第 14 章:交易

Chapter 14: Transactions

数据库系统概念,第 7版。

交易理念 Transaction Concept

- 事务是一个程序执行单元,它访问并可能更新各种数据项
- 例如,将 50 美元从账户 A 转移到账户 B 的交易:
  1. 阅读（A）
  2. A := A – 50
  3. 写(A)
  4. 读(B)
  5. B := B + 50
  6. 写(B)
- 要处理的两个主要问题:
  - 各种故障,如硬件故障和系统故障崩溃
  - 多个事务的并发执行

转帐示例

Example of Fund Transfer

- 将 50 美元从账户 A 转移到账户 B 的交易：
  1. 阅读（一）
  2. A := A – 50
  3. 写（一）
  4. 阅读(B)
  5. B := B + 50
  6. 写(B)
- 原子性要求
  - 如果在第 3 步之后和第 6 步之前交易失败，钱将“丢失”导致不一致的数据库状态
    - 故障可能是由于软件或硬件
  - 系统应确保部分执行事务的更新不会反映在数据库中
- 持久性要求 一旦通知用户交易已经完成（即，50 美元的转移已经发生），即使存在软件或硬件故障，交易对数据库的更新也必须持续存在

转帐示例

Example of Fund Transfer

- 上例中的一致性要求：
  - A 和 B 的总和不会因交易的执行而改变
- 通常，一致性要求包括 明确指定的完整性约束，例如主键和外键

  钥匙
- 隐式完整性约束
  - 例如，所有账户的余额总和减去贷款金额的总和必须
    等值现金
- 事务必须看到一致的数据库
- 在事务执行期间，数据库可能暂时不一致
- 当事务成功完成时，数据库必须是
  持续的
  - 错误的事务逻辑可能导致不一致

转帐示例

- 隔离要求 如果在第 3 步和第 6 步之间,允许另一个事务 T2访问部分更新的数据库,它将看到一个不一致的数据库（ A + B的总和将小于应有的值）

T1	T2
1.阅读（—）	
2. A := A – 50	
3.写（—）	
	读取（A）,读取（B）,打印（A+B）
4.阅读(B)	
5. B := B + 50	
6.写（乙	

- 通过串行运行事务可以轻松确保隔离
  - 也就是说,一个接一个
- 然而,同时执行多个事务具有显着的好处

酸性质 ACID Properties

事务是访问和可能更新各种数据项的程序执行单元。为了保持数据的完整性，数据库系统必须确保：

- 原子性。要么交易的所有操作都正确反映在数据库，或者没有（全有或全无）
- 一致性。隔离执行事务可以保持数据库的一致性
- 隔离。尽管多个事务可以同时执行，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。
    - 也就是说，对于每对事务Ti和Tj ，在Ti看来，要么Tj在Ti开始之前完成执行，要么Tj在Ti完成之后开始执行。
- 耐用性。事务成功完成后，即使出现系统故障，它对数据库所做的更改也会持续存在

交易状态 Transaction State

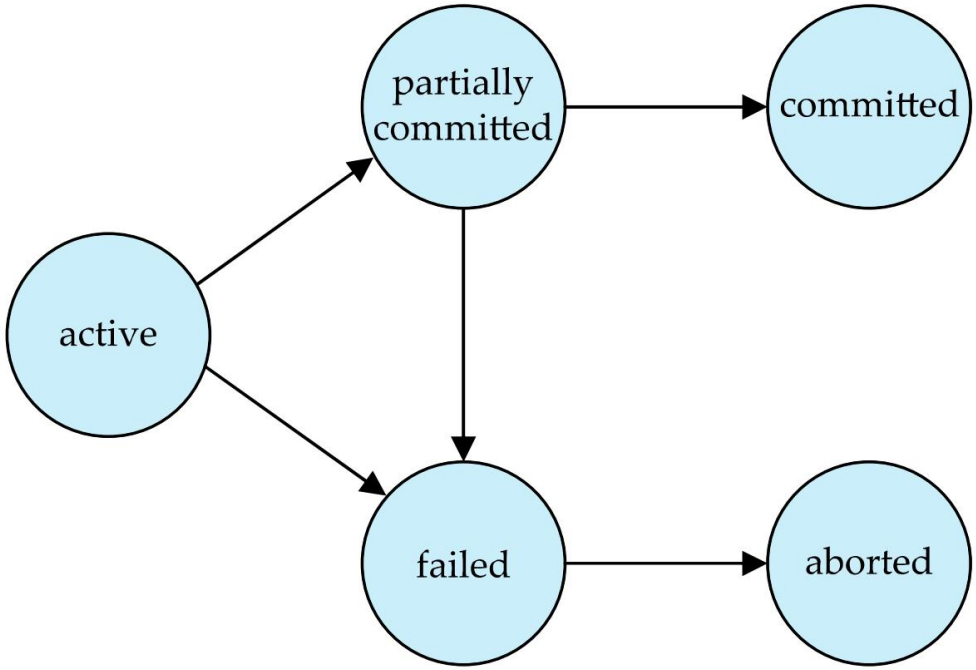
- 活动　初始状态;交易保持在这个状态,而它是执行
- 部分提交　在最终语句执行之后
- 失败　在发现无法继续正常执行之后
- Aborted – 在事务回滚和数据库之后

  恢复到事务开始之前的状态。中止后的两个选项:

  - 重启事务
    - 只有在没有内部逻辑错误时才能完成
  - 终止事务
- Committed – 成功完成后

交易状态

Transaction State



并发执行 Concurrent Executions

- 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，从而实现更好的
    交易吞吐量
    - 例如，一个事务可能正在使用 CPU，而另一个事务正在使用 CPU
      从磁盘读取或写入
  - 减少事务的平均响应时间：短事务无需在长事务之后等待

- 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，防止它们破坏数据库的一致性

丢失更新问题

(a)

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Time

Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

临时更新问题

(b)

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	

Time

Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.

不正确的总结问题

(c)

T_1	T_3
	$sum := 0;$ read_item(A); $sum := sum + A;$ . . .

read_item(X);
$X := X - N;$
write_item(X);	

	read_item(X);
	$sum := sum + X;$
	read_item(Y);
	$sum := sum + Y;$

read_item(Y);
$Y := Y + N;$
write_item(Y);	

T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

不可重复读问题

- 事务 T 读取同一项目两次
- 两次读取之间的另一个事务 T 更改了值
- T 在同一项目的两次读取中收到不同的值

时间表

- 时间表– 指定时间顺序的指令序列
  在其中执行并发事务的指令
  - 一组事务的时间表必须包含所有的指令
    那些交易
  - 必须保持说明出现在每个
    个人交易
- 成功完成执行的事务将有一条提交指令作为最后一条语句
  - 默认情况下,假设事务执行提交指令为
    最后一步

附表1

- 让T1将 50 美元从A转移到B， T2将A余额的 10% 转移从A到B
- T1后跟T2的系列时间表：

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

附表2 Schedule 2

- T2之后是T1的串行时间表

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

附表 3

- 令T1和T2为之前定义的交易。以下
  schedule 不是串行时间表,但它相当于Schedule 1（即T1后跟T2 ）

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

- 在附表 1、2 和 3 中,总和A+B被保留

附表 4

- 以下并发调度不保留(A+B)的值

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

可串行化

- 基本假设 假设每笔交易在其上执行时都是正确的
  自己的
- 假设一组事务的串行执行是正确的
- 正确性标准:每个系列时间表都被认为是正确的
- 如果一个（可能是并发的)调度等效于一个串行调度,那么它是可串行化的。不同形式的调度等价产生了冲突序列化的概念

交易的简化视图

- 我们忽略读写指令以外的操作
- 我们假设事务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

相互矛盾的指令

- 事务Ti和Tj的指令I和J分别当且仅当存在I和J都访问的某个项目Q并且其中至少有一个是写指令时才发生冲突

1. I =读取（Q），J =读取（Q）。 I和J不冲突（I & J的顺序无关紧要）
2. I =读（Q），J =写（Q）。它们冲突（I & J的顺序很重要:先写后读
   写前读会给出不同的读结果）
3. I =写（Q），J =读（Q）。他们冲突（I & J的顺序很重要）
4. I =写（Q），J =写（Q）。它们冲突（I & J的顺序很重要,因为它会影响下一个read(Q)指令的结果）

- 直观地说， I和J之间的冲突会强制它们之间的（逻辑)时间顺序 - 如果I和J在时间表中是连续的并且它们不冲突,即使它们在时间表中互换,它们的结果也将保持不变

冲突可序列化

- 如果一个时间表S可以通过一系列
  非冲突指令的交换,我们说S和S 是冲突等价的

- 或等效地,如果任何两个冲突指令的相对顺序在两个调度中相同,则两个调度是冲突等效的
- 如果一个调度S与串行调度是等价的,我们就说它是冲突可串行化的

冲突可序列化

- 附表 1 可以转换为附表 2,这是一个连续的时间表,其中T2
  在 T1之后,通过一系列非冲突指令的交换。因此附表 1 是冲突可序列化的

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

附表1

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

附表2

冲突可序列化

▪不可序列化冲突的计划示例：

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

▪ T4的更新丢失 ▪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 > ▪这不是冲突可串行化的,因为它不等价到序列号

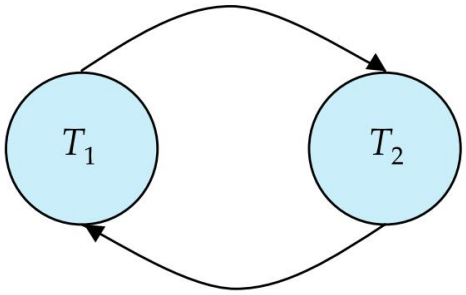
调度 < T3 , T4 >,或串行调度 < T4 , T3 >

可序列化测试 Testing for Serializability

▪考虑一组事务T1 、 T2 、...、 Tn的某个调度S ▪优先图 一个有向图,其中顶点是调度S的事务

▪如果满足以下三个条件之一,我们将绘制一条从Ti到Tj的弧： · Ti 在Tj执行read (Q)之前执行write(Q) · Ti在Tj执行write(Q) 之前执行 read(Q) · Ti执行write (Q)在Tj执行write(Q)之前

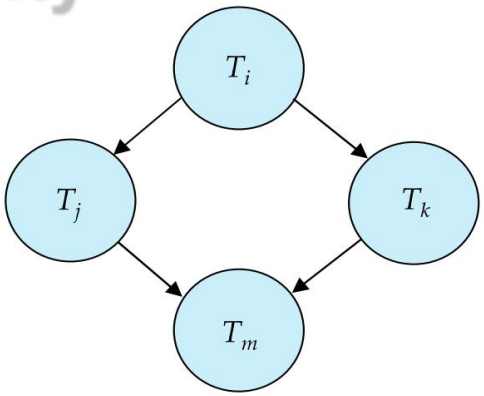
- 如果优先图中存在边Ti → Tj ,则任何等价于S, Ti的串行调度S 必须出现在Tj之前 ▪优先图示例



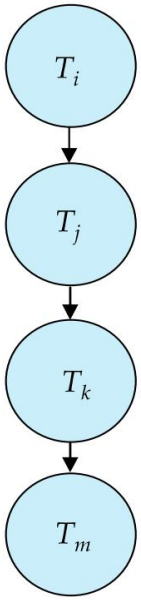
冲突可序列化测试 Testing for Conflict Serializability

- 当且仅当它的优先图是非循环的，一个调度是冲突可序列化的
- 存在需要时间的周期检测算法，其中n是订单n^{2}图中的顶点
- 如果优先图是无环的，则可以通过对图进行拓扑排序得到序列化顺序
  - 这是与图的偏序一致的线性顺序
  - 附表 (a) 的序列化顺序将是

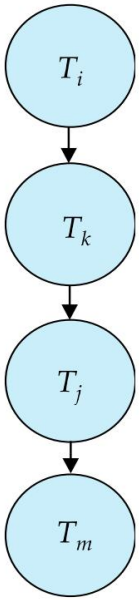
    Ti → Tj → Tk → Tm



(a)



(b)



(c)

可恢复的时间表 Recoverable Schedules

需要解决事务失败对并发运行事务的影响。

- 可恢复计划　如果事务Tj读取事务Ti先前写入的数据项,则 Ti 的提交操作出现在Tj的提交操作之前。

- 以下时间表不可恢复

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

- 如果T8应该中止， T9会读取不一致的数据库状态,但T9已经承诺了。因此,数据库必须确保计划是可恢复的

级联回滚 Cascading Rollbacks

- 级联回滚　单个事务失败导致一系列事务回滚。考虑以下计划，其中尚未提交任何事务（因此计划是可恢复的）

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

如果T10失败， T11和T12也必须回滚。 T11和T12中的读取(A)被称为脏读

- 可能导致大量工作被撤销

无级联时间表

- 无级联计划 级联回滚不能发生；
  - 对于每对事务Ti和Tj ,使得Tj读取一个数据项
    之前由Ti写的， Ti的提交操作出现在Tj的读操作之前

- 每个 Cascadeless 调度也是可恢复的
- 最好将时间表限制为无级联的时间表

一致性水平较弱

- 一些应用程序愿意忍受弱级别的一致性，允许不可序列化的调度
  - 例如，想要获得近似总数的只读事务
    所有账户余额
  - 此类事务不需要相对于其他事务可序列化
    交易
- 以准确性换取绩效

SQL中的事务定义 Transaction Definition in SQL

- 在 SQL 中,事务隐式开始
- SQL 中的事务以以下方式结束:
  - Commit work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 中的事务支持

- 隔离级别
  - 脏读（读取未提交事务的更新）
  - 不可重复读取（另一个事务在两次读取之间更新数据项，以便事务看到两个不同的值）
  - 幻象（如果另一个事务在事务执行期间插入新记录r ，则r在事务开始时不存在，但在事务结束时存在； r称为幻象记录）

	Type of Violation		
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No