# 事务与锁基础

## 课程内容

1. 事务的基础理论
2. 锁的基础以及行锁的特点
3. 锁对于语句的加锁
4. 事务的生命周期
5. 事务重做与回滚日志

## 1. 事务的基础

1. 事务即 transaction，是数据库系统区别于文件系统的重要特性之一。
2. 在文件系统中，如果我们正在写文件，但是操作系统崩溃了，那么文件中的数据可能会丢失。但是数据库可以通过事务机制来确保这一点。

> 事务的必要性

假设有两个用户，不妨记为 a 和 b，假设 a 要给 b 转账 1000 元，那么只需要在表中把 a 对应的记录的相应字段减去 1000，给 b 对应的记录的相应字段增加 1000 即可。
但是当我们给 a 对应的记录减去 1000，但是当执行给 b 对应的记录增加 1000 的时候，服务器突然出现了一些问题，导致该 SQL 并未顺利执行就宕机了。
如果不启用事务，那么结果就是 a 对应的记录减少了 1000，但是 b 对应的记录并没有增加 1000，这是不符合常理的，于是，我们引入了事务机制来保证它的可靠性。

> 数据库系统引入事务的主要目的：

**事务会把数据库从一种一致性状态转换为另一种一致性状态。**在数据库提交的时候，可以确保要么所有提交都保存，要么所有修改都不保存。 **事务可以用来保证数据库的完整性：要么都做，要么不做**。

### 1.1 事务的特性（ACID）

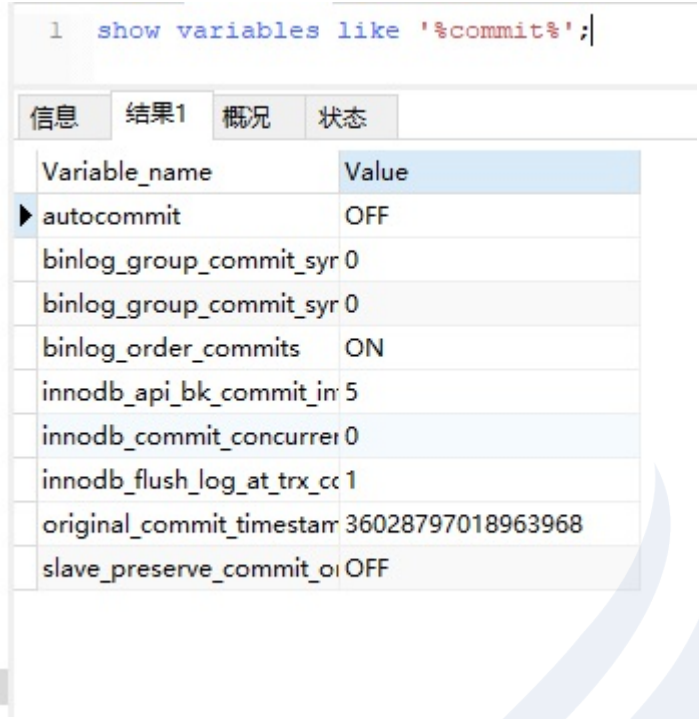
1. 事务要求 ACID 的特性，即：**原子性、一致性、隔离性、持久性**。
2. 所谓原子性，是指整个数据库的每个事务都是不可分割的单位。只有事务中的所有 SQL 语句都执行成功，才算整个事务成功，事务才会被提交。如果事务中任何一个 SQL 语句执行失败，整个事务都应该被回滚。
3. 所谓一致性，是指将数据库从一种一致性状态转换为下一种一致性状态。不允许数据库中的数据出现新老数据都有的情况，要么都是老数据，要么都是新数据。用更书面化的表达就是：数据的完整性约束没有被破坏。
4. 所谓隔离性，是指一个事务的影响在该事务提交前对其他事务都不可见，它通过锁机制来实现。
5. 所谓持久性，是指事务一旦被提交，其结果就是永久性的。即使发生宕机等故障，数据库也能将数据恢复。

### 1.2 事务的语法

1. 在 MySQL 命令行的默认设置下，事务是自动提交的，即执行了SQL 语句之后会马上执行 commit 操作，我们可以设置 set autocommit=0 来禁用当前回话的自动提交。
2. 还可以用 begin 、start transaction 来显式的开始一个事务。
3. commit 在默认设置下是等价于 commit work 的，表示提交事务。
4. rollback 在默认设置下等价于 rollback work，表示事务回滚。
5. savepoint xxx 表示定义一个保存点，在一个事务中可以有多个保存点。
6. release savepoint xxx 表示删除一个保存点，当没有该保存点的时候执行该语句，会抛出一个异常。
7. rollback to [savepoint] xxx 表示回滚到某个保存点。

> 简单操作

```
--查询事务自动提交状态
show variables like '%commit%';
```




以下是事务自动提交修改方式：

```
--全局修改
set global autocommit=0;
show global variables like 'autocommit';
--局部修改
set session autocommit=0;
show global variables like 'autocommit';
```

```
show global variables like 'autocommit';
```

信息 | 结果1 | 概况 | 状态

| Variable_name | Value |
|---|---|
| autocommit | OFF |

```
show session variables like 'autocommit';
```

信息 | 结果1 | 概况 | 状态

| Variable_name | Value |
|---|---|
| autocommit | ON |

事务简单操作：开启两个窗口，一个用于事务新增数据，一个用于事务提交前查询

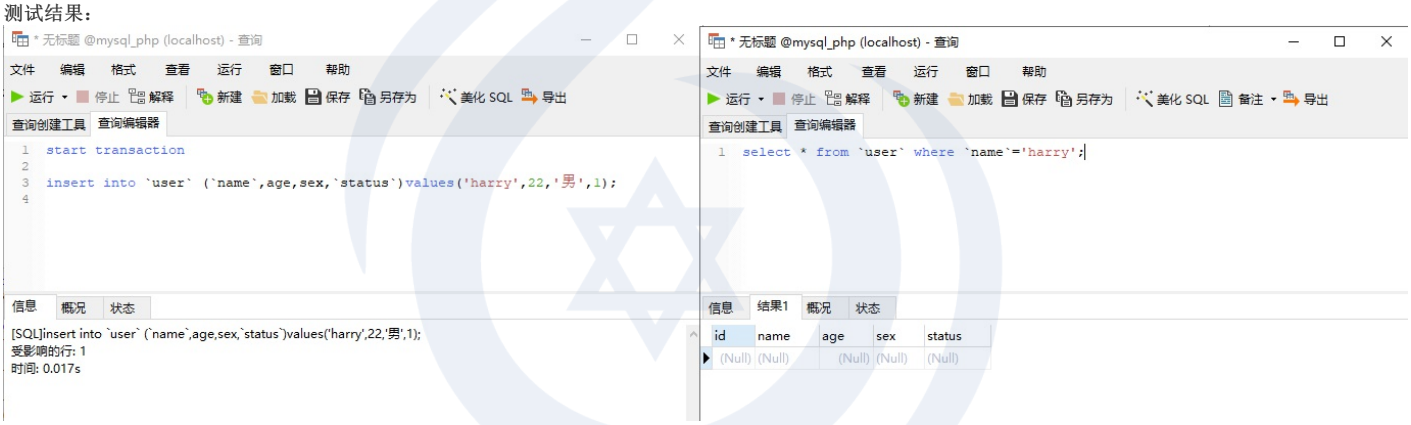
第一次，session1事务没有提交，session2直接查询数据 session1：

```
start transaction--开启事务
insert into `user` (`name`,age,sex,`status`)values('harry',222,'男',1);
```

session2：

```
select * from `user` where name='harry';
```

测试结果：




第二次，session1提交事务，session2等到session1提交完成后查询数据 session1：

```
start transaction--开启事务
insert into `user` (`name`,age,sex,`status`)values('will',222,'男',1);
commit;--提交事务
rollback;--回滚事务
```

session2：

```
select * from `user` where name='harry';
```

测试结果：


通过上面，我们可以感受到事务的隔离性，也就是两个事务之间并不知道对方的存在，在 MySQL 的默认隔离级别下，当一个事务还没有提交的时候，其他事务是无法感知到数据的变化的。

# 2. 锁的基础以及行锁的特点

## 2.1 概念

1. 在开发多用户、数据库驱动的应用时，相当大的一个难点就是解决并发性的问题，目前比较常用的解决方案就是锁机制。
2. 锁机制也是数据库系统区别于文件系统的一个关键特性。
3. InnoDB 存储引擎和 MyISAM 存储引擎使用的是完全不同的策略，我们必须分开来讲。

## 2.2 锁类型

1. 相比其他数据库而言，MySQL 的锁机制比较简单，而且不同的存储引擎支持不同的锁机制。
2. MyISAM 和 Memory 存储引擎使用的是表级锁,BDB 引擎使用的是页级锁，也支持表级锁。由于 BDB 引擎基本已经成为历史，因此就不再介绍了。
3. InnoDB 存储引擎既支持行级锁，也支持表级锁，默认情况下使用行级锁。
4. 所谓表级锁，它直接锁住的是一个表，开销小，加锁快，不会出现死锁的情况，锁定粒度大，发生锁冲突的概率更高，并发度最低。
5. 所谓行级锁，它直接锁住的是一条记录，开销大，加锁慢，发生锁冲突的概率较低，并发度很高。
6. 所谓页级锁，它是锁住的一个页面，在 InnoDB 中一个页面为16KB，它的开销介于表级锁和行级锁中间，也可能会出现死锁，锁定粒度也介于表级锁和行级锁中间，并发度也介于表级锁和行级锁中间。
7. 仅仅从锁的角度来说，表级锁更加适合于以查询为主的应用，只有少量按照索引条件更新数据的应用，比如大多数的 web 应用。
8. 行级锁更适合大量按照索引条件并发更新少量不同的数据，同时还有并发查询的应用，比如一些在线事务处理系统，即 OLTP。

## 2.3 innoDB锁

1. InnoDB 与 MyISAM 的相当大的两点不同在于： (1) 支持事务 (2) 采用行级锁
2. 行级锁本身与表级锁的实现差别就很大，而事务的引入也带来了很多新问题，尤其是事务的隔离性，与锁机制息息相关。
3. 对于事务的基本操作，对于不同隔离级别可能引发的问题，像脏读、不可重复读等问题我们上一节就已经举例说明了，这里就不再赘述了。
4. 数据库实现事务隔离的方式，基本可以分为两种：

> (1) 在操纵数据之前，先对其加锁，防止其他事务对数据进行修改。这就需要各个事务串行操作才可以实现。
>
> (2) 不加任何锁，通过生成一系列特定请求时间点的一致性数据快照，并通过这个快照来提供一致性读取。

5. 上面的第二种方式就是数据多版本并发控制，也就是多版本数据库，一般简称为 MVCC 或者 MCC，它是 Multi Version Concurrency Control 的简写。
6. 数据库的事务隔离越严格，并发的副作用就越小，当然付出的代价也就越大，因为事务隔离机制实质上是使得事务在一定程度上"串行化",这与并行是矛盾的。

## 2.4 innoDB锁类型

1. InnoDB 实现了下面两种类型的锁：

> (1)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。
>
> (2)排他锁(X):允许获得排他锁的事务更新数据，阻止其他事务获得相同数据集的共享读锁和排他写锁。

2. InnoDB 还有两种意向锁，即 Intention Lock，这两种锁都是表锁。意向锁是内部使用的，它是 InnoDB 内部加的，不用用户干预，意向锁分类如下：

```
(1)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。

(2)排他锁(X):允许获得排他锁的事务更新数据，阻止其他事务获得相同数据集的共享读锁和排他写锁。
```

3. 这里有个锁兼容和冲突的概念，如果在加一个锁的时候，另一个锁可以加上去，那么就是锁兼容。如果加上一个锁之后，拒绝其他的锁加上，那么就是锁冲突。
4. 各种锁的兼容冲突情况如下：

```
(1)X 和所有锁都冲突

(2)IX 兼容 IX 和 IS

(3)S 兼容 S 和 IS

(4)IS 兼容 IS、IX 和 S
```

5. 如果一个事务请求的锁模式与当前的锁兼容，InnoDB 就将请求的锁授予该事务，如果两者是冲突的，那么该事务就要等待锁释放。
6. 对于 update、delete、insert 语句，InnoDB 会自动给设计到的数据集加排他锁即 X。
7. 对于 select 语句，InnoDB 不会加任何锁。
8. 我们可以使用如下语句来显式的给数据集加锁：

```
(1)共享锁(S):select * from t1 where ... lock in share mode;

(2)排他锁(X):select * from t1 where ... for update;
```

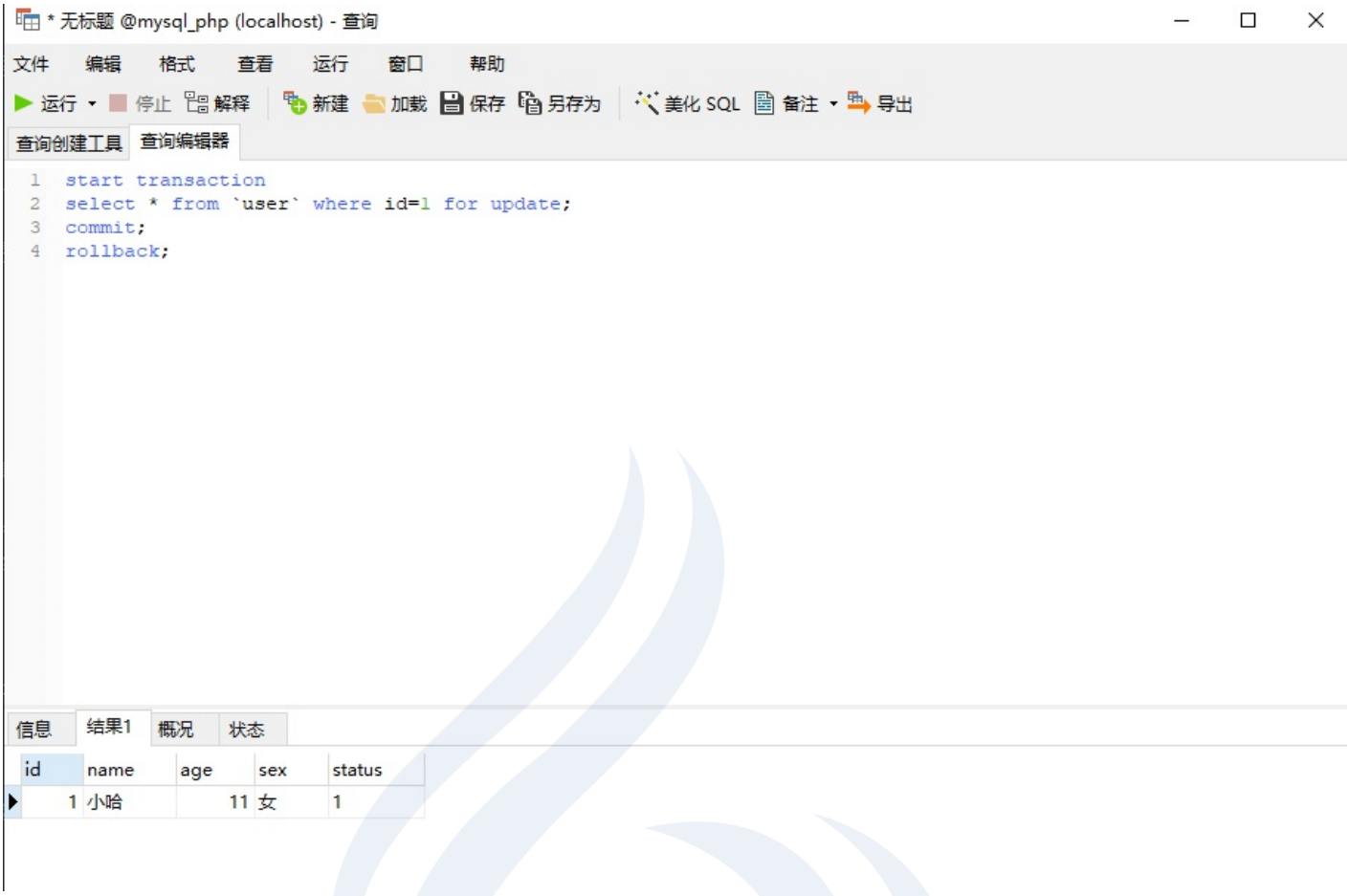
9. 我们可以用 select ...in share mode 来获得共享锁，主要用在数据依存关系时来确认某行记录是否存在，并确认没有人对这个记录进行 update 或者 delete 操作。
10. 我们可以使用 select... for update 来获得排他锁，它会拒绝其他事务在其上加其他锁。

# 3. 锁对于语句的加锁

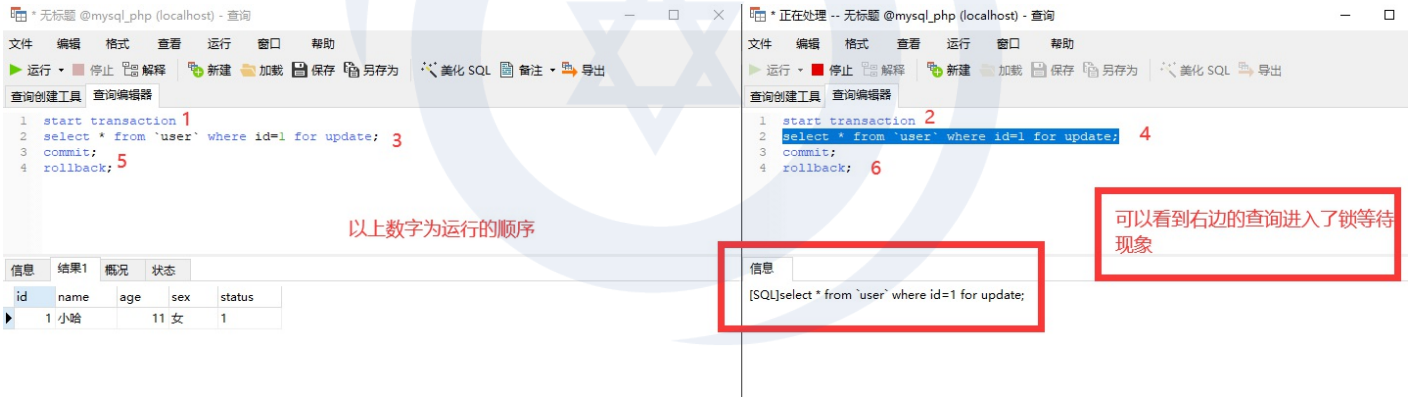
## 3.1 排它锁

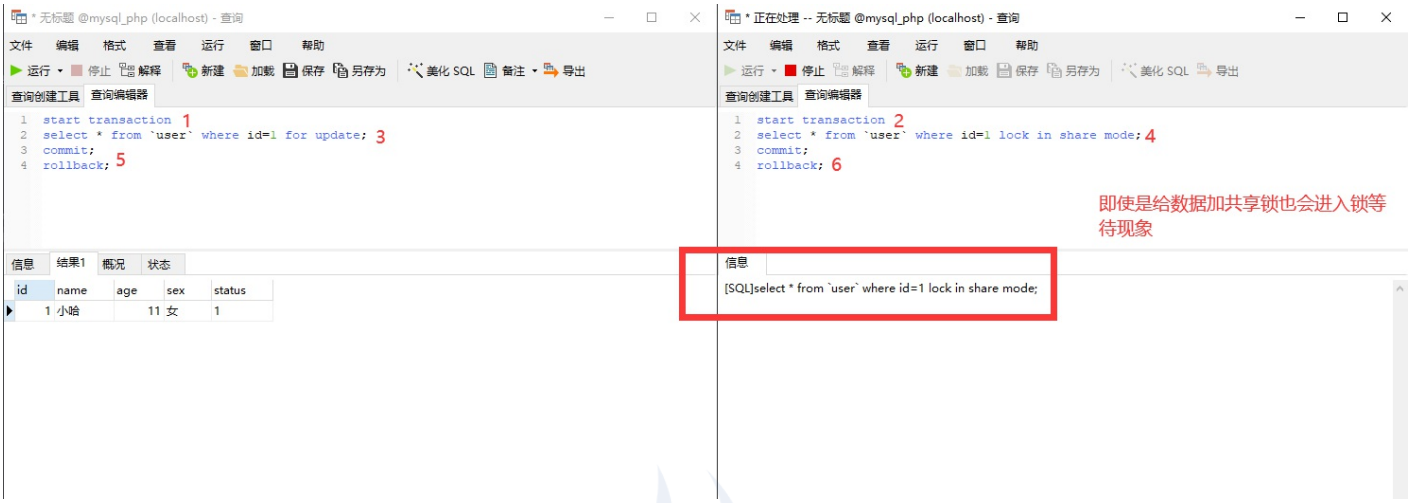
排它锁锁加锁测试：

```
--给`user`表id为1的数据加排它锁
start transaction--开启事务
select * from `user` where id=1 for update;--给id为1的数据加排它锁
commit;--提交事务
rollback;--回滚事务
```




上面的结果我们并不能看出什么效果，但是如果我们同时开启两个session，一起去给同一数据加锁时排它锁的效果就非常明显了，如下测试：






上面的测试结果表名，当前事务给一行数据加锁，那么其他事务将不能在对数据做任何操作，即：**不能读不能写，也不能与其他锁一起使用**

## 3.2 共享锁

共享锁加锁测试：

session1:

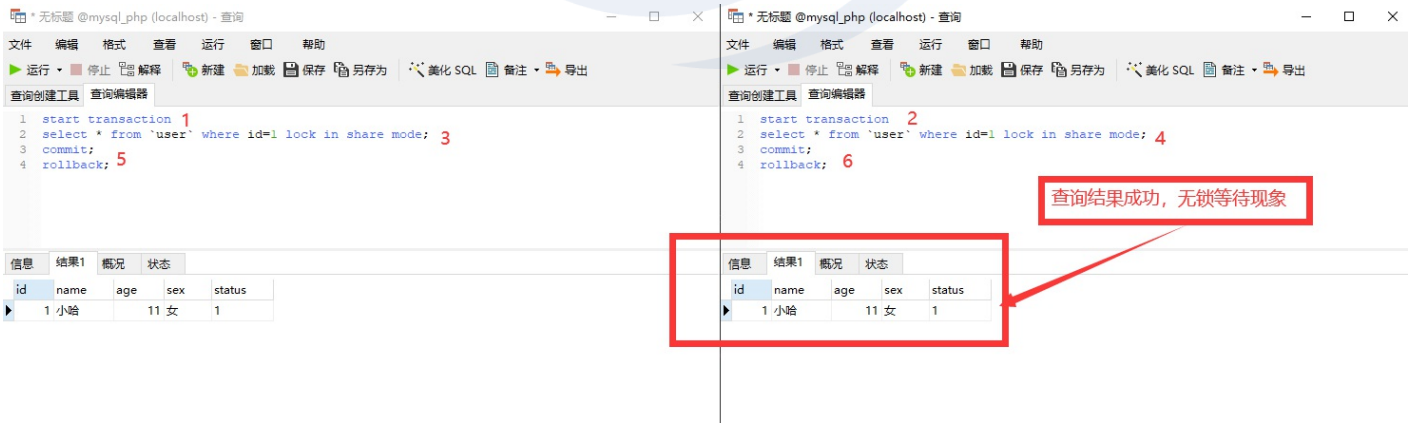
```
--给`user`表id为1的数据加共享锁
start transaction--开启事务
select * from `user` where id=1 lock in share mode;--给id为1的数据加共享锁
commit;--提交事务
rollback;--回滚事务
```

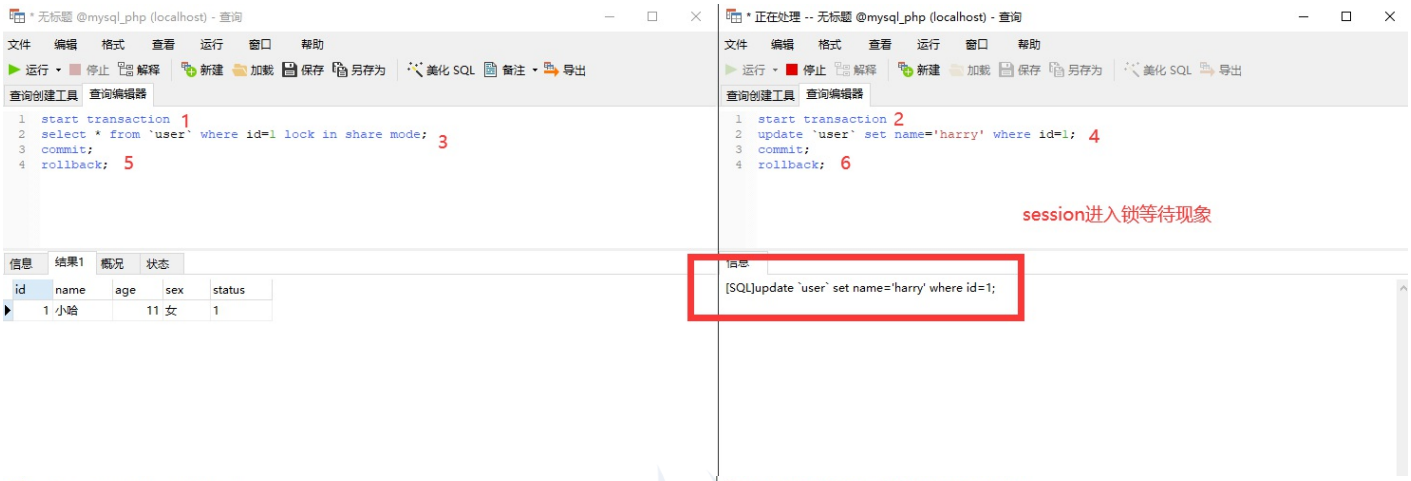
session2:

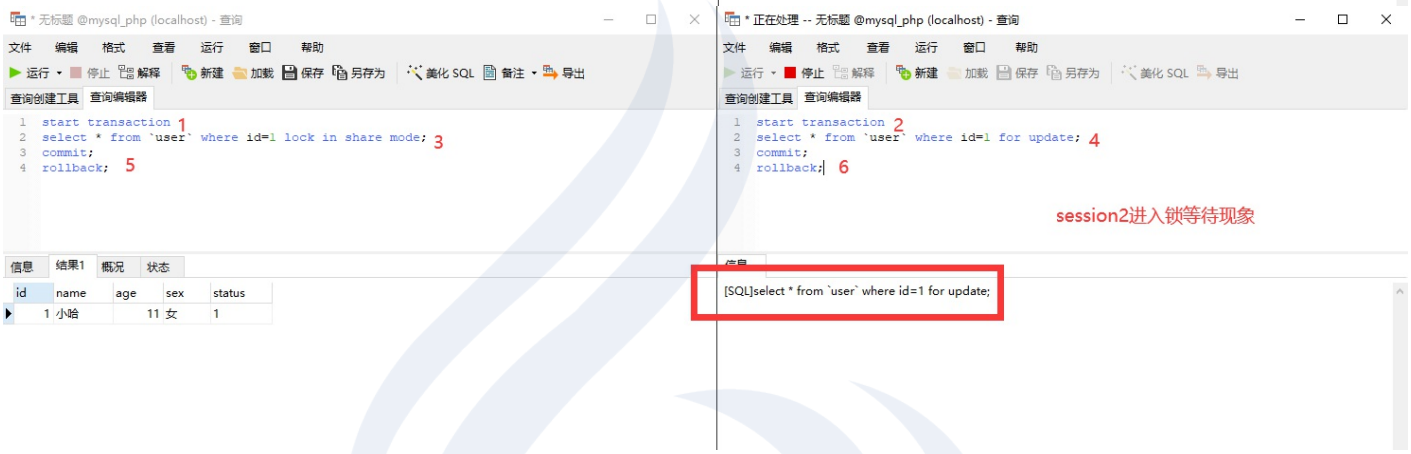
```
--给`user`表id为1的数据加共享锁
start transaction--开启事务
select * from `user` where id=1 lock in share mode;--给id为1的数据加共享锁
commit;--提交事务
rollback;--回滚事务

--给`user`表id为1的数据加共享锁
start transaction--开启事务
update `user` set name='harry' where id=1 for update;
commit;--提交事务
rollback;--回滚事务
```

测试结果：






上面的测试结果表明，当前事务给一行数据加共享锁，那么其他事务可以加共享锁，但不能加排它锁。即：**能读不能写，可以与共享锁一起使用，但不能与排它锁一起使用**

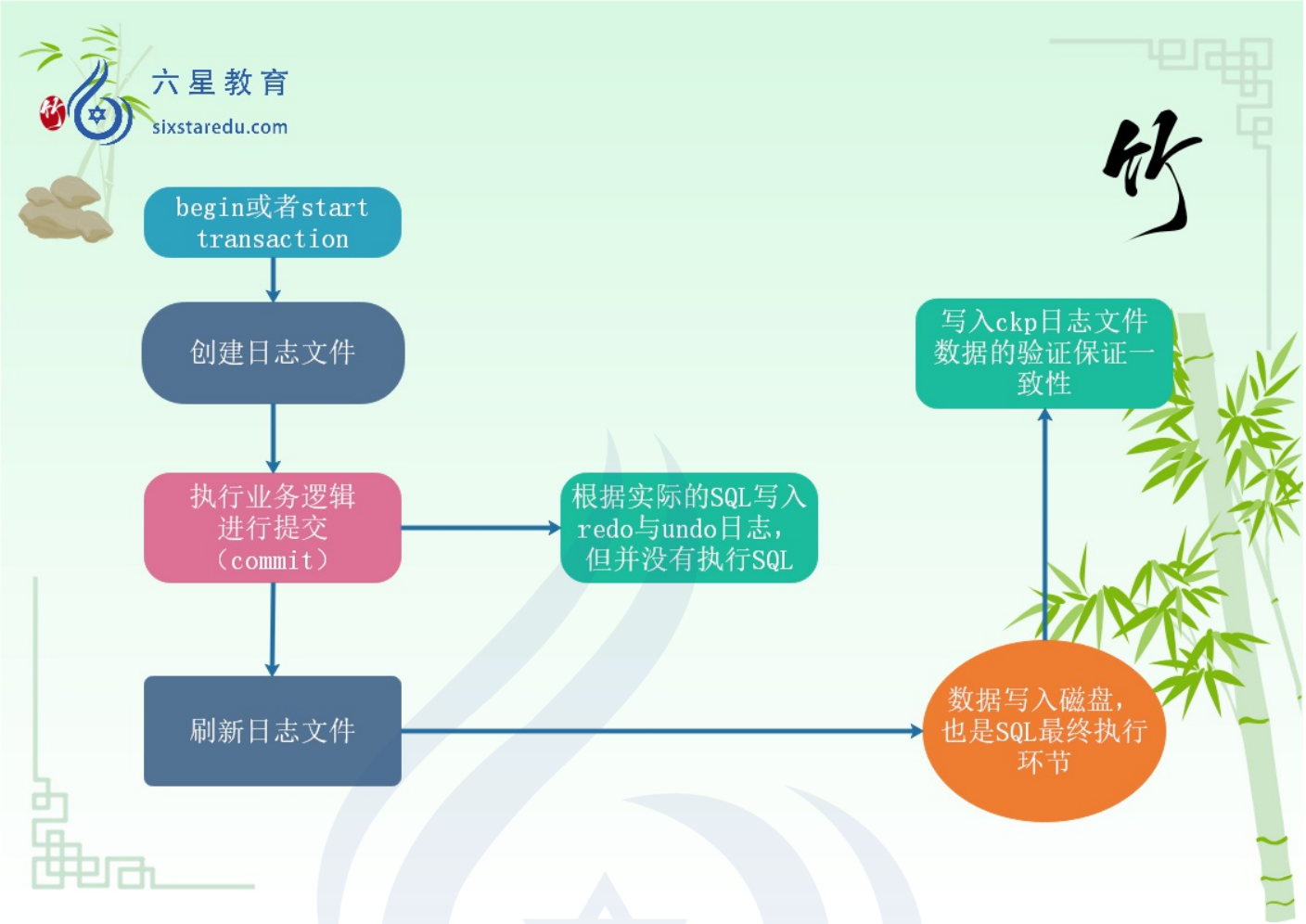
# 4. 事务生命周期

MySQL的checkpoint https://www.cnblogs.com/lintong/p/4381578.html

checkpoint，即检查点。在undolog中写入检查点，表示在checkpoint前的事务都已经完成commit或者rollback了，也就是检查点前面的事务已经不存在数据一致性的问题了(此处暂时不会深入解释)

Innodb的事务日志是指Redo log，简称Log,保存在日志文件ib_logfile里面（去mysql数据目录下看下）。

Innodb还有另外一个日志Undo log，但Undo log是存 放在共享表空间里面的（ibdata*文件，存储的是check point日志序列号）。




# 5. 事务重做日志与回滚日志

```
-- 查看事务日志 ：
show engine innodb status\G;
-- 查看日志文件设置状态
show variables like 'innodb_%';
```

innodb_log_files_in_group:DB中设置几组事务日志，默认是2； innodb_log_group_home_dir:事务日志存放目录，不设置，ib_logfile0...存在在数据文件目录下
Innodb存储引擎可将所有数据存放于ibdata*的共享表空间，也可将每张表存放于独立的.ibd文件的独立表空间

注意：在MySQL中对于数据来说， 最为重要的是日志文件

redo log => ib_logfile0

undo log => ibdata

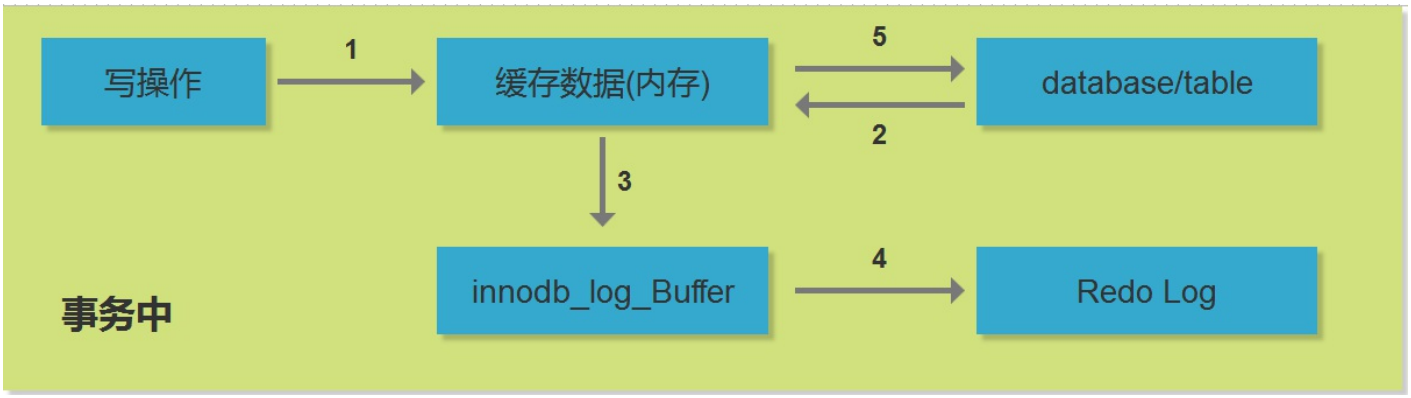
## 5.1 重做日志

### 5.1.1 持久化

事务被提交，数据一定会被写入到数据库中并持久存储起来，通常来说当事务已经被提交之后，就无法再次回滚了。

### 5.1.2 重做日志实现持久化

与原子性一样，事务的持久性也是通过日志来实现的，MySQL 使用重做日志（redo log）实现事务的持久性，重做日志由两部分组成，一是内存中的重做日志缓冲区，因为重做日志缓冲区在内存中，所以它是易失的，另一个就是在磁盘上的重做日志文件，它是持久的。




当我们在一个事务中尝试对数据进行写时，它会先将数据从磁盘读入内存，并更新内存中缓存的数据，然后生成一条重做日志并写入重做日志缓存，当事务真正提交时，MySQL 会将重做日志缓存中的内容刷新到重做日志文件，再将内存中的数据更新到磁盘上，图中的第 4、5 步就是在事务提交时执行的。

> 重做日志执行时间

在mysql中事务执行commit提交了之后，但是服务器挂了，数据还没有写入磁盘，在mysql重启服务之后会重新执行这个重做日志写入数据。

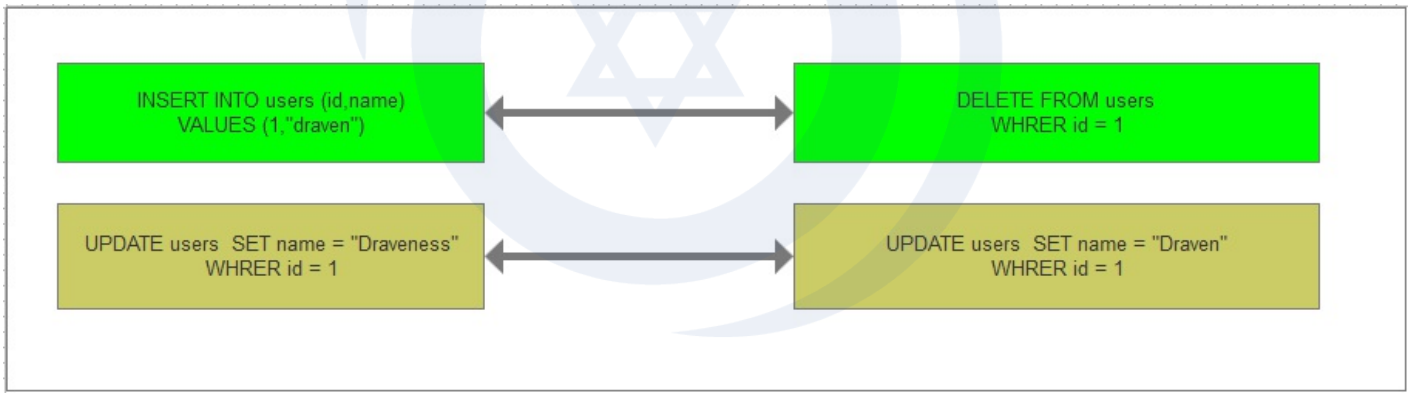
## 5.1 回滚日志

### 5.2.1 原子性

通俗的解释就是：一条绳子上的蚂蚱 专业点：事务就是一系列的操作，要么全部都执行，要都不执行

### 5.2.2 回滚日志实现原子性

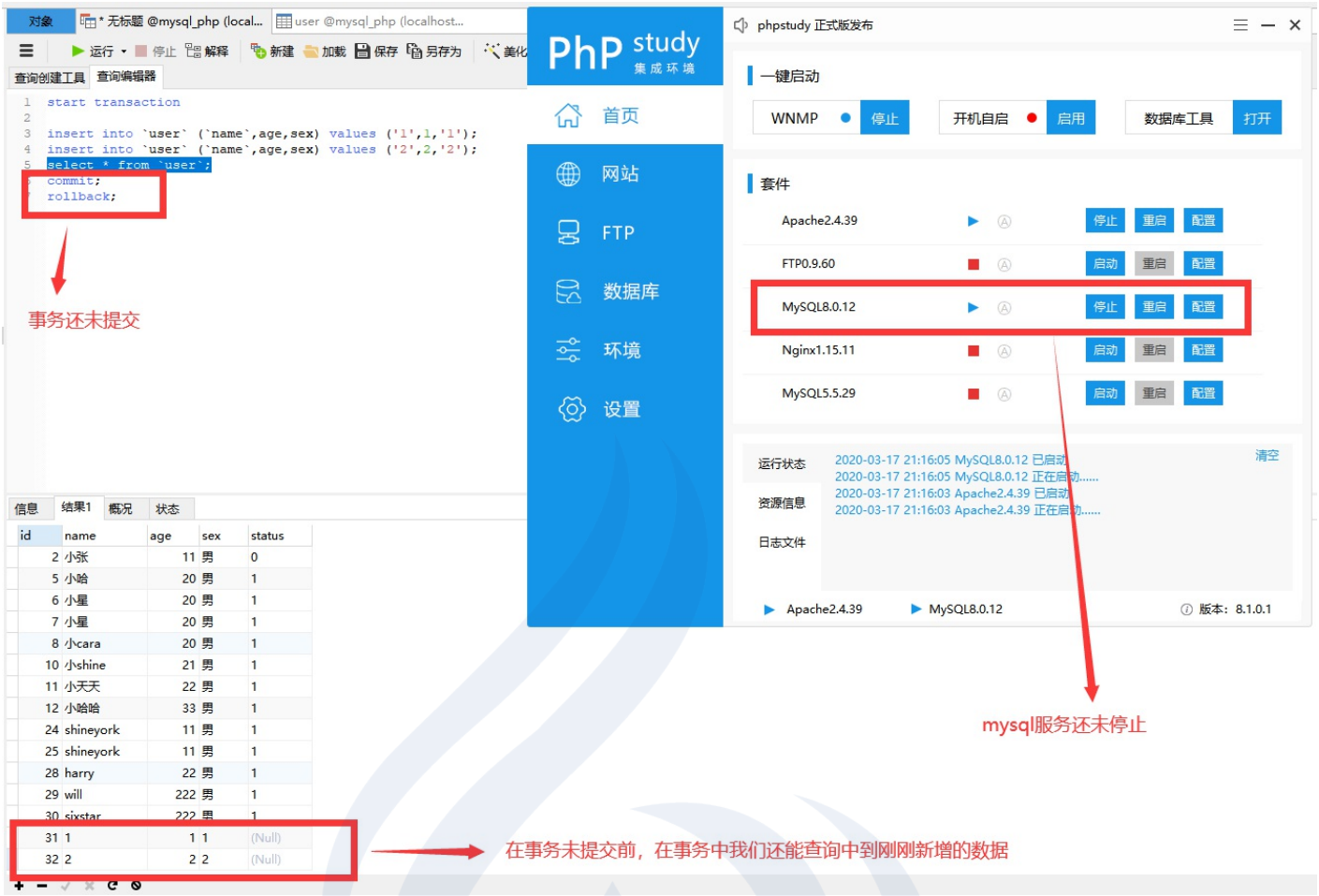
想要保证事务的原子性，就需要在异常发生时，对已经执行的操作进行回滚，而在 MySQL 中，恢复机制是通过回滚日志（undo log）实现的，所有事务进行的修改都会先记录到这个回滚日志中，然后在对数据库中的对应行进行写入。 注意：系统发生崩溃、数据库进程直接被杀死后，当用户再次启动数据库进程时，还能够立刻通过查询回滚日志将之前未完成的事务进行回滚，这也就需要回滚 日志必须先于数据持久化到磁盘上，是我们需要先写日志后写数据库的主要原因。 在日志文件中:在事务中使用的每一条 INSERT 都对应了一条 DELETE，每一条 UPDATE 也都对应一条相反的 UPDATE 语句。




> 回滚日志执行时间

1. 手动执行回滚命令时会执行
2. 如果程序在事务执行之后，提交命令执行之前出现了异常，在下次mysql服务重启的时候会执行

测试： 在事务提交前停止mysql服务：




| id | name | age | sex | status |
|---|---|---|---|---|
| 2 | 小张 | 11 | 男 | 0 |
| 5 | 小哈 | 20 | 男 | 1 |
| 6 | 小星 | 20 | 男 | 1 |
| 7 | 小星 | 20 | 男 | 1 |
| 8 | 小cara | 20 | 男 | 1 |
| 10 | 小shine | 21 | 男 | 1 |
| 11 | 小天天 | 22 | 男 | 1 |
| 12 | 小哈哈 | 33 | 男 | 1 |
| 24 | shineyork | 11 | 男 | 1 |
| 25 | shineyork | 11 | 男 | 1 |
| 28 | harry | 22 | 男 | 1 |
| 29 | will | 222 | 男 | 1 |
| 30 | sixstar | 222 | 男 | 1 |
| 31 | 1 | 1 | 1 | (Null) |
| 32 | 2 | 2 | 2 | (Null) |

停止mysql服务：


再一次开启mysql服务后查询我们新增的数据：

```
start transaction

insert into `user` (`name`,age,sex) values ('1',1,'1');
insert into `user` (`name`,age,sex) values ('2',2,'2');
select * from `user`;
commit;
rollback;
```

信息 | 结果1 | 概况 | 状态

| id | name | age | sex | status |
|---|---|---|---|---|
| 1 | 小哈 | 11 | 女 | 1 |
| 2 | 小张 | 11 | 男 | 0 |
| 5 | 小哈 | 20 | 男 | 1 |
| 6 | 小星 | 20 | 男 | 1 |
| 7 | 小星 | 20 | 男 | 1 |
| 8 | 小cara | 20 | 男 | 1 |
| 10 | 小shine | 21 | 男 | 1 |
| 11 | 小天天 | 22 | 男 | 1 |
| 12 | 小哈哈 | 33 | 男 | 1 |
| 24 | shineyork | 11 | 男 | 1 |
| 25 | shineyork | 11 | 男 | 1 |
| 28 | harry | 22 | 男 | 1 |
| 29 | will | 222 | 男 | 1 |
| 30 | sixstar | 222 | 男 | 1 |

未查询到事务新增的数据。

## 5.3 重做日志与回滚日志

到现在为止我们了解了 MySQL 中的两种日志，回滚日志（undo log）和重做日志（redo log）；在数据库系统中，事务的原子性和持久性是由事务日志（transaction log）保证的，在实现时也就是上面提到的两种日志，前者用于对事务的影响进行撤销，后者在错误处理时对已经提交的事务进行重做，它们能保证两点：

1. 发生错误或者需要回滚的事务能够成功回滚（原子性）；
2. 在事务提交后，数据没来得及写会磁盘就宕机时，在下次重新启动后能够成功恢复数据（持久性）； 在数据库中，这两种日志经常都是一起工作的，我们可以将它们整体看做一条事务日志，其中包含了事务的 ID、修改的行元素以及修改前后的值。