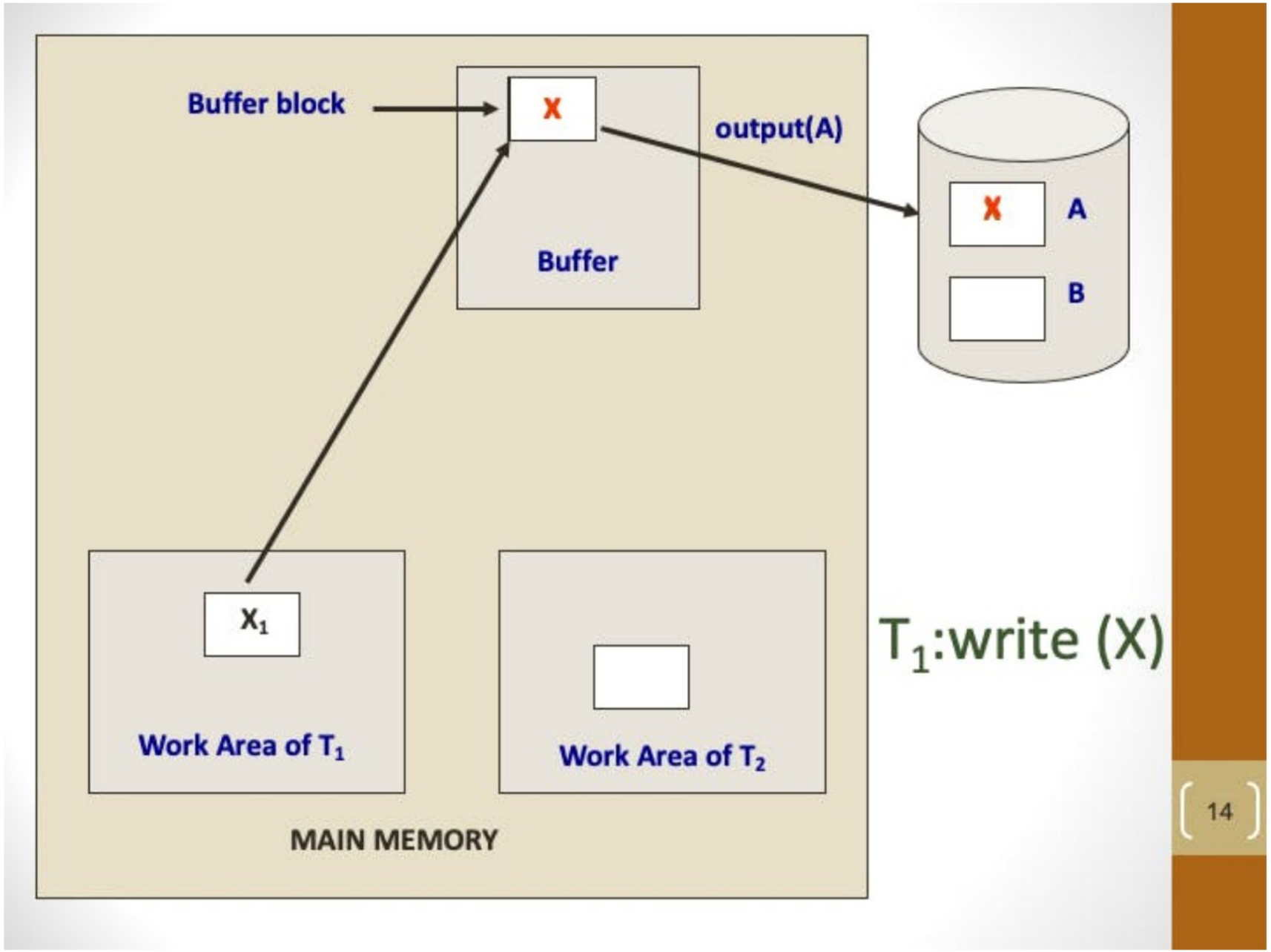
Steps in Read(X)

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable.


Buffer block
X
input(A)
Buffer
X
A
B
X_1
Work Area of T_1
Work Area of T_2
MAIN MEMORY
T_1:read (X)

Steps in Write(X)

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).


Buffer block
X
output(A)
Buffer
X
A
B
X_1
Work Area of T_1
Work Area of T_2
MAIN MEMORY
T_1:write (X)

Transaction: Examples

Given the following database relations:

```
EMPLOYEE (Eno, Name, Address, Sex,
          CivilStatus, Bdate, Position,
          Salary, SuperEno, Dno)
PROJECT (Pno, Pname, Plocation, Dno)
WORKS_ON (Eno, Pno, Hours)
```

Transaction: Example (1)

Update the salary (add 10% to the current salary) of a particular employee with `Eno = x.`

```
EMPLOYEE (Eno, Name, Address, Sex,
          CivilStatus, Bdate, Position,
          Salary, SuperEno, Dno)
```

Transaction consists of 2 dbop and 1 non-dbop.

```
Read(Eno = x,  salary)              dbop
salary =  salary * 1.1
Write(Eno = x, salary)              dbop
```

dbop means "DB Operation"

Transaction: Example (2)

Delete the employee with Eno=x.

Reassign the projects of the employee with Eno=x to new_eno.

```
EMPLOYEE (Eno, Name, Address, Sex,
          CivilStatus, Bdate, Position,
          Salary, SuperEno, Dno)
WORKS_ON (Eno, Pno, Hours)
```

Transaction: Example (2)

```
Delete (Eno = x)                                dbop
for all Works_On records
begin
    read (Pno = pno, Eno)                       dbop
    if (Eno = x) then
        begin
            Eno = new_eno
            write (Pno = pno, Eno)              dbop
        end
end
Write(Eno = x, salary)                          dbop
```

To prevent *Referential Integrity Constraint* violation error,
`Delete (Eno = x)` should be performed last.

Transaction Properties (ACID)

- Atomicity
  - "All or nothing"
  - A transaction is an indivisible unit that is either performed in its entirety or it is not performed at all.
- Consistency
  - A transaction must transform the database from one consistent state to another consistent state.
  - The database must reflect the real-word at all times.

Transaction Properties (ACID)

- Isolation
  - Transactions execute independently of one another.
  - The partial effects of incomplete transactions should not be visible to other transactions.
- Durability
  - The effects of a successfully completed (committed) transactions are permanently recorded in the database.
  - They must not be lost because of a subsequent failure.

Transaction Properties (ACID) An Example

Fund Transfer: Transfer $50 from Account A to Account B.

Pre-State: A = $1000 B = $2000

Post-State: A = $950 B = $2050

```
read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```

CONSISTENCY

- The sum of A and B should be unchanged by the execution of the transaction.
- Ensuring consistency for an individual transaction is the responsibility of the application programmer.

Transaction Properties (ACID) An Example

A = $1000
B = $2000

```
read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
FAIL!
write(B).
```

ATOMICITY

- The DBMS keeps track (on disk) of the old values of any data on which a transaction performs a write.
- If a transaction does not complete its execution, the DBMS restores the old values to make it appear as though the transaction never executed.
- Ensuring atomicity is the responsibility of the transaction-management component of the DBMS.

Transaction Properties (ACID) An Example

A = $1000

B = $2000

```
read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```

DURABILITY

- Updates carried out by the transaction have been written to disk before the transaction completes.
- Information about updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the DBMS is restarted after failure.
- Ensuring durability is the responsibility of the recovery-management component of the DBMS.

Transaction Properties (ACID) An Example

A = $1000

B = $2000

```
T1: read(A);
    A := A - 50;
    write(A);
    read(B);
    B := B + 50;
T2: read(A);
    read(B);
    sum = A + B;
    write(B).
```

ISOLATION

- Execute transactions serially; however performance is affected;
- Other solutions have been developed to allow multiple transactions to execute concurrently.
- Ensuring isolation is the responsibility of the concurrency-control component of the DBMS.

Transactions in DBMS

- The DBMS does not know which updates are grouped together to form a single transaction.
- The user should indicate the boundaries of the transaction through DML commands (delimeters) provided for in SQL2:
  - BEGIN TRANSACTION
  - COMMIT
- If delimeters are not used, the entire program is considered as a single transaction.

Transaction Operations

T	T_1	bal_x
t_1	begin _transaction	100
t_2	read(bal_x)	100
t_3	bal_x=bal_x - 10	100
t_4	write(bal_x)	90
t_5	end_transaction	90
t_6	commit	90

- BEGIN_TRANSACTION
  - Marks the beginning of transaction execution
- READ or WRITE
  - Specify the read or write operations on database items that are executed as part of a transaction

Transaction Operations

- END_TRANSACTION
  - Marks the end limit of transaction execution
  - Specifies that READ and WRITE transaction operations have ended
  - At this point it may be necessary to check whether :
    - The changes introduced by the transaction can be permanently applied (committed) to the database; or
    - The transaction has to be aborted because it violates concurrency control for some other reason.

Transaction Operations

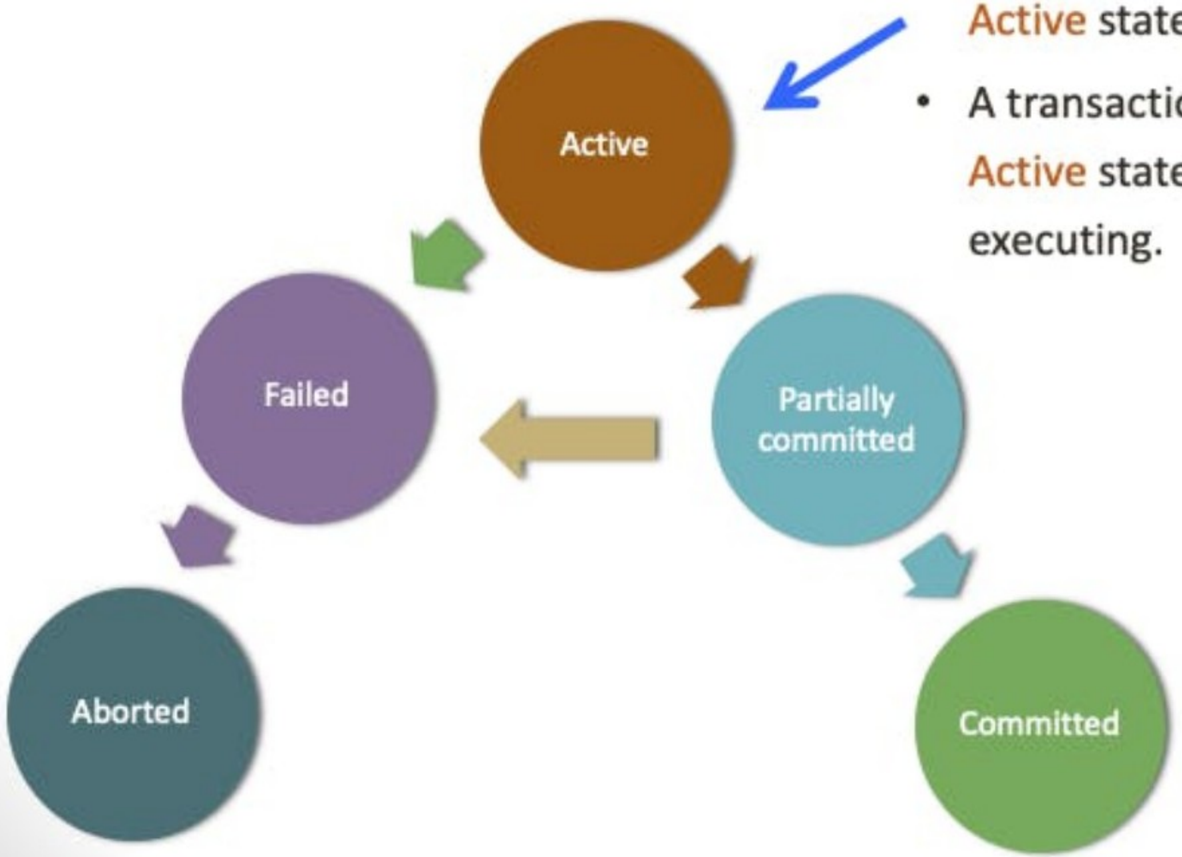
- COMMIT
  - Signals a successful end of the transaction
  - Any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- ROLLBACK
  - Signals that the transaction has ended unsuccessfully
  - Any changes or effects that the transaction may have applied to the database must be undone.

Transaction Operations

- UNDO
  - Similar to rollback except that it applies to a single operation rather than a whole transaction.
- REDO
  - This specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.

Transaction States

- A transaction starts in the Active state.
- A transaction stays in the Active state while it is executing.



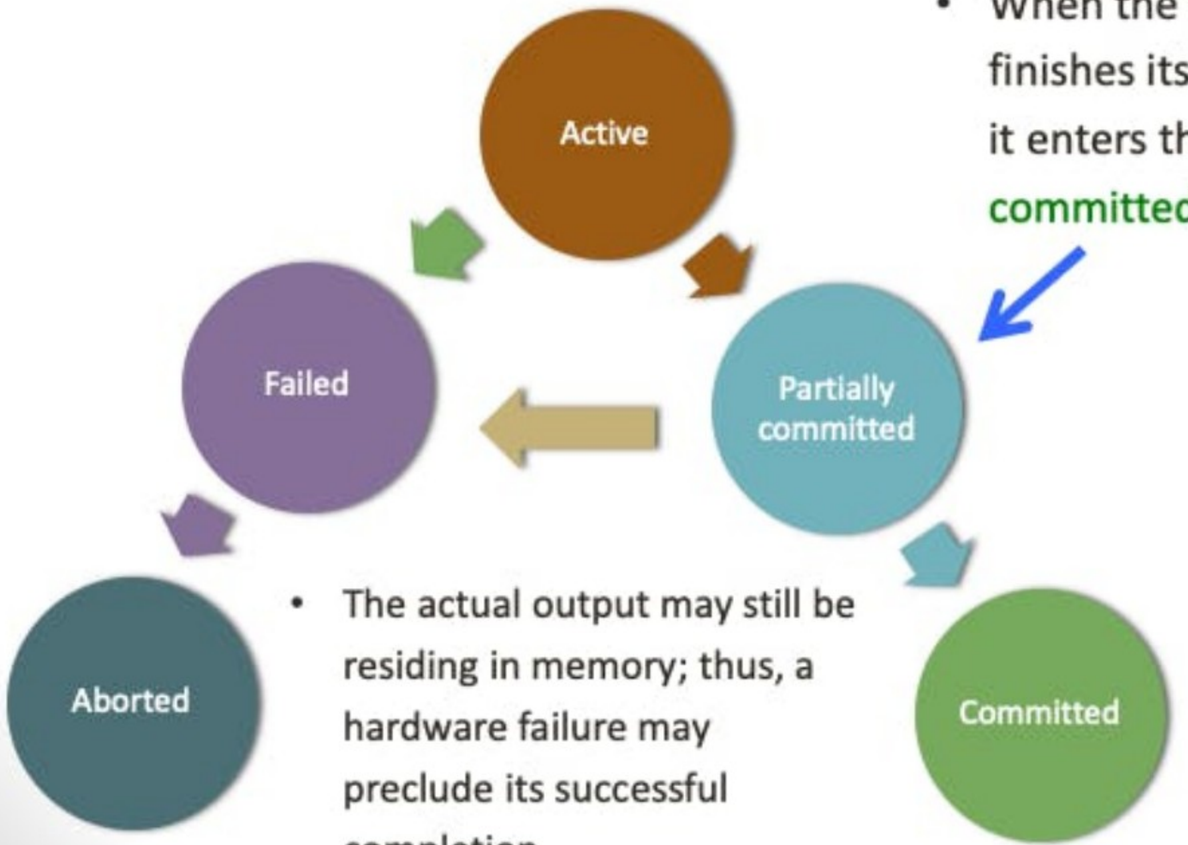
Transaction States: Example

T_1 : Withdrawal transaction

Active

Time	T_1	bal_x
t_1	begin _transaction	100
t_2	read(bal_x)	100
t_3	bal_x=bal_x - 10	100
t_4	write(bal_x)	90
t_5	end_transaction	90
t_6	commit	90

Transaction States



- When the transaction finishes its final statement, it enters the partially committed state.
- The actual output may still be residing in memory; thus, a hardware failure may preclude its successful completion.

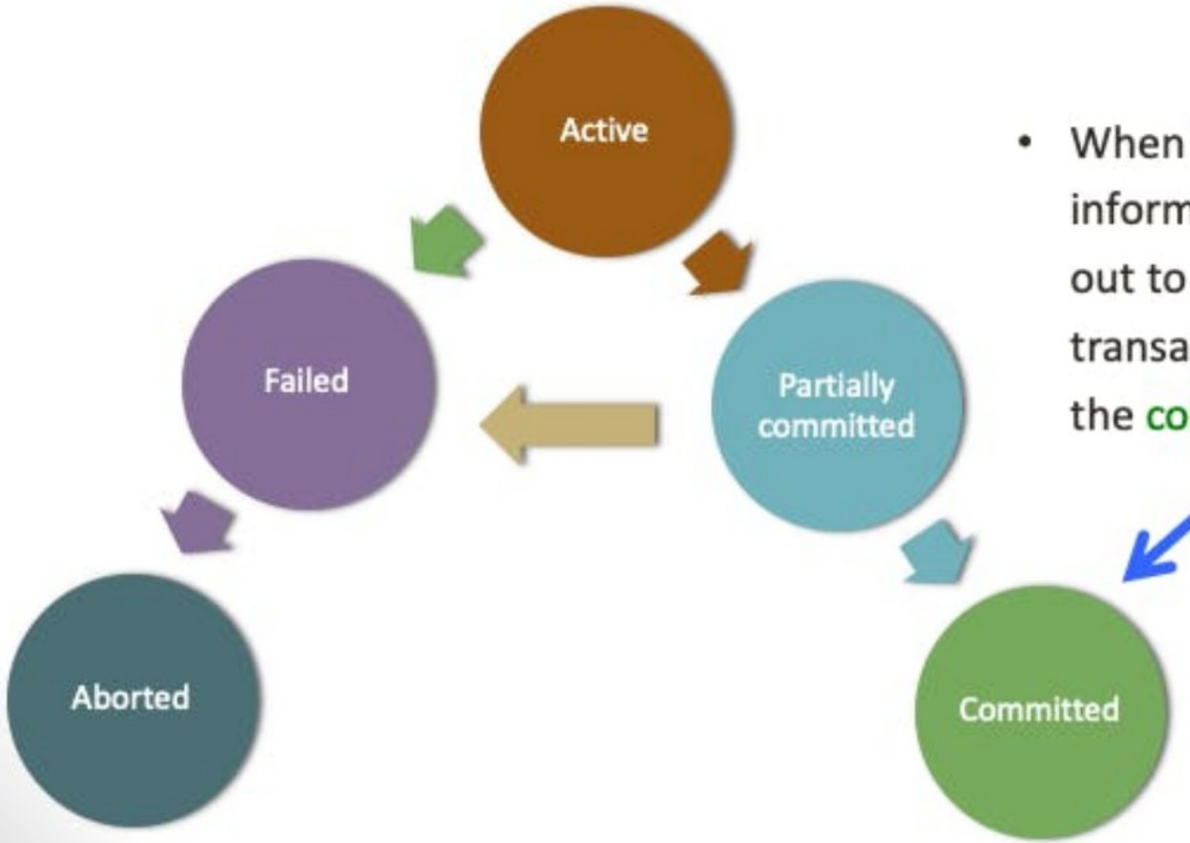
Transaction States: Example

T_1 : Withdrawal transaction

Time	T_1	bal_x
t_1	begin _transaction	100
t_2	read(bal_x)	100
t_3	bal_x=bal_x - 10	100
t_4	write(bal_x)	90
t_5	end_transaction	90
t_6	commit	90

Partially completed

Transaction States



- When the last of the information is written out to disk, the transaction enters the committed state.

Transaction States: Example

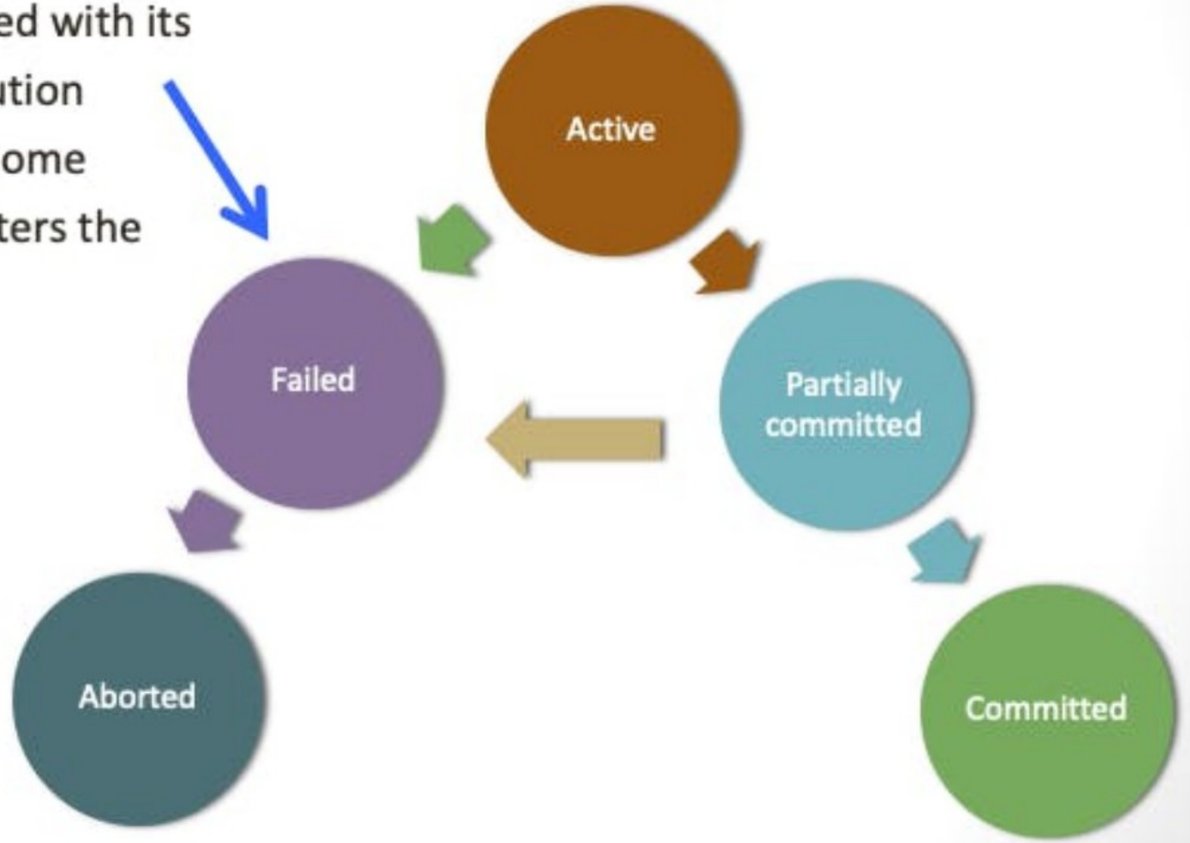
T_1 : Withdrawal transaction

Time	T_1	bal_x
t_1	begin _transaction	100
t_2	read(bal_x)	100
t_3	bal_x=bal_x - 10	100
t_4	write(bal_x)	90
t_5	end_transaction	90
t_6	commit	90

Committed

Transaction States

- If a transaction can no longer proceed with its normal execution (because of some failure), it enters the failed state.



Transaction States: Example

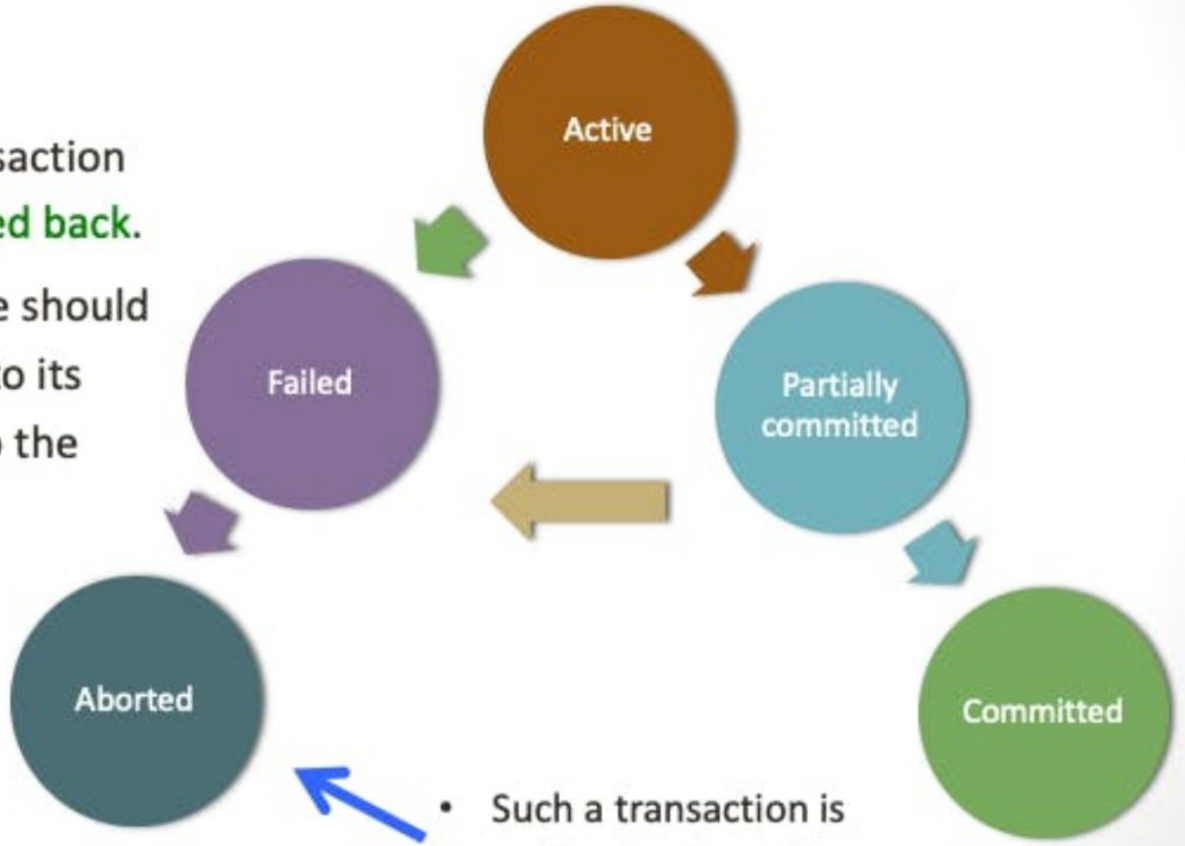
T_4 : Deposit transaction

Time	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	write(bal_x)	200
t_5	… (failure)	
t_6		
t_7		

Failed state

Transaction States

- A failed transaction must be rolled back.
- The database should be restored to its state prior to the start of the transaction.



- Such a transaction is said to be aborted.

Transaction States: Example

T_4 : Deposit transaction

Time	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	write(bal_x)	200
t_5	… (failure)	
t_6	rollback	
t_7		

Transaction Outcomes

- Committed
  - The transaction is successfully completed.
  - The database reaches a new consistent state.
- Aborted
  - The transaction does not execute successfully.
  - The database must be restored to the consistent state it was in before the transaction started.
  - This is called rolled back or undone.

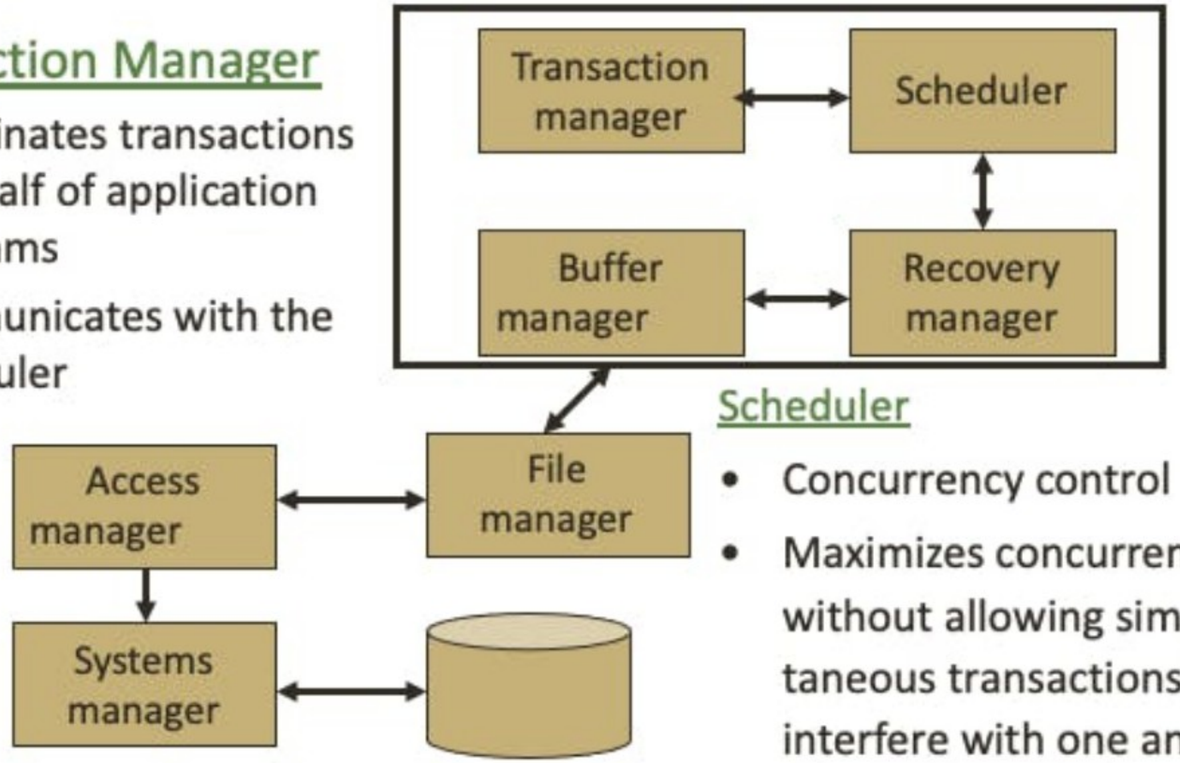
Transaction: Other Processes

- A committed transaction cannot be aborted.
- To reverse the effects of a committed transaction, a compensating transaction must be executed.
- An aborted transaction that is rolled back:
  - Can be restarted later
  - Depending on the cause of failure, it may successfully execute and then commit at that time

DBMS Transaction Subsystem

Transaction Manager

- Coordinates transactions in behalf of application programs
- Communicates with the Scheduler



Scheduler

- Concurrency control
- Maximizes concurrency without allowing simultaneous transactions to interfere with one another

- Recovery Manager - ensures that the database is restored to the right state it was in before a failure occurred
- Buffer Manager – handles data transfer between disk and main memory

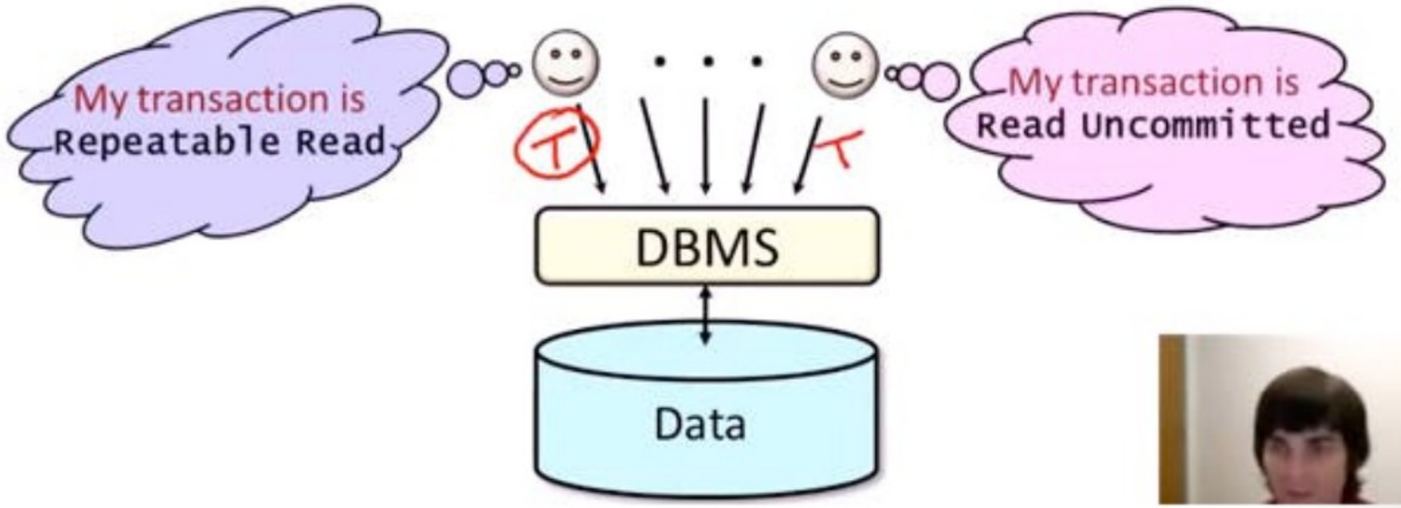
Levels of Isolation Levels

Decreasing level of DB consistency

- Serializable
  - Default
- Repeatable read
  - Only committed records can be read
  - Repeated reads of the same record must return the same value
- Read committed
  - Only committed records can be read
  - Successive reads of record may return different (but committed) values
- Read uncommitted
  - Even uncommitted records may be read

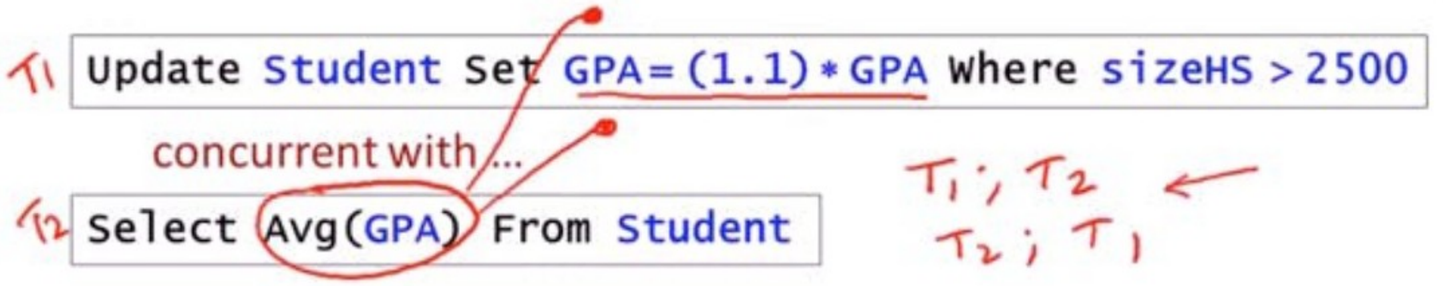
Isolation Levels

- Per transaction
  - Each client can set different isolation levels for its transaction
- "In the eye of the beholder"



Serializable Schedule

- In a Serializable schedule
  - The execution of interleaved transactions, e.g., T1 and T2, should lead to the same results as executing the transactions serially, either <T1, T2> or <T2, T1>



Dirty Reads

- Dirty data item
  - Written by an uncommitted transaction

T1
```
Update College Set enrollment=enrollment+1000
Where cName='Stanford'
```
Commit

concurrent with ...

T2
```
Select Avg(enrollment) From College
```
commit

Dirty read cannot happen within the same transaction

T1
```
Update Student Set GPA=(1.1)*GPA Where sizeHS>2500
```

concurrent with ...

T2
```
Select GPA From Student Where sID=123
```

concurrent with ...

T3
```
Update Student Set sizeHS=2600 Where sID=234
```

Read Uncommitted

- A transaction may perform dirty reads

Serializable

```
Update Student Set GPA=(1.1)*GPA Where sizeHS>2500
```

concurrent with ...

```
Set Transaction Isolation Level Read Uncommitted;
Select Avg(GPA) From Student;
```

- In the given example, the Isolation Level setting allows the Select statement to read dirty data if the Update has not committed yet

Read Committed

Transactions

Isolation Level **Read Committed**

- A transaction may *not* perform dirty reads

  Still does not guarantee global serializability

T1
```
Update Student Set GPA=(1.1)*GPA Where sizeHS>2500
```

concurrent with ...

T2
```
Set Transaction Isolation Level Read Committed;
Select Avg(GPA) From Student;
Select Max(GPA) From Student;
```

- In the given example, the 2 SELECT statements may read committed but different GPA values if the reads were performed before and after the Update statement.
- Is there an equivalent serial schedule?

Repeatable Read

- A transaction may not perform dirty reads
- An item read multiple times cannot change value
- Still does not guarantee global serialization

T1
```
Update Student Set GPA=(1.1)*GPA;
Update Student Set sizeHS=1500 Where sID=123;
```

concurrent with ...

T2
```
Set Transaction Isolation Level Repeatable Read;
Select Avg(GPA) From Student;
Select Avg(sizeHS) From Student;
```

- Why is the schedule still not serializable despite not performing dirty reads?

Repeatable Read

- A relation can change while being read multiple times through "phantom" tuples

T1
```
Insert Into Student [ 100 new tuples ]
```
concurrent with ...

T2
```
Set Transaction Isolation Level Repeatable Read;
Select Avg(GPA) From Student;
Select Max(GPA) From Student;
```

- Repeatable Read allows the AVG operation in T2 to be executed before T1 and MAX to be executed after T1.
- Thus, if the second SELECT statement if T2 is another AVG operation, then the results would not be the same.
- Phantom tuples are those that emerge during an execution.

Repeatable Read

- The previous case where 2 separate Read operations of T2 were allowed to be executed before and after T1 can only happen during Insert, but not Delete. Why?

```
Delete From Student [ 100 tuples ]
```

concurrent with

```
Set Transaction Isolation Level Repeatable Read;
Select Avg(GPA) From Student;
Select Max(GPA) From Student;
```

Isolation Levels: Summary

- Weaker Isolation Levels
  - Increased concurrency + Decreased overhead = Increased performance
  - Weaker consistency guarantees
  - Some systems have default Repeatable Read (e.g., Oracle, MySQL)
- Each transaction's reads must conform to its Isolation level

weak

	dirty reads	nonrepeatable reads	phantoms
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

Transaction Definition in SQL

- SQL standard
  - A transaction begins implicitly
- To end a transaction
  - Commit [work] – commits the current transaction and begins a new one
  - Rollback [work] – causes the current transaction to abort
- By default in almost all DBMS, every SQL statement commits implicitly if it executes successfully
  - To turn off Implicit Commit, e.g., in JDBC

```
connection.setAutoCommit(false);
```

References

- Read the following:

Chapter 14: Transactions
Chapter 15: Concurrency
Chapter 16: Recovery

Silberschatz, A., Korth, H. & Sudarshan, S. (2010). *Database System Concepts, 6th Edition*. McGraw-Hill Book Co.

Chapter 22: Transaction Management

Connolly, T. & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management, 6th Edition. Harlow, Essex: Addison-Wesley

- Watch the videos of Jennifer Widom (Stanford University), eg,

Introduction to Transactions
Transactions – Isolation Levels