



# 第十七讲交易 (第17章)

关继宏教授

电子邮件: [jhguan@tongji.edu.cn](mailto:jhguan@tongji.edu.cn)

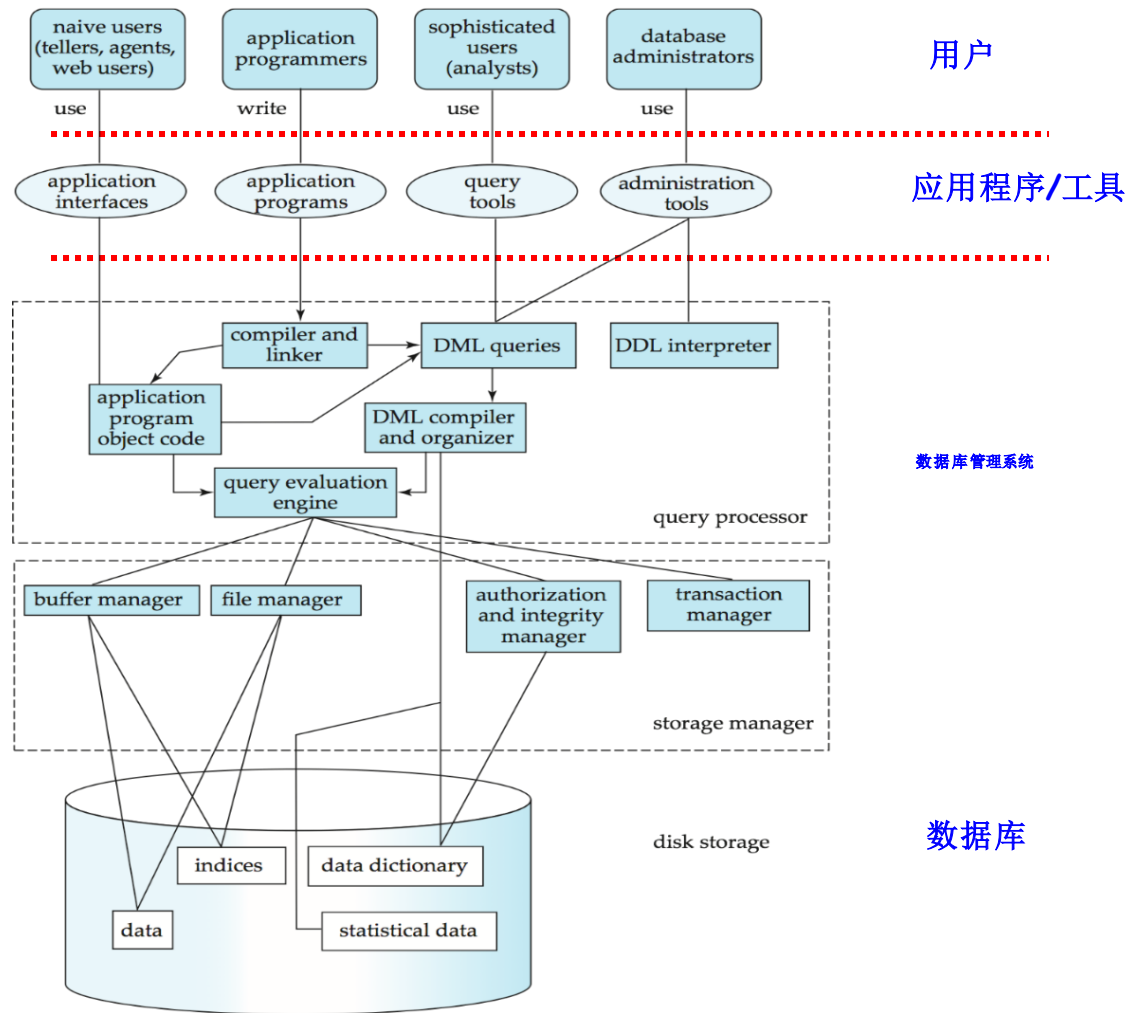
计算机科学与技术系

同济大学

# 课程大纲

- **第0部分:概述**
  - Ch1:介绍
- **Part 1 关系数据库**
  - Ch2:关系模型(数据模型, 关系代数)
  - Ch3&4: SQL(结构化查询语言)
  - Ch5:高级SQL
- **第二部分数据库设计**
  - Ch6:基于E-R模型的数据库设计
  - Ch7:关系型数据库设计
- **第三部分:应用程序设计与开发**
  - Ch8:复杂数据类型
  - Ch9:应用开发
- **Part 4 大数据分析**
  - Ch10:大数据
  - Ch11:数据分析
- **第5部分:数据存储和索引**
  - Ch12:物理存储系统
  - Ch13:数据存储结构
  - Ch14:索引
- **第6部分:查询处理与优化**
  - Ch15:查询处理
  - Ch16:查询优化
- **第7部分:事务管理**
  - **Ch17:交易**
  - Ch18:并发控制
  - Ch19:恢复系统
- **第8部分:并行和分布式数据库**
  - Ch20:数据库系统架构
  - Ch21-23:并行和分布式存储, 查询处理和事务处理
- **第9部分**
  - **DB平台:OceanBase、MongoDB、Neo4J**

# 数据库 系统 结构



## <s:1> 交易概念

- 日程安排
- 可串行化的调度
- 可恢复的时间表
- 可序列化性测试

# 事务的概念

- **事务是由多个操作组成的程序执行单元**
  - 在事务执行过程中，数据库可能不一致
  - 事务提交后，数据库必须是一致的
- **主要有两个问题**
  - **多个事务的并发执行**
  - **硬件故障和系统崩溃**

# ACID属性

- **原子性(Atomicity)**
  - 事务的所有操作要么正确地反映在数据库中，要么没有
- **一致性(Consistency)**
  - 隔离地执行事务可以保持数据库的一致性
- **隔离性**
  - 虽然多个事务可以并发执行，但每个事务必须不知道其他事务
- **持久性(Durability)**
  - 事务成功完成后，即使系统出现故障，它对数据库所做的更改也会持续存在

# 资金转移示例

- 从账户A向账户B转账50美元的交易:

1. 阅读(一)
2.  $A := A - 50$
3. 写(一个)
4. 读(B)
5.  $B := B + 50$
6. 写(B)

- **一致性要求**

- A和B的总和不会因为交易的执行而改变

- **原子性要求**

- 如果事务在步骤3之后和步骤6之前失败，系统应确保其更新没有反映在数据库中。否则，就会出现不一致

# 资金转移示例(续)

- **持久性要求**

- 一旦用户被通知事务已经完成，事务对数据库的更新必须持续，尽管失败

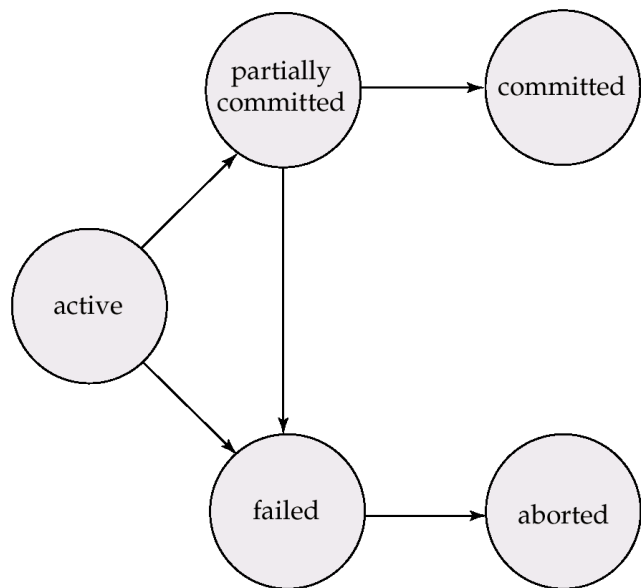
- **隔离的要求**

- 如果在步骤3和6之间，允许另一个事务访问部分更新的数据库，它将看到一个不一致的数据库
- 可以通过连续运行事务，即一个接一个地运行事务来保证。然而，并发地执行多个事务有显著的好处



# 事务状态

- **活动()**
  - 初始状态。事务在执行时保持这种状态
- **部分提交**
  - 在最后一语句被执行之后
- **Failed()**
  - 发现正常执行无法再继续
- **Aborted(中文)**
  - 事务回滚后，数据库恢复到事务开始前的状态
  - 重新启动事务——只有在事务中没有发生内部逻辑错误的情况下
  - 杀死事务——事务出现问题，输入数据，数据库中没有找到需要的数据
- **已提交(Committed)**
  - 成功完成后



- 事务的概念

- ☞ 时间表

- 可串行化的调度
- 可恢复的时间表
- 可序列化性测试

# 同时执行

- **并发执行**

- 允许多个事务在系统中并发运行
- 优势
  - 提高处理器和磁盘利用率
  - 减少平均响应时间

- **并发控制**

- 实现隔离的机制，即控制并发事务之间的交互，以防止它们破坏数据库的一致性

# 日程安排

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

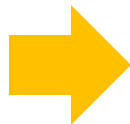
表1

# 附表示例(续)

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

表1



$T_1$	$T_2$
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)

表2

# 附表示例(续)

- **Non-serial schedule**

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

$T_1$	$T_2$
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)	
	read(B) $B := B + temp$ write(B)

表3

# 附表示例(续)

- 下面的并发调度不保留 $A + B$ 的和的值。

- 一个' = 50

- $B' = B + 0.1 *$

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

$T_1$	$T_2$
<u>read(A)</u> $A := A - 50$	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ <u><math>\text{read}(B)</math></u>
<u><math>\text{write}(A)</math></u> $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$	$B := B + \text{temp}$ <u><math>\text{write}(B)</math></u>

表4

- 事务的概念
- 日程安排

## <s:1> Serializable时间表

- 可恢复的时间表
- 序列化性测试



# 序列化性(英文)

- **假设**

- 每个事务保持数据库一致性，因此一组事务的串行执行保持数据库一致性

- **序列化性**

- 如果一个调度相当于一个串行调度，那么它是可串行化的

- **冲突序列化性**

- **查看序列化性()**

- **请注意**

- 我们忽略读写指令以外的操作。我们简化的时间表只包括读写指令

# 冲突序列化性

- **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**

- $I_i = \text{read}(Q)$ ,  $I_j = \text{read}(Q)$ .  $I_i$  and  $I_j$  **(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 they **do not conflict**, their results would remain the same even if they had been **interchanged in the schedule**

# 冲突序列化性(续)

- **Conflict equivalent**
  - 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**
  - We say that a schedule  $S$  is **conflict serializable** if it is **conflict equivalent to a serial schedule**
- **Example** of a schedule that is not conflict serializable
  - We are **unable to 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)

# 冲突序列化性(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**

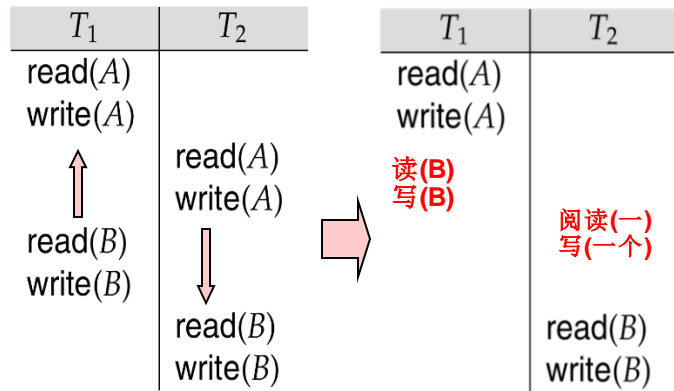



表1

表2

# 冲突序列化性(续)

- 例子

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



- 交换 $w2(A)$ 和 $r1(B)w1(B)$ , 然后我们有

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



- 将 $r2(A)$ 和 $r1(B)w1(B)$ 互换, 则:

星际2 =  $r1(A)r1(B)w1(B)w2(A)w2r2(A)r2(B)w2(B)$



- $Sc2$ 等价于一个可串行化的调度 $t1, t2$

- 那么 $Sc1$ 是可冲突序列化的

# 冲突序列化性(续)

- 可冲突序列化的时间表是可序列化的时间表，但可序列化的时间表并不总是可冲突序列化的。

- 例如，三个事务

$T1 = w1(y) w1(x), t2 = w2(y) w2(x), t3 = w3(x)$

- $L1 = \cancel{w1(y)} \cancel{w1(x)} \cancel{w2(y)} \cancel{w2(x)} w3(x)$ 是可序列化的
- $L2 = \cancel{w1(y)} \cancel{w2(y)} \cancel{w2(x)} \cancel{w1(x)} w3(x)$ 不等同于 $L1$ ，且不可冲突序列化。
- $L2$ 是可序列化的，调度的结果等效于 $L1$ (最终写 $X$ 来自 $T3$ ，最终写 $Y$ 来自 $T2$ )


# 查看序列化性

- $S$  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  $T_i$  must, in schedule  $S'$ , 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  $T_i$  must in schedule  $S'$  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 write( $Q$ ) operation in schedule  $S$  must perform the final write( $Q$ ) operation in schedule  $S'$ .
- As can be seen, view equivalence is also based purely on reads and writes alone.

# 查看序列化性(续)

- 如果一个调度S是可视图序列化的，它就相当于一个串行调度。
- **每个可冲突序列化的调度也是可视图序列化的。**
- **可视图序列化但不可冲突序列化的调度。相当于t3、t4、t6**

- **每**

$T_3$	$T_4$	$T_6$		$T_3$	$T_4$	$T_6$	
read(Q)				read(Q)			
write(Q)	write(Q)						
		write(Q)					
			<b>se</b>		<b>写(Q)</b>		
						write(Q)	<b>读而不读</b>



# 序列化性的其他概念

- 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.

# 大纲

- 事务的概念
- 日程安排
- 可串行化的调度

## 可恢复的时间表

- 序列化性测试

# 可恢复性(英文)

- Recoverable schedule (可恢复调度)

- If a transaction  $T_j$  reads a data items previously written by a transaction  $T_i$ , the commit operation of  $T_i$  appears 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)
read(B)	

# 可恢复性(续)。

- 级联回滚(后排)

- 单个事务失败会导致一系列事务回滚
- 考虑下面的时间表，其中还没有任何事务提交

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

- 如果t10失败，t11和t12也必须回滚

- 会导致大量工作的撤销吗

# 可恢复性(续)。

- 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  $T_i$  appears 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

# SQL中的事务定义

- DML必须包含一个结构，用于指定组成事务的一组操作
- 在SQL中，事务隐式地开始
- SQL中的事务以以下方式结束：
  - **提交工作:提交当前事务并开始一个新的事务。**
  - **回滚工作:导致当前事务中止。**
- SQL-92指定的隔离级别
  - **Serializable -默认:**
  - **可重复读取:**
  - **已提交阅读:**
  - **未提交阅读:**

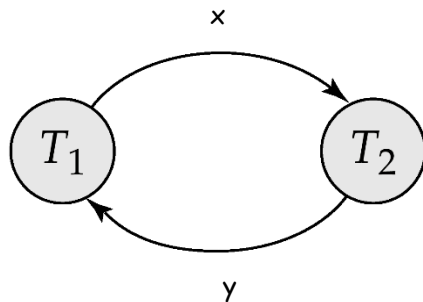
# 大纲

- 事务的概念
- 日程安排
- 可串行化的调度
- 可恢复的时间表

可序列化性测试

# 序列化性测试

- 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**
  - We 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**



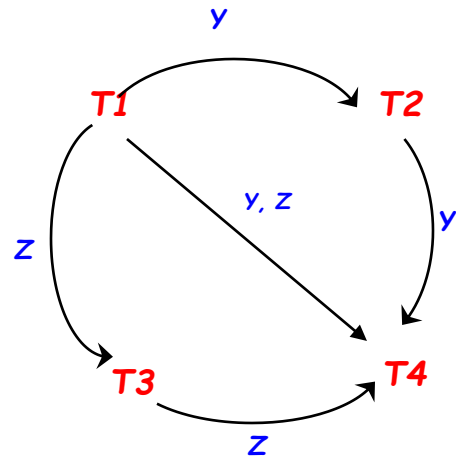
**T1 写(x)在T2读(x)之前**  
**T1 写(x)在T2写(x)之前**  
**T1在T2写(x)之前读(x)**

**T1 读(y)前T2写(y)**  
**T2写(y)在T1写(y)之前**  
**T2在T1写(y)之前读(y)**



## 附表A示例

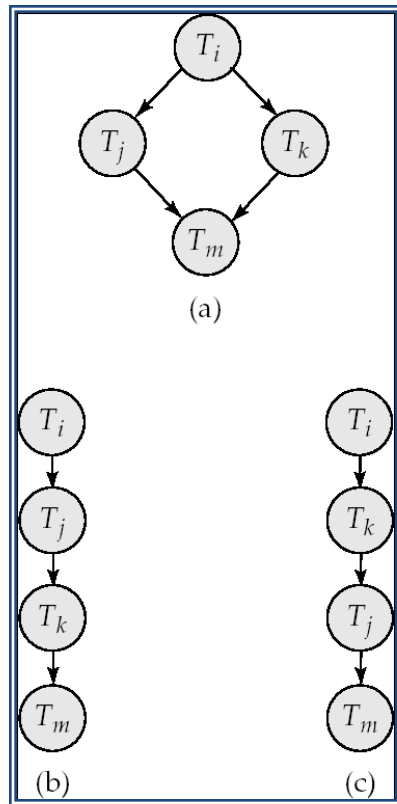
T1	T2	T3	4	5	阅读(X)	阅读(Y)	阅读(Z)
					读(W)	读(V)	
读(W)	阅读(Y)	写(Y)	写(Z)	阅读(U)	读(Y)	写(Y)	读(Z) 写(Z)
读(U)	写(U)						


$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$$
$$T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4 \rightarrow T_5$$

.....???

# 测试冲突序列化性

- 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 **Schedule A** would be  $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$ 
    - Any others?



# 并发控制与序列化性测试

- 在调度执行之后测试它的序列化性已经太晚了
- **目标-开发并发控制协议，确保可序列化性。**
  - 他们通常不会在创建优先级图时检查它
  - 相反，协议将强加一个规则来避免不可串行化的调度
- 可序列化性测试有助于理解并发控制协议正确的原因

# 第17讲结束