目录

[数据库事务的隔离级别 2](#_Toc1442828716)

[数据库ACID理论 2](#_Toc236887888)

[原子性 2](#_Toc2087535725)

[一致性 2](#_Toc1772589036)

[隔离性 2](#_Toc2010776868)

[持久性 3](#_Toc176667637)

[事务隔离级别 3](#_Toc1430574905)

[Read Uncommitted（读取未提交内容） 3](#_Toc445516523)

[Read Committed（读取提交内容） 3](#_Toc1668208619)

[Repeatable Read（可重读） 3](#_Toc35764301)

[Serializable（可串行化） 3](#_Toc1942669394)

[隔离级别的实现  4](#_Toc103135970)

[一致性 4](#_Toc386944661)

[业务一致性 5](#_Toc798434311)

[幂等性（idempotent） 5](#_Toc1807638521)

[业务一致性（Consistency） 5](#_Toc529468338)

[可串行化（Serializable） 6](#_Toc1749607245)

[参考 6](#_Toc155388344)

# 数据库事务的隔离级别

## 数据库ACID理论

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。这个过程被称为一个事务，须满足ACID特性。

ACID特性[1]是指在数据库管理系统（DBMS）中事务所具有的四个特性，它们分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation，又称独立性）、持久性（Durability）。

### 原子性

原子性是指事务是一个不可再分割的工作单元，事务中的操作要么都发生，要么都不发生。

在DBMS中，默认情况下一条SQL就是一个单独事务，事务是自动提交的。只有显式的使用start transaction开启一个事务，才能将一个代码块放在事务中执行。

### 一致性

一致性是指在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。这是说数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

数据库层面会在一个事务执行之前和之后，确保数据符合设置的约束（唯一约束，外键约束,check约束等)和触发器设置；此外，数据库的内部数据结构（如 B 树索引或双向链表）都必须是正确的。业务的一致性一般由开发人员进行保证，亦可转移至数据库层面。

### 隔离性

多个事务并发访问时，事务之间是隔离的，一个事务不应影响其它事务运行效果。

在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看到中间状态的数据。

事务最复杂问题都是由事务隔离性引起的。完全的隔离性是不现实的，完全的隔离性要求数据库同一时间只执行一条事务，这样会严重影响性能。

隔离性中的事务隔离等级是本文要详细分析的。

### 持久性

持久性，意味着在事务完成以后，该事务所对数据库所作的更改要持久的保存在数据库之中，并不会被回滚。（完成的事务是系统永久的部分，对系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持）

## 事务隔离级别

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。

### **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（比如SqlServer，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。

 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，就容易发生问题。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。

### 隔离级别的实现

以上隔离级别是SQL标准的定义。不同的数据库厂商对隔离级别的实现并不一致，如SQL Server[2], DB2[3]，Oracle[4]。也存在对这个标准定义的批评，指出其缺陷[5]。这里不对细节进行说明，我们以MySQL为例作为讨论的基础。

在MySQL中，实现了这四种隔离级别，分别有可能产生问题如下图所示：



1. 隔离级别与read phenomena[6]

## 一致性

让我们回到ACID理论，重点看其中的C：一致性。在ACID中，C是实现关系数据的完整性以及业务逻辑上的一致性的本质需求。除C以外其他特性都是为了实现C而存在的，如果没有C的需求，ACID也就不存在了。

我们多谈谈一致性C。下面直接应用wikipedia关于数据库系统一致性的定义[7]。

**Consistency** in [database systems](https://en.wikipedia.org/wiki/Database_systems" \o "Database systems) refers to the requirement that any given [database transaction](https://en.wikipedia.org/wiki/Database_transaction" \o "Database transaction) must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules, including [constraints](https://en.wikipedia.org/wiki/Integrity_constraints" \o "Integrity constraints), [cascades](https://en.wikipedia.org/wiki/Cascading_rollback" \o "Cascading rollback), [triggers](https://en.wikipedia.org/wiki/Database_trigger" \o "Database trigger), and any combination thereof. This **does not guarantee correctness** of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined database constraints.[[1]](https://en.wikipedia.org/wiki/Consistency_(database_systems)" \l "cite_note-Date2012-1)

数据库系统中的一致性强调要满足数据库系统本身的规则上的限制，它不能保障业务逻辑的正确性（以应用开发者的角度看）。即使我们将隔离级别设置为最高级别：可串行化，也不定能解决业务逻辑上的正确性要求，需要强调的是某些数据库根本就没有实现事务。SQL标准中定义的可串行化也不是我们理解的那个保障完全正确的串行化[8]，而各个数据库对隔离级别的实现也是不同的[2][3][4]，继而考虑到并行（concurrent）的要求，设置最高隔离级别更不现实。

再看看分布式场景，对于分布式存储系统来说，一致性变成一个和数据库系统事务一致性不同的概念[9]。从强一致性到弱一致性，各种一致性概念眼花缭乱，详细的说明可以看[9]。在分布式场景，依靠数据库端解决事务的一致性也是不可行的。

## 业务一致性

业务一致性是指从应用开发者角度理解的业务正确性。如前说述，数据库或者分布式存储系统可以直接帮助我们直接解决部分业务的一致性，但并不是所有，这时需要应用开发者角度实现业务一致性。分布式事务是可以解决很多业务一致性问题的，其中TCC[10]更是解决了在线商品交易的诸多需求。这里再提一个通用的模型，具体方法还是要回到数据库事务的ACID概念，我们使用一个新的概念ICS，分别叫做幂等性（idempotent）、业务一致性（Consistency）和可串行化（Serializable）。这是一个更通用的满足业务一致性要求的模型。

### 幂等性（idempotent）

幂等性要求业务请求可以重复提交，但是不能违反业务一致性。幂等性要求可以部分由数据库系统支持，例如设置人名，连续两次重复设置是一样的结果。而其他业务请求，例如充值，连续两次重复充值则应该和充值一次的结果一致，这一点则必须由应用开发者保证。

### 业务一致性（Consistency）

即业务逻辑的正确性。

### 可串行化（Serializable）

可串行化要求访问同样的数据由同一个角色（进程/线程/Master等）执行，且执行要严格串行化。需要说明的是串形化不能忽略Concurrency。提高concurrency可以通过以更细粒度切分数据的方式实现，具体不再赘述。

## 参考

1. <https://en.wikipedia.org/wiki/ACID>
2. <https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-ver15>
3. <https://www.ibm.com/support/knowledgecenter/SSAE4W_9.5.1/db2/rbafzisollevel.htm>
4. <https://docs.oracle.com/database/121/CNCPT/consist.htm#CNCPT621>
5. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-95-51.pdf>
6. <https://en.wikipedia.org/wiki/Isolation_%28database_systems%29>
7. <https://en.wikipedia.org/wiki/Consistency_(database_systems)>
8. <https://community.oracle.com/thread/4174436>
9. <https://arxiv.org/pdf/1512.00168.pdf>
10. <https://www.cnblogs.com/xybaby/p/7465816.html>