数据库锁

数据中的锁分为两类：悲观锁和乐观锁，锁还有表级锁、行级锁

表级锁例如：

SELECT * FROM table WITH (HOLDLOCK) 其他事务可以读取表，但不能更新删除

SELECT * FROM table WITH (TABLOCKX) 其他事务不能读取表,更新和删除

行级锁例如：

select * from table_name where id = 1 for update;

悲观锁（Pressimistic Locking）

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）。例如：

select * from table_name where id = 'xxx' for update;

这样查询出来的这一行数据就被锁定了,在这个 update 事务提交之前其他外界是不能修改这条数据的，但是这种处理方式效率比较低，一般不推荐使用。

乐观锁（Optimistic Locking）

相对悲观锁而言，乐观锁机制采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，这样的开销往往无法承受。如一个金融系统，当某个操作员读取用户的数据，并在读出的用户数据的基础上进行修改时（如更改用户帐户余额），如果采用悲观锁机制，也就意味着整个操作过程中（从操作员读出数据、开始修改直至提交修改结果的全过程，甚至还包括操作员中途去煮咖啡的时间），数据库记录始终处于加锁状态，可以想见，如果面对几百上千个并发，这样的情况将导致怎样的后果。

乐观锁机制在一定程度上解决了这个问题。乐观锁，大多是基于数据版本（ Version ）

记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。

举个乐观锁的例子(数据库 version 默认为 0)：

不如现在一件衣服就剩一个库存了，但是有两个用户同时下单，如果这时候不加以控制很容易出现库存卖超的情况，这时候我们可以这样操作：

第一个用户将这件衣服读出（version=0），并将库存-1,

第二个用户也将这件衣服读出（version=0），并将库存-1，

第一个个用户完成操作，将数据库版本 version+1，执行更新库存时由于提交的数据版本大于数据库记录的版本，数据被更新，数据库中的 version 被更新为 2。

update goods set store=store-1,version=version+1 where id=xx and version=orginal_version

第二个用户也完成了操作，也将版本 version+1，执行更新库存时发现执行版本和数据库记录的版本相同，不符合提交版本必须大于数据库记录版本的乐观锁策略，所以第二个用户的下单请求被驳回，我们可以通过人性化处理异常给用户提示该商品已售罄等。

乐观锁机制避免了长事务中的数据库加锁开销（两个用户操作过程中，都没有对数据库数据加锁），大大提升了大并发量下的系统整体性能表现。

悲观锁：交给数据库来处理的，由事务（分隐私和显式事务，平时单条 SQL 语句就是一个隐式事务）+锁 那控制的，其中事务相当于锁的作用域，根据事务的提交失败或回滚来释放掉显式事务中开启的锁。（事前处理）

乐观锁：是认为版本号来控制的，这种机制并发性和性能更好（事后处理）