第 6 卷第 10 期
1995 年 10 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 10
Oct. 1995

一个基于 SQL 的并发控制机制*

曲云尧　　施伯乐

(复旦大学计算机科学系,上海 200433)

摘要　传统的读写事务模型(由 read(x)和 write(x)序列组成)不能使调度机制充分利用应用程序的语义信息对事务进行灵活调度,从而不能有效提高系统的并发度.本文根据 SQL 语言的操作语义,给出了基于 SQL 的事务模型.利用这种事务模型并结合 2PL 方法,设计了并发控制机制:Condition－locking.这个机制可以:(1)避免数据库中的幽灵(phantom)问题,(2)利用应用程序的语义信息和完整性约束提高系统的并发度,(3)减少发生死锁的机会.因此,这是一个实用的并发控制机制.

关 键 词　数据库,并发控制,幽灵问题,SQL 语言,两段锁.

1　关系数据库中传统封锁机制的问题

在多用户或分布式数据库系统中,DBMS 并发控制机制的效率对整个系统的吞吐率起着关键性的作用.特别是在事务处理速度要求较高的环境,如证券业、银行业等,就显得更为重要.因此,如何提高数据库系统并发控制机制的性能,有着很现实的意义.

并发控制方法一直是数据库研究领域的主要课题[1,2].自从 Eswaran 等发表了关于并发控制的 2PL 锁方法以来[3],并发控制技术得到了很大发展.人们在不断地试图发现更有效的方法和技术,以适应不同的应用环境.其中,主要可把它们归纳为以下几类:(1)锁方法,(2)时间戳方法,(3)多版本方法,(4)乐观方法等.但是,无论是在哪种原型中[4]或是在成熟的商品中,如 Oracle、SYBASE、INFORMIX 等,锁方法占有绝对的主导地位,其原因不仅是它简单易实现,而且在许多环境中,相对于其它方法来说,其效率是最高的[5].但锁方法也带来了一些问题.例如,如何有效确定封锁单位,如何避免幽灵进出数据库,如何减少发生死锁机会等,都是需要进一步研究的.

在研究新方法的同时,提高和改进原有并发控制机制的效率,即并发度,一直是人们极为感兴趣的问题[6,7].其主要方法是利用事务(或应用程序)的语义信息,去掉事务之间不必要的等待.即人们企图从事务的更高层次上得到更多的关于此事务的语义信息,并利用这些信息对事务进行高效调度.

---

* 本文 1994－04－23 收到,1994－11－22 定稿

本项目受到国家 863 高技术项目资助.作者**曲云尧**,1961 年生,副教授,主要研究领域为面向对象数据库,知识库的体系结构,事务管理技术.**施伯乐**,1935 年生,教授,博士导师,主要研究领域为数据库理论与实现技术,知识库,软件工程等.

本文通讯联系人:曲云尧,上海 200433,复旦大学计算机系

在关系数据库中,2PL 锁机制归结起来有下列主要问题:

(1)封锁单位问题. 关系数据库中,逻辑级上的封锁单位一般分为关系和元组. 物理级上的封锁单位一般分为页面. 用户程序对关系或元组的封锁,DBMS 最终要转换成物理级上对页面的封锁. 例如,在逻辑级上要封锁某几个元组,如这几个元组在页面 P1 中,系统则要锁住这个页面 P1. 因此,一般来讲,系统实际的封锁单位要比逻辑上要求的封锁单位大一些.

例 1:有两个事务

```
T1:  SELECT  A          T2:  UPDATE  R
     FROM    R               SET     (A=a,B=b)
     WHERE   B>3             WHERE   B<=3
```

设关系 R 中某些 B 属性值大于 3 和小于 3 的元组存放在同一页面中. 如果系统在物理级上对页面进行封锁,则这两个事务就发生冲突. 但从逻辑上我们知道 T1 封锁的 R 中 B 属性值大于 3 的元组,而 T2 则相反,所以这两个事务实际上无冲突,可以并发执行. 因此,如果使用大的封锁单位(页面),则要降低并发度. 但是,如果我们只封锁页面中(要使用)的元组,则会出现什么问题呢?

(2)幽灵问题. 幽灵问题[1,3]可以说是封锁方法中的一个较为棘手的问题. 应该说,现在许多商品化 DBMS 都没有很好地解决它. 例 2 说明了一个幽灵问题.

例 2:有一个雇员关系 R 如表 1.

表 1　雇员关系 R

EMPNAME	AGE	SALARY	DEPT
John	30	2000	SAL
Mary	40	3000	TOY
Francis	25	2500	SAL
Susan	27	2800	SAL

有两个事务 T1 和 T2,T1:统计 SAL 部门雇员的平均工资. T2:从 SAL 部门删除 John,插入一元组(Mark,25,2500,SAL)到 R 中. 一个可能的 2PL 调度如下:

(1) T2 从关系 R 中锁住并删除关于 John 的元组(John,30,2000,SAL)

(2) T1 锁住(Francis,25,2500,SAL)和(Susan,27,2800,SAL)两个元组,并统计平均值((2500+2800)/2=2650)

(3) T1 提交(COMMIT)

(4) T2 插入元组(Mark,25,2500,SAL)到 R 中,并锁住

(5) T2 提交

如果让 T1 和 T2 串行执行,则如按调度 T1 T2 执行,T1 的结果为(2000+2500+2800)/3=2433.3;如按 T2 T1 执行,T1 的结果为(2500+2800+2500)/3=2600. 因此,上面的调度是不可串行化的(尽管它们都遵守 2PL 协议). 我们称(John,30,2000,SAL)和(Mark,25,2500,SAL)为幽灵元组. 因为它们出入数据库就象幽灵一样,其它事务不能发现它们. 当然,如果加大封锁单位,例如,锁住整个关系,则可避免幽灵问题,但这对系统的效率影响极大. 可以想象关系 R 中存放了大量关于其它部门职员的有关元组.

(3)事务模型研究和实际应用程序之间脱离.通常在数据库中,我们把事务的操作类型分为:读操作(Read)和写操作(Write).可以说,这两种基本操作不能充分体现事务的操作语义,从而不能使系统利用事务的语义信息来提高并发度.在研究并发控制机制时,传统的研究方法并不考虑事务模型和应用程序之间的语义联系,因此,事务模型语义单调(只由Read(X)和Write(X)序列组成),很难进一步改善并发控制机制的性能.例如,在例1中,实际上根据T1和T2两个事务的操作条件(分别为B>3和B<=3)知它们是不冲突的,可以并发执行.确切地说,这种调度是基于应用程序语义的调度.上面这种调度实际使用了谓词锁[1,3]的概念.谓词封锁单位,或者说,基于操作条件的封锁单位是理想的,它不但可避免幽灵问题,而且这种封锁单位是准确的.但文献[1]认为,当谓词条件复杂时,实现起来较为复杂.

本文结合关系数据库标准于语言SQL和2PL锁方法,给出了基于SQL的事务模型,设计了基于这种模型的并发控制机制——Condition-Locking.这个机制可以避免数据库中的幽灵问题,利用更多的语义信息提高并发度,并在一定程度上减少了发生死锁的机会.

2 基于SQL的事务模型

在传统数据库并发控制方法的研究中,通常把事务抽象成仅由读操作(read(x))和写操作(write(x))组成的序列.在本文,我们把事务表示成由查询操作:Q(R,C)和更新操作:U(R,C),D(R,C),I(R,C)组成的序列.其中Q,U,D和I分别表示SQL程序中的选择(SELECT),修改(UPDATE),删除(DELETE),和插入(INSERT);R表示关系,C表示操作条件.Q(R,C)(U(R,C),D(R,C))表示对关系R中满足条件C的元组查询(修改,删除)I(R,C)表示插入R中的元组满足条件C.

定义1. 设事务Ti的操作Oi要对数据库中满足条件Ci的元组进行操作,则称Ci为Oi的操作条件.满足条件Ci的元组集合记为SAT(Ci).

根据SQL语法,把SQL程序转换成事务操作序列的方法可描述如下.

(1)选择语句的转换

· 在单个关系中选择

```
SELECT  A
FROM    R        转换成 Q(R,C)
WHERE   C
```

· 在多个关系中选择(连接)

```
SELECT  A
FROM    R1,R2,…,Rn        转换成 Q(R1,C1)Q(R2,C2)…Q(Rn,Cn)
WHERE   C1∧C2∧…∧Cn∧φ
```

其中Ci(i∈{1,…,n})是在Ri中做选择运算时的选择条件,φ表示n个关系之间的连接条件.

· 嵌套查询

```
SELECT  A
```

```
FROM    R1
WHERE   C1 AND α  θ  SELECT  B      转换成 Q(R1,C1)Q(R2,C2)
                     FROM    R2
                     WHERE   C2
```

当出现多层嵌套时,处理方法类似.

(2)修改语句的转换

```
UPDATE  R
SET     (A1=W1,A2=W2,…)      转换成 U(R,C)
WHERE   C
```

(3)插入语句的转换

```
INSERT
INTO    R(A1,A2,…,An)      转换成 I(R,C)
VALUE   (a1,a2,…,an)
```

C 为 A1=a AND A2=a2 AND…AND An=an

(4)删除语句的转换

```
DELETE
FROM    R       转换成 D(R,C)
WHERE   C
```

经过上述处理后,我们可把事务 Ti 写成 $Ti=(Oj(Rj,Cj))_{j=1}^{n}$

其中 $Oj\in\{Qi,Ui,Di,Ii\}$,Ti 按下列操作顺序执行 O1(R1,C1) O2(R2,C2) … On(Rn,Cn)

在执行完 On(Cn,Rn)后,再执行 COMMIT 操作表示提交,只要其中有一步无法执行,则整个事务无法执行.

例 3:设有一 SQL 程序如下

```
SELECT *
FROM R,S
WHERE R.A>1 AND R.B=S.B AND S.C >2
UPDATE R
SET A=2,B=3
WHERE B=2
```

根据前面的转换过程,其相应事务 T 的操作序列为:T: Q(R,A>1) Q(S,C>2) U(R,B=2),即事务 T 先对 R 和 S 做查询,其查询条件分别为 A>1 和 C>2,然后对 R 进行修改操作.其条件为 B=2.

3　冲突类型与相关判定算法

根据上节提出的事务模型,现给出事务之间冲突的定义.

定义 2. 设 Oi(Ri,Ci)和 Oj(Rj,Cj)分别表示 Ti 和 Tj 的操作($i\neq j$),则 Ci 和 Cj 相关当且仅当 Ri 和 Rj 为同一关系且存在一元组 $t\in DOM(Ri)$,使 $t\in SAT(Ci)$且 $t\in SAT(Cj)$.

定义 3. 操作 Oi(Ri,Ci)和 Oj(Rj,Cj)是不冲突的,当且仅当 Ci 和 Cj 不相关,或者 Oi 和