说起事务，肯定能够想到这样一句话，这个事情要么不做，要么做完；或者是好兄弟不求同年同月同日生，但求同年同月同日死。有点过分，但是是这个理儿。

四大特性

我们都知道，提到事务，就不能不提事务的四大特性，ACID，即原子性，一致性，隔离性，持久性。

- 原子性（Atom）：事务的一组操作是原子的不可再分割的，这组操作要么同时完成要么同时不完成。
- 一致性（Consistency）：事务在执行前后数据的完整性保持不变。数据库在某个状态下符合所有的完整性约束的状态叫做数据库具有完整性。在解散一个部门时应该同时处理员工表中的员工保证这个事务结束后，仍然保证所有的员工能找到对应的部门，满足外键约束。
- 隔离性（Isolation）：当多个事务同时操作一个数据库时，可能存在并发问题，此时应保证各个事务要进行隔离，事务之间不能互相干扰。
- 持久性（Durability）：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，不能再回滚。

事务有这些特性，但是他又带来了什么样的问题呢?

事务引发的问题

脏读

脏读指一个事务读取了另外一个事务未提交的数据。

这是非常危险的，假设A向B转帐100元，对应sql语句如下所示

1. `update account set money=money+100 where name='B';`
2. `update account set money=money-100 where name='A';`

当第1条sql执行完，第2条还没执行(A未提交时)，如果此时B查询自己的帐户，就会发现自己多了100元钱。如果A等B完成后后再回滚，B就会损失100元。

幻读

出现幻读的情况，数据可能不是错误的，但是可能不符合实际的业务需求。

幻读出现情况：一个事务的两次不同时间的相同查询返回了不同的的结果集。例如:一个 select 语句执行了两次，但是在第二次返回了第一次没有返回的行,那么这些行就是“phantom” row。

例如：银行在做统计报表时统计account表中所有用户的总金额时候，此时总共有三个账户，总共金额为3000元，这时候新增了一个用户账户，并且存入1000元，这时候银行再次统计就会发现账户总金额为4000，造成了幻读情况。

不重复读

不可重复读指在一个事务内读取表中的某一行数据，多次读取结果不同。

例如银行想查询A帐户余额，第一次查询A帐户为200元，此时A向帐户内存了100元并提交了，银行接着又进行了一次查询，此时A帐户为300元了。银行两次查询不一致，可能就会很困惑，不知道哪次查询是准的。

不可重复读和脏读的区别是，脏读是读取前一事务未提交的脏数据，不可重复读是重新读取了前一事务已提交的数据。

很多人认为这种情况就对了，无须困惑，当然是后面的为准。我们可以考虑这样一种情况，比如银行程序需要将查询结果分别输出到电脑屏幕和写到文件中，结果在一个事务中针对输出的目的地，进行的两次查询不一致，导致文件和屏幕中的结果不一致，银行工作人员就不知道以哪个为准了。

更新丢失

丢失更新就是两个不同的事务（或者Java程序线程）在某一时刻对同一数据进行读取后，先后进行修改。导致第一次操作数据丢失。

例如：

时间点	事务A	事务B
1	start transcaction;	
2		start transcaction;
3	update t_customer ts set ts.name='张三' where ts.id='10';	
4		update t_customer t set t.name='李四',t.age=20 where ts.id='10'
5	commit;	
6		commit;

假如原来t_customer表内id为10的行，是一条{id:10，name:"王五"，age:15} 的数据，经过事务A修改后变成{id:10，name:"张三"，age:15}。事务B提交后，该数据变成了{id:10，name:"李四"，age:20}。由事务A所执行的操作在事务B的提交后，数据被冲掉了。这个现象就叫做丢失更新。

既然使用事务有这些问题，那我们应该怎么解决呢？在这里说一下mysql的事务隔离级别

事务的隔离级别

在数据库操作中，为了有效保证并发读取数据的正确性，提出的事务隔离级别，在标准SQL规范中，定义了4个事务隔离级别，不同的隔离级别对事务的处理不同。

未授权读取（Read Uncommitted）

也称为读未提交（Read Uncommitted）：会引发脏读取、不可重复读和虚读，但避免了更新丢失。如果一个事务已经开始写数据，则另外一个事务则不允许同时进行写操作，但允许其他事务读此行数据。该隔离级别可以通过“排他写锁”实现。

eg:

```
-- A窗口

set transaction isolation level  read uncommitted;--设置A用户的数据库隔离级别为Read uncommitted(读未提交)
start transaction;--开启事务
select * from account;--查询A账户中现有的钱，转到B窗口进行操作
select * from account--发现a多了100元，这时候A读到了B未提交的数据（脏读）

-- B窗口
start transaction;--开启事务
update account set money=money+100 where name='A';--不要提交，转到A窗口查询
```

授权读取（Read Committed）

也称为读提交（Read Committed）：会引发不可重复读取和虚读，但避免脏读取。这可以通过“瞬间共享读锁”和“排他写锁”实现。读取数据的事务允许其他事务继续访问该行数据，但是未提交的写事务将会禁止其他事务访问该行。

eg:

```
-- A窗口
set transaction isolation level  read committed;
start transaction;
select * from account;--发现a帐户是1000元，转到b窗口
select * from account;--发现a帐户多了100,这时候，a读到了别的事务提交的数据，两次读取a帐户读到的是不同的结果（不可重复读）


-- B窗口
start transaction;
update account set money=money+100 where name='A';
commit;--转到a窗口
```

可重复读取（Repeatable Read）（mysql默认级别）

可重复读取（Repeatable Read）：禁止不可重复读取和脏读取，但是有时可能出现幻读数据和虚读。这可以通过“共享读锁”和“排他写锁”实现。读取数据的事务将会禁止写事务（但允许读事务），写事务则禁止任何其他事务。

eg:

```
-- A窗口
set transaction isolation level repeatable read;
start transaction;
select * from account;--发现表有4个记录，转到b窗口
select * from account;--可能发现表有5条记录，这时候发生了a读取到另外一个事务插入的数据（虚读）

-- B窗口
start transaction;
insert into account(name,money) values(1002,'ggg',1000);
commit;--转到a窗口
```

序列化（Serializable）

序列化（Serializable）：提供严格的事务隔离。它要求事务序列化执行，事务只能一个接着一个地执行，不能并发执行。仅仅通过“行级锁”是无法实现事务序列化的，必须通过其他机制保证新插入的数据不会被刚执行查询操作的事务访问到。

eg:

```
-- A窗口
set transaction isolation level Serializable;
start transaction;
select * from account;--转到b窗口
-- B窗口
start transaction;
insert into account(id,name,money) values(1003,'gfg',1000);--发现不能插入，只能等待a结束事务才能插入
```

从上面可以看出来，通过选择事务的隔离级别，可以很好的解决上面的4中事务问题，总结一下

问题的解决

脏读

设置事务级别为Read conmiitted或者repeatable read都是可以的。

- A客户端的级别是数据库默认的Repeatable read
- B客户端的级别更改为效率最高的Read committed级别

幻读

修改事务的隔离级别为Repeatable Read，或者是Serializable。

1、Repeatable Read从理论的角度是会出现幻读的，这也就是限制了Repeatable Read这个事务隔离级别使用。一个事务隔离级别推出使用发现其自身带有缺陷，开发者自然会想到完善的方法，所以MySQL内部通过多版本控制机制【实际上就是对读取到的数据加锁】解决这个问题。最后，用户才可以放心大胆使用Repeatable Read这个事务隔离级别。

2、Serializable 和 Repeatable Read都可以防止幻读。但是Serializable 事务隔离级别效率低下，比较耗数据库性能，一般不使用。

不重复读

设置事务级别为repeatable read，Serializable太耗费性能了，不推荐

更新丢失

Serializable虽然可以防止更新丢失，但是效率太低，通常数据库不会用这个隔离级别，所以我们需要其他的机制来防止更新丢失.

1. 使用排它锁(悲观锁)。

经过上面基于数据库锁的介绍可知，丢失更新可以使用写锁(排它锁)进行控制。因为排它锁添加到某个表的时候，事务未经提交，其他的事务根本没法获取修改权，因此排它锁可以用来控制丢失更新。

需要说明的是有时候，当知道某一行会发生并发修改的时候，可以把锁定的范围缩小。例如使用select * from t_account t wheret.id='1' for update; 这样能够比较好地把控上锁的粒度，这种基于行级上锁的方法叫"行级锁"。

2. 使用乐观锁。

乐观锁的原理是：认为事务不一定会产生丢失更新，让事务进行并发修改，不对事务进行锁定。发现并发修改某行数据时，乐观锁抛出异常。让用户解决。

可以通过给数据表添加自增的version字段或时间戳timestamp。进行数据修改时，数据库会检测version字段或者时间戳是否与原来的一致。若不一致，抛出异常。

校验事务B与version值,事务B提交前的version字段值为1，但当前version值为2，禁止事务B提交.抛出异常让用户处理

时间点	事务A	事务B	version值
			1
1	提出修改： update t_account t set t.money=t.money+100 where t.name ='a'	提出修改： update t_account t set t.money=t.money+100 where t.name ='a'	1
2	commit;		校验事务A与version值,version字段值都为"1",通过提交内容，version字段值更新为2
3		commit;	校验事务B与version值,事务B提交前的version字段值为1，但当前version值为2，禁[illegible]务B提交.抛出异常让用户处理

补充：

1、SQL规范所规定的标准，不同的数据库具体的实现可能会有些差异

2、mysql中默认事务隔离级别是可重复读时并不会锁住读取到的行

3、事务隔离级别为读提交时，写数据只会锁住相应的行

4、事务隔离级别为可重复读时，如果有索引（包括主键索引）的时候，以索引列为条件更新数据，会存在间隙锁间隙锁、行锁、下一键锁的问题，从而锁住一些行；如果没有索引，更新数据时会锁住整张表。

5、事务隔离级别为串行化时，读写数据都会锁住整张表

6、隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大，鱼和熊掌不可兼得啊。对于多数应用程序，可以优先考虑把数据库系统的隔离级别设为Read Committed，它能够避免脏读取，而且具有较好的并发性能。尽管它会导致不可重复读、幻读这些并发问题，在可能出现这类问题的个别场合，可以由应用程序采用悲观锁或乐观锁来控制。