**[浅谈SQL Server事务与锁（上篇）](https://www.cnblogs.com/wangjiming/p/8396986.html)**

**一  概述**

在数据库方面，对于非DBA的程序员来说，事务与锁是一大难点，针对该难点，本篇文章试图采用图文的方式来与大家一起探讨。

“浅谈SQL Server 事务与锁”这个专题共分两篇，上篇主讲事务及事务一致性问题，并简略的提及一下锁的种类和锁的控制级别。

下篇主讲SQL Server中的锁机制，锁控制级别和死锁的若干问题。

**二   事务**

**1   何为事务**

 预览众多书籍，对于事务的定义，不同文献不同作者对其虽有细微差别却大致统一，我们将其抽象概括为：

事务：指封装且执行单个或多个操作的单个工作单元，在SqlServer中，其定义表现为显示定义和隐式定义两种方式。

 基于如上的定义，我们可以将事务解剖拆分为如下几个点：

(1)事务是单个工作单元，这一定义，才使事务具有ACID属性

(2)事务是封装操作的，如封装基本的CRUD操作

[复制代码](javascript:void(0);)

1 --事务

2 Begin Tran

3 SELECT \* FROM UserInfo

4 INSERT INTO UserInfo VALUES('Alan\_beijing',**35**)

5 UPDATE UserInfo SET Age=**31** WHERE UserName='Alan\_beijing'

6 DELETE UserInfo WHERE UserName='Alan\_beijing'

7 Commit Tran

[复制代码](javascript:void(0);)

(3)事务在封装操作时，可以封装单个操作，也可以封装多个操作(封装多个操作时，应注意与批处理的区别)

(4)在SqlServer中，事务的定义分为显示定义和隐式定义两种方式

 显示定义：以Begin Tran作为开始，其中提交事务为Commit Tran，回滚事务为RollBack Tran，如我们在一个事务中插入两条操作语句

1 --显示定义事务

2 Begin Tran

3 INSERT INTO UserInfo VALUES('Alan\_shanghai',**30**)

4 INSERT INTO UserInfo VALUES('Alan\_beijing',**35**)

5 Commit Tran

 隐式定义：如果不显示定义事务，SQL Server 默认把每个语句当作一个事务来处理(执行完每个语句之后就自动提交事务)

**2   事务的ACID属性**

 事务作为单个工作单元，该定义使其具有ACID属性，ACID属性指原子性(Atomicity)、一致性(Consisitency)、隔离性(Isolation)和持久性(Durability)。

(1)原子性(Atomicity)

原子性指事务必须是原子工作单元，即对于事务的封装操作，要么全部执行，要么全都不执行。如下情况均会导致事务的撤销或回滚。。。

a.事务提交之前，系统发生故障或重新启动，SQL Server将会撤销在事务中进行的所有操作；

b.事务处理中遇到错误，SQL Server通常会自动回滚事务，但也有少数例外；

c.一些不太严重的错误不会引发事务的自动回滚，如主键冲突，锁超时等；

d.可以使用错误处理代码来捕获一些错误，并采取相应的操作，如把错误记录在日志中，再回滚事务等；

(2)一致性(Consisitency)

一致性主要指数据一致性，即主要对象是数据。从宏观上来说，指某一段时间区间，数据要保持一致性状态，从微观上来说，某个时间点数据要保持一致性状态，我们举个例子，

假若有两个事务A和B对同一张表进行操作，A向表中写数据，B向数据表中读取数据，可以猜测，B读取的数据大致有三种粗粒度可能：

第一种可能：A还没向数据表中写入数据的状态;

第二种可能：A已向数据表中写入部分数据，但还未写完的状态；

第三种可能：A已向数据表中写完数据；

如此，造成了事务的不一致性。

关于事务一致性，可能会发生 丢失更新，脏读，不可重复读和幻读等问题，下文会详细论述这些事务一致性问题。

(3)隔离性(Isolation)

隔离性指当两个及其以上事务对同一边界资源进行操作时，要控制好每个事务的边界，控制好数据访问机制，确保事务只能访问处于期望的一致性级别下的数据。

在SQL Server中，一般采用锁机制来控制，下文中，我们会详细论述。

(4)持久性(Durability)

 我们对数据表进行操作时，一般会按照先后顺序执行如下两步：

第一步：将对数据表操作写入到磁盘上数据库的事务日志中(持久还到磁盘事务日志中)；

第二步：完成第一步后，再将对数据表操作写入到磁盘上数据库的数据分区中(持久化到磁盘上数据库分区中)；

关于如上两步，我们来想想可能发生的问题：

问题1：完成如上第一步之前，系统发生故障(如系统异常，系统重启)，数据库引擎会怎么做？

 由于未完成第一步，提交指令还未记录到磁盘的事务日志中，此时事务并未持久化，系统发生故障后，SQL Server

会检查每个数据库的事务日志，进行恢复处理(恢复处理一般分为重做阶段和撤销阶段)，此时的恢复处理为重做阶段，即提交指令还未记录到磁盘的事务日志中，

数据库引擎会撤销这些事务所做的所有修改，这个过程也成为回滚。

问题2：完成如上第一步但还未完成第二步，系统发生故障(如系统异常，系统重启)，数据库引擎会怎么做？

 完成第一步后，提交指令已记录到磁盘的事务日志中，无论数据操作是否被写入到磁盘的数据分区，此时事务已持久化，系统发生故障后，SQL Server

会检查每个数据库的事务日志，进行恢复处理(恢复处理一般分为重做阶段和撤销阶段)，此时的恢复处理为重做阶段，即由于数据修改还没有运用到数据分区的事务，

数据库引擎会重做这些事务所做的所有修改，这个过程也成为前滚。

**三   事务的隔离级别和隔离级别产生的一致性问题**

**1   未提交读(READ UNCOMMITTED)**

 未提交读(READ UNCOMMITTED)指读取未提交的数据，此时产生的数据不一致性，我们称为数据脏读。

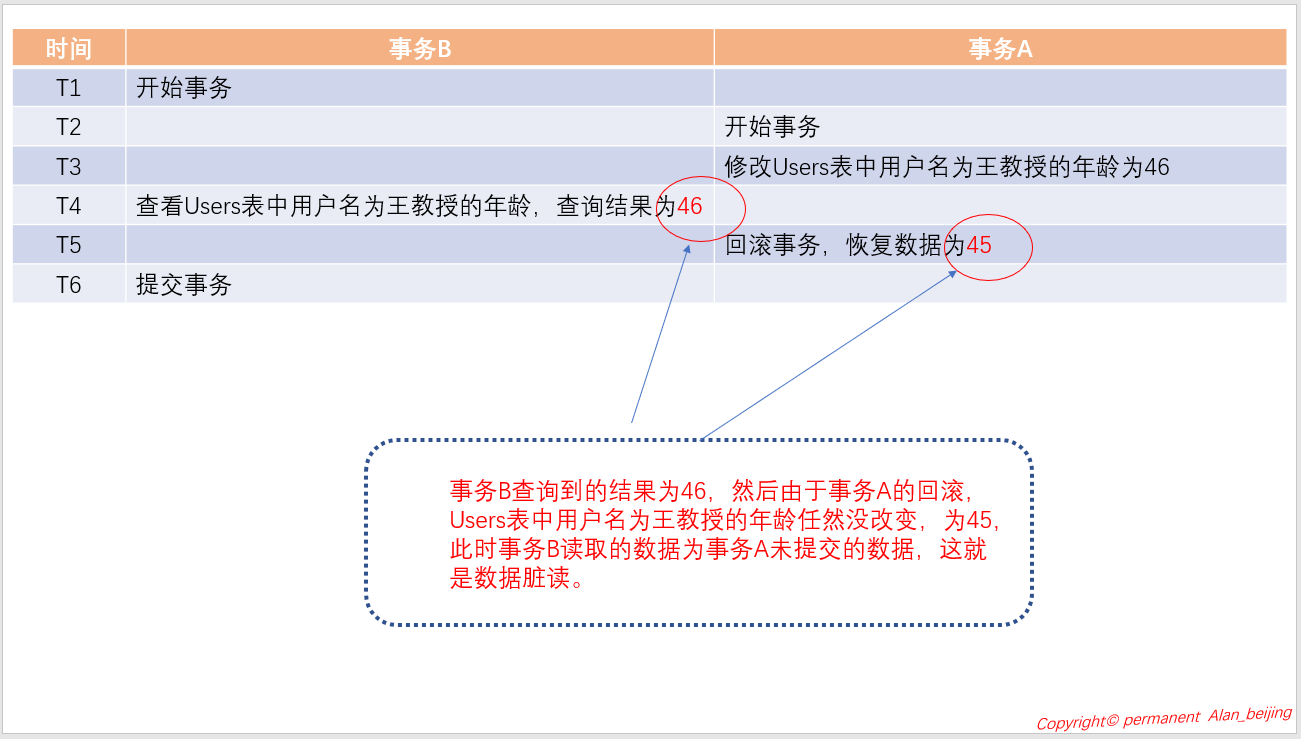
**1.1   未提交读为什么会产生数据脏读**

未提交读是最低级的隔离级别，在这个隔离级别运行的事务，读操作是不需要请求共享锁的，如果读操作不需要共享锁，就不会产生与持有排它锁的事务操作发生冲突，

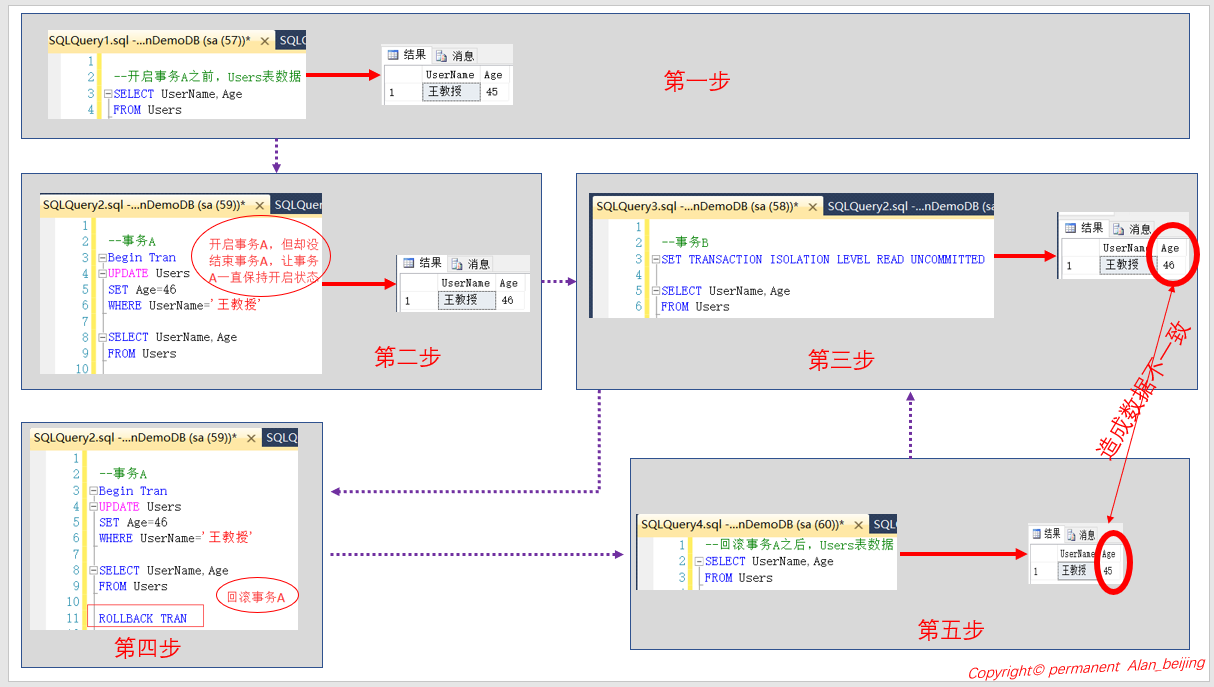
那么也就是说，在这个事务隔离级别，读操作可以与写操作同时进行，互不排斥，读操作可以读取写操作未提交的修改，从而造成数据的不一致性，这种情况，我们称

数据脏读。

**1.2   图解数据脏读**



**1.3   SQL演示数据脏读**



**2   已提交读(READ COMMITTED)**

 已提交读(READ COMMITTED)指只能读取已提交事务的数据，是防止数据脏读的最低隔离级别，也是SQL Server默认的隔离级别，它要求读操作必须获得共享锁后

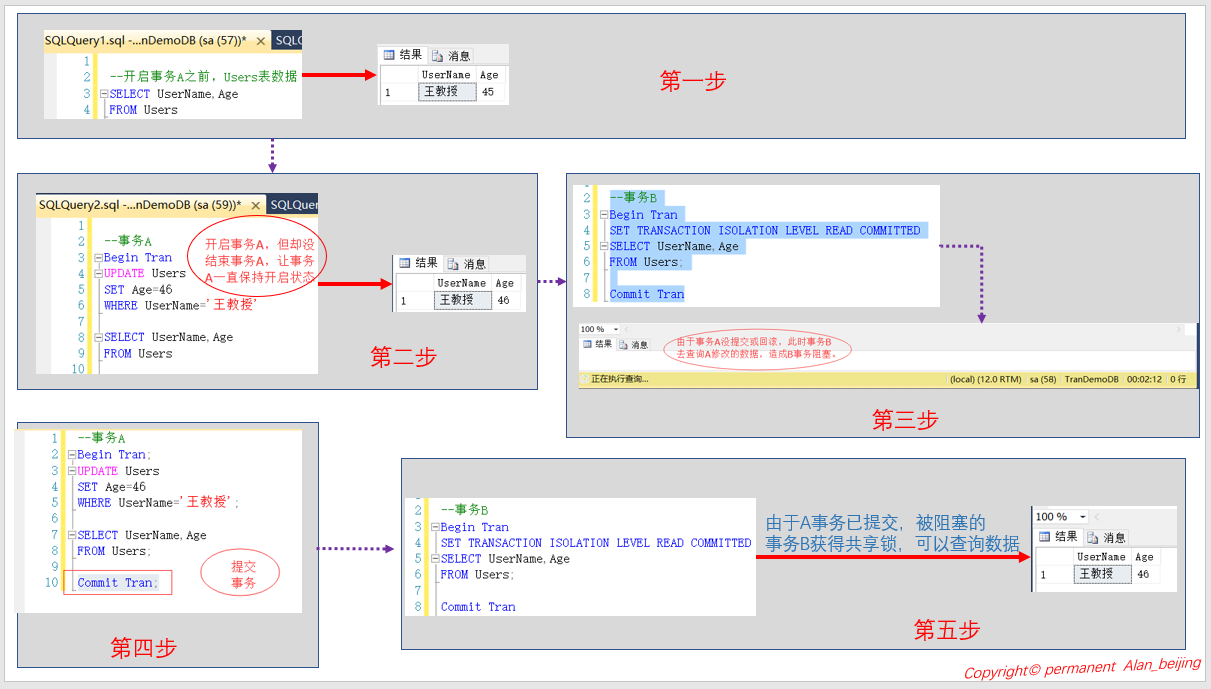
才能进行操作，防止读取未提交的修改，虽然已提交读能防止产生数据脏读，但却不可避免不可重复读数据一致性问题。

**2.1   为什么已提交读能够防止数据脏读**

已提交读只允许读取事务已提交的数据，它要求读操作必须获得共享锁才能尽心操作，而读操作的共享锁与写操作的排他锁是互斥的，两者互斥会发生冲突，所以读操作

在读取数据时，必须等待写操作完成后，才能获取共享锁，然后才能读取数据，此时读取的数据是已经提交结束的数据，因此就防止了数据脏读的问题。

**2.2   SQL演示已提交读**



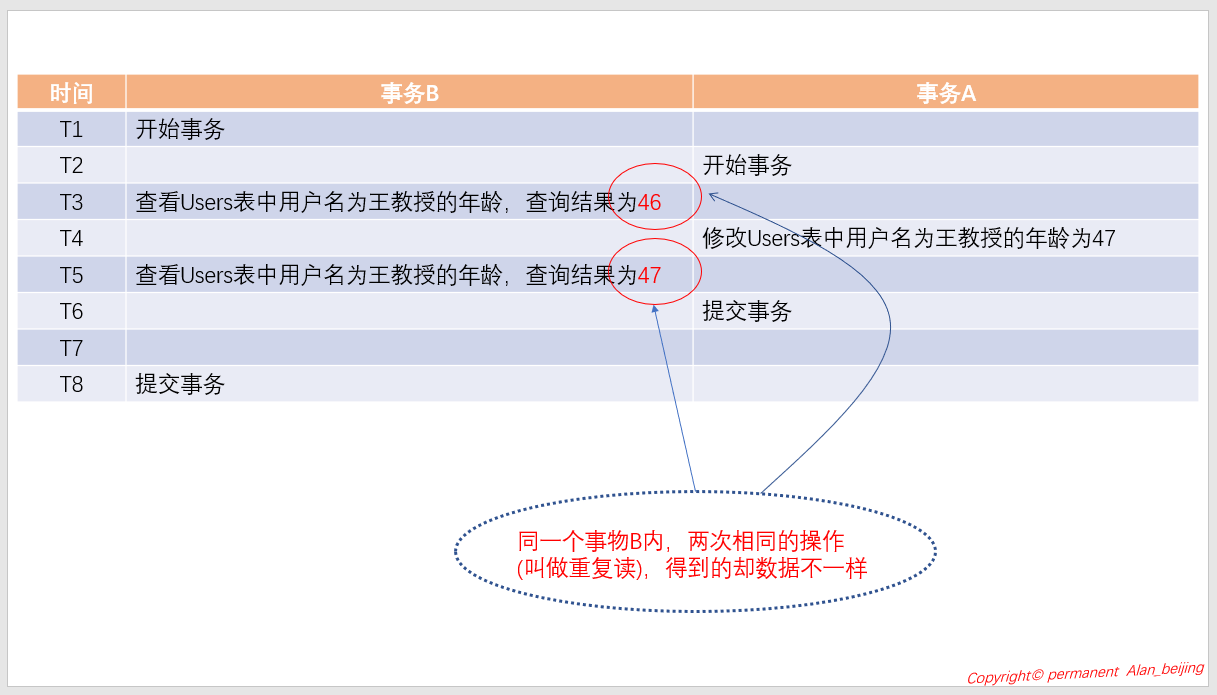
**2.3   为什么已提交读会产生不可重复读问题**

我们知道，虽然已提交读能获得共享锁，然而，读操作一完成，就会立即释放资源上的共享锁(该操作不会在事务持续期间一致保留共享锁)，如此就会产生一个问题，

即在一个事务处理内部对相同数据资源读操作之间，没有共享锁会锁定该资源，导致其他事务可以在两个读操作之间更改数据资源，读操作因而可能每次得到不同的

取值，这种现象称为数据的不可重复读。

**2.4   图解不可重复读**



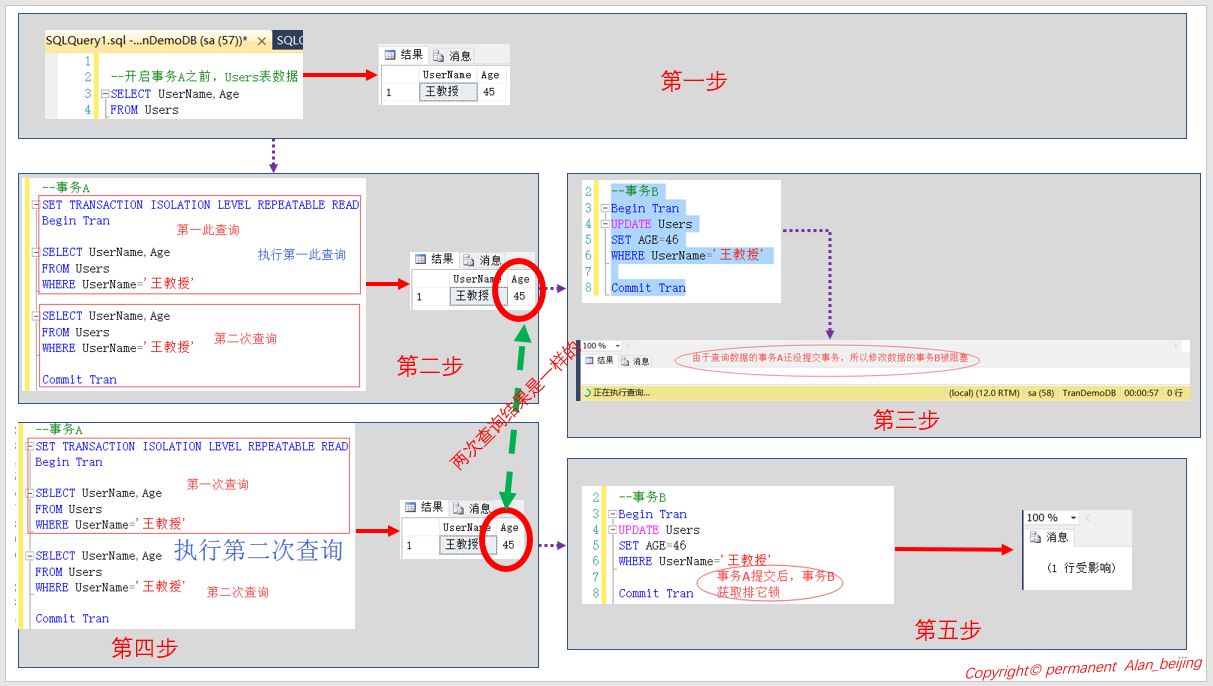
**3  可重复读(REPEATABLE READ)**

 为了防止不可重复读现象，SQL Sever中采用隔离级别升级的方式，即将已提交读升级为可重复读。在可重复读隔离级别下，事务中的读操作不仅能获得共享锁，

而且获得的共享锁一直保持到事务完成为止， 在该事务完成之前，其他事务不可能获得排他锁来修改这一数据，如此，便实现了可重复读，防止了不可重复读造

成的数据不一致性。可重复读不仅能解决不可重复读数据不一致性问题，还能解决丢失更新问题。然而，可重复读也存在问题，那就是死锁和幻读等问题。

**3.1 SQL演示可重复读**

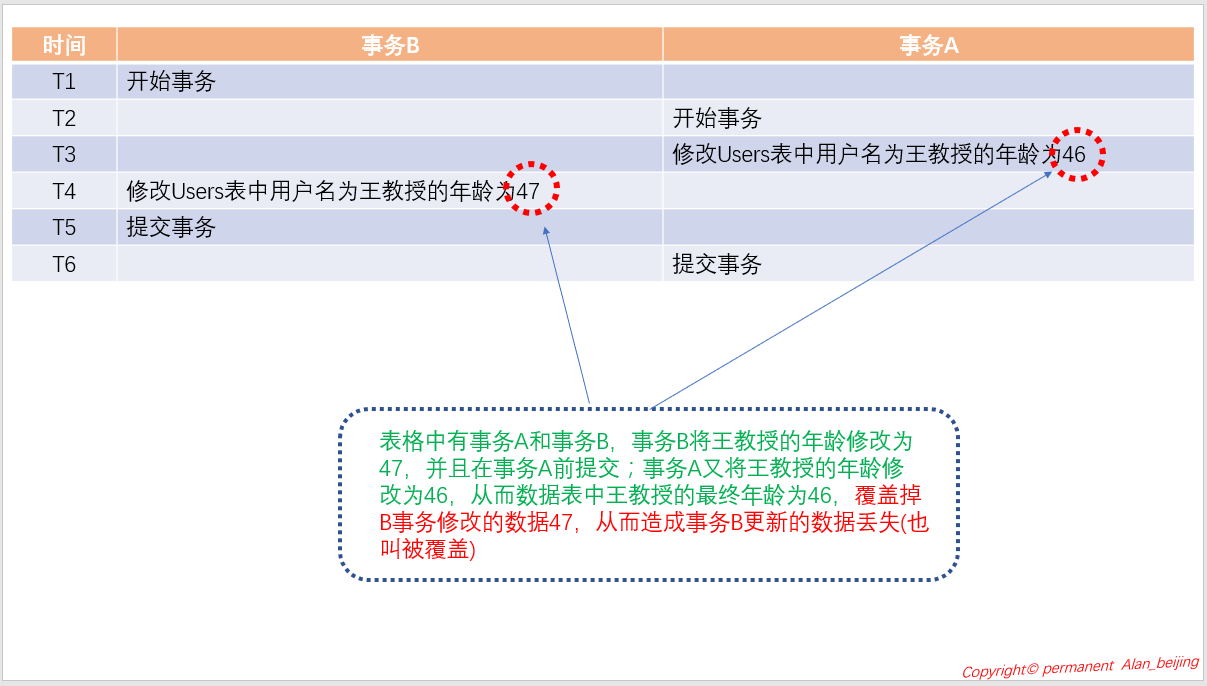


**3.2  何为丢失更新？**

在比可重复读低的隔离级别中，两个事务在读取数据之后就不再持有该资源的任何锁，此时，两个事务都能更新这个值，

从而发生最后事务更新的值覆盖前面事务更新的值，从而造成数据的丢失，这称为丢失更新。

**3.3  图解丢失更新**



**4   可序列化(SERIALIZABLE)**

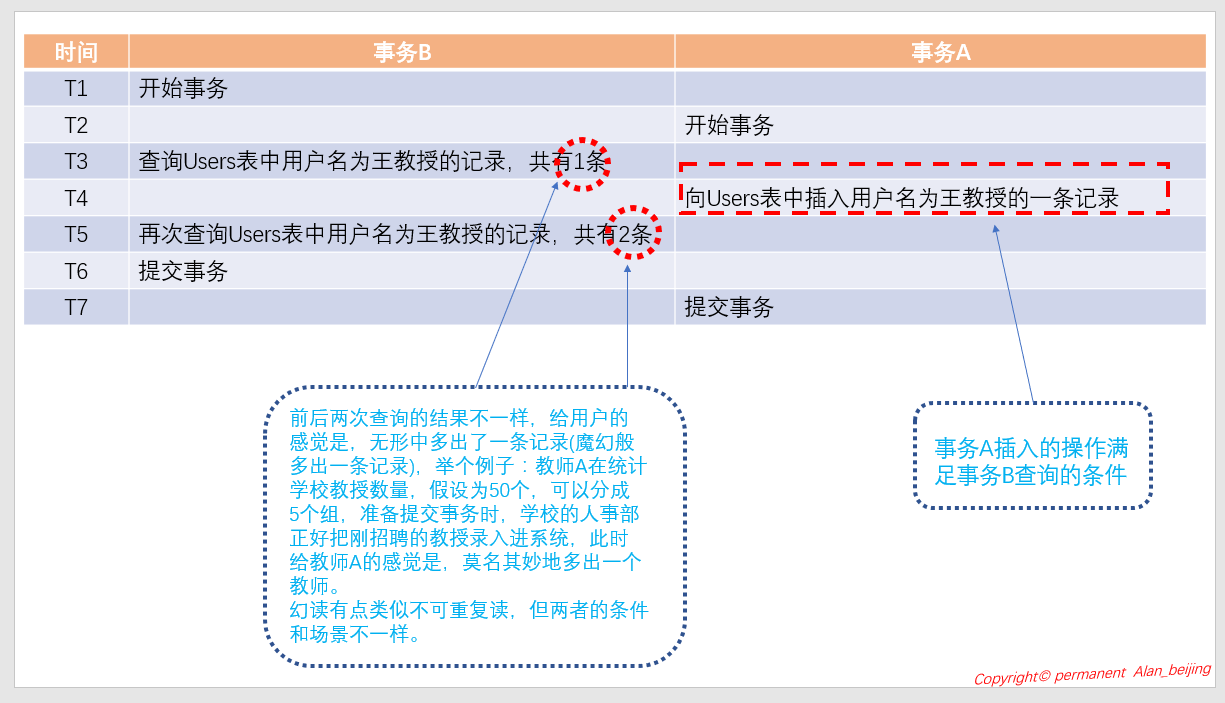
**4.1   何为幻读？**

我们知道，在可重复读隔离级别下，读事务持有的共享锁一直保持到该事务完成为止，但是事务只锁定查询第一次运行时找到的那些数据资源(如，行)，

而不会锁定查询结果范围以外的其他行(其实，控制事务时，有数据库架构级别，表，页和行等)。因此，在同一事务中进行第二次读取之前，若其他事

务插入新行，并且新行能满足读操作的查询过滤条件，那么这些新行也会出现在第二次读操作返回的结果中，这些新行称为幻影子，也叫做幻读。

**4.2  图解幻读**



**4.3  如何解决幻读？**

SQL SERVER中，更高级别的可序列化(SERIALIZABLE)能够解决该问题。

**4.4  何为可序列化(SERIALIZABLE)？**

大多数时候，可序列化(SERIALIZABLE)隔离级别的处理方式和可重复都得处理方式是类似的，只不过，可序列化(SERIALIZABLE)隔离级别

增加了一个新的内容——逻辑上，这个隔离级别会让读操作锁定满足查询搜索条件的键的整范围，这就意味着读操作不仅锁定了满足查询搜索

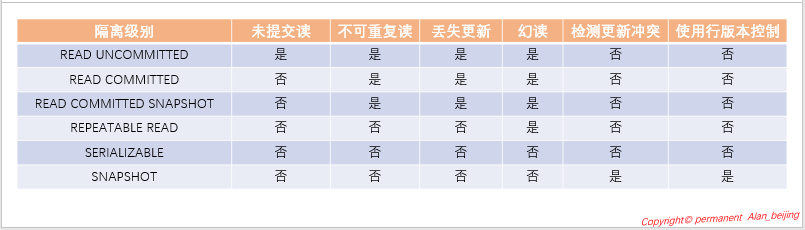
条件的现有的那些行，还锁定了未来可能满足查询搜索条件的行。

**5   SNAPSHOT**

 略。

**四    事务的隔离级别总结**

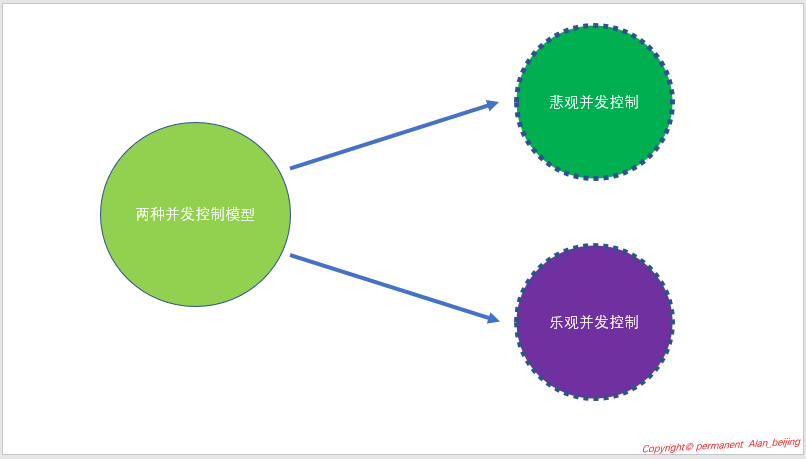
下表总结了每种隔离级别与逻辑一致性问题，检测冲突和行版本控制之间关系



**五   锁定**

**1  两种并发控制模型**

关于并发控制模型，主要有两种，即悲观控制模型和乐观控制模型。



(1)悲观控制模型： 该模型假设总是存在多个事务对同一资源操作(读/写)，即假定冲突总是会发生。在SQL Server中，采用事务

隔离级别来控制(也可叫做采用锁来控制)。一般在事务发生冲突前进行控制，也叫事前控制；

(2)乐观控制模型：该模型与悲观控制模型是对立的，即该模型总是假设系统中并不存在或较少存在多个事务对同一资源操作(读/写)

，即假定冲突是不会发生的或很少发生的。在SQL Server中，采用行版本控制来处理。一般在事务发生冲突后进行控制，也叫事后

控制；

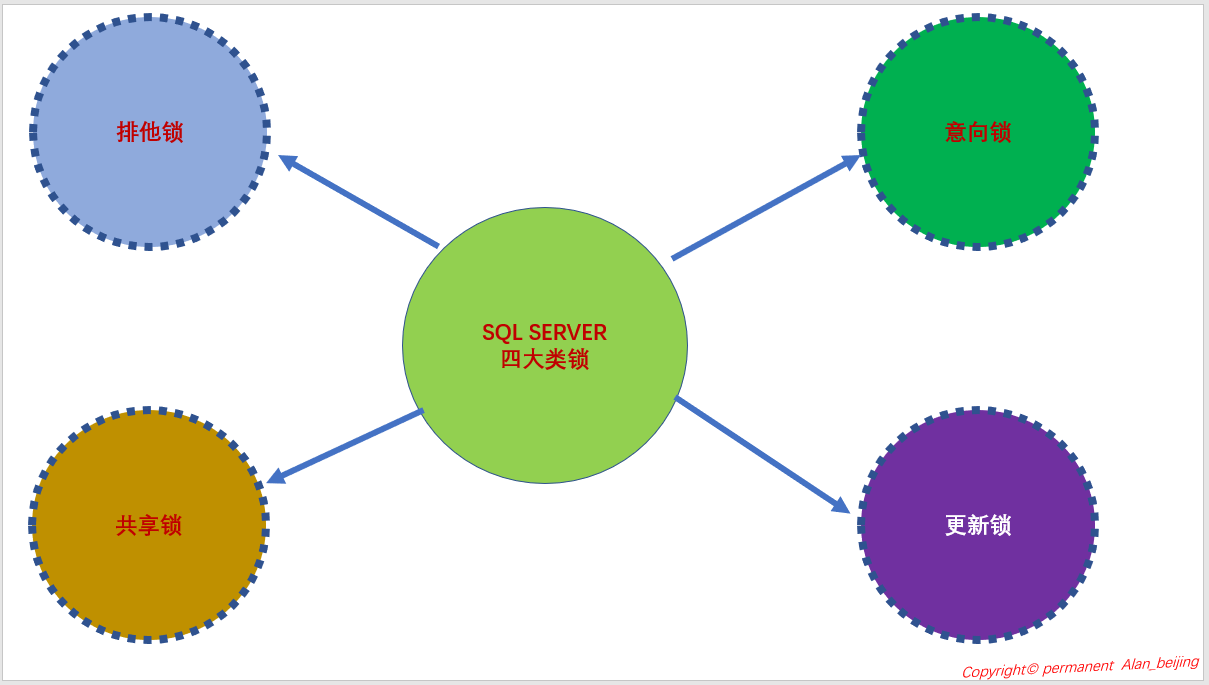
**2 何为锁定及锁定的种类**

**2.1  何为锁定**

锁定，指在并发操作时，确保数据的一致性所采用的一种手段。在SQL Server中，采用锁机制与事务隔离级别来控制数据的一致性，

**2.2 锁定的种类**

常用的四大类锁包括：共享锁，意向锁，更新锁和排他锁。



(1)共享锁：在SQL SERVER中，当事务要读取数据时，需要获取共享锁。

(2)意向锁：在SQL SERVER中，准确来说，意向锁并不是一种独立的锁，其主要作用在于获取锁的控制粒度(如，页，表，行等)。

(3)更新锁：在SQL SERVER中，准确来说，更新锁并不是一种独立的锁，而是由共享锁和排它锁组成的混合锁，其隔离级别高于共享锁，

低于排他锁，更新锁能够预防锁升级而产生的死锁。

(4)排它锁：在SQL SERVER中，当事务要写数据、更细数据和删除数据时，需要获取排他锁。

**3 锁的控制粒度**

 在SQL SERVER中，锁可以控制表，页和行等资源。

