有道翻译 支持免费文档翻译240万字符/年 立即下载→ https://fanyi.youdao.com/download

第 15 讲:并发控制理论

15-445/645 数据库系统(2022 年秋季)

https://15445.courses.cs.cmu.edu/fall2022/

卡内基梅隆大学安迪·帕夫洛

1)动机

- **丢失更新问题(并发控制):**我们如何在同时更新记录时避免竞争条件?
- **持久性问题(恢复):**在断电的情况下，我们如何确保正确的状态?

2 交易

*事务*是在共享数据库上执行一个或多个操作序列(例如，SQL 查询)，以执行一些更高级别的功能。它们是 DBMS 中变化的基本单位。不允许部分事务(即事务必须是原子的)。

例子:把 100 美元从安迪的银行账户转到他的发起人账户

1. 检查一下安迪是否有 100 美元。
2. 从他的账户中扣除 100 美元。
3. 在他的发起人账户上加 100 美元。

要么所有的步骤都必须完成，要么一个都不应该完成。

稻草人系统

一个处理事务的简单系统是在一次使用单个 worker(例如，一个线程)执行一个事务。因此，一次只能运行一个事务。为了执行事务，DBMS 复制整个数据库文件，并对这个新文件进行事务更改。如果事务成功，那么新文件将成为当前数据库文件。如果事务失败，DBMS 将丢弃新文件，并且没有保存事务的任何更改。这种方法很慢，因为它不允许并发事务，并且需要为每个事务复制整个数据库文件。

一个(可能的)更好的方法是允许独立事务的并发执行，同时还保持正确性和公平性(因为所有事务都被同等优先级对待，不会因为永远不被执行而“饿死”)。但是在 DBMS 中执行并发事务是具有挑战性的。它很难确保正确性(例如，如果 Andy 只有 100 美元，并试图一次性支付两个发起者，谁应该得到报酬?)同时也快速执行事务(我们的 strawman 例子保证了顺序正确性，但以并行性为代价)。

任意交错的操作会导致:

临时不一致:不可避免，但不是问题。

- **永久不一致:**不可接受，导致数据的正确性和完整性问题。

事务的范围仅在数据库内部。它不能对外部世界进行更改，因为它不能回滚这些更改。例如，如果事务导致发送电子邮件，如果事务被终止，则 DBMS 无法回滚该邮件。

3 定义

形式上，一个*数据库*可以表示为一组固定的命名数据对象(a, B, C，…)。这些对象可以是属性、元组、页面、表，甚至数据库。我们将讨论的算法适用于任何类型的对象，但所有对象必须是相同类型的。

*事务*是在这些对象上的一系列读写操作(即 R(A)， W(B))。为了简化讨论，这个定义假设数据库是一个固定大小的，因此操作只能是读取和更新，而不是插入或删除。

事务的边界由客户端定义。在 SQL 中，事务以 BEGIN 命令开始。事务的结果是 COMMIT 或 ABORT。对于 COMMIT，要么将事务的所有修改保存到数据库中，要么 DBMS 覆盖它并中止。

对于 ABORT，事务的所有更改都将撤消，因此就像事务从未发生过一样。中断可以是自己造成的，也可以是由 DBMS 引起的。

用于确保数据库正确性的标准由首字母缩写 **ACID** 给出。

- **原子性**:原子性确保事务中的所有操作都发生，或者不发生。
- **一致性**:如果每个事务都是一致的，并且数据库在事务开始时是一致的，那么当事务完成时，数据库保证是一致的。
- **隔离**:隔离是指当一个事务执行时，它应该有一种与其他事务隔离的错觉。

持久性:如果事务提交了，那么它对数据库的影响应该持续。

4 ACID:原子性

DBMS 保证事务是**原子**的。事务要么执行所有操作，要么不执行。有两种方法:

方法一:记录日志

DBMS 记录所有的操作，以便它可以撤销中止的事务的操作。它同时在内存和磁盘上维护撤销记录。由于审计和效率的原因，几乎所有现代系统都使用日志记录。

方法 2:影子分页

DBMS 生成由事务修改的页面的副本，事务对这些副本进行更改。只有当事务提交时，页面才可见。这种方法在运行时通常比基于日志的 DBMS 慢。然而，一个好处是，如果你只是单线程，没有日志记录的需要，所以当事务修改数据库时，有更少的磁盘写入。这也使恢复变得简单，因为你需要做的就是从未提交的事务中删除所有页面。不过，一般来说，更好的运行时性能优于更好的恢复性能，因此在实践中很少使用这一点。

5 ACID:一致性

在高层次上，一致性意味着数据库所代表的“世界”在**逻辑上**是正确的。应用程序对数据提出的所有问题(即查询)都将返回逻辑正确的结果。有两个一致性的概念:

数据库一致性:数据库准确地表示它正在建模的真实世界的实体，并遵循完整性约束。(例如，一个人的年龄不可能不为负数)。此外，未来的事务应该在数据库内部看到过去提交的事务的影响。

事务一致性:如果数据库在事务开始前是一致的，那么它也将是一致**的**

后。确保事务的一致性是应用程序的责任。

6 ACID:隔离

DBMS 给事务提供了它们在系统中单独运行的错觉。它们看不到并发事务的影响。这相当于一个事务按串行顺序执行的系统(即一次执行一个事务)。但是为了获得更好的性能，DBMS 必须在保持隔离假象的同时，将并发事务的操作穿插在一起。

并发控制

*并发控制协议*是 DBMS 在运行时如何决定来自多个事务的操作的适当交错。

有两类并发控制协议:

1. **悲观:**DBMS 假定事务将会发生冲突，因此它首先不会让问题出现。
2. **乐观:**DBMS 假设事务之间的冲突很少，因此它选择在事务提交后发生冲突时处理冲突。

DBMS 执行操作的顺序称为*执行计划*。我们希望交错事务来最大化并发性，同时确保输出是“正确的”。并发控制协议的目标是生成一个等同于某些串行执行的执行时间表:

串行调度:不交错不同事务的操作的调度。

- **等效调度:**对于任何数据库状态，如果执行第一个调度的效果与执行第二个调度的效果相同，则两个调度是等效的。
- 可**串行化调度**:可串行化调度是指与事务的任意串行执行等价的调度。不同的串行执行可以产生不同的结果，但都被认为是“正确的”。

如果两个操作是针对不同的事务，它们在同一个对象上执行，并且至少有一个操作是写操作，那么就会发生*冲突*。冲突有三种变体:

- **读写冲突(“不可重复读取”):**一个事务在多次读取同一对象时无法获得相同的值。
- **写-读冲突(“脏读”):**一个事务在提交其更改之前看到了另一个事务的写效果。
- **写-写冲突(“丢失更新”):**一个事务覆盖另一个并发事务的未提交数据。

可串行化有两种类型:(1)*冲突*和(2)*视图*。这两种定义都不允许人们认为可序列化的所有调度。在实践中，dbms 支持冲突序列化，因为它可以有效地执行。

冲突可串行性

两个调度是*冲突等价*的，因为它们涉及相同事务的相同操作，并且每一对冲突操作在两个调度中以相同的方式排序。如果一个调度与某个*串行调度是冲突等价的，则该调度是冲突可序列化*的。

人们可以通过交换不冲突的操作来验证一个调度是冲突可序列化的，直到一个串行调度形成。对于具有许多事务的调度，这将变得过于昂贵。更好的方法是

验证时间表就是使用*依赖图*(优先图)。

在依赖图中，每个事务都是图中的一个节点。如果来自 Ti 的操作 Oi 与来自 Tj 的操作 Oj 冲突，并且 Oi 发生在调度中的 Oj 之前，则存在从节点 Ti 到 Tj 的有向边。那么，一个调度是冲突可序列化的 iff 依赖图是无环的。

视图可串行性

*视图可序列化性*是可序列化性的一个较弱的概念，它允许所有冲突可序列化和“盲写”(即执行写而不首先读取值)的调度。因此，它允许比冲突可序列化更多的调度，但很难有效地执行。这是因为 DBMS 不知道应用程序将如何“解释”值。

时间表的全域

SerialSchedules⊂冲突的 serializableschedules⊂V 的 viewserializableschedules⊂所有时间表

7 ACID:持久性

所有提交事务的更改必须在崩溃或重启后是**持久**的(即持久性)。DBMS 可以使用日志记录或影子分页来确保所有更改都是持久的。