

事务

1.1 含义

1.1.1 事务控制语言

- transaction control language 事务控制语言
- 事务：一个或者一组sql语句组成一个执行单元，要么这个执行单元全部执行，要么全部不执行，事务类似原子操作，执行过程中不可被外界打断
- 案例：转账

```
张三丰 余额 1000      郭襄 余额 1000
update 表 set 张三丰的余额=500 where name='张三丰'
意外
update 表 set 郭襄的余额=1500 where name='郭襄'
```

事务：事务由单独单元的一个或多个SQL语句组成，在这个单元中，每个MySQL语句是相互依赖的。而整个单独单元作为一个不可分割的整体，如果单元中某条SQL语句一旦执行失败或产生错误，整个单元将会回滚。所有受到影响的数据将返回到事物开始以前的状态；如果单元中的所有SQL语句均执行成功，则事物被顺利执行。

1.1.2 存储引擎

1. 概念：在mysql中的数据用各种不同的技术存储在文件（或内存）中。
2. 通过 `show engines`，来查看mysql支持的存储引擎。
3. 在mysql中用的最多的存储引擎有：`innodb`，`myisam`，`memory` 等。其中 `innodb` 支持事务，而 `myisam`、`memory` 等不支持事务
4. 不是所有的存储引擎都支持事务

1.2 特点

1. **A 原子性**：Atomicity 一个事务是不可再分割的整体，要么都执行要么都不执行
2. **C 一致性**：Consistency 一个事务可以使数据从一个一致状态切换到另外一个一致的状态
3. **I 隔离性**：Isolation 一个事务不受其他事务的干扰，多个事务互相隔离的
4. **D 持久性**：Durability 一个事务一旦提交了，则永久的持久化到本地

1.3 事务使用步骤

1.3.1 隐式事务

1. 事务没有明显的开启和结束的标记，比如 `insert`、`update`、`delete` 语句

```
SHOW VARIABLES LIKE 'autocommit';
```

<input type="checkbox"/>	Variable_name	Value
<input type="checkbox"/>	autocommit	ON

1.3.2 显式事务

1. 事务具有明显的开启和结束的标记；前提：必须先设置自动提交功能为禁用

```
SET autocommit=0;
```

2. 操作步骤

```
# 创建表
CREATE TABLE myaccount(
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(20),
    balance DOUBLE);

# 插入数据
INSERT INTO myaccount(username,balance)
VALUES('join',1000),('mark',1000);

#步骤1: 开启事务
SET autocommit=0;
START TRANSACTION;

#步骤2: 编写一组事务的语句(select insert update delete)
UPDATE account SET balance = 1000 WHERE username='join';
UPDATE account SET balance = 1500 WHERE username='mark';

#步骤3: 结束事务
ROLLBACK;          # 回滚事务
commit;            # 提交事务
```

```
savepoint 节点名;          # 设置保存点
#演示savepoint 的使用
SET autocommit=0;
START TRANSACTION;
DELETE FROM account WHERE id=25;
SAVEPOINT a;              # 设置保存点
DELETE FROM account WHERE id=28;
ROLLBACK TO a;            # 回滚到保存点
```

1.4 并发事务

1. 对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取必要的隔离机制，就会导致各种并发问题：

- 脏读: 对于两个事务 T1, T2, T1 读取了已经被 T2 更新但还没有被提交的字段.之后, 若 T2 回滚, T1读取的内容就是临时且无效的
- 不可重复读: 对于两个事务T1, T2, T1 读取了一个字段, 然后 T2更新了该字段.之后, T1再次读取同一个字段, 值就不同了.

c. 幻读：对于两个事务T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中插入了一些新的行. 之后, 如果 T1 再次读取同一个表, 就会多出几行.

2. 数据库事务的隔离性: 数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题.

3. 一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱.

4. 数据库提供的4种隔离级别

隔离级别	描述
READ UNCOMMITTED (读未提交的数据)	允许事务读取未被其他事务提交的变更, 脏读, 不可重复读和幻读的问题都会出现
READ COMMITTED (读已提交数据)	只允许事务读取已经被其他事务提交的变更。可以避免脏读, 但不可重复读和幻读问题仍然可能出现
REPEATABLE READ (可重复读)	确保事务可以从一个字段中读取相同的值。在这个事务持续期间, 禁止其他事务对这个字段进行更新。可以避免脏读和不可重复读, 但幻读的问题仍然存在
SERIALIZABLE (串行化)	确保事务可以从一个表中读取相同的行。在这个事务持续期间。禁止其他事务对该表执行插入, 更新和删除操作。所有并发问题都可以避免, 但性能十分低下

- Oracle支持的2种事务隔离级别: `READ COMMITTED`, `SERIALIZABLE`。Oracle默认的事务隔离级别为: `READ COMMITTED`
- Mysql支持4种事务隔离级别。mysql默认的事务隔离级别为: `REPEATABLE READ`

5. 在 MySql 中设置隔离级别

- 每启动一个 mysql 程序, 就会获得一个单独的数据库连接, 每个连接都有一个全局变量 `@@tx_isolation`, 表示当前的事务隔离级别

```
# 查看当前的隔离级别:
SELECT @@tx_isolation;
# 在MySQL 8.0.3 中, 变量 tx_isolation 已经被 transaction_isolation 替换了。
# 设置当前 mysql 连接的隔离级别:
set session transaction isolation level read committed;
# 设置数据库系统的全局的隔离级别:
set global transaction isolation level read committed;
```

```
mysql> select @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
1 row in set, 1 warning (0.00 sec)
```

6. 事务的隔离级别

	脏读	不可重复读	幻读
<code>read uncommitted:</code>	✓	✓	✓
<code>read committed:</code>	×	✓	✓
<code>repeatable read:</code>	×	×	✓
<code>serializable:</code>	×	×	×

1.5 数据库隔离级别演示

1.5.1 read uncommitted

```
# session 1
mysql> select @@tx_isolation;           # 查看隔离级别默认为 REPEATABLE-READ
mysql> set session transaction isolation level read uncommitted;
                                           # 设置隔离级别为read uncommitted;

mysql> set autocommit=0;                 # 开启事务
mysql> update account set username='xixihaha123' where id = 3;
                                           # 修改字段

# session 2
mysql> select @@tx_isolation;           # 查看session2隔离级别默认也为
REPEATABLE-READ
mysql> set session transaction isolation level read uncommitted;
mysql> set autocommit=0;                 # 开启事务
mysql> select * from account;            # 查看数据表account
```

1. session2 查看数据表 account 是 session1 更改但还没有提交的数据，session1 回滚，session2 读取的内容就是临时且无效的

1.5.2 read committed

1. 设置隔离级别为 read committed
2. session1 更改数据表中的字段但未提交，session2 读取的数据是 session1 更改前的数据
3. session1 提交后，session2 读取的数据是 session1 提交之后的数据
4. read committed 避免了脏读，不可重复与幻读无法解决

1.5.3 repeatable read

1. 设置隔离级别为 repeatable read
 2. session1 更改数据表中的字段，session2 在同一个事务中多次查询结果都是 session1 更改前的数据
 3. repeatable read 避免了脏读和不可重复读，但幻读无法解决
 4. 幻读：session1 插入一条数据，session2 更改数据时实际多更改了一条
- session1

```
mysql> select * from myaccount;
+----+-----+-----+
| id | username | balance |
+----+-----+-----+
| 1  | join    | 1000    |
| 2  | mark    | 1000    |
+----+-----+-----+
2 rows in set (0.00 sec)

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

mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into myaccount values(null,'xixihaha123', 1500);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

- session2

```
mysql> update myaccount set username='Nihaowa123';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3 Changed: 3 Warnings: 0
```

1.5.4 serializable

- session1 提交之后, session2才能更改, 提交

```
mysql> insert into myaccount values(null, 'xxxxx', 2000);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.09 sec)
```

- session2

```
mysql> set session transaction isolation level serializable;
Query OK, 0 rows affected (0.00 sec)

mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> update myaccount set username='zzzzzz';
Query OK, 4 rows affected (7.65 sec)
Rows matched: 4 Changed: 4 Warnings: 0
```