Day08

需求说明

数据库事务

- 概述
- 事务的例子
- ACID
- 事务相关的 SQL
- 关系型数据库中的事务
- 分布式事务

MySQL 中的事务

事务隔离性

- 读现象（Read phenomena）
  - 脏读（dirty reads）
  - 不可重复读（non-repeatable reads）
  - 幻影读（phantom read）
- 隔离级别
  - Read Uncommitted（读取未提交内容）
  - Read Committed（读取提交内容）
  - Repeatable Read（可重读）
  - Serializable（可串行化）
  - 隔离级别和读现象

课后作业

需求说明

1. 账户表 Account，字段：id，name，money
2. 新建maven项目，项目名：spring-tx-demo，完成 Spring（注解），MyBatis（XML） 整合。
3. 新增两个账户，如：1,张三,100; 2,李四,200
4. 业务层添加转账方法
5. Application 处调用转账方法，方法名：transferMoney

a. 比如从张三账户转账10 元到李四账户

数据库事务

概述

数据库事务（简称：事务）是数据库管理系统执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成。

数据库事务通常包含了一个序列的对数据库的读/写操作。

数据库事务的使用包含有以下两个目的：

- 为数据库操作序列提供了一个从失败中恢复到正常状态的方法，同时提供了数据库即使在异常状态下仍能保持一致性的方法。
- 当多个应用程序在并发访问数据库时，可以在这些应用程序之间提供一个隔离方法，以防止彼此的操作互相干扰。

当一个事务被提交给了数据库管理系统（DBMS），则 DBMS 需要确保该事务中的所有操作都成功完成且其结果被永久保存在数据库中。

如果事务中有的操作没有成功完成，则事务中的所有操作都需要回滚，回到事务开始执行前的状态。

同时，该事务对数据库或者其他事务的执行无影响，所有的事务都好像在独立的运行。

事务的例子

某人要在商店使用电子货币购买100元的东西，当中至少包括两个操作：

- 该人账户减少100元
- 商店账户增加100元

支持事务的数据库管理系统（transactional DBMS）就是要确保以上两个操作（整个“事务”）都能完成，或一起取消；否则就会出现100元平白消失或出现的情况。

在现实情况下，失败的风险很高。

在一个数据库事务的执行过程中，有可能会遇上事务操作失败、数据库系统 / 操作系统出错，甚至是存储介质出错等情况。

这便需要 DBMS 对一个执行失败的事务执行恢复操作，将其数据库状态恢复到一致状态（数据的一致性得到保证的状态）。

为了实现将数据库状态恢复到一致状态的功能，DBMS通常需要维护事务日志以追踪事务中所有影响数据库数据的操作。

并非任意的对数据库的操作序列都是数据库事务。

数据库事务拥有以下四个特性，习惯上被称之为 ACID 特性。

ACID

ACID 是指数据库管理系统（DBMS）在写入或更新资料的过程中，为保证事务（transaction）是正确可靠的，所必须具备的四个特性：
原子性（atomicity，或称不可分割性）
一致性（consistency）
隔离性（isolation，又称独立性）
持久性（durability）。

我们知道，在数据库系统中，一个事务是指：由一系列数据库操作组成的一个完整的逻辑过程。

例如银行转帐，从原账户扣除金额，以及向目标账户添加金额，这两个数据库操作的总和，构成一个完整的逻辑过程，不可拆分。

上面例子中的每个过程被称为一个事务，而事务必然具有 ACID 特性（ACID 的概念在 ISO/IEC 10026-1:1992 文件的第四段内有所说明）。

ACID 说明如下：

- 原子性（Atomicity）：
    - 一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，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。
    - 事务不可分割、不可约简。

- 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约束、触发器、级联回滚等。
- 事务隔离（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：
    - 未提交读（Read uncommitted）
    - 提交读（read committed）
    - 可重复读（repeatable read）
    - 序列化（Serializable）
- 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。

事务相关的 SQL

SQL 国际标准使用 start transaction 开始一个事务。commit 语句使事务成功完成。

rollback 语句结束事务，放弃从事务开始的一切变更。

如果 autocommit 被 start transaction 的使用禁止，在事务结束时 autocommit 会重新激活。

关系型数据库中的事务

关系型数据库传统上是由具有固定大小的字段和记录的表组成。

在启动事务后，数据库记录或对象会被锁定，只读，或读写。事务结束后才可以进行读写操作。

一旦事务完全完成，变化就会被原子性地提交或回滚，这样在事务结束时就不会出现不一致的情况。

分布式事务

数据库系统实现分布式事务是指在多个节点上访问数据的事务。

一个分布式事务在多个节点上执行 ACID 属性，可能包括数据库、存储管理器、文件系统、消息系统和其他数据管理器等系统。

在一个分布式事务中，通常有一个实体协调所有的过程，以确保事务的所有部分都适用于所有相关系统。

MySQL 中的事务

默认情况下, MySQL启用自动提交模式(变量 autocommit 为ON)。这意味着,只要你执行 DML 操作的语句，MySQL会立即隐式提交事务(Implicit Commit)。

查询：

```sql
show session variables like 'autocommit';
```

修改：

```sql
set session autocommit=1;
```

除此之外，还可以通过 SQL 的方式：

```sql
start transaction;
commit;
rollback;
```

事务隔离性

事务隔离（Transaction Isolation）定义了数据库系统中一个事务中操作的结果在何时以何种方式对其他并发事务操作可见。

另外，隔离也是事务 ACID（原子性、一致性、隔离性、持久性）四大属性之一。

读现象（Read phenomena）

对于事务隔离性，在 ANSI/ISO SQL 92 标准中，描述了三种不同的一个事务读取另外一个事务可能修改的数据的“读现象”。

假设有两个事务，事务 1 执行语句 1。接着，事务 2 执行语句 2 并且提交，最后事务 1 再执行语句 1。

users

id	name	age
1	Joe	20
2	Jill	25

下面是几种常见的读现象。

脏读（dirty reads）

当一个事务允许读取另外一个事务修改但未提交的数据时，就可能发生脏读（dirty reads）。

脏读和不可重复读类似，不同点在于事务2不需要提交就能造成语句1两次执行的结果不同。在未提交读隔离级别唯一禁止的是更新混乱，即早期的更新可能出现在后来更新之前的结果集中。

下面的例子中，事务 2 修改了一行，但是没有提交，事务 1 读了这个没有提交的数据。

如果事务 2 回滚了刚才的修改或者做了另外的修改的话，事务 1 中查到的数据就是不正确的了。

在这个例子中，事务 2 回滚后就没有 id 是 1，age 是 21 的数据行了。

Transaction 1	Transaction 2

`/* Query 1 */`
`SELECT age FROM users WHERE id = 1;`
`/* will read 20 */`	

	`/* Query 2 */`
	`UPDATE users SET age = 21 WHERE id = 1;`
	`/* No commit here */`

`/* Query 1 */`
`SELECT age FROM users WHERE id = 1;`
`/* will read 21 */`	

	`ROLLBACK; /* lock-based DIRTY READ */`

不可重复读（non-repeatable reads）

在一次事务中，当一行数据获取两遍得到不同的结果表示发生了不可重复读（non-repeatable reads）。

在基于锁的并发控制中“不可重复读”现象发生在当执行 SELECT 操作时没有获得读锁或者SELECT操作执行完后马上释放了读锁；

多版本并发控制中当没有要求一个提交冲突(commit conflict)的事务回滚也会发生“不可重复读”现象。

事务 1	事务 2
`/* Query 1 */` `SELECT * FROM users WHERE id = 1;`	

	`/* Query 2 */`
	`UPDATE users SET age = 21 WHERE id = 1;`
	`COMMIT; /* in multiversion concurrency control, or lock-based READ COMMITTED */`

`/* Query 1 */`
`SELECT * FROM users WHERE id = 1;`
`COMMIT; /* lock-based REPEATABLE READ */`	

上面的例子中，事务2提交成功，因此他对id为 1 的行的修改就对其他事务可见了。

但是事务1在此前已经从这行读到了另外一个“age”的值。

在可序列化（SERIALIZABLE）和可重复读的隔离级别中，数据库在第二次 SELECT 请求的时必须返回更新之前的值。

在提交读和未提交读中，返回的是更新之后的值，这个现象就是不可重复读。

有两种策略可以避免不可重复读。

一个是要求事务2延迟到事务1提交或者回滚之后再执行。这种方式实现了T1, T2 的串行化调度。串行化调度可以支持可重复读。

另一种被用在多版本并发控制的策略是允许事务2先提交，这样能得到更好的并发性能。

但因为事务1 在事务2 之前开始，事务1 必须在其开始执行时间点的数据库的快照上面操作。

当事务1最终提交时候，数据库会检查其结果是否等价于T1, T2串行调度。

如果等价，则允许事务1提交，如果不等价，事务1需要回滚并抛出个串行化失败的错误。

使用基于锁的并发控制，在可重复读的隔离级别中，ID=1 的行会被锁住，在事务1 提交或回滚前一直阻塞语句2 的执行。在提交读的级别，语句1 第二次执行，age 已经被修改了。

在多版本并发控制机制下，可序列化(SERIALIZABLE)级别，两次SELECT语句读到的数据都是事务1 开始的快照，因此返回同样的数据。

但是，如果事务1试图UPDATE这行数据，事务1会被要求回滚并抛出一个串行化失败的错误。

在提交读隔离级别，每个语句读到的是语句执行前的快照，因此读到更新前后不同的值。

在这种级别不会有串行化的错误（因为这种级别不要求串行化），事务1 也不要求重试。

幻影读（phantom read）

在事务执行过程中，当两个完全相同的查询语句执行得到不同的结果集。

这种现象称为“幻影读（phantom read）”。

当事务没有获取范围锁的情况下执行 SELECT ... WHERE 操作可能会发生“幻影读”。

“幻影读”是不可重复读的一种特殊场景：当事务1 两次执行 SELECT ... WHERE 检索一定范围内数据的操作中间，事务2 在这个表中创建了(如 INSERT)了一行新数据，这条新数据正好满足事务1 的“WHERE”子句。

事务 1	事务 2

`/* Query 1 */`
`SELECT * FROM users`
`WHERE age BETWEEN 10 AND 30;`	

	`/* Query 2 */`
	`INSERT INTO users VALUES ( 3, 'Bob', 27 );`
	`COMMIT;`

`/* Query 1 */`
`SELECT * FROM users`
`WHERE age BETWEEN 10 AND 30;`	

需要指出的是事务1 执行了两遍同样的查询语句。

如果设置了最高的隔离级别，两次会得到同样的结果集，这也正是数据库在可序列化（SERIALIZABLE）隔离级别上需要满足的。

但是在较低的隔离级别上，第二次查询可能会得到不同的结果集。

在可序列化隔离级别，查询语句1 在 age 从 10 到 30 的记录上加锁，事务2 只能阻塞直至事务1 提交。在可重复读级别，这个范围不会被锁定，允许记录插入，因此第二次执行语句1的结果中会包括新插入的行。

隔离级别

隔离级别主要有四种：
Read uncommitted (未提交读)
Read committed (已提交读)
Repeatable read (可重复读)
Serializable (可序列化)

MySQL中查看隔离级别：

```sql
select @@transaction_isolation; -- MySQL 8
select @@tx_isolation -- MySQL 8 之前的版本
```

设置隔离级别：

```sql
set session transaction isolation level read uncommitted;
set session transaction isolation level read committed;
set session transaction isolation level repeatable read;
set session transaction isolation level serializable;
```

SQL标准定义了4类隔离级别，包括了一些具体规则，用来限定事务内外的哪些改变是可见的，哪些是不可见的。低级别的隔离级一般支持更高的并发处理，并拥有更低的系统开销。

隔离级别在 MySQL 文档中也有详细说明：
https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html

Read Uncommitted（读取未提交内容）

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

Read Committed（读取提交内容）

这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。

Repeatable Read（可重读）

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影”行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

Serializable（可串行化）

这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

隔离级别和读现象

这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生读现象。例如：

- 脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。
- 不可重复读(Non-repeatable read):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。
- 幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

在 MySQL 中，实现了这四种隔离级别，分别有可能产生问题如下所示：

隔离级别	脏读	不可重复读	幻读
Read uncommited	√	√	√
Read commited	×	√	√
Repeatable read	×	×	√
Serializable	×	×	×

脏读：允许读取未提交的信息，原因：Read uncommitted，解决方案： Read committed（表级读锁）
不可重复读：读取过程中单个数据发生了变化，解决方案： Repeatable read （行级写锁）
幻读：读取过程中数据条目发生了变化，解决方案： Serializable（表级写锁）

课后作业

- 继续完成 spring-tx-demo
- 熟悉 MySQL 中事务操作，开始，提交，回滚，
- 背的内容：ACID，隔离性，隔离级别。