数据库系统概论-11-并发控制

2022-06-16 | 2025-11-26 | 南京大学软件学院本科课程， 2022Spring-数据库系统概论 | 175

4.1k | 4 分钟

1. 并发控制概述

事务是并发控制的基本单位。

并发控制机制的任务：

1. 对并发操作进行正确调度
2. 保证事务的隔离性
3. 保证数据库的一致性

并发控制主要技术：

1. 封锁（Locking）
2. 时间戳（Timestamp）
3. 乐观控制法
4. 多版本并发控制（MVCC）

1.1 多事务执行方式

1.1.1 事务串行执行

每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行。

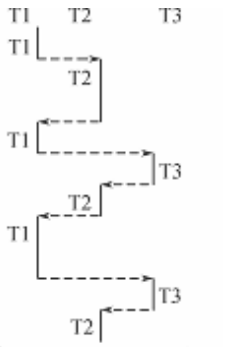
不能充分利用系统资源，发挥数据库共享资源的特点

事务的串行执行方式

1.1.2 交叉并发方式

交叉并发方式（Interleaved Concurrency），在单处理机系统中，事务的并行执行是这些并行事务的并行操作轮流交叉运行。

单处理机系统中的并行事务并没有真正地并行运行，但能够减少处理机的空闲时间，提高系统的效率



(b) 事务的交叉并发执行方式

1.1.3 同时并发方式

同时并发方式（simultaneous concurrency），多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行。

最理想的并发方式，但受制于硬件环境。

1.2 并发操作带来的数据不一致性

1.2.1 丢失修改

两个事务 T_1 和 T_2 读入同一数据并修改， T_2 的提交结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失。

1.2.2 不可重复读

不可重复读是指事务 T_1 读取数据后，事务 T_2 执行更新操作，使 T_1 无法再现前一次读取结果（例如为了校对需要重复读）。

不可重复读包括三种情况，后两种不可重复读有时也称为幻影现象（Phantom Row）：

1. 一读一改：事务 T_1 读取某一数据后，事务 T_2 对其做了修改，当事务 T_1 再次读该数据时，得到与前一次不同的值 。
2. 一读一删：事务 T_1 按一定条件从数据库中读取了某些数据记录后，事务 T_2 删除了其中部分记录，当 T_1 再次按相同条件读取数据时，发现某些记录神秘地消失了。
3. 一读一加：事务 T_1 按一定条件从数据库中读取某些数据记录后，事务 T_2 插入了一些记录，当 T_1 再次按相同条件读取数据时，发现多了一些记录。

1.2.3 读“脏”数据

读“脏”数据是指：

1. 事务 T_1 修改某一数据，并将其写回磁盘
2. 事务 T_2 读取同一数据后， T_1 由于某种原因被撤销
3. 这时 T_1 已修改过的数据恢复原值， T_2 读到的数据就与数据库中的数据不一致
4. T_2 读到的数据就为“脏”数据，即不正确的数据

2. 封锁

封锁就是事务 T 在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁。

加锁后事务 T 就对该数据对象有了一定的控制，在事务 T 释放它的锁之前，其它的事务不能更新此数据对象。

基本封锁类型：

1. 排它锁（eXclusive Locks，简记为 X 锁）

2. 共享锁（Share Locks，简记为 S 锁）

排它锁又称为写锁，表示** 正在写，其他事务不能读 **。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其它任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。保证其他事务在 T 释放 A 上的锁之前不能再读取和修改 A。

共享锁又称为读锁，表示** 正在读，其他事务不能写 **。若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，其它事务只能再对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁。保证其他事务可以读 A，但在 T 释放 A 上的 S 锁之前不能对 A 做任何修改 。

2.1 封锁协议

2.1.1 一级封锁协议

事务 T 在**修改数据 R 之前必须先对其加 X 锁**，直到事务结束（COMMIT 或 ROLLBACK）才释放。

一级封锁协议可**防止丢失修改**，并**保证事务 T 是可恢复的。**在一级封锁协议中，如果仅仅是读数据不对其进行修改，是不需要加锁的，所以它**不能保证可重复读**和**不读“脏”数据。**

2.1.2 二级封锁协议

一级封锁协议加上事务 T 在**读取数据 R 之前必须先对其加 S 锁**，**读完后**即可释放 S 锁。

二级封锁协议可以**防止丢失修改**和**读“脏”数据。**在二级封锁协议中，由于读完数据后即可释放 S 锁，所以它**不能保证可重复读。**

2.1.3 三级封锁协议

一级封锁协议加上事务 T 在**读取数据 R 之前必须先对其加 S 锁**，直到**事务结束**才释放。

三级封锁协议可**防止丢失修改、读脏数据和不可重复读。**

	防止丢失修改	**防止读“脏”数据**	**保证可重复读**
一级封锁协议	Y		
二级封锁协议	Y	Y	
三级封锁协议	Y	Y	Y

2.2 活锁

活锁指优先级较低的事务有可能总是得不到资源。

解决活锁的方式是使用先来先服务策略。

2.3 死锁

两个（或多个）事务相互等待对方释放各自已经持有的资源，形成死锁。

2.3.1 一次封锁法

一次封锁法，要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。

1. 降低系统并发度。
2. 难于事先精确确定封锁对象：数据库中数据是不断变化的，原来不要求封锁的数据，在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象。
    1. 解决方法：将事务在执行过程中可能要封锁的数据对象全部加锁，这就进一步降低了并发度。

2.4 顺序封锁法

顺序封锁法，预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。

1. 维护成本：数据库系统中封锁的数据对象极多，并且随数据的插入、删除等操作而不断地变化，要维护这样的资源的封锁顺序非常困难，成本很高。
2. 难以实现：事务的封锁请求可以随着事务的执行而动态地决定，很难事先确定每一个事务要封锁哪些对象，因此也就很难按规定的顺序去施加封锁

2.5 诊断解除法

超时法：如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。

等待图法：并发控制子系统周期性地（比如每隔数秒）生成事务等待图，检测事务。如果发现图中存在回路，则表示系统中出现了死锁。

解除死锁：

1. 选择一个处理死锁代价最小的事务，将其撤消
2. 释放此事务持有的所有的锁，使其它事务能继续运行下去

3. 事务调度

串行调度是正确的。执行结果等价于串行调度的调度也是正确的，称为**可串行化调度**。多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。

可串行性（Serializability）是并发事务正确调度的准则：一个给定的并发调度，当且仅当它是可串行化的，才认为是正确调度。

3.1 冲突可串行化

冲突操作：是指不同的事务对同一数据的读写操作和写写操作（如 $R_i(x)$ 与 $W_j(x)$、$W_i(x)$ 与 $W_j(x)$）

一个调度 Sc 在保证冲突操作的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度 Sc’ ，如果 Sc’ 是串行的，称调度 Sc 是**冲突可串行化**的调度。

冲突可串行化调度是可串行化调度的充分条件，不是必要条件。

> 下图是一个视图可串行化的例子，但不是冲突可串行化。如果一个调度是可串行化但不是冲突可串行化，则必然是多了若干盲写。
>
> Example（视图可串行化调度）
>
> | T_1 | T_2 | T_3 |
> |---|---|---|
> | r(A) | | |
| | w(A) | |
> | w(A) | | |
| | | w(A) |
>
> $\equiv_{view}$
>
> | T_1 | T_2 | T_3 |
> |---|---|---|
> | r(A) | | |
> | w(A) | | |
| | w(A) | |
| | | w(A) |
>
>

3.2 两段锁协议

两段锁协议，指所有事务必须分两个阶段对数据项加锁和解锁：

1. 在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁
2. 在释放一个封锁之后，事务不再申请和获得任何其他封锁

在此协议下，事务分为两个阶段：

1. 第一阶段是获得封锁，也称为扩展阶段：事务可以申请获得任何数据项上的任何类型的锁，但是**不能释放任何锁**
2. 第二阶段是释放封锁，也称为收缩阶段：事务可以释放任何数据项上的任何类型的锁，但是**不能再申请任何锁**

事务遵守两段锁协议是可串行化调度的充分条件，而不是必要条件。

但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁。

4. 封锁粒度

封锁对象的大小称为封锁粒度（Granularity）。封锁的对象：逻辑单元，物理单元。在关系数据库中的封锁对象：

1. 逻辑单元: 属性值、属性值的集合、元组、关系、索引项、整个索引、整个数据库等
2. 物理单元：页（数据页或索引页）、物理记录等

封锁粒度与系统的并发度和并发控制的开销密切相关：封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；封锁的粒度越小，并发度较高，但系统开销也就越大

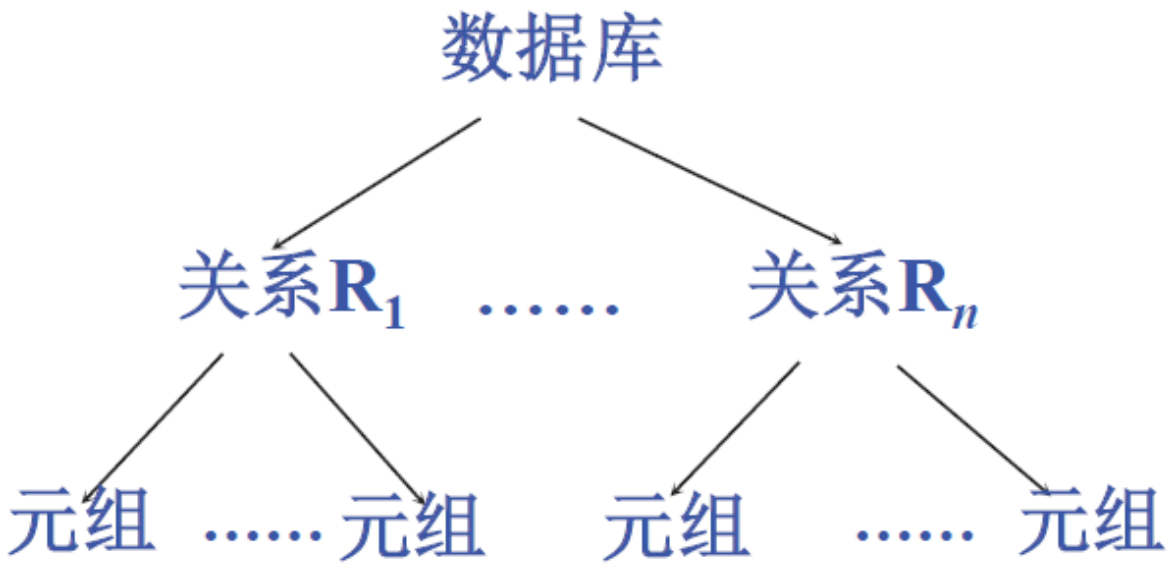
4.1 多粒度封锁

多粒度封锁（Multiple Granularity Locking），在一个系统中同时支持多种封锁粒度供不同的事务选择。选择封锁粒度，同时考虑封锁开销和并发度两个因素, 适当选择封锁粒度：

1. 需要处理多个关系的大量元组的用户事务：以数据库为封锁单位
2. 需要处理大量元组的用户事务：以关系为封锁单元
3. 只处理少量元组的用户事务：以元组为封锁单位

多粒度树：以树形结构来表示多级封锁粒度，根节点是整个数据库，表示最大的数据粒度，叶节点表示最小的数据粒度。

例：三级粒度树。根节点为数据库，数据库的子节点为关系，关系的子节点为元组。



多粒度封锁协议允许多粒度树中的每个节点被独立地加锁。对一个节点加锁意味着这个节点的所有后裔节点也被加以同样类型的锁。在多粒度封锁中一个数据对象可能以两种方式封锁：

1. **显式封锁**：直接加到数据对象上的封锁
2. **隐式封锁**：是该数据对象没有独立加锁，是由于其上级节点加锁而使该数据对象加上了锁

显式封锁和隐式封锁的效果是一样的。系统检查封锁冲突时既要检查显式封锁，还要检查隐式封锁。对某个数据对象加锁，系统要检查：

1. 该数据对象：有无显式封锁与之冲突
2. 所有上级节点：检查本事务的显式封锁是否与该数据对象上的隐式封锁冲突（由上级节点已加的封锁造成的）冲突
3. 所有下级节点：看上面的显式封锁是否与本事务的隐式封锁（将加到下级节点的封锁）冲突

4.2 意向锁

引进意向锁（intention lock）目的：提高对某个数据对象加锁时系统的检查效率。如果对一个节点加意向锁，则说明该节点的下层节点正在被加锁。对任一节点加基本锁，必须先对它的上层节点加意向锁。

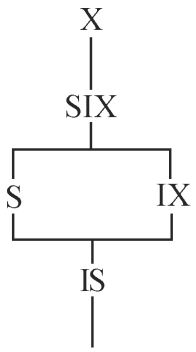
1. 意向共享锁（Intent Share Lock，IS 锁）：表示它的后裔节点拟（意向）加 S 锁。
2. 意向排它锁（Intent Exclusive Lock，IX 锁）：表示它的后裔节点拟（意向）加 X 锁。
3. 共享意向排它锁（Share Intent Exclusive Lock，SIX 锁）：表示对它加 S 锁，再加 IX /锁，即 SIX = S + IX。

锁的强度是指它对其他锁的排斥程度。一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然。锁的强度有偏序关系。强度越大，表示该锁所能相容的锁越少。

T_1 \ T_2	S	X	IS	IX	SIX	–
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
–	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵



(b) 锁的强度的偏序关系

申请封锁时应该按自上而下的次序进行，释放封锁时则应该按自下而上的次序进行。具有意向锁的多粒度封锁方法：提高了系统的并发度，减少了加锁和解锁的开销，在实际的数据库管理系统产品中得到广泛应用。

打赏

原创

数据库系统概论-10-数据库恢复技术 《NoSQL 精粹》第一章-为什么使用 NoSQL