30丨锁：悲观锁和乐观锁是什么？

2019-08-19 陈旸

SQL必知必会 进入课程 >

讲述：陈旸
时长 10:57 大小 12.55M

索引和锁是数据库中的两个核心知识点，不论在工作中，还是在面试中，我们都会经常跟它们打交道。之前我们已经从不同维度对索引进行了了解，比如 B+ 树、Hash 索引、页结构、缓冲池和索引原则等，了解它们的工作原理可以加深我们对索引的理解。同时在基础篇的部分中，我也讲解了事务的 4 大原则以及不同的隔离级别。这些隔离级别的实现都是通过锁来完成的，你可以思考下为什么我们需要给数据加锁呢？

实际上加锁是为了保证数据的一致性，这个思想在程序开发领域中同样很重要。在程序开发中也会存在多线程同步的问题。当多个线程并发访问某个数据的时候，尤其是针对一些敏感的数据（比如订单、金额等），我们就需要保证这个数据在任何时刻最多只有一个线程在进行访问，保证数据的完整性和一致性。

今天的内容主要包括以下几个方面：

1. 就分类而言，锁的划分有多种方式，这些划分方式都包括哪些？
2. 为什么共享锁会发生死锁？
3. 乐观锁和悲观锁的思想是什么？乐观锁有两种实现方式，这两种实现方式是什么？
4. 多个事务并发，发生死锁时该如何解决？怎样降低死锁发生的概率？

按照锁粒度进行划分

锁用来对数据进行锁定，我们可以从锁定对象的粒度大小来对锁进行划分，分别为行锁、页锁和表锁。

顾名思义，行锁就是按照行的粒度对数据进行锁定。锁定力度小，发生锁冲突概率低，可以实现的并发度高，但是对于锁的开销比较大，加锁会比较慢，容易出现死锁情况。

页锁就是在页的粒度上进行锁定，锁定的数据资源比行锁要多，因为一个页中可以有多个行记录。当我们使用页锁的时候，会出现数据浪费的现象，但这样的浪费最多也就是一个页上的数据行。页锁的开销介于表锁和行锁之间，会出现死锁。锁定粒度介于表锁和行锁之间，并发度一般。

表锁就是对数据表进行锁定，锁定粒度很大，同时发生锁冲突的概率也会较高，数据访问的并发度低。不过好处在于对锁的使用开销小，加锁会很快。

行锁、页锁和表锁是相对常见的三种锁，除此以外我们还可以在区和数据库的粒度上锁定数据，对应区锁和数据库锁。不同的数据库和存储引擎支持的锁粒度不同，InnoDB 和 Oracle 支持行锁和表锁。而 MyISAM 只支持表锁，MySQL 中的 BDB 存储引擎支持页锁和表锁。SQL Server 可以同时支持行锁、页锁和表锁，如下表所示：

	行锁	页锁	表锁
InnoDB	√		√
MyISAM			√
BDB		√	√
Oracle	√		√
SQL Server	√	√	√

这里需要说明下，每个层级的锁数量是有限制的，因为锁会占用内存空间，锁空间的大小是有限的。当某个层级的锁数量超过了这个层级的阈值时，就会进行锁升级。锁升级就是用更大粒度的锁替代多个更小粒度的锁，比如 InnoDB 中行锁升级为表锁，这样做的好处是占用的锁空间降低了，但同时数据的并发度也下降了。

从数据库管理的角度对锁进行划分

除了按照锁粒度大小对锁进行划分外，我们还可以从数据库管理的角度对锁进行划分。共享锁和排它锁，是我们经常会接触到的两把锁。

共享锁也叫读锁或 S 锁，共享锁锁定的资源可以被其他用户读取，但不能修改。在进行SELECT的时候，会将对象进行共享锁锁定，当数据读取完毕之后，就会释放共享锁，这样就可以保证数据在读取时不被修改。

比如我们想给 product_comment 在表上加共享锁，可以使用下面这行命令：

复制代码

```
1 LOCK TABLE product_comment READ;
```

当对数据表加上共享锁的时候，该数据表就变成了只读模式，此时我们想要更新 product_comment 表中的数据，比如下面这样：

复制代码

```
1 UPDATE product_comment SET product_id = 10002 WHERE user_id = 912178;
```

系统会做出如下提示：

复制代码

```
1 ERROR 1099 (HY000): Table 'product_comment' was locked with a READ lock and can't be upd
```

也就是当共享锁没有释放时，不能对锁住的数据进行修改。

如果我们想要对表上的共享锁进行解锁，可以使用下面这行命令：

复制代码

```
UNLOCK TABLE;
```

如果我们想要给某一行加上共享锁呢，比如想对 user_id=912178 的数据行加上共享锁，可以像下面这样：

复制代码

```
SELECT comment_id, product_id, comment_text, user_id FROM product_comment WHERE user_id
```

排它锁也叫独占锁、写锁或 X 锁。排它锁锁定的数据只允许进行锁定操作的事务使用，其他事务无法对已锁定的数据进行查询或修改。

如果我们想给 product_comment 数据表添加排它锁，可以使用下面这行命令：

复制代码

```
LOCK TABLE product_comment WRITE;
```

这时只有获得排它锁的事务可以对 product_comment 进行查询或修改，其他事务如果想要在 product_comment 表上查询数据，则需要等待。你可以自己开两个 MySQL 客户端来模拟下。

这时我们释放掉排它锁，使用这行命令即可。

复制代码

```
UNLOCK TABLE;
```

同样的，如果我们想要在某个数据行上添加排它锁，比如针对 user_id=912178 的数据行，则写成如下这样：

复制代码

```
SELECT comment_id, product_id, comment_text, user_id FROM product_comment WHERE user_id
```

另外当我们对数据进行更新的时候，也就是INSERT、DELETE或者UPDATE的时候，数据库也会自动使用排它锁，防止其他事务对该数据行进行操作。

当我们想要获取某个数据表的排它锁的时候，需要先看下这张数据表有没有上了排它锁。如果这个数据表中的某个数据行被上了行锁，我们就无法获取排它锁。这时需要对数据表中的行逐一排查，检查是否有行锁，如果没有，才可以获取这张数据表的排它锁。这个过程是不是有些麻烦？这里就需要用到意向锁。

意向锁（Intent Lock），简单来说就是给更大一级别的空间示意里面是否已经上过锁。举个例子，你可以给整个房子设置一个标识，告诉它里面有人，即使你只是获取了房子中某一个房间的锁。这样其他人如果想要获取整个房子的控制权，只需要看这个房子的标识即可，不需要再对房子中的每个房间进行查找。这样是不是很方便？

返回数据表的场景，如果我们给某一行数据加上了排它锁，数据库会自动给更大一级的空间，比如数据页或数据表加上意向锁，告诉其他人这个数据页或数据表已经有人上过排它锁了，这样当其他人想要获取数据表排它锁的时候，只需要了解是否有人已经获取了这个数据表的意向排他锁即可。

如果事务想要获得数据表中某些记录的共享锁，就需要在数据表上添加意向共享锁。同理，事务想要获得数据表中某些记录的排他锁，就需要在数据表上添加意向排他锁。这时，意向锁会告诉其他事务已经有人锁定了表中的某些记录，不能对整个表进行全表扫描。

为什么共享锁会发生死锁的情况？

当我们使用共享锁的时候会出现死锁的风险，下面我们用两个 MySQL 客户端来模拟一下事务查询。

首先客户端 1 开启事务，然后采用读锁的方式对`user_id=912178`的数据行进行查询，这时事务没有提交的时候，这两行数据行上了读锁。

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM product_comment WHERE user_id = 912178 LOCK IN SHARE MODE;
+------------+------------+----------------------+---------------------+---------+
| comment_id | product_id | comment_text         | comment_time        | user_id |
+------------+------------+----------------------+---------------------+---------+
|      10001 |      10001 | b35cdab0766675de84e9 | 2018-01-01 00:00:14 |  912178 |
|     320411 |      10001 | 1f4f043b61277114fd98 | 2018-04-18 19:46:12 |  912178 |
+------------+------------+----------------------+---------------------+---------+
2 rows in set (0.00 sec)
```

然后我们用客户端 2 开启事务，同样对`user_id=912178`获取读锁，理论上获取读锁后还可以对数据进行修改，比如执行下面这条语句：

复制代码

```
1 UPDATE product_comment SET product_i = 10002 WHERE user_id = 912178;
```

当我们执行的时候客户端 2 会一直等待，因为客户端 1 也获取了该数据的读锁，不需要客户端 2 对该数据进行修改。这时客户端 2 会提示等待超时，重新执行事务。

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM product_comment WHERE user_id = 912178 LOCK IN SHARE MODE;
+------------+------------+----------------------+---------------------+---------+
| comment_id | product_id | comment_text         | comment_time        | user_id |
+------------+------------+----------------------+---------------------+---------+
|      10001 |      10001 | b35cdab0766675de84e9 | 2018-01-01 00:00:14 |  912178 |
|     320411 |      10001 | 1f4f043b61277114fd98 | 2018-04-18 19:46:12 |  912178 |
+------------+------------+----------------------+---------------------+---------+
2 rows in set (0.00 sec)

mysql> UPDATE product_comment SET product_id = 10002 WHERE user_id = 912178;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

你能看到当有多个事务对同一数据获得读锁的时候，可能会出现死锁的情况。

从程序员的角度对进行划分

如果从程序员的视角来看锁的话，可以将锁分成乐观锁和悲观锁，从名字中也可以看出这两种锁是两种看待数据并发的思维方式。

乐观锁（Optimistic Locking）认为对同一数据的并发操作不会总发生，属于小概率事件，不用每次都对数据上锁，也就是不采用数据库自身的锁机制，而是通过程序来实现。在程序上，我们可以采用版本号机制或者时间戳机制实现。

乐观锁的版本号机制

在表中设计一个版本字段 version，第一次读的时候，会获取 version 字段的取值。然后对数据进行更新或删除操作时，会执行`UPDATE ... SET version=version+1 WHERE version=version`。此时如果已经有事务对这条数据进行了更改，修改就不会成功。

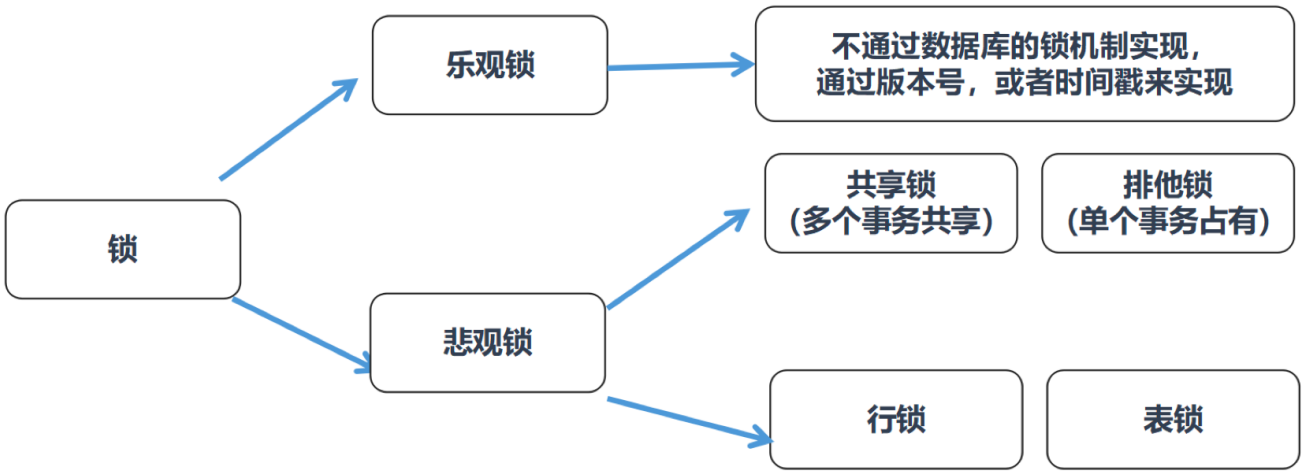
这种方式类似我们熟悉的 SVN、CVS 版本管理系统，当我们修改了代码进行提交时，首先会检查当前版本号与服务器上的版本号是否一致，如果一致就可以直接提交，如果不一致就需要更新服务器上的最新代码，然后再进行提交。

乐观锁的时间戳机制

时间戳和版本号机制一样，也是在更新提交的时候，将当前数据的时间戳和更新之前取得的时间戳进行比较，如果两者一致则更新成功，否则就是版本冲突。

你能看到乐观锁就是程序员自己控制数据并发操作的权限，基本是通过给数据行增加一个戳（版本号或者时间戳），从而证明当前拿到的数据是否最新。

悲观锁（Pessimistic Locking）也是一种思想，对数据被其他事务的修改持保守态度，会通过数据库自身的锁机制来实现，从而保证数据操作的排它性。



从这两种锁的设计思想中，你能看出乐观锁和悲观锁的适用场景：

1. 乐观锁适合读操作多的场景，相对来说写的操作比较少。它的优点在于程序实现，不存在死锁问题，不过适用场景也会相对乐观，因为它阻止不了除了程序以外的数据库操作。
2. 悲观锁适合写操作多的场景，因为写的操作具有排它性。采用悲观锁的方式，可以在数据库层面阻止其他事务对该数据的操作权限，防止读 - 写和写 - 写的冲突。

总结

今天我们讲解了数据库中锁的划分，你能看到从不同维度都可以对锁进行划分，需要注意的是，乐观锁和悲观锁并不是锁，而是锁的设计思想。

既然有锁的存在，就有可能发生死锁的情况。死锁就是多个事务（如果是在程序层面就是多个进程）在执行过程中，因为竞争某个相同的资源而造成阻塞的现象。发生死锁，往往是因为在事务中，锁的获取是逐步进行的。

我在文章中举了一个例子，在客户端 1 获取某数据行共享锁的同时，另一个客户端 2 也获取了该数据行的共享锁，这时任何一个客户端都没法对这个数据进行更新，因为共享锁会阻止其他事务对数据的更新，当某个客户端想要对锁定的数据进行更新的时候，就出现了死锁的情况。当死锁发生的时候，就需要一个事务进行回滚，另一个事务获取锁完成事务，然后将锁释放掉，很像交通堵塞时候的解决方案。

我们都不希望出现死锁的情况，可以采取一些方法避免死锁的发生：

1. 如果事务涉及多个表，操作比较复杂，那么可以尽量一次锁定所有的资源，而不是逐步来获取，这样可以减少死锁发生的概率；
2. 如果事务需要更新数据表中的大部分数据，数据表又比较大，这时可以采用锁升级的方式，比如将行级锁升级为表级锁，从而减少死锁产生的概率；
3. 不同事务并发读写多张数据表，可以约定访问表的顺序，采用相同的顺序降低死锁发生的概率。

当然在数据库中，也有一些情况是不会发生死锁的，比如采用乐观锁的方式。另外在 MySQL MyISAM 存储引擎中也不会出现死锁，这是因为 MyISAM 总是一次性获得全部的锁，这样的话要么全部满足可以执行，要么就需要全部等待。

最后你有没有想过，使用 MySQL InnoDB 存储引擎时，为什么对某行数据添加排它锁之前，会在数据表上添加意向排他锁呢？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起来进步。

上一篇　29 | 为什么没有理想的索引？

精选留言 (6)

写留言

老毕
2019-08-19

意向锁是一种协作机制，用于表锁和行锁的共生场景。

意向锁表达的信息很明确：某事务打算获得某种行锁，或某事务已持有某种行锁。

这样一来，打算锁表的事务就能迅速获得足够的锁信息并决定下一步行动。...
展开

许童童
2019-08-19

使用 MySQL InnoDB 存储引擎时，为什么对某行数据添加排它锁之前，会在数据表上添加意向排他锁呢？
这样就不需要逐行扫描，看每一行上是否有排它锁了，通过大粒度，来节省资源。

捞鱼的搬砖奇

2019-08-19

请问xmind可以提供下吗

展开

lmtoo

2019-08-19

添加意向排它锁是为了防止在锁升级时需要全表扫描；两个读锁，其中一个读锁升级为写锁，这不算是死锁吧

夜空中最亮的星（华仔...

2019-08-19

我又回来了老师

展开

安静的boy

2019-08-19

有一点不明白，为什么MyISAM不存在死锁的情况？MyISAM一个事务一次获取全部的共享锁，另一个事务也获取全部的共享锁，然后再更新数据，这个时候不还是会发生死锁吗？

1