# 1 前言

数据库大并发操作要考虑死锁和锁的性能问题。看到网上大多语焉不详(尤其更新锁），所以这里做个简明解释，为下面描述方便，这里用T1代表一个数据库执行请求，T2代表另一个请求，也可以理解为T1为一个线程，T2 为另一个线程。T3,T4以此类推。下面以SQL Server(2005)为例。

# 2 锁的种类

### 1.共享锁(Shared lock)。

例1：

----------------------------------------

T1: select \* from table (请想象它需要执行1个小时之久，后面的sql语句请都这么想象）

T2: update table set column1='hello'

过程：

T1运行 （加共享锁)

T2运行

If T1 还没执行完

T2等......

else

锁被释放

T2执行

endif

T2之所以要等，是因为T2在执行update前，试图对table表加一个排他锁，

而数据库规定同一资源上不能同时共存共享锁和排他锁。所以T2必须等T1

执行完，释放了共享锁，才能加上排他锁，然后才能开始执行update语句。

例2：

----------------------------------------

T1: select \* from table

T2: select \* from table

这里T2不用等待T1执行完，而是可以马上执行。

分析：

T1运行，则table被加锁，比如叫lockA

T2运行，再对table加一个共享锁，比如叫lockB。

两个锁是可以同时存在于同一资源上的（比如同一个表上）。这被称为共

享锁与共享锁兼容。这意味着共享锁不阻止其它session同时读资源，但阻

止其它session update

例3：

----------------------------------------

T1: select \* from table

T2: select \* from table

T3: update table set column1='hello'

这次，T2不用等T1运行完就能运行，T3却要等T1和T2都运行完才能运行。

因为T3必须等T1和T2的共享锁全部释放才能进行加排他锁然后执行update

操作。

例4：（死锁的发生）

----------------------------------------

T1:

begin tran

select \* from table (holdlock) (holdlock意思是加共享锁，直到事物结束才释放)

update table set column1='hello'

T2:

begin tran

select \* from table(holdlock)

update table set column1='world'

假设T1和T2同时达到select，T1对table加共享锁，T2也对加共享锁，当

T1的select执行完，准备执行update时，根据锁机制，T1的共享锁需要升

级到排他锁才能执行接下来的update.在升级排他锁前，必须等table上的

其它共享锁释放，但因为holdlock这样的共享锁只有等事务结束后才释放，

所以因为T2的共享锁不释放而导致T1等(等T2释放共享锁，自己好升级成排

他锁），同理，也因为T1的共享锁不释放而导致T2等。死锁产生了。

例5：

----------------------------------------

T1:

begin tran

update table set column1='hello' where id=10

T2:

begin tran

update table set column1='world' where id=20

这种语句虽然最为常见，很多人觉得它有机会产生死锁，但实际上要看情

况，如果id是主键上面有索引，那么T1会一下子找到该条记录(id=10的记

录），然后对该条记录加排他锁，T2，同样，一下子通过索引定位到记录，

然后对id=20的记录加排他锁，这样T1和T2各更新各的，互不影响。T2也不

需要等。

但如果id是普通的一列，没有索引。那么当T1对id=10这一行加排他锁后，

T2为了找到id=20，需要对全表扫描，那么就会预先对表加上共享锁或更新

锁或排他锁(依赖于数据库执行策略和方式，比如第一次执行和第二次执行

数据库执行策略就会不同）。但因为T1已经为一条记录加了排他锁，导致

T2的全表扫描进行不下去，就导致T2等待。

死锁怎么解决呢？一种办法是，如下：

例6：

----------------------------------------

T1:

begin tran

select \* from table(xlock) (xlock意思是直接对表加排他锁)

update table set column1='hello'

T2:

begin tran

select \* from table(xlock)

update table set column1='world'

这样，当T1的select 执行时，直接对表加上了排他锁，T2在执行select时，就需要等T1事物完全执行完才能执行。排除了死锁发生。

但当第三个user过来想执行一个查询语句时，也因为排他锁的存在而不得不等待，第四个、第五个user也会因此而等待。在大并发

情况下，让大家等待显得性能就太友好了，所以，这里引入了更新锁。

### 2.更新锁(Update lock)

为解决死锁，引入更新锁。

例7：

----------------------------------------

T1:

begin tran

select \* from table(updlock) (加更新锁)

update table set column1='hello'

T2:

begin tran

select \* from table(updlock)

update table set column1='world'

更新锁的意思是：“我现在只想读，你们别人也可以读，但我将来可能会做更新操作，我已经获取了从共享锁（用来读）到排他锁

（用来更新）的资格”。一个事物只能有一个更新锁获此资格。

T1执行select，加更新锁。

T2运行，准备加更新锁，但发现已经有一个更新锁在那儿了，只好等。

当后来有user3、user4...需要查询table表中的数据时，并不会因为T1的select在执行就被阻塞，照样能查询，相比起例6，这提高

了效率。

例8:

----------------------------------------

T1: select \* from table(updlock) (加更新锁）

T2: select \* from table(updlock) (等待，直到T1释放更新锁，因为同一时间不能在同一资源上有两个更新锁）

T3: select \* from table (加共享锁，但不用等updlock释放，就可以读）

这个例子是说明：共享锁和更新锁可以同时在同一个资源上。这被称为共享锁和更新锁是兼容的。

例9:

----------------------------------------

T1:

begin

select \* from table(updlock) (加更新锁）

update table set column1='hello' (重点：这里T1做update时，不需要等T2释放什么，而是直接把更新锁升级为排他锁，然后执行update)

T2:

begin

select \* from table (T1加的更新锁不影响T2读取）

update table set column1='world' (T2的update需要等T1的update做完才能执行)

我们以这个例子来加深更新锁的理解，

第一种情况：T1先达，T2紧接到达；在这种情况中，T1先对表加更新锁，T2对表加共享锁，假设T2的select先执行完，准备执行update，

发现已有更新锁存在，T2等。T1执行这时才执行完select，准备执行update，更新锁升级为排他锁，然后执行update，执行完成，事务

结束，释放锁，T2才轮到执行update。

第二种情况：T2先达，T1紧接达；在这种情况，T2先对表加共享锁，T1达后，T1对表加更新锁，假设T2 select先结束，准备

update，发现已有更新锁，则等待，后面步骤就跟第一种情况一样了。

这个例子是说明：排他锁与更新锁是不兼容的，它们不能同时加在同一子资源上。

### 3.排他锁（独占锁，Exclusive Locks)

这个简单，即其它事务既不能读，又不能改排他锁锁定的资源。

例10

T1: update table set column1='hello' where id<1000

T2: update table set column1='world' where id>1000

假设T1先达，T2随后至，这个过程中T1会对id<1000的记录施加排他锁.但不会阻塞T2的update。

例11 (假设id都是自增长且连续的）

T1: update table set column1='hello' where id<1000

T2: update table set column1='world' where id>900

如同例10，T1先达，T2立刻也到，T1加的排他锁会阻塞T2的update.

### 4.意向锁(Intent Locks)

意向锁就是说在屋（比如代表一个表）门口设置一个标识，说明屋子里有人（比如代表某些记录）被锁住了。另一个人想知道屋子

里是否有人被锁，不用进屋子里一个一个的去查，直接看门口标识就行了。

当一个表中的某一行被加上排他锁后，该表就不能再被加表锁。数据库程序如何知道该表不能被加表锁？一种方式是逐条的判断该

表的每一条记录是否已经有排他锁，另一种方式是直接在表这一层级检查表本身是否有意向锁，不需要逐条判断。显然后者效率高。

例12：

----------------------------------------

T1: begin tran

select \* from table (xlock) where id=10 --意思是对id=10这一行强加排他锁

T2: begin tran

select \* from table (tablock) --意思是要加表级锁

假设T1先执行，T2后执行，T2执行时，欲加表锁，为判断是否可以加表锁，数据库系统要逐条判断table表每行记录是否已有排他锁，

如果发现其中一行已经有排他锁了，就不允许再加表锁了。只是这样逐条判断效率太低了。

实际上，数据库系统不是这样工作的。当T1的select执行时，系统对表table的id=10的这一行加了排他锁，还同时悄悄的对整个表

加了意向排他锁(IX)，当T2执行表锁时，只需要看到这个表已经有意向排他锁存在，就直接等待，而不需要逐条检查资源了。

例13：

----------------------------------------

T1: begin tran

update table set column1='hello' where id=1

T2: begin tran

update table set column1='world' where id=1

这个例子和上面的例子实际效果相同，T1执行，系统对table同时对行家排他锁、对页加意向排他锁、对表加意向排他锁。

### 5. 计划锁(Schema Locks)

例14：

----------------------------------------

alter table .... (加schema locks，称之为Schema modification (Sch-M) locks

DDL语句都会加Sch-M锁

该锁不允许任何其它session连接该表。连都连不了这个表了，当然更不用说想对该表执行什么sql语句了。

例15:

----------------------------------------

用jdbc向数据库发送了一条新的sql语句，数据库要先对之进行编译，在编译期间，也会加锁，称之为：Schema stability (Sch-S) locks

select \* from tableA

编译这条语句过程中，其它session可以对表tableA做任何操作(update,delete，加排他锁等等），但不能做DDL(比如alter table)操作。

### 6.Bulk Update Locks 主要在批量导数据时用（比如用类似于oracle中的imp/exp的bcp命令）。不难理解，程序员往往也不需要关心，不赘述了。

# 3.何时加锁？

如何加锁，何时加锁，加什么锁，你可以通过hint手工强行指定，但大多是数据库系统自动决定的。这就是为什么我们可以不懂锁也可

以高高兴兴的写SQL。

例15:

----------------------------------------

T1: begin tran

update table set column1='hello' where id=1

T2: SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED -- 事物隔离级别为允许脏读

go

select \* from table where id=1

这里，T2的select可以查出结果。如果事物隔离级别不设为脏读，则T2会等T1事物执行完才能读出结果。

数据库如何自动加锁的？

1) T1执行，数据库自动加排他锁

2) T2执行，数据库发现事物隔离级别允许脏读，便不加共享锁。不加共享锁，则不会与已有的排他锁冲突，所以可以脏读。

例16:

----------------------------------------

T1: begin tran

update table set column1='hello' where id=1

T2: select \* from table where id=1 --为指定隔离级别，则使用系统默认隔离级别，它不允许脏读

如果事物级别不设为脏读，则：

1) T1执行，数据库自动加排他锁

2) T2执行，数据库发现事物隔离级别不允许脏读，便准备为此次select过程加共享锁，但发现加不上，因为已经有排他锁了，所以就

等啊等。直到T1执行完，释放了排他锁，T2才加上了共享锁，然后开始读....

# 4.锁的粒度

锁的粒度就是指锁的生效范围，就是说是行锁，还是页锁，还是整表锁. 锁的粒度同样既可以由数据库自动管理，也可以通过手工指定hint来管理。

例17：

----------------------------------------

T1: select \* from table (paglock)

T2: update table set column1='hello' where id>10

T1执行时，会先对第一页加锁，读完第一页后，释放锁，再对第二页加锁，依此类推。假设前10行记录恰好是一页(当然，一般不可能

一页只有10行记录），那么T1执行到第一页查询时，并不会阻塞T2的更新。

例18：

----------------------------------------

T1: select \* from table (rowlock)

T2: update table set column1='hello' where id=10

T1执行时，对每行加共享锁，读取，然后释放，再对下一行加锁;T2执行时，会对id=10的那一行试图加锁，只要该行没有被T1加上行锁，

T2就可以顺利执行update操作。

例19：

----------------------------------------

T1: select \* from table (tablock)

T2: update table set column1='hello' where id = 10

T1执行，对整个表加共享锁. T1必须完全查询完，T2才可以允许加锁，并开始更新。

以上3例是手工指定锁的粒度，也可以通过设定事物隔离级别，让数据库自动设置锁的粒度。不同的事物隔离级别，数据库会有不同的

加锁策略（比如加什么类型的锁，加什么粒度的锁）。具体请查联机手册。

# 5.锁与事物隔离级别的优先级

手工指定的锁优先，

例20:

----------------------------------------

T1: GO

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

GO

BEGIN TRANSACTION

SELECT \* FROM table (NOLOCK)

GO

T2: update table set column1='hello' where id=10

T1是事物隔离级别为最高级，串行锁，数据库系统本应对后面的select语句自动加表级锁，但因为手工指定了NOLOCK，所以该select

语句不会加任何锁，所以T2也就不会有任何阻塞。

# 6. 数据库的其它重要Hint以及它们的区别

1) holdlock 对表加共享锁，且事物不完成，共享锁不释放。

2) tablock 对表加共享锁，只要statement不完成，共享锁不释放。

与holdlock区别，见下例：

例21

----------------------------------------

T1:

begin tran

select \* from table (tablock)

T2:

begin tran

update table set column1='hello' where id = 10

T1执行完select，就会释放共享锁，然后T2就可以执行update. 此之谓tablock. 下面我们看holdlock

例22

----------------------------------------

T1:

begin tran

select \* from table (holdlock)

T2:

begin tran

update table set column1='hello' where id = 10

T1执行完select，共享锁仍然不会释放，仍然会被hold(持有），T2也因此必须等待而不能update. 当T1最后执行了commit或

rollback说明这一个事物结束了，T2才取得执行权。

3) TABLOCKX 对表加排他锁

例23：

----------------------------------------

T1: select \* from table(tablockx) (强行加排他锁）

其它session就无法对这个表进行读和更新了，除非T1执行完了，就会自动释放排他锁。

例24：

----------------------------------------

T1: begin tran

select \* from table(tablockx)

这次，单单select执行完还不行，必须整个事物完成（执行了commit或rollback后）才会释放排他锁。

4) xlock 加排他锁

那它跟tablockx有何区别呢？

它可以这样用，

例25:

----------------------------------------

select \* from table(xlock paglock) 对page加排他锁

而TABLELOCX不能这么用。

xlock还可这么用：select \* from table(xlock tablock) 效果等同于select \* from table(tablockx)

# 7.锁的超时等待

例26

SET LOCK\_TIMEOUT 4000 用来设置锁等待时间，单位是毫秒，4000意味着等待

4秒可以用select @@LOCK\_TIMEOUT查看当前session的锁超时设置。-1 意味着

永远等待。

T1: begin tran

udpate table set column1='hello' where id = 10

T2: set lock\_timeout 4000

select \* from table wehre id = 10

T2执行时，会等待T1释放排他锁，等了4秒钟，如果T1还没有释放排他锁，T2就会抛出异常： Lock request time out period exceeded.

# 8 附：各种锁的兼容关系表

| Requested mode | IS | S | U | IX | SIX | X |

| Intent shared (IS) | Yes | Yes | Yes | Yes | Yes | No |

| Shared (S) | Yes | Yes | Yes | No | No | No |

| Update (U) | Yes | Yes | No | No | No | No |

| Intent exclusive (IX) | Yes | No | No | Yes | No | No |

| Shared with intent exclusive (SIX) | Yes | No | No | No | No | No |

| Exclusive (X) | No | No | No | No | No | No |

# 9 如何提高并发效率

1. 悲观锁：利用数据库本身的锁机制实现。通过上面对数据库锁的了解，可以根据具体业务情况综合使用事务隔离级别与合理的手工指定锁的方式比如降低锁的粒度等减少并发等待。
2. 乐观锁：利用程序处理并发。原理都比较好理解，基本一看即懂。方式大概有以下3种
   1. 对记录加版本号.
   2. 对记录加时间戳.
   3. 对将要更新的数据进行提前读取、事后对比。

不论是数据库系统本身的锁机制，还是乐观锁这种业务数据级别上的锁机制，本质上都是对状态位的读、写、判断。

<https://www.cnblogs.com/zhouqianhua/archive/2011/04/15/2017049.html>