* --ASP丶NET

**[Mysql(三)------事务的特性、事务并发、事务读一致性问题](https://www.cnblogs.com/flgb/p/12154348.html)**

Posted on 2020-01-05 23:31  [FLGB](https://www.cnblogs.com/flgb/)  阅读(9)  评论(0)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=12154348)  [收藏](javascript:void(0))

# 1 什么是数据库的事务？

1.1 事务的典型场景

在项目里面，什么地方会开启事务，或者配置了事务？无论是在方法上加注解，还

是配置切面

[复制代码](javascript:void(0);)

<tx:advice id="txAdvice" transaction-manager="transactionManager">

　　<tx:attributes> <tx:method name="save\*" rollback-for="Throwable" />

　　<tx:method name="add\*" rollback-for="Throwable" />

　　<tx:method name="send\*" rollback-for="Throwable" />

　　<tx:method name="insert\*" rollback-for="Throwable" />

　　</tx:attributes>

</tx:advice>

[复制代码](javascript:void(0);)

比如下单，会操作订单表，资金表，物流表等等，这个时候我们需要让这些操作都

在一个事务里面完成。当一个业务流程涉及多个表的操作的时候，我们希望它们要么是

全部成功的，要么都不成功，这个时候我们会启用事务。

在金融的系统里面事务配置是很常见的，比如行内转账的这种操作，如果我们把它

简单地理解为一个账户的余额增加，另一个账户的余额减少的情况（当然实际上要比这

复杂），那么这两个动作一定是同时成功或者同时失败的，否则就会造成银行的会计科

目不平衡。

1.2 事务的定义

什么是事务？

维基百科的定义：事务是数据库管理系统（DBMS）执行过程中的一个逻辑单位，由

一个有限的数据库操作序列构成。

这里面有两个关键点，第一个，它是数据库最小的工作单元，是不可以再分的。第

二个，它可能包含了一个或者一系列的 DML 语句，包括 insert delete update。

（单条 DDL（create drop）和 DCL（grant revoke）也会有事务）

1.3 哪些存储引擎支持事务

在我们第一天的课里面说到了，InnoDB 支持事务，这个也是它成为默认的存储引擎

的一个重要原因：

https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

另一个是 NDB。

1.4 事务的四大特性

事务的四大特性：ACID。

第一个，原子性，Atomicity，也就是我们刚才说的不可再分，也就意味着我们对数

据库的一系列的操作，要么都是成功，要么都是失败，不可能出现部分成功或者部分失

败的情况。以转账的场景为例，一个账户的余额减少，对应一个账户的增加，这两个一

定是同时成功或者同时失败的。

让它全部失败呢？这个时候我们必须要回滚。

原子性，在 InnoDB 里面是通过 undo log 来实现的，它记录了数据修改之前的值（逻

辑日志），一旦发生异常，就可以用 undo log 来实现回滚操作。

第二个，一致性，consistent，指的是数据库的完整性约束没有被破坏，事务执行的前后都是合法的数据状态。比如主键必须是唯一的，字段长度符合要求。

除了数据库自身的完整性约束，还有一个是用户自定义的完整性。

比如说转账的这个场景，A 账户余额减少 1000，B 账户余额只增加了 500，这个时

候因为两个操作都成功了，按照我们对原子性的定义，它是满足原子性的， 但是它没有

满足一致性，因为它导致了会计科目的不平衡。

还有一种情况，A 账户余额为 0，如果这个时候转账成功了，A 账户的余额会变成

-1000，虽然它满足了原子性的，但是我们知道，借记卡的余额是不能够小于 0 的，所以

也违反了一致性。用户自定义的完整性通常要在代码中控制。

第三个，隔离性，Isolation，我们有了事务的定义以后，在数据库里面会有很多的

事务同时去操作我们的同一张表或者同一行数据，必然会产生一些并发或者干扰的操作，

那么我们对隔离性的定义，就是这些很多个的事务，对表或者行的并发操作，应该是透

明的，互相不干扰的。通过这种方式，我们最终也是保证业务数据的一致性。

最后一个叫做持久性，Durable，事务的持久性是什么意思呢？我们对数据库的任意

的操作，增删改，只要事务提交成功，那么结果就是永久性的，不可能因为我们系统宕

机或者重启了数据库的服务器，它又恢复到原来的状态了。这个就是事务的持久性。

持久性怎么实现呢？数据库崩溃恢复（crash-safe）是通过什么实现的？

持久性是通过 redo log 和 double write 双写缓冲来实现的，我们操作数据的时候，

会先写到内存的 buffer pool 里面，同时记录 redo log，如果在刷盘之前出现异常，在

重启后就可以读取 redo log 的内容，写入到磁盘，保证数据的持久性。

当然，恢复成功的前提是数据页本身没有被破坏，是完整的，这个通过双写缓冲

（double write）保证。

原子性，隔离性，持久性，最后都是为了实现一致性。

1.5 数据库什么时候会出现事务

无论是我们在 Navicat 的这种工具里面去操作，还是在我们的 Java 代码里面通过

API 去操作，还是加上@Transactional 的注解或者 AOP 配置，其实最终都是发送一个

指令到数据库去执行，Java 的 JDBC 只不过是把这些命令封装起来了。

我们先来看一下我们的操作环境。版本（5.7），存储引擎（InnnoDB），事务隔离

级别（RR）。

select version();

show variables like '%engine%';

show global variables like "tx\_isolation";

执行这样一条更新语句的时候，它有事务吗？

update student set sname = '猫老公 111' where id=1;

实际上，它自动开启了一个事务，并且提交了，所以最终写入了磁盘。

这个是开启事务的第一种方式，自动开启和自动提交。

InnoDB 里面有一个 autocommit 的参数（分成两个级别， session 级别和 global

级别）。

show variables like 'autocommit';

它的默认值是 ON。autocommit 这个参数是什么意思呢？是否自动提交。如果它的

值是 true/on 的话，我们在操作数据的时候，会自动开启一个事务，和自动提交事务。

否则，如果我们把 autocommit 设置成 false/off，那么数据库的事务就需要我们手

动地去开启和手动地去结束。

手动开启事务也有几种方式，一种是用 begin；一种是用 start transaction。

那么怎么结束一个事务呢？我们结束也有两种方式，第一种就是提交一个事务，

commit；还有一种就是 rollback，回滚的时候，事务也会结束。还有一种情况，客户端

的连接断开的时候，事务也会结束。

后面我们会讲到，当我们结束一个事务的时候，事务持有的锁就会被释放，无论是

提交还是回滚。

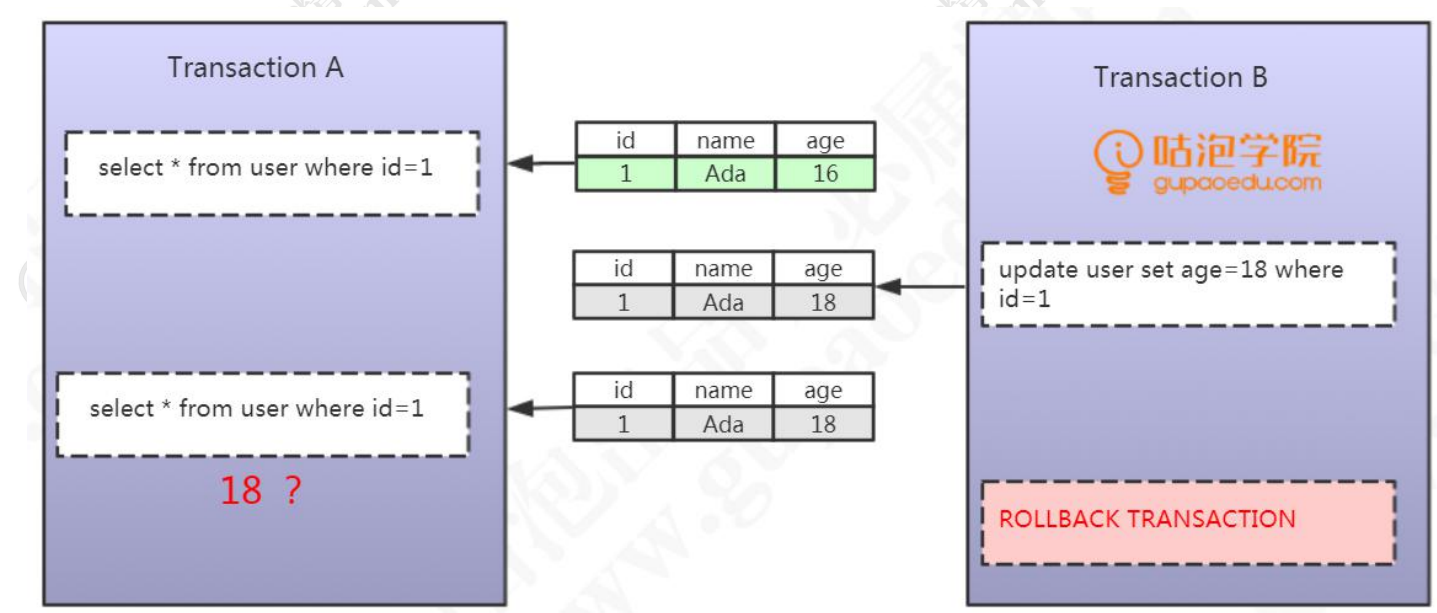
我们用 begin 手工开启一个事务，执行第二个 update，但是数据没有写入磁盘，因为事务还没有提交，这个时候 commit 一下，

再刷新一下，OK，写入了。这个就是我们开启和结束事务的两种方式。

1.6 事务并发会带来什么问题？

当很多事务并发地去操作数据库的表或者行的时候，如果没有我们刚才讲的事务的

Isolation 隔离性的时候，会带来哪些问题呢？



我们有两个事务，一个是 Transaction A，一个是 Transaction B，在第一个事务里

面，它首先通过一个 where id=1 的条件查询一条数据，返回 name=Ada，age=16 的

这条数据。然后第二个事务，它同样地是去操作 id=1 的这行数据，它通过一个 update

的语句，把这行 id=1 的数据的 age 改成了 18，但是注意，它没有提交。

这个时候，在第一个事务里面，它再次去执行相同的查询操作，发现数据发生了变

化，获取到的数据 age 变成了 18。那么，这种在一个事务里面，由于其他的时候修改了

数据并且没有提交，而导致了前后两次读取数据不一致的情况，这种事务并发的问题，

我们把它定义成什么？

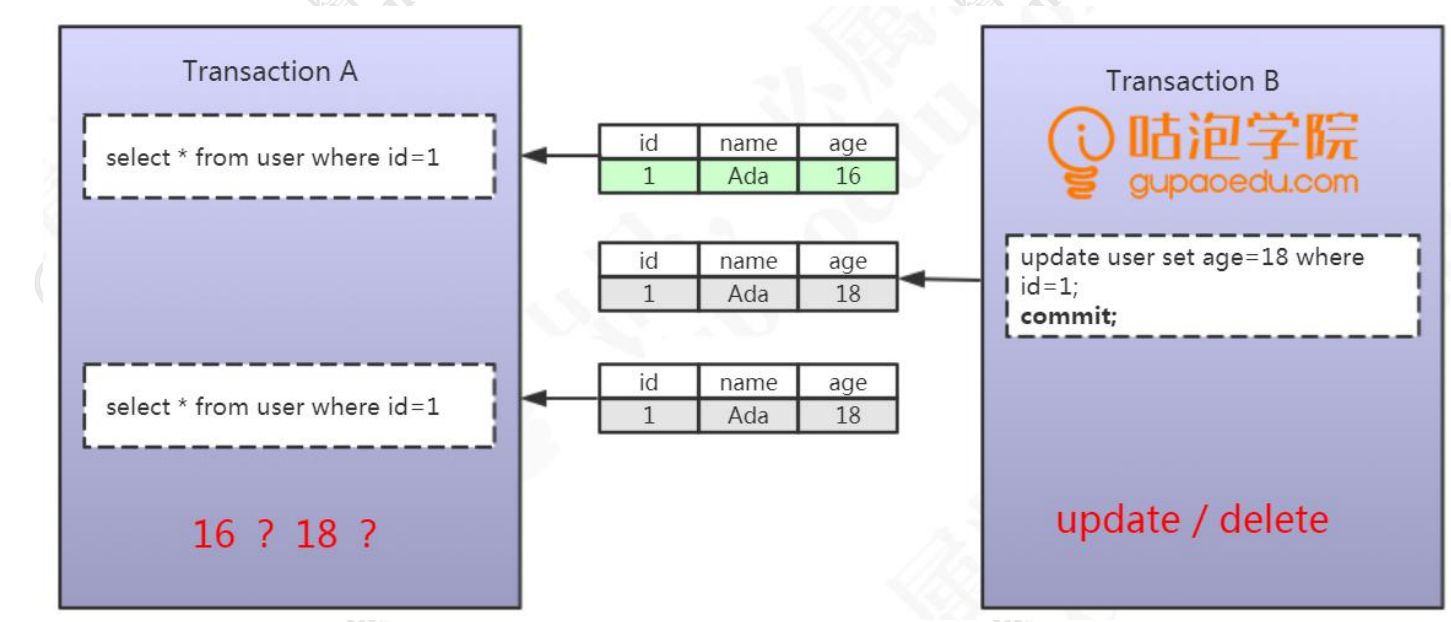
这个叫做脏读。

如果在转账的案例里面，我们第一个事务基于读取到的第二个事务未提交的余额进

行了操作，但是第二个事务进行了回滚，这个时候就会导致数据不一致。

这种读取到其他事务未提交的数据的情况，我们把它叫做脏读。

我们再来看第二个。



同样是两个事务，第一个事务通过 id=1 查询到了一条数据。然后在第二个事务里面

执行了一个 update 操作，这里大家注意一下，执行了 update 以后它通过一个 commit

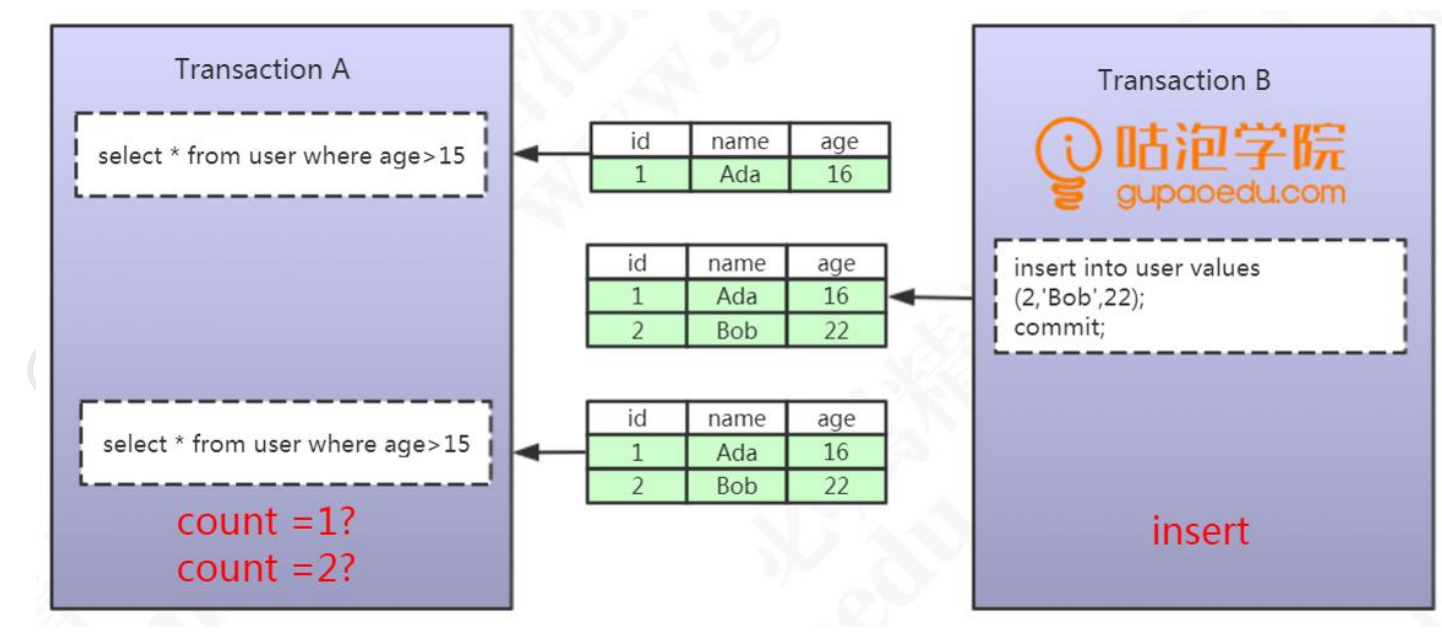
提交了修改。然后第一个事务读取到了其他事务已提交的数据导致前后两次读取数据不

一致的情况，就像这里，age 到底是等于 16 还是 18，那么这种事务并发带来的问题，

我们把它叫做什么？

这种一个事务读取到了其他事务已提交的数据导致前后两次读取数据不一致的情

况，我们把它叫做不可重复读。



在第一个事务里面我们执行了一个范围查询，这个时候满足条件的数据只有一条。

在第二个事务里面，它插入了一行数据，并且提交了。重点：插入了一行数据。在第一

个事务里面再去查询的时候，它发现多了一行数据。这种情况，我们把它叫做什么呢？

一个事务前后两次读取数据数据不一致，是由于其他事务插入数据造成的，这种情

况我们把它叫做幻读。

不可重复读和幻读，的区别在那里呢？

不可重复读是修改或者删除，幻读是插入。

小结：我们刚才讲了事务并发带来的三大问题，现在来给大家总结一下。无论是脏

读，还是不可重复读，还是幻读，它们都是数据库的读一致性的问题，都是在一个事务

里面前后两次读取出现了不一致的情况。

读一致性的问题，必须要由数据库提供一定的事务隔离机制来解决。就像我们去饭

店吃饭，基本的设施和卫生保证都是饭店提供的。那么我们使用数据库，隔离性的问题

也必须由数据库帮助我们来解决。

1.7 SQL92 标准

所以，就有很多的数据库专家联合制定了一个标准，也就是说建议数据库厂商都按

照这个标准，提供一定的事务隔离级别，来解决事务并发的问题，这个就是 SQL92 标准。

我们来看一下 SQL92 标准的官网。

http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

这里面有一张表格（搜索\_iso），里面定义了四个隔离级别，右边的 P1 P2 P3 就是

代表事务并发的 3 个问题，脏读，不可重复读，幻读。Possible 代表在这个隔离级别下，

这个问题有可能发生，换句话说，没有解决这个问题。Not Possible 就是解决了这个问

题。

我们详细地分析一下这 4 个隔离级别是怎么定义的。

第一个隔离级别叫做：Read Uncommitted（未提交读），一个事务可以读取到其

他事务未提交的数据，会出现脏读，所以叫做 RU，它没有解决任何的问题。

第二个隔离级别叫做：Read Committed（已提交读），也就是一个事务只能读取

到其他事务已提交的数据，不能读取到其他事务未提交的数据，它解决了脏读的问题，

但是会出现不可重复读的问题。

第三个隔离级别叫做：Repeatable Read (可重复读)，它解决了不可重复读的问题，

也就是在同一个事务里面多次读取同样的数据结果是一样的，但是在这个级别下，没有

定义解决幻读的问题。

最后一个就是：Serializable（串行化），在这个隔离级别里面，所有的事务都是串

行执行的，也就是对数据的操作需要排队，已经不存在事务的并发操作了，所以它解决

了所有的问题。

这个是 SQL92 的标准，但是不同的数据库厂商或者存储引擎的实现有一定的差异，

比如 Oracle 里面就只有两种 RC（已提交读）和 Serializable（串行化）。那么 InnoDB

的实现又是怎么样的呢？

1.8 MySQL InnoDB 对隔离级别的支持

在 MySQL InnoDB 里面，不需要使用串行化的隔离级别去解决所有问题。那我们来

看一下 MySQL InnoDB 里面对数据库事务隔离级别的支持程度是什么样的。



InnoDB 支持的四个隔离级别和 SQL92 定义的基本一致，隔离级别越高，事务的并

发度就越低。唯一的区别就在于，InnoDB 在 RR 的级别就解决了幻读的问题。这个也是

InnoDB 默认使用 RR 作为事务隔离级别的原因，既保证了数据的一致性，又支持较高的

并发度。

1.9 两大实现方案

那么大家想一下，如果要解决读一致性的问题，保证一个事务中前后两次读取数据

结果一致，实现事务隔离，应该怎么做？我们有哪一些方法呢？你的思路是什么样的呢？

总体上来说，我们有两大类的方案。

1.9.1 LBCC

第一种，我既然要保证前后两次读取数据一致，那么我读取数据的时候，锁定我要

操作的数据，不允许其他的事务修改就行了。这种方案我们叫做基于锁的并发控制 Lock

Based Concurrency Control（LBCC）。

如果仅仅是基于锁来实现事务隔离，一个事务读取的时候不允许其他时候修改，那

就意味着不支持并发的读写操作，而我们的大多数应用都是读多写少的，这样会极大地

影响操作数据的效率。

1.9.2 MVCC

所以我们还有另一种解决方案，如果要让一个事务前后两次读取的数据保持一致，

那么我们可以在修改数据的时候给它建立一个备份或者叫快照，后面再来读取这个快照

就行了。这种方案我们叫做多版本的并发控制 Multi Version Concurrency Control

（MVCC）。

MVCC 的核心思想是： 我可以查到在我这个事务开始之前已经存在的数据，即使它

在后面被修改或者删除了。在我这个事务之后新增的数据，我是查不到的。

问题：这个快照什么时候创建？读取数据的时候，怎么保证能读取到这个快照而不

是最新的数据？这个怎么实现呢？

InnoDB 为每行记录都实现了两个隐藏字段：

DB\_TRX\_ID，6 字节：插入或更新行的最后一个事务的事务 ID，事务编号是自动递

增的（我们把它理解为创建版本号，在数据新增或者修改为新数据的时候，记录当前事

务 ID）。

DB\_ROLL\_PTR，7 字节：回滚指针（我们把它理解为删除版本号，数据被删除或记

录为旧数据的时候，记录当前事务 ID）。

我们把这两个事务 ID 理解为版本号。

https://www.processon.com/view/link/5d29999ee4b07917e2e09298 MVCC 演示图

第一个事务，初始化数据（检查初始数据）

Transaction 1

begin;

insert into mvcctest values(NULL,'qingshan') ;

insert into mvcctest values(NULL,'jack') ;

commit;

此时的数据，创建版本是当前事务 ID，删除版本为空：



第二个事务，执行第 1 次查询，读取到两条原始数据，这个时候事务 ID 是 2：

Transaction 2

begin;

select \* from mvcctest ; -- (1) 第一次查询

第三个事务，插入数据：

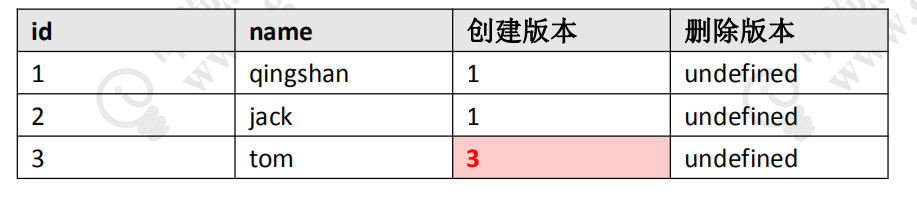
Transaction 3

begin;

insert into mvcctest values(NULL,'tom') ;

commit;

此时的数据，多了一条 tom，它的创建版本号是当前事务编号，3：



第二个事务，执行第 2 次查询：

Transaction 2

select \* from mvcctest ; (2) 第二次查询

MVCC 的查找规则：只能查找创建时间小于等于当前事务 ID 的数据，和删除时间大

于当前事务 ID 的行（或未删除）。

也就是不能查到在我的事务开始之后插入的数据，tom 的创建 ID 大于 2，所以还是

只能查到两条数据。

第四个事务，删除数据，删除了 id=2 jack 这条记录：

Transaction 4

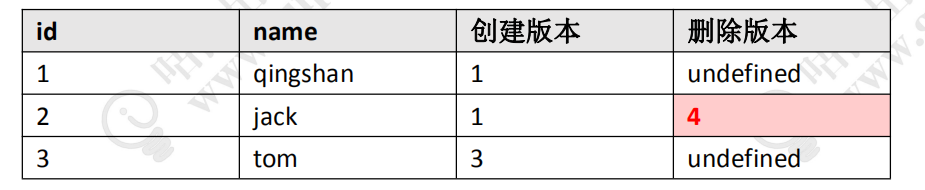
begin;咕泡出品，必属精品 www.gupaoedu.com

14

delete from mvcctest where id=2;

commit;

此时的数据，jack 的删除版本被记录为当前事务 ID，4，其他数据不变：



在第二个事务中，执行第 3 次查询：

Transaction 2

select \* from mvcctest ; (3) 第三次查询

查找规则：只能查找创建时间小于等于当前事务 ID 的数据，和删除时间大于当前事

务 ID 的行（或未删除）。

也就是，在我事务开始之后删除的数据，所以 jack 依然可以查出来。所以还是这两

条数据。

第五个事务，执行更新操作，这个事务事务 ID 是 5：

Transaction 4

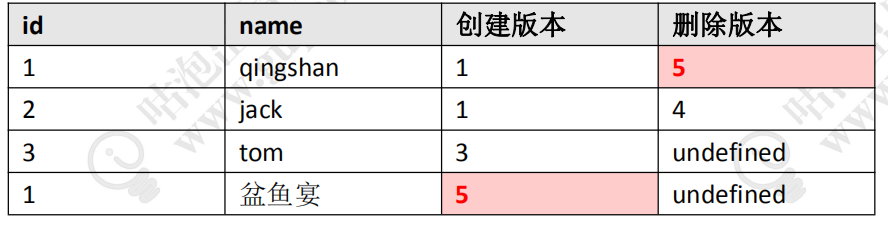
begin;

update mvcctest set name ='盆鱼宴' where id=1;

commit;

此时的数据，更新数据的时候，旧数据的删除版本被记录为当前事务 ID 5（undo），

产生了一条新数据，创建 ID 为当前事务 ID 5：



第二个事务，执行第 4 次查询：

Transaction 2咕泡出品，必属精品 www.gupaoedu.com

select \* from mvcctest ; (4) 第四次查询

查找规则：只能查找创建时间小于等于当前事务 ID 的数据，和删除时间大于当前事

务 ID 的行（或未删除）。

因为更新后的数据 penyuyan 创建版本大于 2，代表是在事务之后增加的，查不出

来。

而旧数据 qingshan 的删除版本大于 2，代表是在事务之后删除的，可以查出来。

通过以上演示我们能看到，通过版本号的控制，无论其他事务是插入、修改、删除，

第一个事务查询到的数据都没有变化。

在 InnoDB 中，MVCC 是通过 Undo log 实现的。

Oracle、Postgres 等等其他数据库都有 MVCC 的实现。

需要注意，在 InnoDB 中，MVCC 和锁是协同使用的，这两种方案并不是互斥的。

第一大类解决方案是锁，锁又是怎么实现读一致性的呢？