



事务 Transactions

李文根/Wengen Li

Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室 (ADMIS)

<https://admis.tongji.edu.cn/main.htm>



- **Part 0: Overview**
 - Ch1: Introduction
- **Part 1 Relational Databases**
 - Ch2: Relational model
 - Ch3: Introduction to SQL
 - Ch4: Intermediate SQL
 - Ch5: Advanced SQL
- **Part 2 Database Design**
 - Ch6: Database design based on E-R model
 - Ch7: Relational database design
- **Part 3 Application Design & Development**
 - Ch8: Complex data types
 - Ch9: Application development
- **Part 4 Big data analytics**
 - Ch10: Big data
 - Ch11: Data analytics
- **Part 5 Data Storage & Indexing**
 - Ch12: Physical storage system
 - Ch13: Data storage structure
 - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
 - Ch15: Query processing
 - Ch16: Query optimization
- **Part 7 Transaction Management**
 - **Ch17: Transactions**
 - Ch18: Concurrency control
 - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
 - Ch20: Database system architecture
 - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Part 9**
 - DB Platform: **OceanBase**, MongoDB, Neo4J

- **事务的概念**
- **事务的调度**
- **可串行化调度**
- **可恢复调度**
- **可串行性检测**

► 事务 (Transaction)



- A transaction is a unit of program execution consisting of multiple operations
 - During transaction execution, the database may be **inconsistent**
 - After the transaction is committed, the database must be **consistent**
- **Two main issues**
 - Concurrent execution of multiple transactions -> **Concurrency control** (Ch18)
 - Recover from hardware failures and system crashes -> **Recovery** (Ch19)

- **Atomicity (原子性)**
 - Either all operations of the transaction are properly reflected in the database or none are
- **Consistency (一致性)**
 - Execution of a transaction in isolation preserves the consistency of the database
- **Isolation (隔离性)**
 - Although multiple transactions may execute concurrently, each transaction must be unaware of other transactions
- **Durability (持久性)**
 - After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

► 例：Fund Transfer

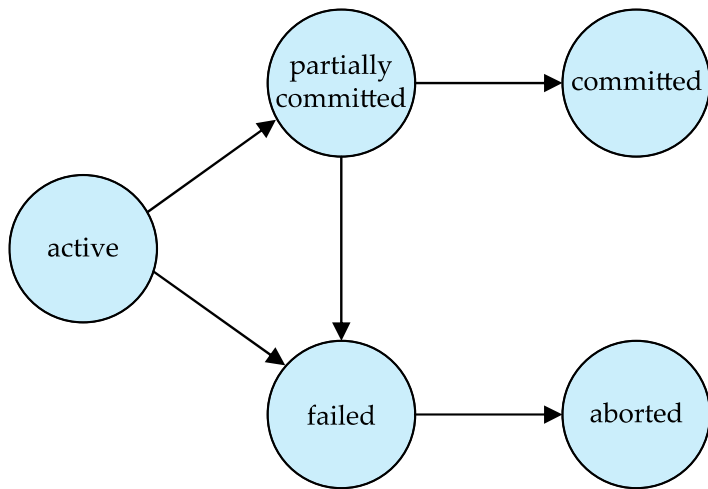


- A transaction to transfer \$50 from account A to account B:

1. read(A)
2. $A := A - 50$
3. write(A)
4. read(B)
5. $B := B + 50$
6. write(B)

- **Consistency requirement**
 - The sum of A and B is unchanged after the execution of the transaction
- **Atomicity requirement**
 - If the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database. Otherwise, an inconsistency will occur
- **Isolation requirement**
 - If between steps 3 and 6, another transaction accesses the partially updated database, it will see an inconsistent database
 - Can be ensured trivially by running transactions serially, i.e., one after the other. However, executing multiple transactions concurrently has significant benefits
- **Durability requirement**
 - Once the transaction has completed, the updates to the database by the transaction must persist despite failures

- **Active(活跃)**
 - The initial state. The transaction stays in this state while it is executing
- **Partially committed(部分提交)**
 - After the final statement has been executed
- **Failed(失败)**
 - After discovering that normal execution can no longer proceed
- **Aborted(中止)**
 - The transaction has been rolled back and the database restores to its state prior to the start of the transaction
 - Restart the transaction – only if no internal logical error happens in the transaction
 - Kill the transaction – problems arising with the transaction, input data, no desirable data found in the database
- **Committed(提交)**
 - After successful completion



- 事务的概念
- **事务的调度**
- 可串行化调度
- 可恢复调度
- 可串行性检测

- **Concurrent execution**

- Multiple transactions run concurrently in the system
- Advantages
 - Increase the utilization of processors and disks
 - Reduce the average response time

- **Concurrency control**

- Mechanisms to achieve isolation, i.e., to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

- **Schedule**

- Sequences that indicate the chronological order in which instructions of concurrent transactions are executed
- A schedule for a set of transactions must consist of all the instructions of those transactions
- Must preserve the order in which the instructions appear in each individual transaction

- **Example**

- Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B
- Schedule 1 is a serial schedule (串行调度), in which T_1 is followed by T_2

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule 1

- Another serial schedule where T_2 is followed by T_1

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule 1



T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule 2

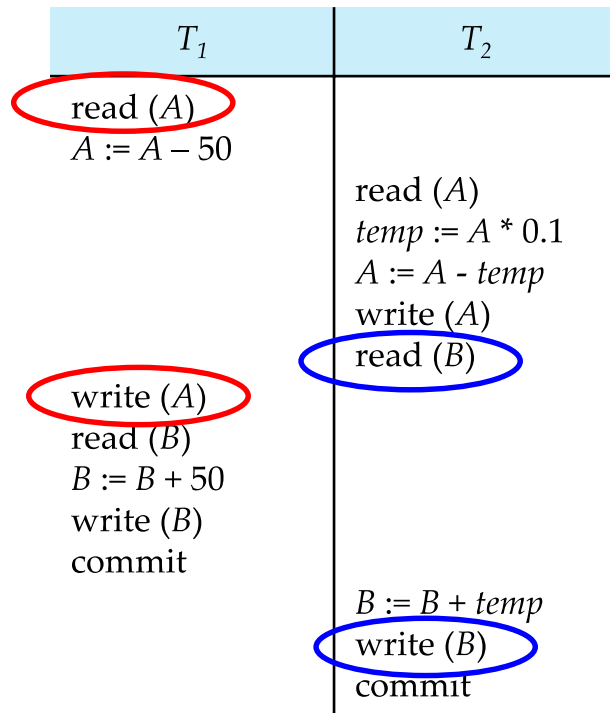
• Non-serial schedule

- Let T_1 and T_2 be the transactions defined previously
- Schedule 3 is not a serial schedule, but equivalent to Schedule 1
 - $A' = (A - 50) * 0.9$
 - $B' = B + 50 + (A - 50) * 0.1$
 - $A' + B' = A + B$

T_1	T_2
read (A) $A := A - 50$ write (A)	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	
	read (B) $B := B + temp$ write (B) commit

Schedule 3

- $A' = A - 50$
- $B' = B + A * 0.1$
- $A' + B' = A + B - 50 + A * 0.1 \neq A + B$



Schedule 4

- 事务的概念
- 事务的调度
- **可串行化调度**
- 可恢复调度
- 可串行性检测

► 可串行化 (Serializability)



- **Assumption**
 - Each transaction preserves database consistency, thus serial execution of a set of transactions preserves database consistency
- **Serializability**
 - A schedule is serializable if it is equivalent to a serial schedule
 - **Conflict serializability (冲突可串行性)**
 - **View serializability (视图可串行性)**
- **Note**
 - We ignore operations other than **read** and **write** instructions

- **Conflict**

- Given instructions I_i and I_j of transactions T_i and T_j respectively, conflict occurs iff there exists some item Q accessed by both I_i and I_j , and at least one of these instructions wrote Q
- **Four cases**
 - $I_i = \text{read}(Q), I_j = \text{read}(Q)$. (**no conflict**)
 - $I_i = \text{read}(Q), I_j = \text{write}(Q)$. (**conflict**)
 - $I_i = \text{write}(Q), I_j = \text{read}(Q)$. (**conflict**)
 - $I_i = \text{write}(Q), I_j = \text{write}(Q)$. (**conflict**)
- Intuitively, a conflict between I_i and I_j forces a (logical) **temporal order** between them
- If I_i and I_j are consecutive in a schedule and do not conflict, their results would remain the same even if they are interchanged in the schedule

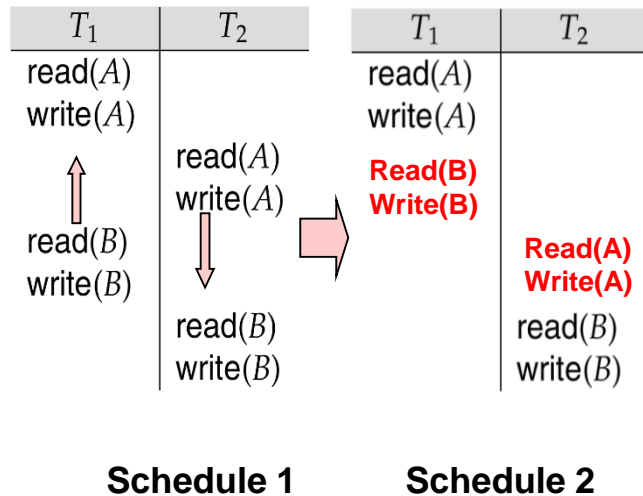
- **Conflict equivalent (冲突等价)**
 - If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**
 - A schedule S is **conflict serializable (冲突可串行化)** if it is **conflict equivalent** to a serial schedule (串行调度)
- Example of a schedule that is not conflict serializable
 - Cannot swap instructions in the following schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$, or the serial schedule $\langle T_4, T_3 \rangle$.

T_3	T_4
read (Q)	write (Q)
write (Q)	

► 冲突可串行化 (续)



- Schedule 1 can be transformed into Schedule 2, a serial schedule where T_2 follows T_1 , by a series of swaps of non-conflicting instructions
- Therefore, Schedule 1 is conflict serializable



► 冲突可串行化 (续)



• 例

Sc1=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)

- Swap w2(A) and r1(B), then we have

r1(A)w1(A)r2(A)r1(B)w1(B)w2(A)r2(B)w2(B)

- Swap r2(A) and r1(B), then:

Sc2=r1(A)w1(A)r1(B)w1(B)r2(A)w2(A)r2(B)w2(B)

- Sc2 is **equivalent to a serializable schedule** T_1, T_2
- Then Sc1 is conflict serializable

► 冲突可串行化 (续)



- A conflict serializable schedule is a serializable schedule, but a serializable schedule is not always conflict serializable.
- E.g., three transactions: $T1=W1(Y)W1(X)$, $T2=W2(Y)W2(X)$, $T3=W3(X)$
 - $S1=W1(Y)W1(X)W2(Y)W2(X)W3(X)$ is serializable
 - $S2=W1(Y)W2(Y)W2(X)W1(X)W3(X)$ is not conflict equivalent to $S1$, and not conflict serializable
 - $S2$ is serializable, and its result is equivalent to $S1$

► 视图可串行化 (View Serializability)




- **S and S' are view equivalent if the following three conditions are met:**
 - For each data item Q , if transaction T_i reads the initial value of Q in schedule S , then transaction T_i in schedule S' should also read the initial value of Q
 - For each data item Q , if transaction T_i executes $\text{read}(Q)$ in schedule S , and that value was produced by transaction T_j (if any), then transaction T_i in schedule S' should also read the value of Q that was produced by transaction T_j
 - For each data item Q , the transaction (if any) that performs the final $\text{write}(Q)$ operation in schedule S must perform the final $\text{write}(Q)$ operation in schedule S'
- View equivalence is also based purely on reads and writes

► 视图可串行化 (续)



- If a schedule S is view serializable, it is view equivalent to a serial schedule
- Every conflict serializable schedule is also view serializable
- E.g., the following schedule, equivalent to T_3, T_4, T_6 , is view-serializable but not conflict serializable

T_3	T_4	T_6		T_3	T_4	T_6
read(Q)				read(Q)		
write(Q)	write(Q)			Write(Q)		
		write(Q)			Write(Q)	write(Q)

► 其他可串行性情形



- The following schedule produces the same outcome as the serial schedule $\langle T_1, T_5 \rangle$, yet it is not conflict equivalent or view equivalent

T_1	T_5
read (A) $A := A - 50$ write (A)	
	read (B) $B := B - 10$ write (B)
read (B) $B := B + 50$ write (B)	
	read (A) $A := A + 10$ write (A)

- Determining such equivalence requires analysis of operations other than read and write

- 事务的概念
- 事务的调度
- 可串行化调度
- **可恢复调度**
- 可串行性检测

► 可恢复性 (Recoverability)



- **Recoverable schedule (可恢复调度)**
 - If a transaction T_j reads a data item previously written by a transaction T_i , the commit operation of T_i should appear before the commit operation of T_j
 - The following schedule is not recoverable if T_9 commits immediately after the read

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- **Cascading rollback(级联回滚)**

- A single transaction failure leads to a series of transaction rollbacks
- Consider the following schedule where none of the transactions has yet committed

T_{10}	T_{11}	T_{12}
read (A) read (B) write (A)		
	read (A) write (A)	
abort		read (A)

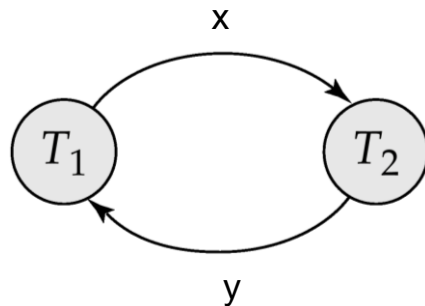
- If T_{10} fails, T_{11} and T_{12} must also be rolled back
- **Could lead to the undoing of a significant amount of work**

- **Cascadeless schedules (无级联回滚调度)**
 - For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i should appear before the read operation of T_j
 - Cascading rollbacks cannot occur and every cascadeless schedule is also recoverable
 - It is desirable to restrict the schedules to those that are cascadeless

- DML must include a construct for specifying the set of actions that comprise a transaction
- In SQL, a transaction begins implicitly
- A transaction in SQL ends by:
 - Commit work: commits current transaction and starts a new one
 - Rollback work: causes current transaction to abort
- Levels of isolation specified by SQL-92
 - Serializable – **default**: 保证可串行化调度
 - Repeatable read: 只允许读取已提交数据, 两次读取之间数据不能更新
 - Read committed: 只允许读取已提交数据, 不要求可重复读
 - Read uncommitted: 允许读取未提交数据

- 事务的概念
- 事务的调度
- 可串行化调度
- 可恢复调度
- **可串行性检测**

- Given a set of transactions T_1, T_2, \dots, T_n
- **Precedence graph (优先图)**
 - A direct graph where the vertices are the transactions
 - Draw an arc from T_i to T_j if the two transactions conflict, and T_i accessed the data item on which the conflict arose earlier
 - We label the arc by the data item that was accessed
- **Example**



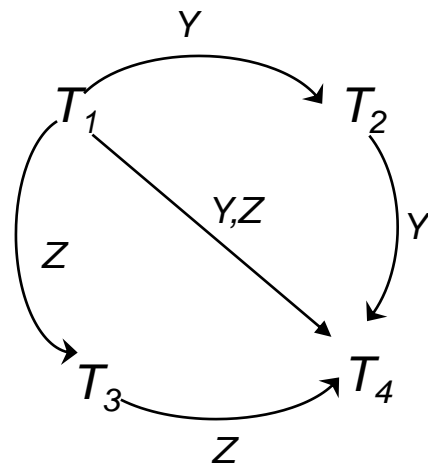
T1 write(x) before T2 read(x)
T1 write(x) before T2 write(x)
T1 read(x) before T2 write(x)

T2 write(y) before T1 read(y)
T2 write(y) before T1 write(y)
T2 read(y) before T1 write(y)

调度与优先图



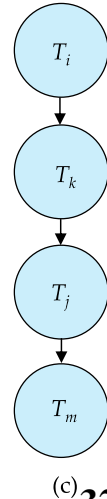
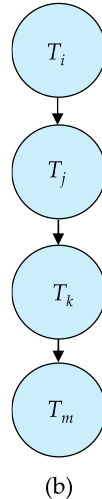
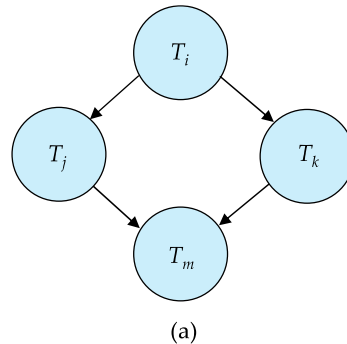
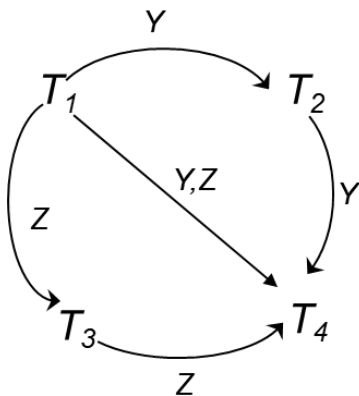
T_1	T_2	T_3	T_4	T_5
read(Y) read(Z)	read(X)			read(V) read(W) read(W)
	read(Y) write(Y)	write(Z)		
read(U)			read(Y) write(Y) read(Z) write(Z)	
read(U) write(U)				



► 可串行性检测 (续)



- A schedule is conflict serializable if and only if its precedence graph is acyclic (无环)
- If precedence graph is acyclic, the serializability order can be obtained by a topological sorting of the graph
 - For example, a serializability order for the Schedule on the previous page would be $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
 - Any others?



- **Concurrency control**
 - Develop concurrency control protocols to assure serializability
 - Not examine the precedence graph as it is being created
- Testing for serializability helps understand why a concurrency control protocol is correct