

数据库系统原理 (CS307)

第 15 讲:交易

Zhong-Qiu Wang

计算机科学与工程系
南方科技大学

· 大部分内容来自 Stéphane Faroult 和《数据库系统概念（第 7 版）》作者制作的幻灯片。 · 他们的原始幻灯片已修改以适应南方科技大学 CS307 的时间表。 · 幻灯片主要基于马宇欣博士提供的幻灯片

现实生活中的交易

- “用商品换取金钱” 一系列步骤 全有或全无



Flickr:BracketingLife (克拉伦斯)



计算机交易

- 事务是程序执行的一个单位,它访问并可能更新各种数据项
- 数据库中的经典示例:汇款

例如从账户A向账户B转账50元人民币:

```
1. read(A) // 从磁盘读取到主存 2. A := A - 50 3. write(A) // 从主存写入到磁盘 4. read(B) //  
从磁盘读取到主存 5. B := B +  
50 6. write(B) //写入磁盘
```

PostgreSQL 事务示例

·开始、提交、回滚



```
begin;  -- Start a transaction
```

```
update people_1 set num_movies = 50000 where peopleid = 1;
```

```
select * from people_1 where peopleid = 1;
```

```
delete from people_1 where peopleid > 100 and peopleid < 200;
```

```
commit;  -- start executing all the queries above
```

```
-- or "rollback;", which means to revoke the operationso of all the queries
```

计算机交易

- 事务是程序执行的一个单位,它访问并可能更新各种数据项
- 数据库中的经典示例:汇款

例如从账户A向账户B转账50元人民币的交易:

```
1. read(A)
2. A := A - 50
3. 写入 (A)
4. 读取 (B)
5. B := B + 50
6. 写入(B)
```

- 需要处理两个主要问题:

- 各种故障,如硬件故障和系统崩溃
- 多个事务的并发执行

交易要求

- 原子性要求

- 如果交易在第 3 步之后和第 6 步之前失败,资金将“丢失”,导致数据库状态不一致

- 故障可能是由于软件或硬件

- 系统应确保部分执行的事务的更新不会反映在数据库中

例如从账户A向账户B转账50元人民币:

- 1.读 (A)
2. $A := A - 50$
- 3.写入 (A)
4. 读取 (B)
5. $B := B + 50$
- 6.写入(B)

交易要求

- 耐久性要求

- 一旦通知用户交易已完成（即已转账 50 元），即使出现软件或硬件故障，该交易对数据库的更新也必须保留。
-

交易要求

·一致性要求

- 明确指定的完整性约束,如主键和外键
- 隐式完整性约束
- 依赖于应用程序的一致性约束,这些约束过于复杂,无法使用 SQL 结构来表达数据完整性
- 例如,所有帐户余额总和

– 示例中:交易执行后,A 和 B 的总和不会发生变化

例如从账户A向账户B转账50元人民币:

- 1.读 (A)
2. $A := A - 50$
- 3.写入 (A)
- 4.读取 (B)
5. $B := B + 50$
- 6.写入(B)



交易要求

- 隔离要求

· 如果在步骤 3 和步骤 6 之间, **另一个事务T2**被允许访问部分更新的数据库, 它将看到 不一致的数据库

- A + B 的总和将小于应有的值

T1	T2
1.读 (A) 2. A := A - 50 3. 写入 (A)	
	读取 (A) ,读取 (B) ,打印 (A+B)
4.阅读 (B) 5. B := B + 50 6.写入(B)	

· 可以通过串行运行事务 (即一个接一个地运行事务)轻松地确保隔离性。
其他

- 然而,同时执行多个事务具有显著的好处

酸性质

- 事务是程序执行的一个单位,它访问并可能更新各种数据项
- 为了保持数据的完整性,数据库系统必须确保:

原子性:事务的所有操作要么都正确反映在数据库中,要么都不反映

一致性:隔离执行事务可保持数据库的一致性。

隔离性:尽管多个事务可以并发执行,但每个事务必须不知道其他并发执行的事务。中间事务结果必须对其他并发执行的事务隐藏。

持久性: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仍会保留下来。

- 也就是说,对于每一对事务 T_i 和 T_j ,在 T_i 看来,要么 T_j 在 T_i 开始之前完成执行,要么 T_j 在 T_i 完成之后开始执行。

交易状态

- 积极的

- 初始状态;交易在进行期间保持此状态。
执行

- 部分承诺

- 执行完最后一条语句后

- 失败的

- 发现无法继续正常执行后

- 中止 – 事务回滚后,数据库恢复到事务开始前的状态。中止后有两个选项:

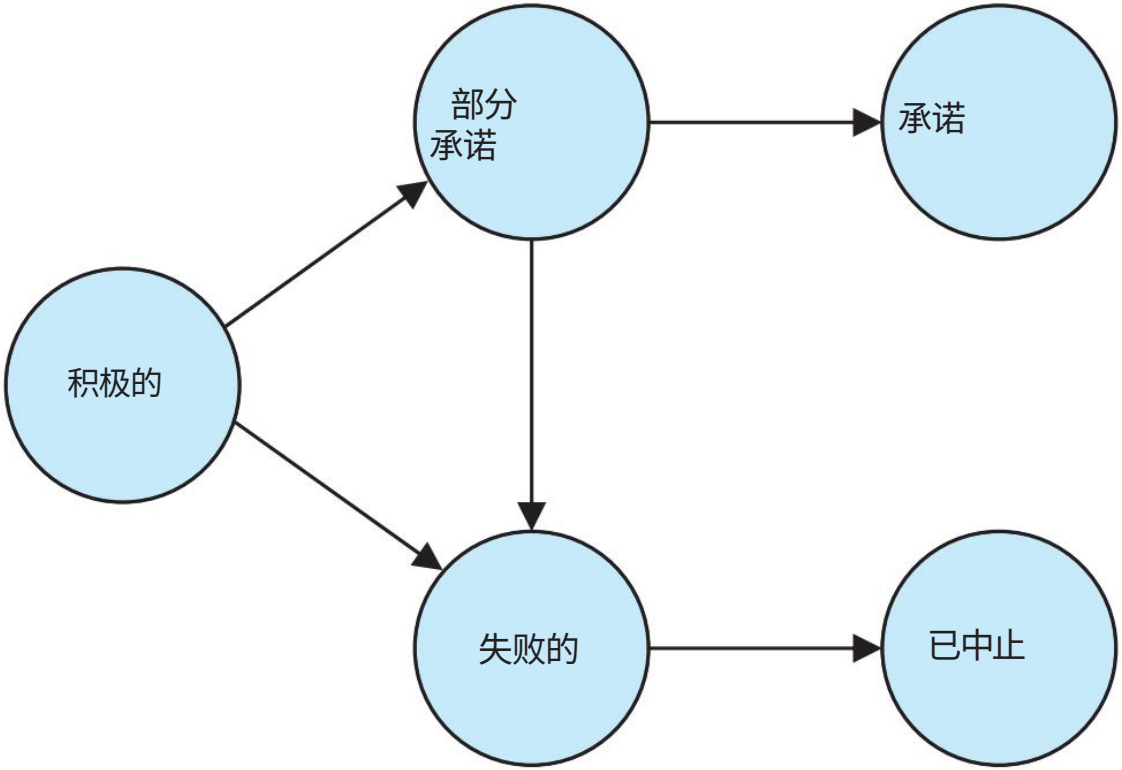
- 重新开始交易

- 仅当不存在内部逻辑错误时才可执行

- 终止交易

- 坚定的

- 成功完成后



并发执行

- 通过串行（一个接一个）运行事务可以确保隔离

- 允许多个事务在系统中同时运行

优点是：

- 提高处理器和磁盘利用率,从而提高交易吞吐量

- 例如,一个事务可能正在使用 CPU,而另一个事务正在读取或写入到磁盘

- 减少交易的平均响应时间

- 短期交易无需等待长期交易

- 并发控制方案 实现隔离的机制

- 即控制并发事务之间的交互,以便

- 防止它们破坏数据库的一致性

时间表

·**时间表** 一系列指令,指定并发事务指令的执行**时间**顺序

- 一组交易的时间表必须包含这些交易的所有指令
- 必须保留每笔交易中指令出现的顺序
- 成功完成执行的事务将以提交指令作为最后一条语句
- 默认情况下,事务假定执行提交指令作为其最后一步
- 未能成功完成执行的事务将中止
指令作为最后一条语句

附表1

- 假设T1从 A 向 B 转账 50 元人民币,并且 T2将余额的 10% 从 A 转移到乙
- 串行调度,其中T1之后是T2
- :

T1	T2
阅读 (A) 一个 := 一个 - 50 写 (A) 阅读 (B) B := B + 50 写 (B) 犯罪	阅读 (A) 温度 := A * 0.1 A := 临时 - 写 (A) 阅读 (B) B := B + 温度 写 (B) 犯罪

附表 2

·遵循T2的串行计划
由T1

T1	T2
	读取 (A)温度 := A * 0.1 A := A 临时写入 (A) 读取 (B)
阅读 (A) A := A - 50 写 入 (A)读取 (B) B := B + 50 写 入 (B)提交	B := B + 临时写入 (B)提交

附表 3

- 假设T1和T2为之前定义的事务
- 当 DBMS 同时执行多个事务时,相应的调度不再是串行的
- 以下时间表不是连续的附表,但相当于附表1
- 在附表 1、2 和 3 中,A + B 之和被保留。

T1	T2
阅读 (A) 一个 := 一个 - 50 写 (A)	阅读 (A) 温度 := A * 0.1 A := 临时 - 写 (A)
阅读 (B) B := B + 50 写 (B) 犯罪	阅读 (B) B := B + 温度 写 (B) 犯罪

附表 3 与 1

- 假设T1和T2为之前定义的事务
- 当 DBMS 同时执行多个事务时,相应的调度不再是串行的
- 以下时间表不是连续的附表,但相当于附表1
 - 在附表 1、2 和 3 中,A + B 之和被保留。

T1	T2
阅读 (A) 一个 := 一个 - 50 写 (A)	阅读 (A) 温度 := A * 0.1 A := 临时 - 写 (A)
阅读 (B) B := B + 50 写 (B) 犯罪	阅读 (B) B := B + 温度 写 (B) 犯罪

附表 3

T1	T2
阅读 (A) 一个 := 一个 - 50 写 (A) 阅读 (B) B := B + 50 写 (B) 犯罪	阅读 (A) 温度 := A * 0.1 A := 临时 - 写 (A) 阅读 (B) B := B + 温度 写 (B) 犯罪

附表1

附表4

- 可以有多个执行序列,因为两个事务中的各种指令现在可以交错
- 无法准确预测在 CPU 切换到另一个事务之前将执行多少条事务指令
- 并非所有并发执行都会产生正确的状态
- 以下并发计划不保留 $(A + B)$ 的值
 - A 扣掉 50,但 B 增加 $A*0.1$

T1	T2
阅读 (A) 一个 := 一个 - 50	阅读 (A) 温度 := A * 0.1 A := 临时 - 写 (A) 阅读 (B)
写 (A) 阅读 (B) B := B + 50 写 (B) 犯罪	B := B + 温度 写 (B) 犯罪

可序列化

- 确保数据库一致性
- 确保执行的任何计划都与串行计划具有相同的效果

- 基本假设：
 - 每个事务都保持数据库一致性
 - 因此，一组事务的串行执行可以保持数据库的一致性
-

- 如果一个（可能并发的）调度与一个串行调度等价，那么它就是可串行化的
-

- 不同形式的进度等值产生了以下概念：

- 1. 冲突可序列化^{*}
- 2. 查看可序列化性

简化的交易视图

- 我们忽略除读写指令之外的操作
- 我们假设交易可以对数据进行任意计算
读取和写入之间的本地缓冲区。
- 我们的简化时间表仅包含读写指令。

相冲突的指令

·事务T和Tj的指令li和lj分别发生冲突当且仅当存在某个项目Q被li和lj访问,并且这些指令中至少有一条写入了Q

- 1. $li = \text{read}(Q), lj = \text{read}(Q)$. li和lj不冲突
- 2. $li = \text{read}(Q), lj = \text{write}(Q)$.它们冲突。
- 3. $li = \text{write}(Q), lj = \text{read}(Q)$.它们冲突
- 4. $li = \text{write}(Q), lj = \text{write}(Q)$.它们冲突
- *如果li和lj指的是不同的项目,则不存在冲突

·直观地看, li和lj之间的冲突迫使li和lj之间存在 (逻辑上的)时间顺序——
他们。

·如果li和lj在调度中是连续的并且它们不冲突,那么即使它们在调度中互换,它们的结果也会保持不变。

冲突可串行化

· 如果通过一系列非冲突指令的交换,调度S可以转换为调度S₂,则我们称S和S₂是冲突等价的

· 如果调度S是冲突等价的,则我们称其为冲突可序列化

按照连续时间表

冲突可串行化

- 时间表 3 可以转换为时间表 6,这是一个连续的时间表,其中 T2跟随T1
 - 通过一系列非冲突指令的交换
- 因此,附表3是冲突可序列化的。

对不同数据进行操作

- ...因此可以按时间顺序交换

T1	T2
阅读 (A) 写 (A)	阅读 (A) 写 (A)
阅读 (B) 写 (B)	
	阅读 (B) 写 (B)

附表 3

T_1	T_2
read(<i>A</i>) write(<i>A</i>)	read(<i>A</i>)
read(<i>B</i>)	
write(<i>B</i>)	write(<i>A</i>)
	read(<i>B</i>) write(<i>B</i>)

附表4

T1	T2
阅读 (A) 写 (A) 阅读 (B) 写 (B)	阅读 (A) 写 (A) 阅读 (B) 写 (B)

附表 6

冲突可串行化

·非冲突序列化的计划示例：

T3	T4
阅读（问）	写（问）
写（问）	

·我们无法交换上述调度中的指令以获得串行调度< T3, T4 >或串行调度< T4, T3 >。

测试可序列化性

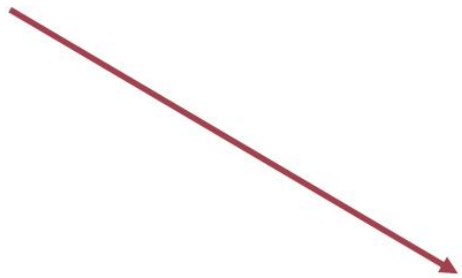
- 考虑一组事务 T_1 、 T_2 、...、 T_n 的某个调度

- 优先图** 以交易（交易名称）

为顶点的有向图 如果两个交易发生冲突,我们从 T_i 到 T_j 画一条弧

- 这意味着,在时间表 S 中, T_i 必须早于 T_j

- 我们可能会根据访问的项目来标记弧。

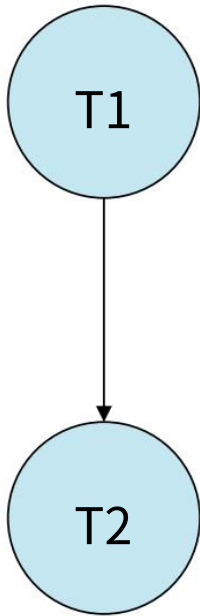


冲突 – 对于数据项 Q ,至少存在下列情况之一：

- T_i : 写入 (Q) \rightarrow T_j : 读取 (Q)
- T_i : 读取 (Q) \rightarrow T_j : 写入 (Q)
- T_i : 写入 (Q) \rightarrow T_j : 写入 (Q)

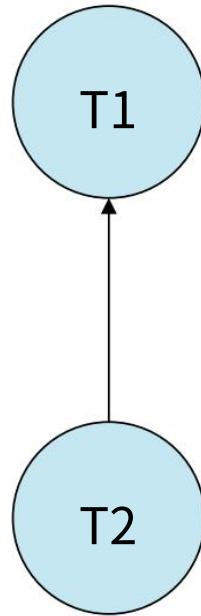
测试可序列化性

T1	T2
阅读 (A) A := A - 50 写 入 (A)读取 (B) B := B + 50 写 入 (B)提交	读取 (A)温度 := A * 0.1 A := A 临时写入 (A) 读取 (B) B := B + 临时写入 (B)提交



附表1

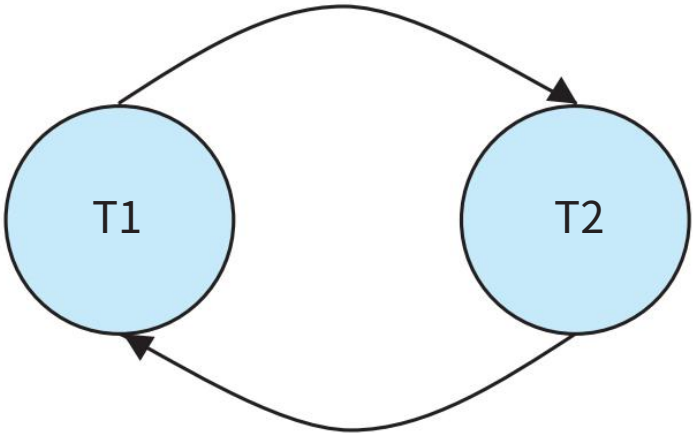
T1	T2
读取 (A) $A := A - 50$ 写 入 (A)读取 (B) $B := B + 50$ 写 入 (B)提交	读取 (A)温 * 度 := A 0.1 $A := A$ 临时写入 (A) 读取 (B) $B := B +$ 临时写入 (B)提交



附表 2

测试可序列化性

T1	T2
阅读 (A) 一个 := 一个 - 50	读取 (A) 温度 := A * 0.1 A := A 临时写入 (A) 读取 (B)
写 (A) 读 (B) B := B + 50 写 入 (B) 提交	 B := B + 临时写入 (B) 提交



T1 -> T2, T1 在 T2 写入 A 之前读取 A
T2 -> T1, T2 在 T1 写入 A 之前读取 B

附表4

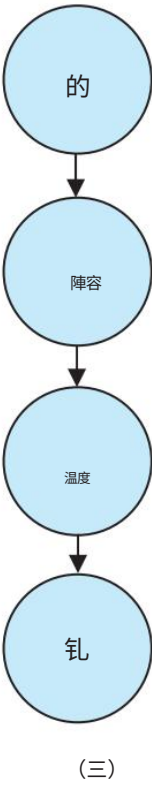
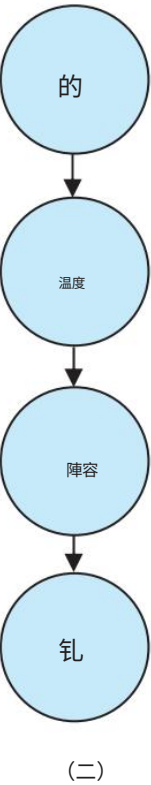
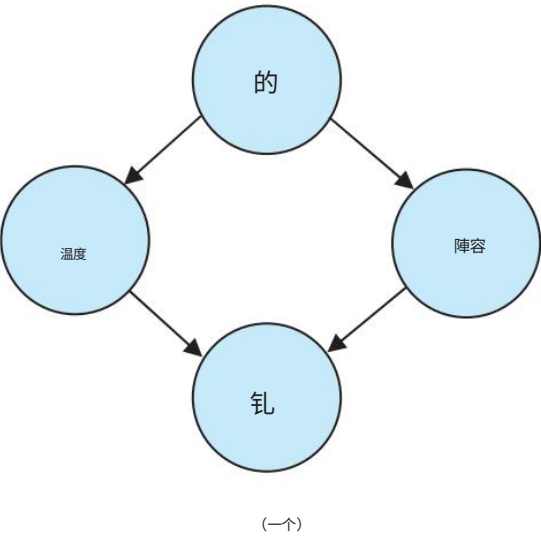
测试可序列化性

·调度是冲突可序列化的当且仅当其
优先图是非循环的

循环检测:存在循环检测算法,需要 n^2 时间,其中 n 是图中的顶点数。

·如果优先图是非循环的,则
序列化顺序可以通过图的拓扑排序来获得

·例如,(a) 的拓扑顺序可以是 (b) 和 (c)



可恢复时间表

- 需要解决事务失败对并发运行的影响
交易
- 可恢复调度 如果事务Tj读取了事务Ti先前写入的数据项,则Ti的提交操作会出现在Tj的提交操作之前。

·以下时间表不可恢复

Q8	T9
阅读 (A)	
写 (A)	
	阅读 (A)
	犯罪
阅读 (B)	

·如果T8中止， T9将会读取（并可能向用户显示)不一致的数据库状态。因此,数据库必须确保计划可恢复。

·为了可恢复， T9需要在T8之后提交

一致性较弱

- 如果每个事务单独执行时都能保持一致性,那么
 - 可串行化可以确保并发执行保持一致性
 - 但对于某些应用程序来说,并发性太低
- 一些应用程序愿意忍受较弱的一致性,允许不可序列化的调度

- 例如,一个只读事务想要获取所有
 账户

- 此类交易不需要相对于其他交易进行序列化
 - 目的:准确度和性能之间的权衡
-

一致性级别（在 SQL-92 中）

- 可序列化（最强）

- 默认

- 可重复读取 仅读取已提交的记录。 重复读取相同记录必须返回相同值。

- 但是,事务可能不是可序列化的 它可能会发现由交易但找不到其他。

- 读取已提交 只能读取已提交的记录。

- 连续读取记录可能会返回不同的（但已提交的）值。

- 读取未提交（最弱） 甚至可以读取未提交的记录。

一致性级别

- 较低程度的一致性可能有助于收集近似
有关数据库的信息
-

- 警告:**有些数据库系统不能通过以下方式确保可序列化的调度:
默认

·例如,Oracle (以及 9 之前的 PostgreSQL)默认支持称为**快照隔离**的一致性级别 (不是 SQL 标准的一部分)

- 警告2:**所有 SQL-92 一致性级别都推断禁止脏写

·**脏写**- 当一个事务覆盖先前由另一个事务写入的值时
另一笔仍在进行中的交易
