**MySql的隔离级别和锁的关系**

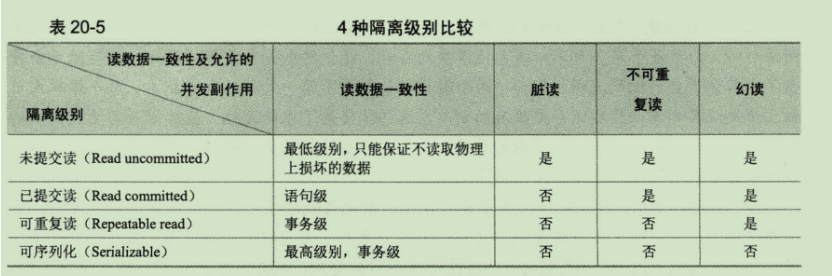
**一、事务的4个基本特征**

**Atomic（原子性）：**   
事务中包括的操作被看做一个逻辑单元。这个逻辑单元中的操作要么所有成功。要么所有失败。  
**Consistency（一致性）：**  
仅仅有合法的数据能够被写入数据库，否则事务应该将其回滚到最初状态。  
**Isolation（隔离性）：**   
事务同意多个用户对同一个数据进行并发訪问，而不破坏数据的正确性和完整性。同一时候。并行事务的改动必须与其它并行事务的改动   
相互独立。   
**Durability（持久性）：**   
事务结束后。事务处理的结果必须可以得到固化。

**二、数据库隔离级别**

数据库事务的隔离级别有4个。由低到高依次为Read uncommitted、Read committed、Repeatable read、Serializable。这四个级别能够逐个解决脏读、不可反复读、幻读这几类问题。MySql设置的隔离级别默觉得Repeatable Read。可反复读级别。

隔离级别能够配置。



注意：我们讨论隔离级别的场景，主要是在多个事务并发的情况下。因此，接下来的解说都环绕事务并发。

**1、读不提交(Read Uncommited，RU)**

  这种隔离级别下，事务间完全不隔离，会产生脏读，可以读取未提交的记录，实际情况下不会使用。

**2、读提交(Read commited，RC)**

  本事务读取到的是最新的数据（其他事务提交后的）。问题是，在同一个事务里，前后两次相同的SELECT会读到不同的结果（不重复读）

**3、可重复读（Repeatable Read，RR）【MySQL 默认的级别】**

  在同一个事务里，SELECT的结果是事务开始时时间点的状态，因此，同一个事务同样的SELECT操作读到的结果会是一致的。但是，会有幻读现象

**4、 串行化（SERIALIZABLE）**。读操作会隐式获取共享锁，可以保证不同事务间的互斥

**Read uncommitted 读未提交**

READ UNCOMMITTED是限制性最弱的隔离级别。由于该级别忽略其它事务放置的锁。使用READ UNCOMMITTED级别运行的事务，能够读取尚未由其它事务提交的改动后的数据值，这些行为称为“脏”读。我们所说的脏读，两个并发的事务，“事务A：领导给singo发工资”、“事务B：singo查询工资账户”，事务B读取了事务A尚未提交的数据。比方，事务1改动一行，事务2在事务1提交之前读取了这一行。

假设事务1回滚，事务2就读取了一行没有提交的数据。这种数据我们觉得是不存在的。

**Read committed 读提交**

该级别通过指定语句不能读取其它事务已改动可是尚未提交的数据值。禁止运行脏读。在当前事务中的各个语句运行之间，其它事务仍能够改动、插入或删除数据。从而产生无法反复的读操作。或“影子”数据。比方，事务1读取了一行。事务2改动或者删除这一行而且提交。假设事务1想再一次读取这一行，它将获得改动后的数据或者发现这一样已经被删除。因此事务的第二次读取结果与第一次读取结果不同，因此也叫不可反复读。

大多数数据库的默认级别就是Read committed。比方Sql Server , Oracle。怎样解决不可反复读这一问题。请看下一个隔离级别。

**Repeatable read 反复读**

REPEATABLE READ是比READ COMMITTED限制性更强的隔离级别。

该级别包含READ COMMITTED，而且另外指定了在当前事务提交之前。其它不论什么事务均不能够改动或删除当前事务已读取的数据。并发性低于 READ COMMITTED。由于已读数据的共享锁在整个事务期间持有，而不是在每一个语句结束时释放。

这个隔离级别仅仅是说，不可以改动和删除，可是并没有强制不能插入新的满足条件查询的数据行。

此可以得出结论：REPEATABLE READ隔离级别保证了在同样的查询条件下，同一个事务中的两个查询。第二次读取的内容肯定包换第一次读到的内容。注：Mysql的默认隔离级别就是Repeatable read。

这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。例如：

**脏读(Drity Read)：**某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

**不可重复读(Non-repeatable read):**在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

**幻读(Phantom Read):**在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

**反复读与幻读**

反复读是为了保证在一个事务中，相同查询条件下读取的数据值不发生改变，可是不能保证下次相同条件查询。结果记录数不会添加。

幻读就是为了解决问题而存在的，他将这个查询范围都加锁了。所以就不能再往这个范围内插入数据。这就是SERIALIZABLE 隔离级别做的事情。

**Serializable 序列化**

SERIALIZABLE 是限制性最强的隔离级别，由于该级别锁定整个范围的键。并一直持有锁，直到事务完毕。该级别包含REPEATABLE READ。并添加了在事务完毕之前，其它事务不能向事务已读取的范围插入新行的限制。比方，事务1读取了一系列满足搜索条件的行。事务2在运行SQL statement产生一行或者多行满足事务1搜索条件的行时会冲突。则事务2回滚。这时事务1再次读取了一系列满足同样搜索条件的行。第二次读取的结果和第一次读取的结果同样。

**三、锁**

**一次封锁or两段锁？**  
由于有大量的并发訪问，为了预防死锁。一般应用中推荐使用一次封锁法。就是在方法的開始阶段。已经预先知道会用到哪些数据，然后所有锁住，在方法执行之后，再所有解锁。

这样的方式能够有效的避免循环死锁，但在数据库中却不适用，由于在事务開始阶段，数据库并不知道会用到哪些数据。  
数据库遵循的是两段锁协议，将事务分成两个阶段，加锁阶段和解锁阶段（所以叫两段锁）

加锁阶段：在该阶段能够进行加锁操作。在对不论什么数据进行读操作之前要申请并获得S锁（共享锁，其他事务能够继续加共享锁，但不能加排它锁），在进行写操作之前要申请并获得X锁（排它锁，其他事务不能再获得不论什么锁）。加锁不成功，则事务进入等待状态，直到加锁成功才继续运行。  
解锁阶段：当事务释放了一个封锁以后，事务进入解锁阶段。在该阶段仅仅能进行解锁操作不能再进行加锁操作。  
事务                       加锁/解锁处理  
begin。   
insert into test ...... 加insert相应的锁  
update test set ...... 加update相应的锁  
delete from test ...... 加delete相应的锁  
commit; 事务提交时，同一时候释放insert、update、delete相应的锁  
这样的方式尽管无法避免死锁。可是两段锁协议能够保证事务的并发调度是串行化（串行化非常重要，尤其是在数据恢复和备份的时候）的。

不可反复读和幻读的差别  
非常多人easy搞混不可反复读和幻读，确实这两者有些相似。

但不可反复读重点在于update和delete。而幻读的重点在于insert。

假设使用锁机制来实现这两种隔离级别。在可反复读中，该sql第一次读取到数据后。就将这些数据加锁，其他事务无法改动这些数据。就能够实现可反复读了。但这样的方法却无法锁住insert的数据。所以当事务A先前读取了数据，或者改动了所有数据，事务B还是能够insert数据提交，这时事务A就会发现莫名其妙多了一条之前没有的数据。这就是幻读。不能通过行锁来避免。

须要Serializable隔离级别 。读用读锁，写用写锁，读锁和写锁相互排斥，这么做能够有效的避免幻读、不可反复读、脏读等问题，但会极大的减少数据库的并发能力。

所以说不可反复读和幻读最大的差别，就在于怎样通过锁机制来解决他们产生的问题。

上文说的，是使用悲观锁机制来处理这两种问题，可是MySQL、ORACLE、PostgreSQL等成熟的数据库。出于性能考虑，都是使用了以乐观锁为理论基础的MVCC（多版本号并发控制）来避免这两种问题。

**悲观锁和乐观锁  
悲观锁**  
正如其名，它指的是对数据被外界（包含本系统当前的其它事务，以及来自外部系统的事务处理）改动持保守态度。因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现。往往依靠数据库提供的锁机制（也仅仅有数据库层提供的锁机制才干真正保证数据訪问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会改动数据）。

在悲观锁的情况下，为了保证事务的隔离性，就须要一致性锁定读。

读取数据时给加锁，其他事务无法改动这些数据。

改动删除数据时也要加锁，其他事务无法读取这些数据。

**乐观锁**  
相对悲观锁而言，乐观锁机制採取了更加宽松的加锁机制。悲观锁大多数情况下依靠数据库的锁机制实现，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销。特别是对长事务而言，这种开销往往无法承受。

而乐观锁机制在一定程度上攻克了这个问题。乐观锁，大多是基于数据版本号（ Version ）记录机制实现。何谓数据版本号？即为数据添加一个版本号标识，在基于数据库表的版本号解决方式中，通常是通过为数据库表添加一个 “version” 字段来实现。

读取出数据时，将此版本号号一同读出，之后更新时，对此版本号号加一。

此时。将提交数据的版本号数据与数据库表相应记录的当前版本号信息进行比对，假设提交的数据版本号号大于数据库表当前版本号号，则予以更新。否则觉得是过期数据。

要说明的是，MVCC的实现没有固定的规范，每一个数据库都会有不同的实现方式，这里讨论的是InnoDB的MVCC。

**MVCC在MySQL的InnoDB中的实现**  
在InnoDB中，会在每行数据后加入两个额外的隐藏的值来实现MVCC，这两个值一个记录这行数据何时被创建，另外一个记录这行数据何时过期（或者被删除）。 在实际操作中。存储的并非时间，而是事务的版本，每开启一个新事务，事务的版本就会递增。 在可重读Repeatable reads事务隔离级别下：

* SELECT时。读取创建版本<=当前事务版本。删除版本为空或>当前事务版本。
* INSERT时，保存当前事务版本为行的创建版本
* DELETE时，保存当前事务版本为行的删除版本
* UPDATE时，插入一条新纪录。保存当前事务版本为行创建版本，同一时候保存当前事务版本到原来删除的行
* 通过MVCC，尽管每行记录都须要额外的存储空间，很多其它的行检查工作以及一些额外的维护工作。但能够降低锁的使用，大多数读操作都不用加锁，读数据操作非常easy，性能非常好，而且也能保证仅仅会读取到符合标准的行。也仅仅锁住必要行。

我们无论从数据库方面的教课书中学到。还是从网络上看到，大都是上文中事务的四种隔离级别这一模块列出的意思，RR级别是可反复读的，但无法解决幻读。而仅仅有在Serializable级别才干解决幻读。

于是我就加了一个事务C来展示效果。在事务C中加入了一条teacher\_id=1的数据commit。RR级别中应该会有幻读现象，事务A在查询teacher\_id=1的数据时会读到事务C新加的数据。

可是測试后发现，在MySQL中是不存在这样的情况的。在事务C提交后，事务A还是不会读到这条数据。可见在MySQL的RR级别中。是攻克了幻读的读问题的。參见下图

**Serializable**  
这个级别非常easy。读加共享锁。写加排他锁，读写相互排斥。使用的悲观锁的理论，实现简单，数据更加安全。可是并发能力非常差。假设你的业务并发的特别少或者没有并发，同一时候又要求数据及时可靠的话，能够使用这样的模式。

这里要吐槽一句，不要看到select就说不会加锁了。在Serializable这个级别，还是会加锁的。

**然后是关于数据库的各种锁的总结：**

1.共享锁（又称读锁）、排它锁（又称写锁）：

InnoDB引擎的锁机制：**InnoDB支持事务，支持行锁和表锁用的比较多，Myisam不支持事务，只支持表锁。**

共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。  
排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。  
意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。  
意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

说明：

**1）共享锁和排他锁都是行锁，意向锁都是表锁，应用中我们只会使用到共享锁和排他锁，意向锁是mysql内部使用的，不需要用户干预。**

**2）对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁，事务可以通过以下语句显示给记录集加共享锁或排他锁。**  
共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE。  
排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE。

\*\*对于锁定行记录后需要进行更新操作的应用，应该使用Select...For update 方式，获取排它锁。（用共享锁，在读了之后再写会阻塞，会导致死锁）

**这里说说Myisam：MyISAM在执行查询语句(SELECT)前，会自动给涉及的所有表加读锁，在执行更新操作(UPDATE、DELETE、INSERT等)前，会自动给涉及的表加写锁。**

**3）InnoDB行锁是通过给索引上的索引项加锁来实现的，因此InnoDB这种行锁实现特点意味着：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！**

2.乐观锁、悲观锁：

**悲观锁：**悲观锁，正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）

1）使用悲观锁，我们必须关闭mysql数据库的自动提交属性，采用手动提交事务的方式，因为MySQL默认使用autocommit模式，也就是说，当你执行一个更新操作后，MySQL会立刻将结果进行提交。

2）需要注意的是，在事务中，只有SELECT ... FOR UPDATE 或LOCK IN SHARE MODE 同一笔数据时会等待其它事务结束后才执行，一般SELECT ... 则不受此影响。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)。

3）补充：MySQL select…for update的Row Lock与Table Lock

使用select…for update会把数据给锁住，不过我们需要注意一些锁的级别，MySQL InnoDB默认Row-Level Lock，所以只有「明确」地指定主键（或有索引的地方），MySQL 才会执行Row lock (只锁住被选取的数据) ，否则MySQL 将会执行Table Lock (将整个数据表单给锁住)。

**乐观锁：**

乐观锁（ Optimistic Locking ） 相对悲观锁而言，乐观锁假设认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做（**一般是回滚事务**）。那么我们如何实现乐观锁呢，一般来说有以下2种方式：

1）.使用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。何谓数据版本？即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的 “version” 字段来实现。当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的version值进行比对，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据。

2）.乐观锁定的第二种实现方式和第一种差不多，同样是在需要乐观锁控制的table中增加一个字段，名称无所谓，字段类型使用时间戳（timestamp）, 和上面的version类似，也是在更新提交的时候检查当前数据库中数据的时间戳和自己更新前取到的时间戳进行对比，如果一致则OK，否则就是版本冲突。

**总结：两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适。**

**另外，高并发情况下个人认为乐观锁要好于悲观锁，因为悲观锁的机制使得各个线程等待时间过长，极其影响效率，乐观锁可以在一定程度上提高并发度。**

3.表锁、行锁

表级锁(table-level locking)：MyISAM和MEMORY存储引擎

行级锁(row-level locking) ：InnoDB存储引擎

页面锁(page-level-locking)：BDB存储引擎

**表级锁：**开销小，加锁快;不会出现死锁;锁定粒度大，发生锁冲突的概率最高,并发度最低。

**行级锁：**开销大，加锁慢;会出现死锁;锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

**页面锁：**开销和加锁时间界于表锁和行锁之间;会出现死锁;锁定粒度界于表锁和行锁之间，并发度一般。