锁机制与结构

课程内容

1. 死锁
2. MVCC-乐观锁
3. 乐观锁与悲观锁
4. 间隙锁
5. mysql大体结构

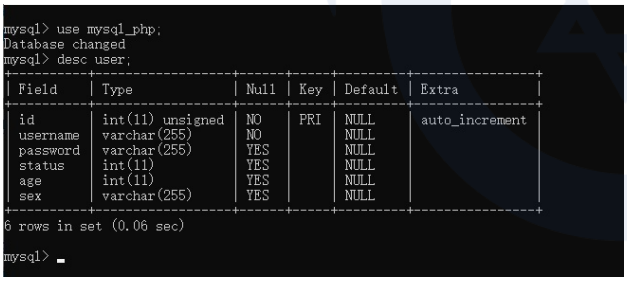
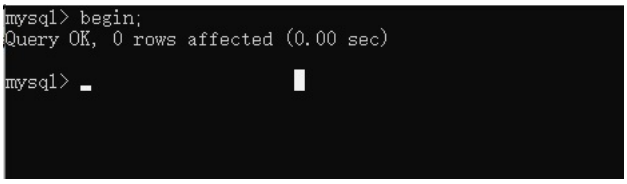
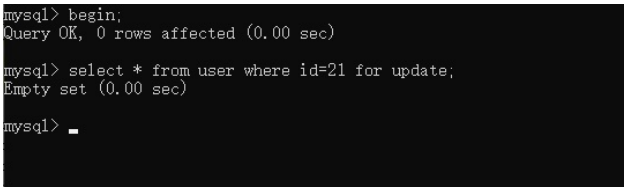
1. 死锁（DeadLock）

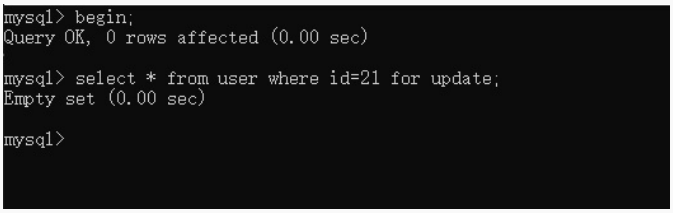
1.1 什么情况下会产生死锁

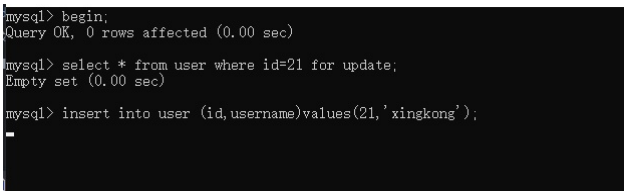
死锁是指两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去.此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等待的进程称为死锁进程.表级锁不会产生死锁.所以解决死锁主要还是针对于最常用的InnoDB

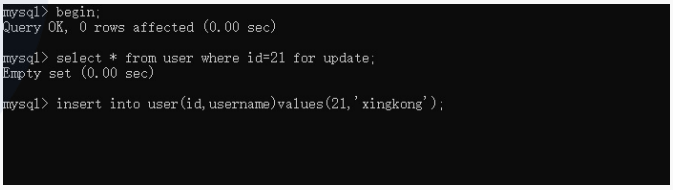
类型	死锁情况
表级	否
页级	可能会
行级	是

1.2 mysql死锁演示

session2	session1
查看表结构	
	
begin(begin-end用于定义一组语句块)查询数据	
	
	□
	







通常在应用中，在事务隔离级别RR级别下，两个事务对相同的数据使用了排它锁，在没有符合该条件的情况下，两个事务都会加锁成功。如果是在程序中发现记录不存在，就新增一条数据，如果两个事务都这么做，就会发生死锁的情况。

> mysql处理死锁的方式

- 等待，直到超时（innodb_lock_wait_timeout=50s）。
- 发起死锁检测，主动回滚一条事务，让其他事务继续执行（innodb_deadlock_detect=on）。

死锁检测

死锁检测的原理是构建一个以事务为顶点、锁为边的有向图，判断有向图是否存在环，存在即有死锁。

回滚

检测到死锁之后，选择插入更新或者删除的行数最少的事务回滚，基于 INFORMATION_SCHEMA.INNODB_TRX 表中的 trx_weight 字段来判断。

2. MVCC-乐观锁

2.1 什么是MVCC?

英文全称为Multi-Version Concurrency Control,翻译为中文即 多版本并发控制。MVCC使得InnoDB的事务隔离级别下执行一致性读操作有了保证，换言之，就是为了查询一些正在被另一个事务更新的行， 并且可以看到它们被更新之前的值。这是一个可以用来增强并发性的强大的技术，因为这样一来的话查询就不用等待另一个事务释放锁。这项技术在数据库领域并不是普遍使用的。一些其它的数据库产品， 以及mysql其它的存储引擎并不支持它。

mysql的innodb表除了实际的数据之外，还会加上3个隐藏的字段，如下：

实际数据	create_no(创建版本号或者创建时间)	update_no(每次修改的版本号或者修改时间)	delete_no(删除版本号或者删除时间)

```
insert：当我们新增一条数据时，这条数据会加上创建的版本号
update：修改当前的字段，每修改一次数据，修改版本号都会依次增加一次
delete：删除当前的数据，其实并不会真实的删除，他会先在删除版本号字段记录下删除的版本号，在过了一段时间后会进行清除或者刷新
```

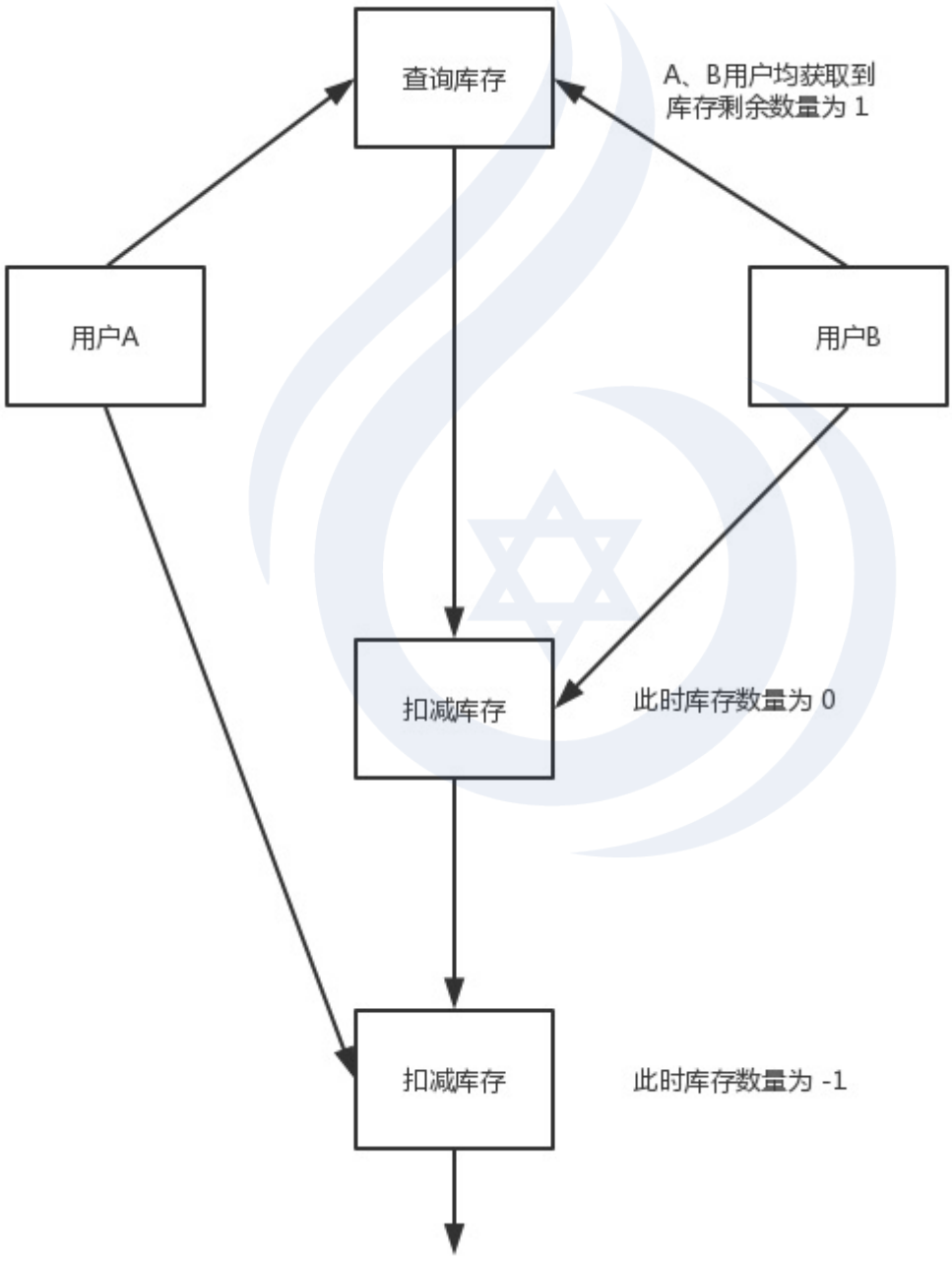
> MVCC是乐观锁的一种实现方式，但并不是MVCC就等于乐观锁。

3. 悲观锁

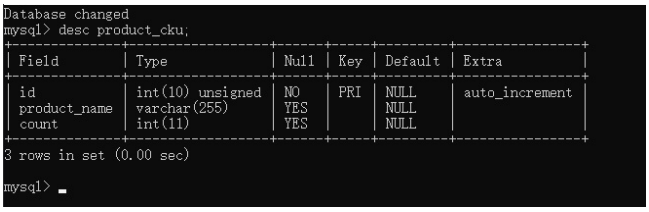
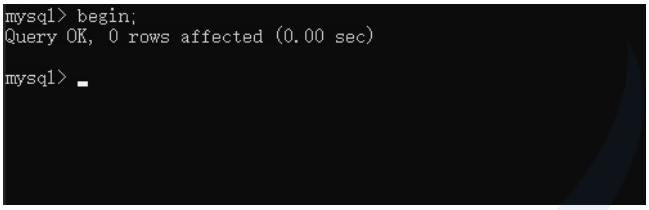
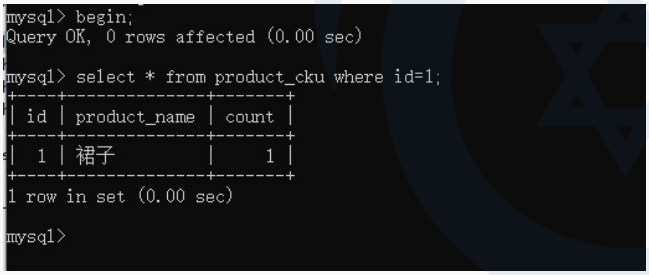
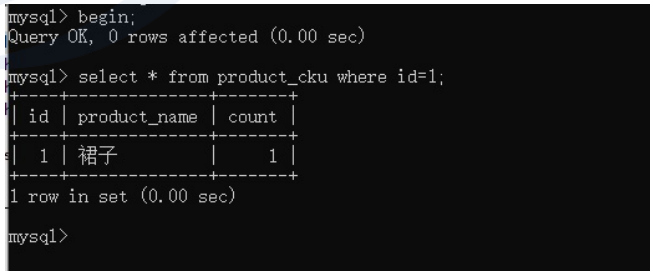
悲观并发控制（又名“悲观锁”，Pessimistic Concurrency Control，缩写“PCC”）是一种并发控制的方法。它可以阻止一个事务以影响其他用户的方式来修改数据。如果一个事务执行的操作读某行数据应用了锁，那只有当这个事务把锁释放，其他事务才能够执行与该锁冲突的操作。

> 使用场景

在商品购买场景中，当有多个用户对某个库存有限的商品同时进行下单操作。若采用先查询库存，后减库存的方式进行库存数量的变更，将会导致超卖的产生。



以下是超卖问题演示：

session1	session
查看数据表结构	
	
开启事务1查看库存是否满足	
	
	开启事务2查看库存是否满足
	
	
	
满足库存则减少库存数量	

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

mysql> select * from product_cku where id=1;
+----+--------------+-------+
| id | product_name | count |
+----+--------------+-------+
|  1 | 裙子         |     1 |
+----+--------------+-------+
1 row in set (0.00 sec)

mysql> update product_cku set count=count-1 where id=1;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

事务2查询满足也减少数量

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

mysql> select * from product_cku where id=1;
+----+--------------+-------+
| id | product_name | count |
+----+--------------+-------+
|  1 | 裙子         |     1 |
+----+--------------+-------+
1 row in set (0.00 sec)

mysql> update product_cku set count=count-1 where id=1;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

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

mysql> select * from product_cku where id=1;
+----+--------------+-------+
| id | product_name | count |
+----+--------------+-------+
|  1 | 裙子         |     1 |
+----+--------------+-------+
1 row in set (0.00 sec)

mysql> update product_cku set count=count-1 where id=1;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

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

mysql>
```

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

mysql> select * from product_cku where id=1;
+----+--------------+-------+
| id | product_name | count |
+----+--------------+-------+
|  1 | 裙子         |     1 |
+----+--------------+-------+
1 row in set (0.00 sec)

mysql> update product_cku set count=count-1 where id=1;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

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

mysql>
```

事务1与事务2提交后查询数据，发现出现超卖现象

```
mysql> select * from product_cku where id=1;
+----+--------------+-------+
| id | product_name | count |
+----+--------------+-------+
|  1 | 裙子         |    -1 |
+----+--------------+-------+
1 row in set (0.00 sec)

mysql>
```

使用悲观锁可以解决该问题：我们可以在查询库存是就加把排它锁，这样当事务2在查询的时候就会出现锁等待现象，只有到前面的事务1释放，事务2才能继续操作。而当事务2可以查询时事务1已经修改了数据，判定为不符合库存。

4. 间隙锁

当我们用范围条件而不是相等条件检索数据，并请求共享或排他锁时，InnoDB会给符合条件的已有数据记录的索引项加锁；对于键值在条件范围内但不存在的记录，叫做"间隙(GAP)"，InnoDB也会对这个"间隙"加锁，这种锁机制就是所谓的间隙锁(NEXT-KEY)锁。

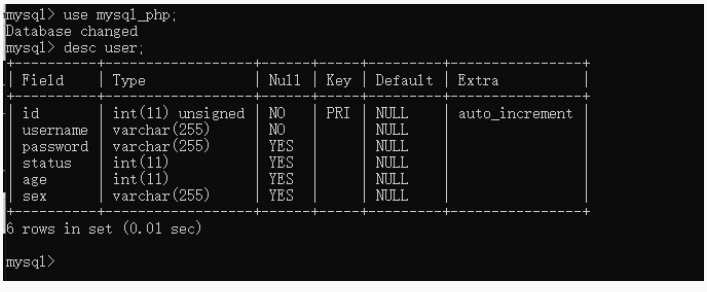
session1	session2
start transaction	start transaction
select * from user_s where id between 1 and 10;	
此时已经锁住了id为1-10的数据	
	先修改id为12的数据试试
	update user_s set username='start' where id=12
	修改成功
	在来试试id为5的数据
	update user_s set username='start' where id=5
	进入锁等待
释放	
commit	
	修改成功

5. 行锁升级为表锁

InnoDB 行级锁是通过给索引上的索引项加锁来实现的，InnoDB行级锁只有通过索引条件检索数据，才使用行级锁；否则，InnoDB使用表锁 在不通过索引（主键）条件查询的时候，InnoDB是表锁而不是行锁。

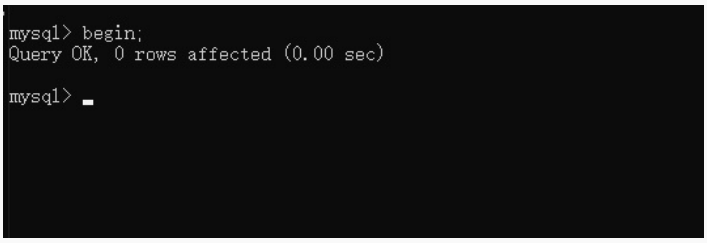
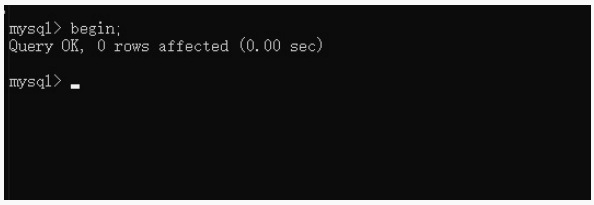
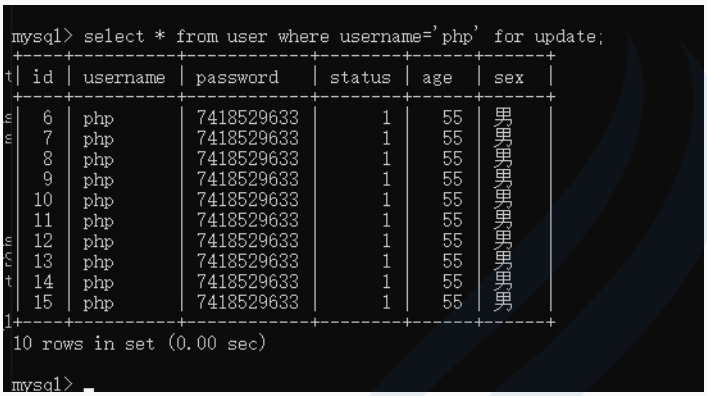
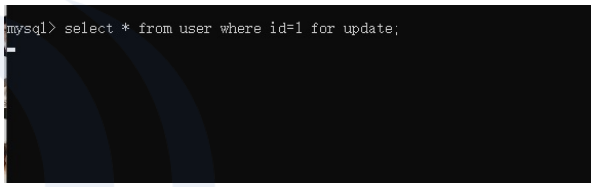
> 通常begin-end用于定义一组语句块

InnoDB表锁定机制的列子

session1	session2
查看数据表结构	
	

```
mysql> use mysql_php;
Database changed
mysql> desc user;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(11) unsigned | NO   | PRI | NULL    | auto_increment |
| username | varchar(255)     | NO   |     | NULL    |                |
| password | varchar(255)     | YES  |     | NULL    |                |
| status   | int(11)          | YES  |     | NULL    |                |
| age      | int(11)          | YES  |     | NULL    |                |
| sex      | varchar(255)     | YES  |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)

mysql>
```

begin(begin-end用于定义一组语句块)查询数据	
	
查询数据并加上锁机制	
	
	
commit	这个时候就可以查询数据了

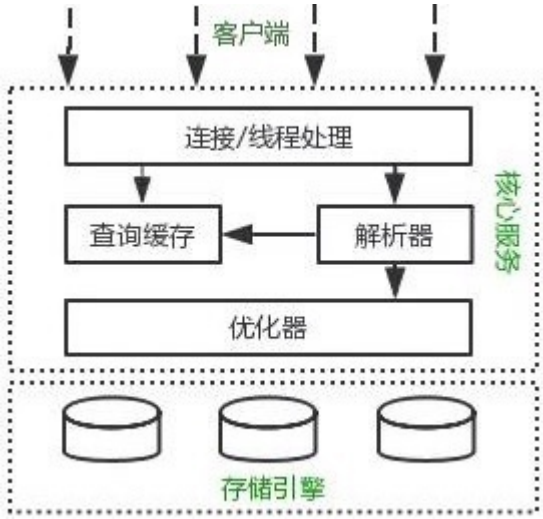
总结：就是在没有使用索引的情况下InnoDB就会使用表级锁（共享锁不会有这个情况）

6. 事务的使用建议

innodb存储引擎由于实现了行几所，颗粒更小，实现更复杂。但是innodb行锁在并发性能上远远要高于表锁页锁。在使用方面可以尽量做到以下几点；

1. 控制事务大小，减少锁定的资源量和锁定时间长度。
2. 人所有的数据检索都通过索引来完成，从而避免因为无法通过索引加锁而升级为表锁。
3. 减少基于范围的数据检索过滤条件，避免因为间隙锁带来的负面影响而锁定了不该锁定的数据。
4. 在业务条件允许下，尽量使用较低隔离级别的事务隔离。减少隔离级别带来的附加成本。
5. 河里使用索引，让innodb在索引上面加锁的时候更加准确。
6. 在应用中尽可能做到访问的顺序执行
7. 如果容易死锁，就可以考虑使用表锁来减少死锁的概率

7. mysql的大体结构



客户端 = Connection(语言连接器例如：PHP-pdo，MySQLi)

服务端 = SQL层 + 存储引擎层

SQL层 = 链接/线程处理 + 查询缓存 + 分析器 + 优化器

存储引擎 = InnoDB + MariaDB + ... Connection：这一块其实主要是其他语言的连接，并不属于MySQL本身；主要是其他语言对于MySQL的连接操作的工具比如PHP中的：pdo，mysqli或者 Navicat for MySQL SQL层：

功能主要包括权限判断，SQL解析功能和查询缓存处理等。

1. 链接/线程处理：客户端通过 连接/线程层 来连接MySQL数据库，连接/线程层主要用来处理客户端的请求、身份验证和数据库安全性验证等。
2. 查询缓存和查询分析器是SQL层的核心部分，其中主要涉及查询的解析、优化、缓存、以及所有内置的函数，存储过程，触发器，视图等功能。
3. 优化器主要负责存储和获取所有存储在MySQL中的数据。