MySQL事务

为什么要使用事务?

在执行`SQL`语句时，某些业务要求，一系列操作必须全部执行，而不能仅执行一部分。例如一个转账操作：

```
-- 从id=1的账户给id=2的账户转账100元
-- 第一步：将id=1的A账户余额减去100
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
-- 第二步：将id=2的B账户余额加上100
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
```

这两条`SQL`语句必须全部执行，或者由于某些原因，第一条语句执行成功，第二条语句执行失败，两条语句就必须全部撤销。如果没有使用事务，第一条并没有撤销，id=1的人转账给id=2的人，扣了100元，而由于某些原因第二条语句执行失败，id=2的人并没有得到一百元，这样的话就出现了很大的问题。

这种把多条语句作为一个整体进行操作的功能，被称为**数据库事务**。数据库事务可以确保该事务范围内的所有操作都可以全部成功或者全部失败。如果事务失败，那么效果就和没有执行这些`SQL`一样，不会对数据库数据有任何改动。

事务的4个特性

事务拥有`ACID`这4个特性：

- A：Atomic，原子性，将所有`SQL`作为原子工作单位执行，要么全部执行，要么全部不执行，如上例中两条语句可看作在同一个原子；
- C：Consistent，一致性，事务完成后，所有数据的状态都是一致性的，即A账户减去100，那么B账户就必须加上100；
- I：Isolation，隔离性，如果有多个事务并发执行，每个事务作出的修改必须与其他事务隔离；
- D：Duration，持久性，即事务完成后，对数据库的修改被持久化存储。

隐式事务和显式事务

隐式事务

单条`SQL`语句，数据库系统自动将其作为一个事务执行，这种事务被称为**隐式事务**。

显式事务

要手动把多条`SQL`语句作为一个事务执行，使用`BEGIN`开启一个事务，使用`COMMIT`提交一个事务，这种事务被称为**显式事务**，例如，把上述的转账操作作为一个显式事务：

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

很显然多条SQL语句要想作为一个事务执行，就必须使用显式事务。

`COMMIT`是指提交事务，即试图把事务内的所有SQL所做的修改永久保存。如果`COMMIT`语句执行失败了，整个事务也会失败。

有些时候，我们希望主动让事务失败，这时，可以用`ROLLBACK`回滚事务，整个事务会失败：

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
ROLLBACK;
```

隔离级别

对于**两个并发执行的事务**，如果涉及到操作同一条记录的时候，可能会发生问题。因为并发操作会带来数据的*不一致性，包括**脏读**、**不可重复读**、**幻读***等。数据库系统提供了隔离级别来让我们有针对性地选择事务的隔离级别，避免数据不一致的问题。

`SQL`标准定义了4种隔离级别，分别对应可能出现的数据不一致的情况：

ISOLATION LEVEL	脏读（Dirty Read）	不可重复读（Non Repeatable Read）	幻读（Phantom Read）
Read Uncommitted	Yes	Yes	Yes
Read Committed	-	Yes	Yes
Repeatable Read	-	-	Yes
Serializble	-	-	-

如果没有指定隔离级别，数据库就会使用默认的隔离级别。在`MySQL`中，如果使用`InnoDB`引擎，**默认的隔离级别**是**Repeatable Read。**

脏读

隔离级别最低的一种事务级别。在这种隔离级别下，一个事务会读到另一个事务更新后但未提交的数据，如果另一个事务回滚，那么当前事务读到的数据就是脏数据，这就是脏读（Dirty Read）。

首先，我们准备好`students`表的数据，该表仅一行记录：

```
mysql> select * from students;
+----+-------+
| id | name  |
+----+-------+
|  1 | Alice |
+----+-------+
1 row in set (0.00 sec)
```

然后，分别开启两个`MySQL`客户端连接，按顺序依次执行事务A和事务B，同时事务A和事务B是并发的：

时刻	事务A	事务B
1	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
2	BEGIN;	BEGIN;
3	UPDATE students SET name = 'Bob' WHERE id = 1;	
4		SELECT * FROM students WHERE id = 1;
5	ROLLBACK;	
6		SELECT * FROM students WHERE id = 1;
7		COMMIT;

当事务A执行完第三步时，将id=1的记录name列改为Bob，然后紧接着事务B执行第四步，读取了id=1的记录，此时该条记录的name列为Bob,然后事务A回滚了，相当于name还是原来的没有被改变。接下来当事务B执行第六步时，读出来的数据name还是原来的，和事务B第四步读出来的不一样，这种情况称为**脏读。**可见一个事务可能读取到另一个事务更新但未提交的数据，这个数据有可能是**脏数据。**

不可重复读

不可重复读是指，在一个事务内，多次读同一数据，在这个事务还没有结束时，如果另一个事务恰好修改了这个数据，那么，在第一个事务中，两次读取的数据就可能不一致。

我们仍然先准备好`students`表的数据：

```
mysql> select * from students;
+----+-------+
| id | name  |
+----+-------+
|  1 | Alice |
+----+-------+
1 row in set (0.00 sec)
```

然后，分别开启两个MySQL客户端连接，按顺序依次执行事务A和事务B：

时刻	事务A	事务B
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2	BEGIN;	BEGIN;
3		SELECT * FROM students WHERE id = 1;
4	UPDATE students SET name = 'Bob' WHERE id = 1;	
5	COMMIT;	
6		SELECT * FROM students WHERE id = 1;
7		COMMIT;

和脏读类似，不可重复读出现时，是一个事务A提交了，导致事务B两次读到的记录不一致。

幻读

幻读是指，在一个事务中，第一次查询某条记录，发现没有，但是，当试图更新这条不存在的记录时，竟然能成功，并且，再次读取同一条记录，它就神奇地出现了。

我们仍然先准备好`students`表的数据：

```
mysql> select * from students;
+----+-------+
| id | name  |
+----+-------+
|  1 | Alice |
+----+-------+
1 row in set (0.00 sec)
```

然后，分别开启两个MySQL客户端连接，按顺序依次执行事务A和事务B：

时刻	事务A	事务B
1	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2	BEGIN;	BEGIN;
3		SELECT * FROM students WHERE id = 99;
4	INSERT INTO students (id, name) VALUES (99, 'Bob');	
5	COMMIT;	
6		SELECT * FROM students WHERE id = 99;
7		UPDATE students SET name = 'Alice' WHERE id = 99;
8		SELECT * FROM students WHERE id = 99;
9		COMMIT;

先执行事务B的第三步，id=99的记录是不存在的，执行事务A的第四步插入id=99的记录，再执行事务B的接下来步骤时id=99的记录是存在的。可见，幻读就是没有读到的记录，以为不存在，但其实是可以更新成功的，并且，更新成功后，再次读取，就出现了。

待续.....