TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Database
Lesson 12. Transaction management

Viet-Trung Tran
https://is.hust.edu.vn/~trungtv/

Learning Map

Sequence	Title
1	Introduction to Databases
2	Relational Databases
3	Relational Algebra
4	Structured Query Language – Part 1
5	Structured Query Language – Part 2
6	Constraints and Triggers
7	Entity Relationship Model
8	Functional Dependency
9	Normalization
10	Storage - Indexing
11	Query Processing
12	Transaction Management – Part 1
13	Transaction Management – Part 2

Outline

- Transaction
  - Definition
  - ACID properties
  - Transaction states
  - Transaction management interfaces
- Concurrency control
  - Objective
  - Scheduling
  - Serializability
  - Locks
  - Locking protocol
  - 2 Phase Locking (2PL)

Objectives

- Upon completion of this lesson, students will be able to:
  - Understanding transaction and ACID properties
  - Understanding locking and 2 phase locking algorithm

1. Transaction

- Definition
- ACID properties
- Transaction states
- Transaction management interfaces

1.1. What is transaction

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- Example: transfer \$50 from account *A* to account *B*
  - 1. $R(A)$
  - 2. $A \leftarrow A - 50$
  - 3. $W(A)$
  - 4. $R(B)$
  - 5. $B \leftarrow B + 50$
  - 6. $W(B)$
- Two main issues:
  - 1. concurrent execution of multiple transactions
  - 2. failures of various kind (e.g., hardware failure, system crash)

1.2. ACID Properties

- Database system must guarantee ACID for transactions:
  - **Atomicity**: either all operations of the transaction are executed or none
  - **Consistency**: execution of a transaction in isolation preserves the consistency of the database
  - **Isolation**: although multiple transactions may execute concurrently, each transaction must be unaware of the other concurrent transactions.
  - **Durability**: After a transaction completes successfully, changes to the database persist even in case of system failure.

1.2.1. Atomicity

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- **What if failure (hardware or software) after step 3?**
  - money is lost
  - database is inconsistent
- **Atomicity:**
  - either all operations or none
  - updates of partially executed transactions not reflected in database

1.2.2. Consistency

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- **Consistency in example**: sum $A + B$ must be unchanged
- **Consistency in general:**
  - explicit integrity constraints (e.g., foreign key)
  - implicit integrity constraints (e.g., sum of all account balances of a
  - bank branch must be equal to branch balance)
- **Transaction:**
  - must see consistent database
  - during transaction inconsistent state allowed
  - after completion database must be consistent again

1.2.3. Isolation

- Example: transfer \$50 from account A to account B
  - 1. $R(A)$
  - 2. $A \leftarrow A - 50$
  - 3. $W(A)$
  - 4. $R(B)$
  - 5. $B \leftarrow B + 50$
  - 6. $W(B)$
- Imagine second transaction $T2$:
  - $T2 : R(A), R(B), print(A + B)$
  - $T2$ is executed between steps 3 and 4
  - $T2$ sees an inconsistent database and gives wrong result

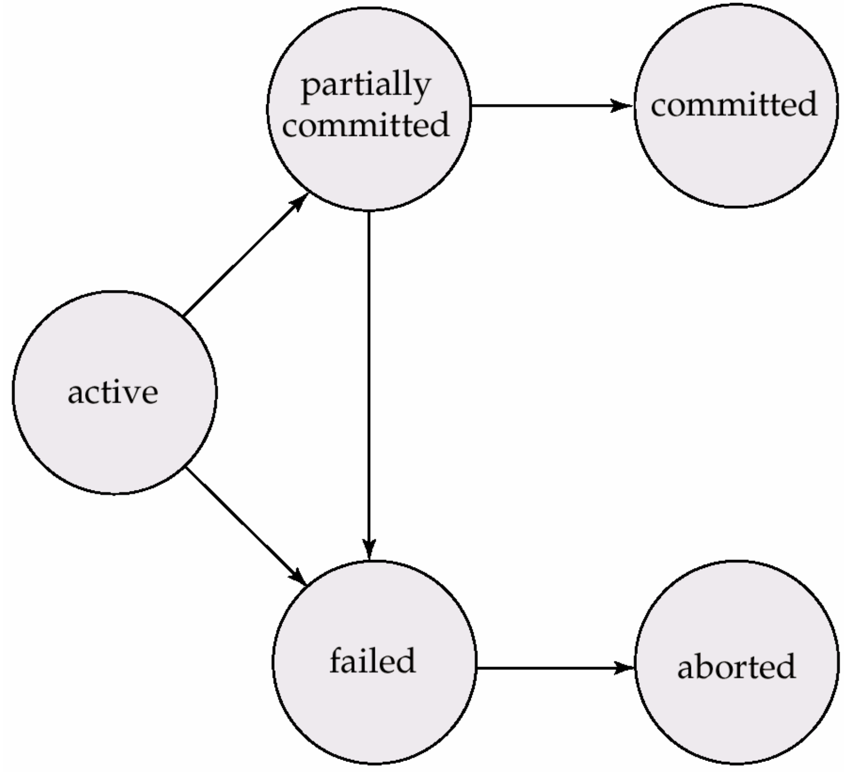
1.2.3. Isolation

- **Trivial isolation:** run transactions serially
- Isolation for concurrent transactions:
  - For every pair of transactions *Ti* and *Tj*,
  - it appears to *Ti* as if either *Tj* finished execution before *Ti* started
  - or *Tj* started execution after *Ti* finished.

1.2.4. Durability

- When a transaction is done it commits.
- **Example**: transaction commits too early
  - transaction writes *A*, then commits
  - *A* is written to the disk buffer
  - then system crashes
  - value of *A* is lost
- **Durability**: After a transaction has committed, the changes to the database persist even in case of system failure.
- Commit only after all changes are permanent:
  - either written to log file or directly to database
  - database must recover in case of a crash

1.3. Transaction states



1.4. Transaction management interfaces

- Begin Trans
- Commit ()
- Abort()
- Savepoint Save()
- Rollback (savepoint)

  (savepoint = 0 ==> Abort)

2. Concurrency control

- Objective
- Scheduling
- Serializability
- Locks
- Locking protocol
- 2 Phase Locking (2PL)

2.1. Objective

- Objective:
  - ensures that database transactions are performed concurrently without violating the data integrity
  - guarantees that no effect of committed transactions is lost, and no effect of aborted (rolled back) transactions remains in the related database.
- Example

T0	T1

read(A);
A := A -50;
write(A);
read(B);
B := B + 50;
write(B);
	read(A);
	temp := A *0.1;
	A := A -temp;
	write(A);
	read(B);
	B := B + temp;
	write(B);

2.1. Scheduling

T_0	T_1
read(A)	
A := A - 50	
write(A)	
read(B)	
B := B + 50	
write(B)	
	read(A)
	temp := A * 0.1
	A := A -temp
	write(A)
	read(B)
	B := B + temp
	write(B)

(1)

T_0	T_1
	read(A)
	temp := A * 0.1
	A := A -temp
	write(A)
	read(B)
	B := B + temp
	write(B)
read(A)	
A := A - 50	
write(A)	
read(B)	
B := B + 50	
write(B)	

(2)

T_0	T_1
read(A)	
A := A - 50	
	read(A)
	temp := A * 0.1
	A := A -temp
	write(A)
	read(B)
write(A)	
read(B)	
B := B + 50	
write(B)	
	B := B + temp
	write(B)

(3)

2.2. Serializability

- A schedule of a set of transactions is a linear ordering of their actions
  - e.g. for the simultaneous deposits example:

    R1(X) R2(X) W1(X) W2(X)
- A serial schedule is one in which all the steps of each transaction occur consecutively
- A serializable schedule is one which is equivalent to some serial schedule

2.3. Locks

- A lock is a mechanism to control concurrency on a data item.
  - one way of enforcing concurrency control policies
- Two types of locks on a data item A:
  - exclusive – $xL(A)$: data item A can be both read and written
  - shared – $sL(A)$: data item A can only be read.
- Lock request are made to concurrency control manager.
- Transaction is blocked until lock is granted.
- Unlock A – $uL(A)$: release the lock on a data item A

2.3.1. Lock Compatibility

- Lock compatibility matrix:

$T_1 \downarrow \quad T_2 \rightarrow$	shared	exclusive
shared	true	false
exclusive	false	false

- *T*1 **holds shared lock on** *A*:
  - shared lock is granted to *T*2
  - exclusive lock is not granted to *T*2
- *T*2 **holds exclusive lock on** *A*:
  - shared lock is not granted to *T*2
  - exclusive lock is not granted to *T*2
- Shared locks can be shared by any number of transactions.

2.3.2. Example

T0: LX(A);	T1: LX(A);
read(A);	read(A);
A := A -50;	temp := A
*0.1;	
write(A);	A := A -temp;
LX(B);	write(A)
read(B);	LX(B);
B := B + 50;	read(B);
write(B);	B:=B+temp;
UN(A);	write(B);
UN(B);	UN(A);
	UN(B);

2.4. Locking protocol

- Example transaction $T2$ with locking:
  1. $sL(A)$, $R(A)$, $uL(A)$
  2. $sL(B)$, $R(B)$, $uL(B)$
  3. $print(A + B)$
- $T2$ uses locking, but is not serializable
  - A and/or B could be updated between steps 1 and 2
  - printed sum may be wrong
- Locking protocol:
  - set of rules followed by all transactions while requesting/releasing locks
  - locking protocol restricts the set of possible schedules

2.4.1. Pitfalls of locking protocols – Deadlock

- **Example**: two concurrent money transfers

T0		T1	
T0: LX(B);	(1)	T1: LX(A);	(4)
read(B);	(2)	read(A);	(5)
B := B +50;	(3)	temp := A *0.1;	(6)
write(B);	(8)	A := A -temp;	(7)
LX(A);	(10)	write(A)	(9)
read(A);		LX(B);	
A := A - 50;		read(B);	
write(A);		B:=B+temp;	
UN(A);		write(B);	
UN(B);		UN(A);	
		UN(B);	

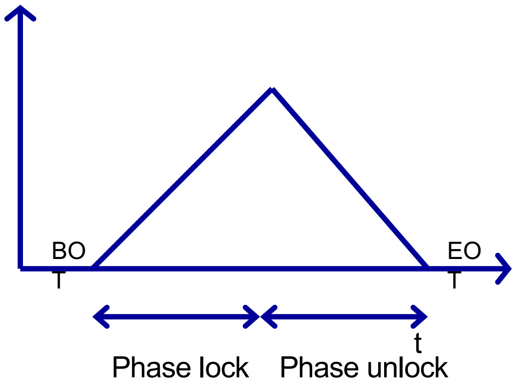
- **Deadlock**: situation when transactions block each other
- **Handling deadlocks:**
  - one of the transactions must be rolled back (i.e., undone)
  - rolled back transaction releases locks

2.4.2. Pitfalls of locking protocols – Starvation

- **Starvation**: transaction continues to wait for lock
- Examples:
  - the same transaction is repeatedly rolled back due to deadlocks
  - a transaction continues to wait for an exclusive lock on an item while a sequence of other transactions are granted shared locks
- Well-designed concurrency manager avoids starvation.

2.5. 2 Phase Locking (2PL)

- Protocol that guarantees serializability.
  - Phase 1
    - locks are acquired and no locks are released
  - Phase 2
    - locks are released and no locks are acquired



2.5.1. Example

T_1
Lock(A)
Read(A)
Lock(B)
Read(B)
B:=B+A
Write(B)
Unlock(A)
Unlock(B)

T_2
Lock(B)
Read(B)
Lock(A)
Read(A)
Unlock(B)
A:=A+B
Write(A)
Unlock(A)

2PL

T_3
Lock(B)
Read(B)
B=B-50
Write(B)
Unlock(B)
Lock(A)
Read(A)
A=A+50
Write(A)
Unlock(A)

T_4
Lock(A)
Read(A)
Unlock(A)
Lock(B)
Read(B)
Unlock(B)
Pritn(A+B)

Not 2PL

Summary

- Transaction and ACID properties
- Concurrency control based on locking mechanisms
- 2 phase locking (2PL)

Next lesson: Transaction management

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database Systems: The Complete Book. Pearson Prentice Hall. the 2nd edition. 2008: Chapter 7
- Nguyen Kim Anh, Nguyên lý các hệ cơ sở dữ liệu, NXB Giáo dục. 2004: Chương 7