04 | 事务：账户余额如何设计，保证可靠

账户系统负责记录和管理用户账户的余额，这个余额就是每个用户临时存在电商的钱，来源可能是用户充值或者退货退款等多种途径。

账户系统的用途也非常广泛，不仅仅是电商，各种互联网内容提供商、网络游戏服务商，电信运营商等等，都需要账户系统来管理用户账户的余额，或者是虚拟货币。包括银行的核心系统，也同样包含一个账户系统。

从业务需求角度来分析，一个最小化的账户系统，它的数据模型可以用下面这张表来表示：这个表包括用户 ID、账户余额和更新时间三个字段。每次交易的时候，根据用户 ID 去更新这个账户的余额就可以了。

列名	数据类型	主键	非空	说明
user_id	BIGINT	是	是	用户ID
balance	BIGINT		是	账户余额
timestamp	DATE		是	更新时间

为什么总是对不上账?

每个账户系统都**不是孤立存在的，至少要和财务、订单、交易这些系统有着密切的关联**。理想情况下，账户系统内的数据应该是自洽的。所有用户的账户余额加起来，应该等于这个电商公司在银行专用账户的总余额。账户系统的数据也应该和其他系统的数据能对的上。

比如说，每个用户的余额应该能和交易系统中充值记录，以及订单系统中的订单对的上。不过，由于**业务和系统的复杂性**，现实情况却是，很少有账户系统能够做到一点不差的对上每一笔账。所以，稍微大型一点儿的系统，都会有一个专门的对账系统，来核对、矫正账户系统和其他系统之间的数据差异。对不上账的原因非常多，比如业务变化、人为修改了数据、系统之间数据交换失败等等。

那作为系统的设计者，我们只关注“如何避免由于技术原因导致的对不上账”就可以了，有哪些是因为技术原因导致的呢？**比如说：网络请求错误，服务器宕机、系统 Bug 等。“对不上账”是通俗的说法，它的本质问题是，冗余数据的一致性问题**。这里面的冗余数据并不是多余或者重复的数据，而是多份含有相同信息的数据。

比如，我们完全可以通过用户的每一笔充值交易数据、消费的订单数据，来计算出这个用户当前的账户余额是多少。也就是说，账户余额数据和这些账户相关的交易记录，都含有“账

户余额”这个信息，**那它们之间就互为冗余数据。**在设计系统的存储时，原则上不应该存储冗余数据，

一是浪费存储空间，

二是让这些冗余数据保持一致是一件非常麻烦的事儿。

但有些场景下存储冗余数据是必要的，比如用户账户的余额这个数据。这个数据在交易过程中会被非常频繁地用到，总不能每次交易之前，先通过所有历史交易记录计算一下当前账户的余额，这样做速度太慢了，性能满足不了交易的需求。**所以账户系统保存了每个用户的账户余额，这实际上是一种用*存储空间换计算时间*的设计**

如果说只是满足功能需求，账户系统只记录余额，每次交易的时候更新账户余额就够了。但是这样做有一个问题，如果账户余额被篡改，是没有办法追查的，所以在记录余额的同时，还需要记录每一笔交易记录，也就是账户的流水。

流水的数据模型至少需要包含：流水 ID、交易金额、交易时间戳以及交易双方的系统、账户、交易单号等信息。虽然说，流水和余额也是互为冗余数据，但是记录流水，可以有效地修正由于系统 Bug 或者人为篡改导致的账户余额错误的问题，也便于账户系统与其他外部系统进行对账，所以账户系统记录流水是非常必要的。

使用数据库事务来保证数据一致性

ACID

我们先看一下如何来使用 MySQL 的事务，实现一笔交易。比如说，在事务中执行一个充值 100 元的交易，先记录一条交易流水，流水号是 888，然后把账户余额从 100 元更新到 200 元。对应的 SQL 是这样的：

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

mysql> insert into account_log ...;  -- 写入交易流水
Query OK, 1 rows affected (0.01 sec)

mysql> update account_balance ...;  -- 更新账户余额
Query OK, 1 rows affected (0.00 sec)

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

使用事务的时候，只需要在之前执行begin，标记开始一个事务，然后正常执行多条 SQL 语句，在事务里面的不仅可以执行更新数据的 SQL，查询语句也是可以的，最后执行commit，提交事务就可以了。

我们来看一下，事务可以给我们提供什么样的保证？首先，它可以保证，记录流水和更新余额这两个操作，要么都成功，要么都失败，即使是在数据库宕机、应用程序退出等等这些异常情况下，也不会出现，只更新了一个表而另一个表没更新的情况。**这是事务的原子性（Atomic）。**

事务还可以保证，数据库中的数据总是从一个一致性状态（888 流水不存在，余额是 100 元）转换到另外一个一致性状态（888 流水存在，余额是 200 元）。对于其他事务来说，不存在任何中间状态（888 流水存在，但余额是 100 元）。其他事务，在任何一个时刻，如果它读到的流水中没有 888 这条流水记录，它读出来的余额一定是 100 元，这是交易前的状态。如果它能读到 888 这条流水记录，它读出来的余额一定是 200 元，这是交易之后的状态。也就是说，事务保证我们读到的数据（交易和流水）总是一致的，**这是事务的一致性 (Consistency)。**

实际上，这个事务的执行过程无论多快，它都是需要时间的，那修改流水表和余额表对应的数据，也会有先后。那一定存在一个时刻，流水更新了，但是余额还没更新，也就是说每个事务的中间状态是事实存在的。数据库为了实现一致性，必须保证每个事务的执行过程中，中间状态对其他事务是不可见的。比如说我们在事务 A 中，写入了 888 这条流水，但是还没有提交事务，那在其他事务中，都不应该读到 888 这条流水记录。**这是事务的隔离性 (Isolation)。**

最后，只要事务提交成功，数据一定会被持久化到磁盘中，后续即使发生数据库宕机，也不会改变事务的结果。**这是事务的持久性 (Durability)。**

你会发现，我上面讲的就是事务的 ACID 四个基本特性。你需要注意的是，这四个特性之间是紧密关联在一起的，不用去纠结每一个特性的严格定义，更重要的是理解事务的行为，也就是我们的系统在使用事务的时候，各种情况下，事务对你的数据会产生什么影响，这是使用事务的关键。理解事务的隔离级别

理解事务的隔离级别

隔离级别	脏读 (DR，Dirty Read)	不可重复读 (NR，NonRepeatable Read)	幻读 (PR，Phantom Read)
能读到未提交的数据，RU，READ-UNCOMMITTED	y	y	y
能读到已提交的数据，RC，READ-COMMITTED	N	y	y
可重复读 RR,REPEATABLE-READ	N	N	y
串行执行 SERIALIZABLE	N	N	N