MySQL事务

1.事务简介

(1)在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务。
(2)事务处理可以用来维护数据库的完整性，保证成批的 SQL 语句要么全部执行，要么全部不执行。
(3)事务用来管理 insert,update,delete 语句。

2.事务四大特征

一般来说，事务是必须满足4个条件（ACID）：

（1）原子性（Atomicity）

一个事务（transaction）中的所有操作，**要么全部完成，要么全部不完成**，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

（2）一致性（Consistency）

在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。(**比如：A向B转账，不可能A扣了钱，B却没有收到**)

（3）隔离性（Isolation）

数据库**允许多个并发事务同时对其数据进行读写和修改的能力**，隔离性可以防止多个事务并发执行时由于**交叉执行而导致数据的不一致**。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。（比u人：A正在从一张银行卡里面取钱，在A取钱的过程中，B不能向这张银行卡打钱）

（4）持久性（Durability）

事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

3.事务提交、回滚

```
-- UNSIGNED代表无符号数，不能是负数
create table user(
    id int primary key auto_increment,
    name VARCHAR(20),
    balance DECIMAL(10,2) UNSIGNED
);

insert into user VALUES (1,'楠哥',200);
insert into user VALUES (2,'楠哥老婆',50000);
```

```
-- 转账业务，必须都成功，或者都失败，所以不能一句一句执行，万一执行了一半，断电了咋办
-- 所以要编程一个整体
-- 都成功
-- begin;
start transaction;
UPDATE user set balance = balance - 200 where id = 1;
UPDATE user set balance = balance + 200 where id = 2;
commit;

-- 都失败
start transaction;
UPDATE user set balance = balance - 200 where id = 1;
UPDATE user set balance = balance + 200 where id = 2;
rollback;
```

3.1、实现的原理简单介绍

mysql每执行一条语句记录一条日志，

1、start transaction，先记个日志，真正执行执行。

2、UPDATE user set balance = balance - 200 where id = 1，先记个日志，真正执行。

2.1如果此时断电了，当然不能继续执行了，过了一会来电了，启动mysql会检查日志，发现有个事务没有执行完毕，没有commit，就会安装反向的操作把他回滚了。

3、UPDATE user set balance = balance + 200 where id = 2，先记个日志，真正执行。

4、如commit，记个记录，执行，结束了，日志就能删除了。如果rollback，就会按照日志反向操作，回滚。

4.事务特性--隔离性

隔离强调的是两个或两个以上同时发生（并发）的业务同时操作一个数据库，为了让两个事务一方面能都看到、得到正确的结果，一方面还要保证一定的效率而产生的不同的隔离级别。

4.1 隔离性有隔离级别(4个)

（1）读未提交：read uncommitted
（2）读已提交：read committed
（3）可重复读：repeatable read
（4）串行化：serializable

	脏读	不可重复读	幻读
Read uncommitted	√	√	√
Read committed	×	√	√
Repeatable read	×	×	√
Serializable	×	×	×

查看个设置事务的隔离级别：

```
SELECT @@global.tx_isolation, @@tx_isolation;

set session transaction isolation level repeatable read;

SET  transaction isolation level read uncommitted;
SET  transaction isolation level read committed;
set  transaction isolation level repeatable read;
SET  transaction isolation level serializable;

SET GLOBAL transaction isolation level read uncommitted;
SET GLOBAL transaction isolation level read committed;
set GLOBAL transaction isolation level repeatable read;
SET GLOBAL transaction isolation level serializable;

其中，SESSION 和 GLOBAL 关键字用来指定修改的事务隔离级别的范围：
SESSION：表示修改的事务隔离级别将应用于当前 session（当前 cmd 窗口）内的所有事务；
GLOBAL：表示修改的事务隔离级别将应用于所有 session（全局）中的所有事务，且当前已经存在的 session 不受影响；
如果省略 SESSION 和 GLOBAL，表示修改的事务隔离级别将应用于当前 session 内的下一个还未开始的事务。
```

4.2 读未提交

- 事物A和事物B，事物A未提交的数据，事物B可以读取到
- 这里读取到的数据叫做“脏数据”，叫脏读
- 这种隔离级别最低，这种级别一般是在理论上存在，数据库隔离级别一般都高于该级别

简而言之第一个事务没提交，别的事物就能读，这种数据不一定是正确的因为人家可能回滚呀！

案例：

楠哥发工资了，老婆让楠哥把工资打到他老婆的账号上，但是该事务并未提交，就让老婆去查看，老婆一看真的打了钱了，高高兴兴关了网页，此时楠哥急中生智进行回滚，钱瞬间回来，一次蒙混了一个月工资。所以楠哥老婆看到的数据我们称之为“脏数据”。

必须开两个事务

```
use test;
SET  transaction isolation level read uncommitted;
```

1-楠哥，转账

```
start transaction;
UPDATE user set balance = balance - 10000 where id = 1;
UPDATE user set balance = balance + 10000 where id = 2;
```

2-楠哥老婆，查账，不错，钱已到账

```
start transaction;
select * from user where id = 2;
commit;
```

3-楠哥,回马枪，回滚

```
rollback;
```

4-楠哥老婆某天查账，哎，怎么少了一万

```
start transaction;
select * from user where id = 2;
commit;
```

出现上述情况，即我们所说的脏读 ，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。

4.3 读已提交

能读到别的事物已经提交的数据。

A事务在本次事务中，对自己操作过的数据，进行了多次读取发现数据不一致，不可重复读。

简单点说就是不能让我好好的重复读，一个事务里读出来的数据都不一样，让不让人干活了。

针对的语句update和delete，会导致不可重复读

楠哥拿着工资卡去消费，系统读取到卡里确实有10200元，而此时她的老婆也正好在网上转账，把楠哥工资卡的2000元转到另一账户，并在 楠哥之前提交了事务，当楠哥扣款时，系统检查到楠哥的工资卡和上次读取的不一样了，楠哥十分纳闷，明明卡里有钱，为何.....

```
SET  transaction isolation level read committed;
```

1-楠哥去消费了，显示有余额，贼高兴

```
start transaction;
select * from user where id = 1;
```

2-老婆转账

```
start transaction;
UPDATE user set balance = balance + 10000 where id = 2;
UPDATE user set balance = balance - 10000 where id = 1;
commit;
```

3-楠哥查账，同一个事务里，发现钱少了。

```
select * from user where id = 1;
```

当隔离级别设置为Read committed 时，避免了脏读，但是可能会造成不可重复读。

大多数数据库的默认级别就是Read committed，比如Sql Server , Oracle。如何解决不可重复读这一问题，请看下一个隔离级别。

4.4 可重复读

A事务在本次事务中对未操作的数据进行多次查询，发现第一次没有，第二次出现了就像幻觉一样。或者第一次有而第二次没有。针对delete和insert。

案例

楠哥的老婆在银行部门工作，她时常通过银行内部系统查看楠哥的账户信息。有一天，她正在查询到楠哥账户信息时发现楠哥只有一个账户，心想这家伙应该没有私房钱。此时楠哥在另外一家分行右开了一个账户，准备存私房钱。一次同时楠哥老婆点击了打印，结果打印出的楠哥账户居然多了一个，真实奇怪。

```
set  transaction isolation level repeatable read;
```

1-楠哥开启事务

```
start transaction;
```

2-老婆查账户

```
start transaction;
select * from user where name = '楠哥';
```

3-楠哥趁机开户

```
insert into user values(3,'楠哥',10000);
commit;
```

4-老婆再查询并打印，应该发现楠哥多了一个账户，但是没有。

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

MySQL 通过多版本并发控制（MVCC）（快照读/一致性读）其实解决了幻读问题。

原理：事务开启后，将历史数据存一份快照，其他事务增加与删除的数据，对于当前事务来说是不可见的。

当然还能这样测一下

```
set  transaction isolation level repeatable read;
```

1-楠哥开启事务

```
start transaction;
```

2-老婆查账户，给楠哥开了个账户

```
start transaction;
select * from user where name = '楠哥';
insert into user values(3,'楠哥',10000);
```

3-楠哥不知道老婆给他开了账户，自己也开一个，看见自己没有这个3号账户，居然不能插入，很奇幻。

```
select * from user where name = '楠哥';
insert into user values(3,'楠哥',10000);
```

4.5 串行化

- 事务A和事务B，事务A在操作数据库时，事务B只能排队等待
- 这种隔离级别很少使用，吞吐量太低，用户体验差
- 这种级别可以避免“幻像读”，每一次读取的都是数据库中真实存在数据，事务A与事务B串行，而不并发。
- 别的地方一用这个数据就不能修改删除，直到别的地方提交

```
SET  transaction isolation level serializable;
```

1-楠哥

```
begin;
select * from user;
```

2-老婆

```
begin;
select * from user;
```

3-楠哥操作发现卡住了

```
delete from user where id = 9;
```

4-老婆这边一提交，那边就能操作了

```
commit;
```