MQL事务

事务：一个最小的不可再分的工作单元；通常一个事务对应一个完整的业务(例如银行账户转账业务，该业务就是一个最小的工作单元)

一个完整的业务需要批量的DML(insert、update、delete)语句共同联合完成

事务只和DML语句有关，或者说DML语句才有事务。这个和业务逻辑有关，业务逻辑不同，DML语句的个数不同

MySQL 事务主要用于处理操作量大，复杂度高的数据。比如说，在人员管理系统中，你删除一个人员，你既需要删除人员的基本资料，也要删除和该人员相关的信息，如信箱，文章等等，这样，这些数据库操作语句就构成一个事务！

事务是为解决数据安全操作提出的，事务控制实际上就是控制数据的安全访问。

用一个简单例子说明：银行转帐业务，账户A要将自己账户上的1000元转到B账户下面，A账户余额首先要减去1000元，然后B账户要增加1000元。假如在中间网络出现了问题，A账户减去1000元已经结束，B因为网络中断而操作失败，那么整个业务失败，必须做出控制，要求A账户转帐业务撤销。这才能保证业务的正确性，完成这个操作就需要事务，将A账户资金减少和B账户资金增加放到同一个事务里，要么全部执行成功，要么全部撤销，这样就保证了数据的安全性。

- 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。
- 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。
- 事务用来管理 insert,update,delete 语句

事务四大特征(ACID)

- 一般来说，事务是必须满足4个条件（ACID）：：原子性（**A**tomicity，或称不可分割性）、一致性（**C**onsistency）、隔离性（**I**solation，又称独立性）、持久性（**D**urability）。
    - **原子性：**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。
    - **一致性：**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。
    - **隔离性：**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）
    - **持久性：**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。

关于事务的一些术语

###

1、用 BEGIN, ROLLBACK, COMMIT来实现

- **BEGIN** 开始一个事务
- **ROLLBACK** 事务回滚
- **COMMIT** 事务确认

2、直接用 SET 来改变 MySQL 的自动提交模式:

- **SET AUTOCOMMIT=0** 禁止自动提交
- **SET AUTOCOMMIT=1** 开启自动提交

事务开启的标志？事务结束的标志？

开启标志：

```
- 任何一条DML语句(insert、update、delete)执行，标志事务的开启
```

结束标志(提交或者回滚)：

```
- 提交：成功的结束，将所有的DML语句操作历史记录和底层硬盘数据来一次同步
- 回滚：失败的结束，将所有的DML语句操作历史记录全部清空
```

在 MySQL 命令行的默认设置下，事务都是自动提交的，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT=0，用来禁止使用当前会话的自动提交。

```
1、关闭自动提交
SET autocommit=0;
2、开启事务
START TRANSACTION;

3、事务语句
ALTER TABLE girls MODIFY gname VARCHAR(20) NOT NULL;
INSERT INTO girls VALUES(0,'张三123');

4、明显的结束标记
COMMIT;  #提交
ROLLBACK;  #回滚
SELECT * FROM girls;
#查看

注意点：
1、关闭自动提交
SET autocommit=0;
2、开启事务
START TRANSACTION;
3、需要执行的sql代码
sql1;
sql2;
.......
4、明显的结束标志
commit / rollback
```

代码

```
mysql> use RUNOOB;
Database changed
mysql> CREATE TABLE runoob_transaction_test( id int(5)) engine=innodb;  # 创建数据表
Query OK, 0 rows affected (0.04 sec)

mysql> select * from runoob_transaction_test;
Empty set (0.01 sec)

mysql> begin;  # 开始事务
Query OK, 0 rows affected (0.00 sec)

mysql> insert into runoob_transaction_test value(5);
Query OK, 1 rows affected (0.01 sec)

mysql> insert into runoob_transaction_test value(6);
Query OK, 1 rows affected (0.00 sec)
```

```
mysql> commit; # 提交事务
Query OK, 0 rows affected (0.01 sec)

mysql>  select * from runoob_transaction_test;
+------+
| id   |
+------+
| 5    |
| 6    |
+------+
2 rows in set (0.01 sec)

mysql> begin;    # 开始事务
Query OK, 0 rows affected (0.00 sec)

mysql>  insert into runoob_transaction_test values(7);
Query OK, 1 rows affected (0.00 sec)

mysql> rollback;   # 回滚
Query OK, 0 rows affected (0.00 sec)

mysql>   select * from runoob_transaction_test;   # 因为回滚所以数据没有插入
+------+
| id   |
+------+
| 5    |
| 6    |
+------+
2 rows in set (0.01 sec)

mysql>
```

脏读,不可重复读,幻读

事前准备

事前准备数据

```
mysql> create table city(
    -> id int(10) auto_increment,
    -> name varchar(30),
    -> primary key (id)
    -> )engine=innodb charset=utf8mb4;

insert into city(name) values('武汉市');

mysql> select * from city;
+----+-----------+
| id | name |
+----+-----------+
| 1 | 武汉市 |
+----+-----------+
```

脏读

> 会话B开启一个事务，把id=1的name为武汉市修改成温州市，此时另外一个会话A也开启一个事务，读取id=1的name，此时的查询结果为温州市，会话B的事务最后回滚了刚才修改的记录，这样会话A读到的数据是不存在的，这个现象就是脏读。（脏读只在读未提交隔离级别才会出现）

不可重复读

> 会话A开启一个事务，查询id=1的结果，此时查询的结果name为武汉市。接着会话B把id=1的name修改为温州市（隐式事务，因为此时的autocommit为1，每条SQL语句执行完自动提交），此时会话A的事务再一次查询id=1的结果，读取的结果name为温州市。会话B再此修改id=1的name为杭州市，会话A的事务再次查询id=1，结果name的值为杭州市，这种现象就是不可重复读。

幻读

幻读（Phantom）

> 一个事务先根据某些条件查询出一些记录，之后另一个事务又向表中插入了符合这些条件的记录，原先的事务再次按照该条件查询时，能把另一个事务插入的记录也读出来。（幻读在读未提交、读已提交、可重复读隔离级别都可能会出现）

> 会话A开启一个事务，查询id>0的记录，此时会查到name=武汉市的记录。接着会话B插入一条name=温州市的数据（隐式事务，因为此时的autocommit为1，每条SQL语句执行完自动提交），这时会话A的事务再以刚才的查询条件（id>0）再一次查询，此时会出现两条记录（name为武汉市和温州市的记录），这种现象就是幻读。

查看隔离级别

查看事务隔离级别

在 MySQL 中，可以通过 `show variables like '%tx_isolation%'` 或 `select @@tx_isolation;` 语句来查看当前事务隔离级别。

查看当前事务隔离级别的 SQL 语句和运行结果如下：

```
mysql> show variables like '%tx_isolation%';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+
1 row in set, 1 warning (0.17 sec)
mysql> select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set, 1 warning (0.00 sec)
```

结果显示，目前 MySQL 的事务隔离级别是 REPEATABLE-READ。

事务的四种隔离级别

> 数据库内部定义了四种隔离级别，用于解决三种隔离问题

1、Serializable：可避免脏读、不可重复读、虚读情况的发生。（串行化、序列化）

2、Repeatable Read：可避免脏读、不可重复读情况的发生，不可以避免虚读（可重复读）

3、Read Committed：可避免脏读情况发生（读已提交）

4、Read uncommitted：最低级别，以上情况均无法保证。(读未提交)

MySQL的隔离级别的作用就是让事务之间互相隔离，互不影响，这样可以保证事务的一致性。

隔离级别比较：可串行化>可重复读>读已提交>读未提交

隔离级别对性能的影响比较：可串行化>可重复读>读已提交>读未提交

由此看出，隔离级别越高，所需要消耗的MySQL性能越大（如事务并发严重性），为了平衡二者，一般建议设置的隔离级别为可重复读，MySQL默认的隔离级别也是可重复读。

读未提交

读未提交（READ UNCOMMITTED）

> 在读未提交隔离级别下，事务A可以读取到事务B修改过但未提交的数据。
>
> 可能发生脏读、不可重复读和幻读问题，一般很少使用此隔离级别。

读已提交

读已提交（READ COMMITTED）

> 在读已提交隔离级别下，事务B只能在事务A修改过并且已提交后才能读取到事务B修改的数据。
>
> 读已提交隔离级别解决了脏读的问题，但可能发生不可重复读和幻读问题，一般很少使用此隔离级别。

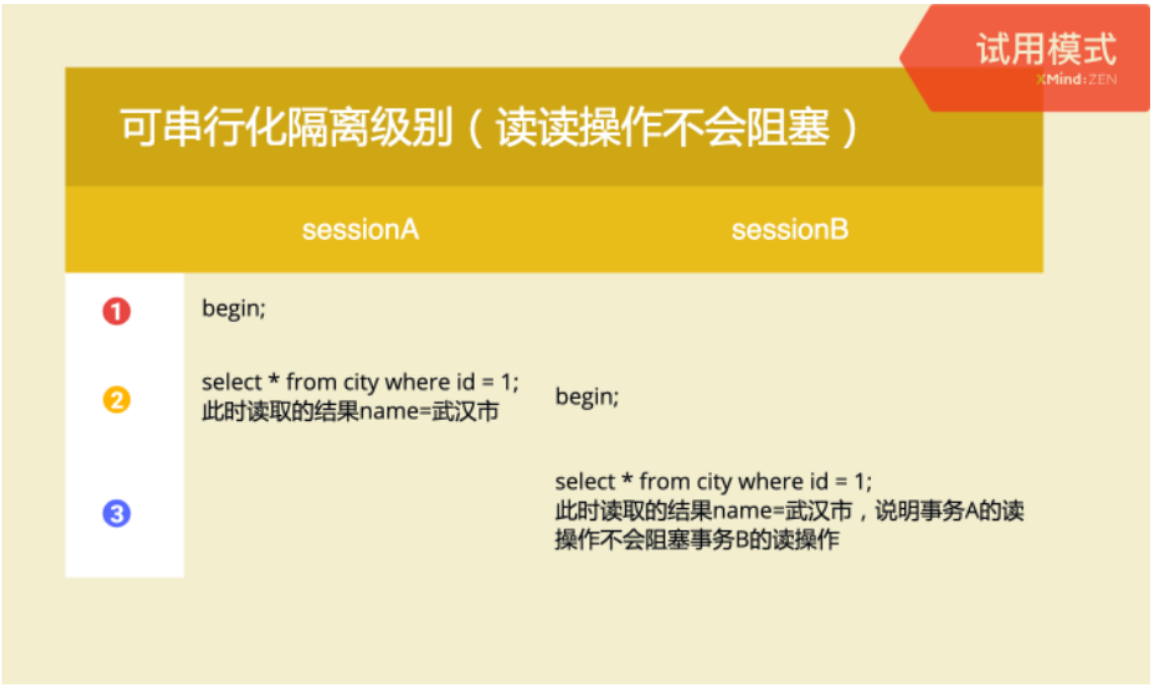
可重复读

> 在可重复读隔离级别下，事务B只能在事务A修改过数据并提交后，自己也提交事务后，才能读取到事务B修改的数据。
>
> 可重复读隔离级别解决了脏读和不可重复读的问题，但可能发生幻读问题。
>
> 提问：为什么上了写锁（写操作），别的事务还可以读操作?
>
> 因为InnoDB有MVCC机制（多版本并发控制），可以使用快照读，而不会被阻塞。

可串行化

读读操作

可串行化（SERIALIZABLE）



读写操作

写读操作

写写操作

各种问题（脏读、不可重复读、幻读）都不会发生，通过加锁实现（读锁和写锁）。

阻塞区别


试用模式
XMind:ZEN
可串行化隔离级别的阻塞区别
事务A读操作
事务A写操作
事务B读操作
不阻塞
阻塞
事务B写操作
阻塞
阻塞