Nguyen Tat Thanh Institute of International Education (NIIE)

DATABASE MANAGEMENT SYSTEMS

(Creadits 3)

MSc. Luong Tran Ngoc Khiet

May - 2021

Course content

Chap 1. Overview

Chap 2. Data storage management

Chap 3. Programming with Cursors

Chap 4. Query optimization

Chap 5. Continuous transaction processing

Chap 5. Continuous transaction processing

MSc. Luong Tran Ngoc Khiet

NTT Institute of International Education (NIIE)

Goal

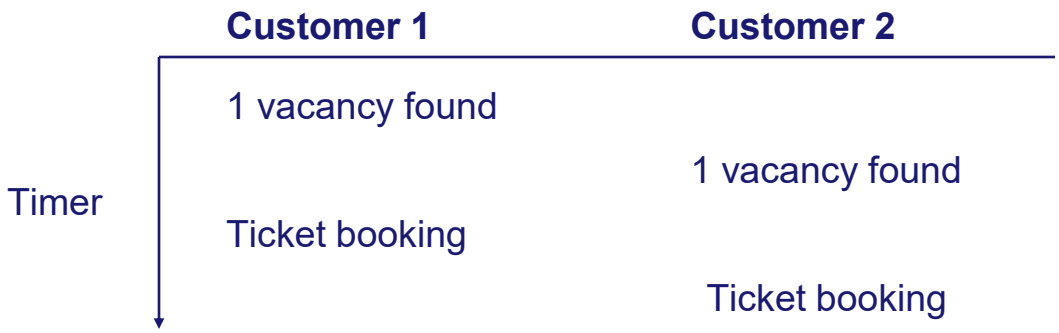
- The concept of transaction.
- Problems occur when many people exploit the Database at the same time.
- Solutions to the above problems.
- Use transactions in SQL Server.

Transaction

- A transaction is a series of operations to be performed on the database under a single unit, meaning either performing all operations or doing none at all.

Transaction

- For example:
  - Bank transaction system
  - Flight ticket booking system
- DBMS is a multi-user environment
  - Multiple access operations on the same data unit
  - Multiple operations execute simultaneously



2 customers booked the same empty seat???

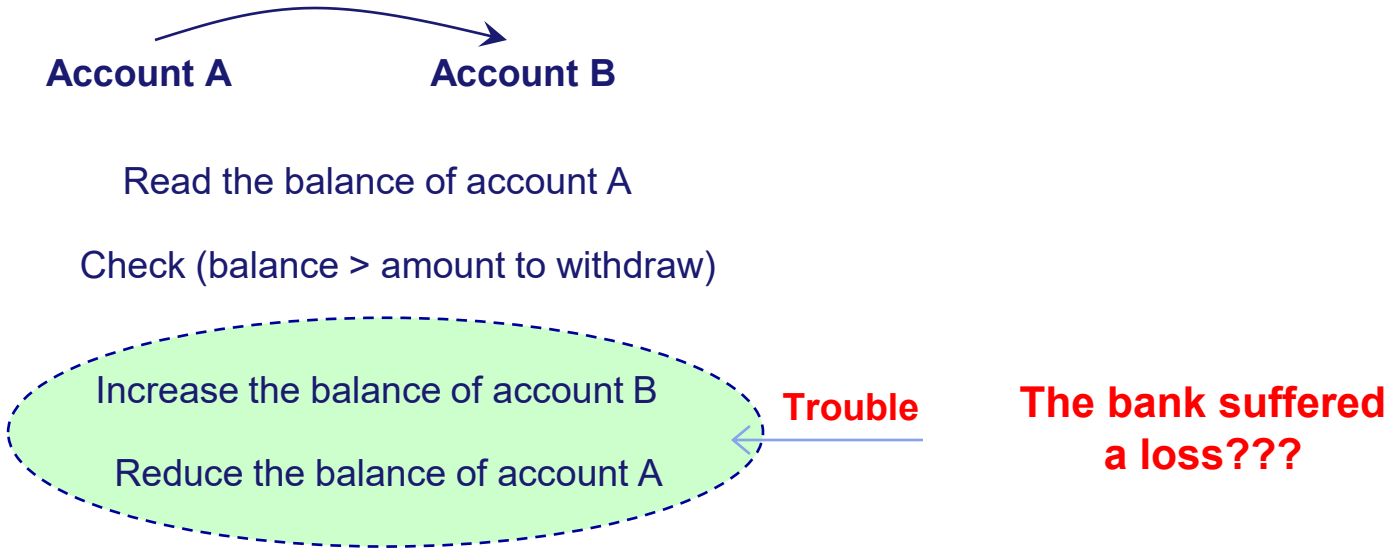
Sequential mechanism

Transaction

- For example, a transfer transaction from A → B includes the following 2 operations:
  - Subtract money A
  - Add money B
- Transfers are made in the form of a transaction, which means either subtracting money A and adding money B, or if there is a problem, do nothing and notify the transaction of failure.

Transaction

- When DBMS crashes
  - Operations that may cause the database state to be incorrect



Concurrent access problems

- The data loss issue has been updated
- The problem cannot be reread
- The problem of inconsistent data

Concurrent access problems

The data loss issue has been updated

- Example: The bookstore has 500 books left.
  - At time T1, employee A requests to buy 400 books from customer X.
  - Also at T1, employee B receives a request to buy 300 books from customer Y.
  - A and B read the data and saw that there were 500 copies left, so they both agreed to sell

Concurrent access problems

The data loss issue has been updated

- Example: The bookstore has 500 books left.
  - At time T2, employee A will update the number of books from 500 to 100.
  - At time T3, employee B will update the number of books from 500 to 200.
- So A's update operation has no effect or will the data that A updates be lost because B updates later??? (last in wind)

Concurrent access problems

The problem cannot be reread

- Example: The bookstore has 200 books left.
  - At the time T1, employee A sells 150 books to the customer, the number of books will be updated from 200 to 50. (The transaction is not completed, for example because the delivery of money has not been completed)
  - Then at T2, B receives a request to buy 100 books. If B reads the incomplete data, B will refuse to sell these 100 books.

Concurrent access problems

The problem cannot be reread

- Example: The bookstore has 200 books left.
  - If at T3 for some reason, such as not having enough money, A's customer does not buy 150 books. A's sales transaction cannot be processed, so it returns to the status of 200 books.
  - But B refused the customer.
  - What if B cannot read the data from T1 to T3?

Concurrent access problems

The problem of inconsistent data

- Example: Suppose employee C needs to synthesize 5 lines of data 1 2 3 4 5 to make a report.
  - T1:C reads and puts lines 1 2 3 4 into the report
  - T2:D delete line 1 and replace it with line 6.
  - T3:C continues reading lines 5 and 6 and puts them in the report
  - So this report handles both old and new data → FALSE

ACID nature of transactions

- **A**tomicity - Nguyên tố
  - Either all transaction activity is properly reflected in the database or there is no activity at all
- **C**onsistency - Nhất quán
  - A transaction is executed independently of other transactions processed concurrently with it to ensure database consistency.
- **I**solation - Cô lập
  - A transaction does not care about other transactions processing concurrently with it
- **D**urability - Bền vững
  - Every change a transaction makes to the database must be persistently recorded

ACID nature of transactions - Sample

```
T: Read(A,t);
   t:=t-50;
   Write(A,t);
   Read(B,t);
   t:=t+50;
   Write(B,t);
```

- Consistency - Nhất quán
  - The sum A+B is unchanged
  - If the database is consistent before T is executed, then after T completes the database will still be consistent

ACID nature of transactions - Sample

```
T: Read(A,t);
   t:=t-50;
   Write(A,t);
   Read(B,t);
   t:=t+50;
   Write(B,t);
```

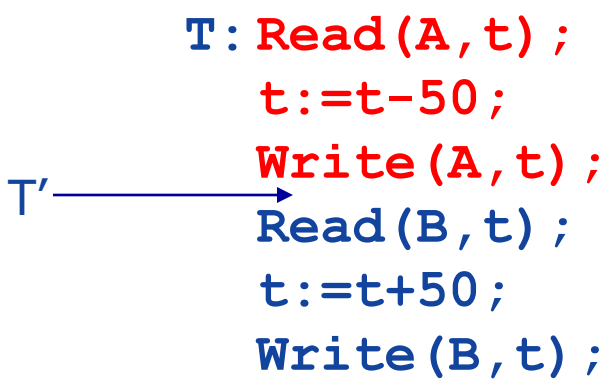
- Atomicity - Nguyên tố
  - A=100, B=200 (A+B=300)
  - At the moment after write(A,t)
    - A=50, B=200 (A+B=250) - CSDL not consistent
  - At the moment after write(B,t)
    - A=50, B=250 (A+B=300) - CSDL consistent
  - If T never begins execution or T is guaranteed to complete, then inconsistent state will not occur

ACID nature of transactions - Sample

```
T: Read(A,t);
   t:=t-50;
   Write(A,t);
   Read(B,t);
   t:=t+50;
   Write(B,t);
```

- Durability - Bền vững
  - When T finishes successfully
  - Data will not be lost even if a system failure occurs

ACID nature of transactions - Sample

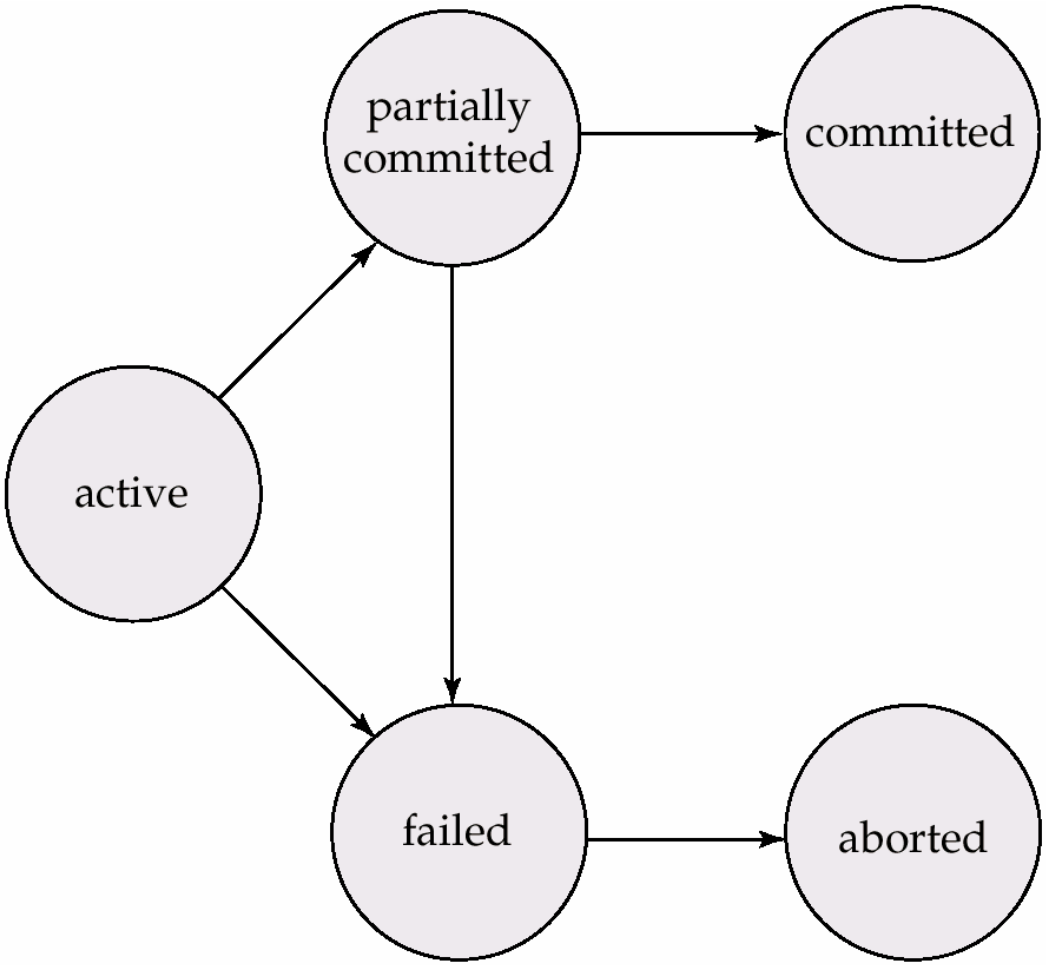


- Isolation - Cô lập
  - Suppose there is a transaction T' that performs operation A+B and intervenes in the execution time of T
  - T' ends: A+B=50+200=250
  - T ends: A+B=50+250=300
  - The system of transactions performed concurrently has a state equivalent to the system state of transactions performed sequentially in a certain order.

Status of the transaction

- Active: As soon as the read/write operation begins
- Partially committed: After the final execution command executes
- Failed: After realizing the actions can no longer be performed
- Aborted: After the transaction is rolled back and the database is restored to the state before the transaction started state
  - Restart the transaction (if possible)
  - Cancel transaction.
- Committed: After all actions complete successfully

Transaction state diagram



Transaction classification

- Explicit transaction - Giao dịch tường minh
- Implicit transaction - Giao dịch ngầm định
- Commit transaction - Giao dịch xác nhận

Locks

- Lock Lock was born to limit access rights in multi-user environments.
- Microsoft SQL Server 200X uses locks to ensure transaction integrity and database consistency.
- If locks are not used, the data within the database may be logically corrupted, and queries run on it will produce unexpected results.
- The nature of lock is that one person wants to privately access a table, so the server will lock that table for that person alone.

Classification of Locks in SQL Server

- Pessimistic Lock
- Optimistic Lock
- Shared Locks
- Exclusive Locks
- Update Locks

Locking technique

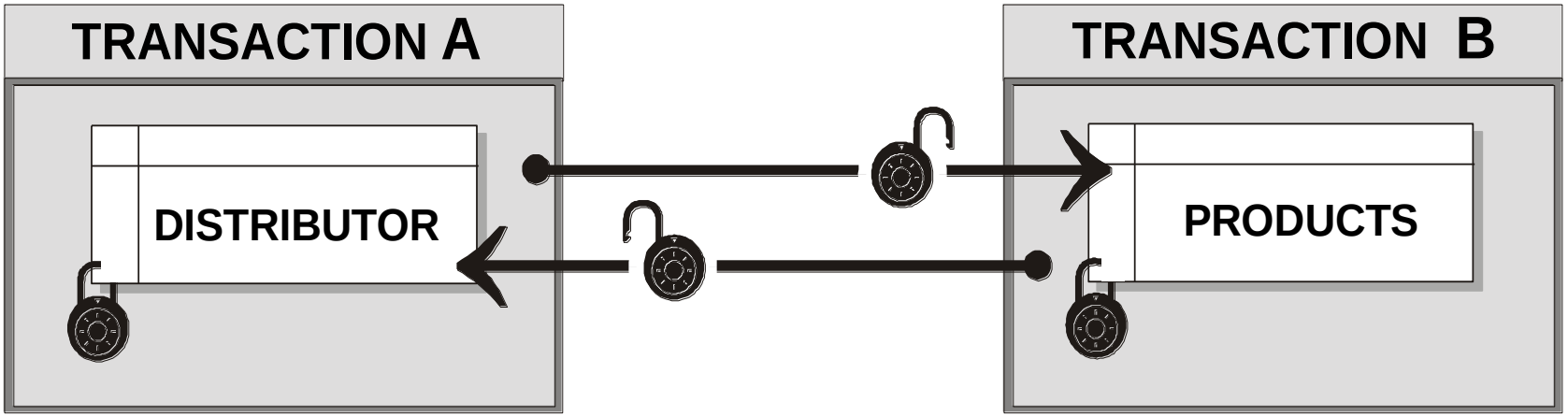
- **Share Lock**: When a transaction is reading data X, X will be shared lock. This means that other transactions at the same time only have the right to read X and do not have the right to edit X
- **Exclusive Lock**: When a transaction is updating data X, X will be exclusively locked. This means that other transactions at the same time cannot read or modify X.

Locking technique

- 2-stage locking technique
- Locking techniques on hierarchical data
- Dead lock: is a situation in which two or more transactions are in a state waiting for the transaction to release the resources needed to complete the transaction..

Locking technique

- Dead lock



Deadlock

- A deadlock occurs when two users (or two sessions) have placed locks on two separate objects, and each user wants to place locks on the other user's objects. Each user must wait for the other to release their key so they can set the key.
- SQL Server automatically recognizes deadlocks and resolves them by selecting one application and forcing it to release the lock, while allowing the other application to continue running.
- The best way to avoid deadlock is to avoid it. One way to avoid it is to not run concurrent transactions.

Solve Deadlock

- SET DEADLOCK_PRIORITY
- SET LOCK_TIMEOUT

Solve Deadlock

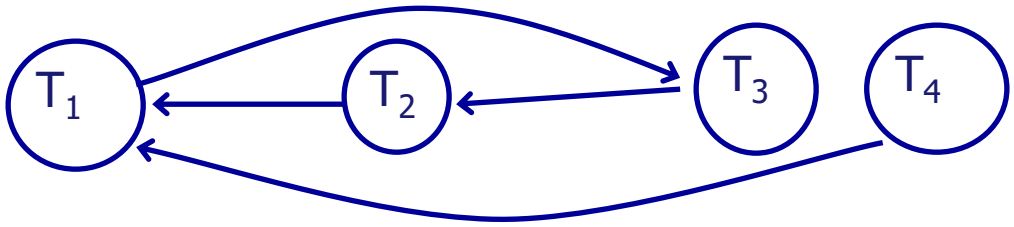
- Detect
  - Allow the deadlock state to occur and then attempt to restore the system: Select a transaction to rollback
  - Method: Wait-for graph
- Prevent
  - Manage transactions so that there is never a deadlock
  - Method
    - Rresource ordering (Sắp thứ tự tài nguyên)
    - Timeout
    - Wait-die
    - Wound-wait

Waiting graph

- The graph consists of vertices of transactions holding locks or waiting for locks. The arc goes from vertex T to U when
  - U is holding a lock on data unit A
  - T is waiting for a lock on A
  - T cannot lock data unit A if U does not release the lock
- If the wait graph has no cycles, transactions can complete
- On the contrary, no transaction in the cycle can continue to execute deadlock→ deadlock

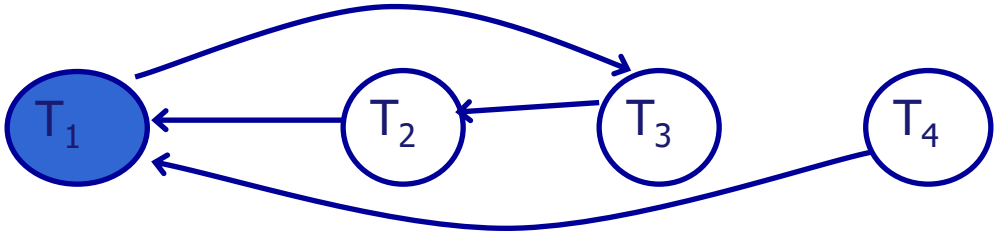
Waiting graph - Example

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D);
5		L(A)		R(D)
6		↓	L(C)	
7		Chờ	↓	L(A)
8	L(B)		Chờ	↓
	↓ Chờ			Chờ



Waiting graph - Example

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D);
5		L(A)		R(D)
6			L(C)	
7				L(A)
8	L(B)			



Sort resources in order

- Imposes a certain order on data units
- If transactions lock data units in this order
- Then no deadlock occurs while waiting

Timeout

- Limit transactions to only be performed within a certain period of time
- If the transaction exceeds this time
- Then the transaction must be rolled back

Review

- Timeout
  - Simple
  - It is difficult to choose the appropriate timeout period
  - There is a phenomenon of starvation
  - Transaction repeats the process: start, deadlock, rollback
- Resource ordering
  - Unrealistic
  - Waiting a lot $\rightarrow$ deadlock potential

Transaction with T-SQL

- BEGIN TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION
- SAVE TRANSACTION

Discussion