**概述**

事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。

* ACID  
  事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。

## An Example of ACID?

It might help to look at ACID and its concepts using an example.  Consider a banking transaction where you’re withdrawing money from checking to deposit in your saving account.  As part of the transaction, a journal entry is made as an audit record.  How would ACID help in this situation?

Since the transaction is **Atomic**, the money can’t be taken out of your checking account without being subsequently deposited in savings.  If the transaction was interrupted for some reason, your account balance would remain unchanged.

Since a record of every transaction is kept in a journal, **Consistency** ensures that the transfer can’t complete without successfully writing the journal entry.  If the journal is full, then the transfer is aborted.  Your account balances are returned to their original balances.

**Isolation** ensures that other banking transactions don’t affect the outcome of your transfer.  Other transaction to alter your checking balance must wait until your transaction completes.

Being **Durable**, once the transaction is saved or committed, it can’t be “lost.”  That is, a power outage or system crash won’t cause any of the data to go missing.

* 隔离级别  
  ACID这4个特征中，最难理解的是隔离性。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。4个隔离级别分别是：读未提及（READ\_UNCOMMITTED）、读已提交（READ\_COMMITTED）、可重复读（REPEATABLE\_READ）、顺序读（SERIALIZABLE）。
* 事务并发引起的问题  
  数据库在不同的隔离性级别下并发访问可能会出现以下几种问题：脏读（Dirty Read）、不可重复读（Unrepeatable Read）、幻读（Phantom Read）。

**事务的思维导图**

A picture containing screenshot

Description automatically generated

**ACID**

**1. 原子性（Atomicity）**

事务的原子性是指事务必须是一个原子的操作序列单元。事务中包含的各项操作在一次执行过程中，只允许出现两种状态之一。

* 全部执行成功
* 全部执行失败

任何一项操作都会导致整个事务的失败，同时其它已经被执行的操作都将被撤销并回滚，只有所有的操作全部成功，整个事务才算是成功完成。

**2. 一致性（Consistency）**

事务的一致性是指事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处以一致性状态。

比如：如果从A账户转账到B账户，不可能因为A账户扣了钱，而B账户没有加钱。

**3. 隔离性（Isolation）**

事务的隔离性是指在并发环境中，并发的事务是互相隔离的，一个事务的执行不能被其它事务干扰。也就是说，不同的事务并发操作相同的数据时，每个事务都有各自完整的数据空间。

一个事务内部的操作及使用的数据对其它并发事务是隔离的，并发执行的各个事务是不能互相干扰的。

隔离性分4个级别，下面会介绍。

**4. 持久性（Duration）**

事务的持久性是指事务一旦提交后，数据库中的数据必须被永久的保存下来。即使服务器系统崩溃或服务器宕机等故障。只要数据库重新启动，那么一定能够将其恢复到事务成功结束后的状态。

**事务隔离级别**

There are mainly four types of common concurrency problems: dirty read, lost read, non-repeatable read and phantom reads.

**1. 读未提交（READ\_UNCOMMITTED）**

读未提及，该隔离级别允许脏读取，其隔离级别是最低的。换句话说，如果一个事务正在处理某一数据，并对其进行了更新，但同时尚未完成事务，因此还没有提交事务；而以此同时，允许另一个事务也能够访问该数据。

**脏读示例：**

在事务A和事务B同时执行时可能会出现如下场景：

| **时间** | **事务A（存款）** | **事务B（取款）** |
| --- | --- | --- |
| T1 | 开始事务 | —— |
| T2 | —— | 开始事务 |
| T3 | —— | 查询余额（1000元） |
| T4 | —— | 取出1000元（余额0元） |
| T5 | 查询余额（0元） | —— |
| T6 | —— | 撤销事务（余额恢复1000元） |
| T7 | 存入500元（余额500元） | —— |
| T8 | 提交事务 | —— |

余额应该为1500元才对。请看T5时间点，事务A此时查询的余额为0，这个数据就是**脏数据**，他是事务B造成的，很明显是事务没有进行隔离造成的。

脏读：读到了还没有提交的数据

**2. 读已提交（READ\_COMMITTED）**

读已提交是不同的时候执行的时候只能获取到已经提交的数据。  
这样就不会出现上面的脏读的情况了。

**不可重复读示例**

可是解决了脏读问题，但是还是解决不了不可重复读问题。

| **时间** | **事务A（存款）** | **事务B（取款）** |
| --- | --- | --- |
| T1 | 开始事务 | —— |
| T2 | —— | 开始事务 |
| T3 | —— | 查询余额（1000元） |
| T4 | 查询余额（1000元） | —— |
| T5 | —— | 取出1000元（余额0元） |
| T6 | —— | 提交事务 |
| T7 | 查询余额（0元） | —— |
| T8 | 提交事务 | —— |

事务A其实除了查询两次以外，其它什么事情都没做，结果钱就从1000变成0了，这就是不可重复读的问题。

**3. 可重复读（REPEATABLE\_READ）**

可重复读就是保证在事务处理过程中，多次读取同一个数据时，该数据的值和事务开始时刻是一致的。因此该事务级别禁止了不可重复读取和脏读，但是有可能出现幻读的数据。

**幻读**

幻读就是指同样的事务操作，在前后两个时间段内执行对同一个数据项的读取，可能出现不一致的结果。

| **时间** | **事务A（统计总存款）** | **事务B（存款）** |
| --- | --- | --- |
| T1 | 开始事务 | —— |
| T2 | —— | 开始事务 |
| T3 | 统计总存款（10000元） | —— |
| T4 | —— | 存入100元 |
| T5 | —— | 提交事务 |
| T6 | 提交总存款（10100） | —— |
| T7 | 提交事务 | —— |

银行工作人员在一个事务中多次统计总存款时看到结果不一样。如果要解决幻读，那只能使用顺序读了。

**4. 顺序读（SERIALIZABLE）**

顺序读是最严格的事务隔离级别。它要求所有的事务排队顺序执行，即事务只能一个接一个地处理，不能并发。

**事务隔离级别对比**

| **事务隔离级别** | **脏 读** | **不可重复读** | **幻 读** |
| --- | --- | --- | --- |
| 读未提及（READ\_UNCOMMITTED） | 允许 | 允许 | 允许 |
| 读已提交（READ\_COMMITTED） | 禁止 | 允许 | 允许 |
| 可重复读（REPEATABLE\_READ） | 禁止 | 禁止 | 允许 |
| 顺序读（SERIALIZABLE） | 禁止 | 禁止 | 禁止 |

4种事务隔离级别从上往下，级别越高，并发性越差，安全性就越来越高。  
一般数据默认级别是读以提交或可重复读。

备注：不可重复度和幻读的区别

当然,   从总的结果来看,   似乎两者都表现为两次读取的结果不一致.  
  
但如果你从控制的角度来看,   两者的区别就比较大  
对于前者,   只需要锁住满足条件的记录  
对于后者,   要锁住满足条件及其相近的记录

-----------------------------------------------------------

我这么理解是否可以？  
避免不可重复读需要锁行就行  
避免幻影读则需要锁表

------------------------

####不可重复读和幻读的区别####  
很多人容易搞混不可重复读和幻读，确实这两者有些相似。但不可重复读重点在于update和delete，而幻读的重点在于insert。

如果使用锁机制来实现这两种隔离级别，在可重复读中，该sql第一次读取到数据后，就将这些数据加锁，其它事务无法修改这些数据，就可以实现可重复 读了。但这种方法却无法锁住insert的数据，所以当事务A先前读取了数据，或者修改了全部数据，事务B还是可以insert数据提交，这时事务A就会 发现莫名其妙多了一条之前没有的数据，这就是幻读，不能通过行锁来避免。需要Serializable隔离级别 ，读用读锁，写用写锁，读锁和写锁互斥，这么做可以有效的避免幻读、不可重复读、脏读等问题，但会极大的降低数据库的并发能力。

**所以说不可重复读和幻读最大的区别，就在于如何通过锁机制来解决他们产生的问题。**

上文说的，是使用悲观锁机制来处理这两种问题，但是MySQL、ORACLE、PostgreSQL等成熟的数据库，出于性能考虑，都是使用了以乐观锁为理论基础的MVCC（多版本并发控制）来避免这两种问题。