<https://mp.weixin.qq.com/s/kyRG_urd-HcVUbvg0T3Evg>

### 什么是事务？

事务就是「一组原子性的SQL查询」，或者说一个独立的工作单元。如果数据库引擎能够成功地对数据库应用该组查询的全部语句，那么就执行该组查询。如果其中有任何一条语句因为崩溃或其他原因无法执行，那么所有的语句都不会执行。也就是说，事务内的语句，要么全部执行成功，要么全部执行失败。

### 事务控制语法知道吗？

1. BEGIN 或 START TRANSACTION 显式地开启一个事务；  
   2、COMMIT / COMMIT WORK二者是等价的。提交事务，并使已对数据库进行的所有修改成为永久性的；  
   3、ROLLBACK / ROLLBACK WORK。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；  
   4、SAVEPOINT identifier 在事务中创建一个保存点，一个事务中可以有多个 SAVEPOINT；  
   5、RELEASE SAVEPOINT identifier 删除一个事务的保存点；  
   6、ROLLBACK TO identifier 把事务回滚到标记点；  
   7、SET TRANSACTION 用来设置事务的隔离级别。InnoDB 存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 和 SERIALIZABLE

### 用通俗的语言说说你理解的事务

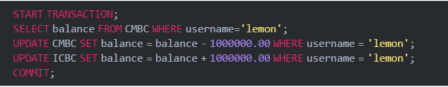
用银行业务举个栗子，用户lemon有两银行卡，一张是招商银行CMBC的工资卡，另一张是工商银行ICBC的储蓄卡，每月10号发工资都要把招行卡的100万转到建设银行储蓄卡账户。记住这里的银行缩写后面就是对应的数据表名称，你要记不住，我给你理一理。

这个转账的操作可以简化抽成一个事务，包含如下步骤：

1. 查询CMBC账户的余额是否大于100万
2. 从CMBC账户余额中减去100万
3. 在ICBC账户余额中增加100万

以下语句对应创建了一个转账事务：

以下语句对应创建了一个转账事务：



### 事务的ACID特性是什么？

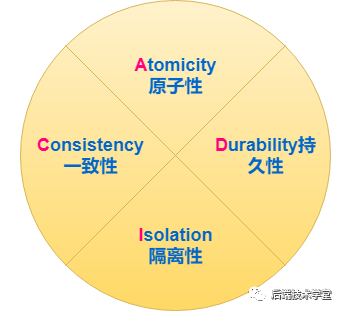
ACID其实是事务特性的英文首字母缩写，具体的含义是这样的：

原子性（atomicity)  
一个事务必须被视为一个不可分割的最小工作单元，整个事务中的所有操作要么全部提交成功，要么全部失败回滚，对于一个事务来说，不可能只执行其中的一部分操作。

一致性（consistency)  
数据库总是从一个一致性的状态转换到另外一个一致性的状态。在前面的例子中，一致性确保了，即使在执行第三、四条语句之间时系统崩溃，CMBC账户中也不会损失100万，不然lemon要哭死，因为事务最终没有提交，所以事务中所做的修改也不会保存到数据库中。

隔离性（isolation)  
通常来说，一个事务所做的修改在最终提交以前，对其他事务是不可见的。在前面的例子中，当执行完第三条语句、第四条语句还未开始时，此时如果有其他人准备给lemon的CMBC账户存钱，那他看到的CMBC账户里还是有100万的。

持久性（durability)  
一旦事务提交，则其所做的修改就会永久保存到数据库中。此时即使系统崩溃，修改的数据也不会丢失。持久性是个有点模糊的概念，因为实际上持久性也分很多不同的级别。有些持久性策略能够提供非常强的安全保障，而有些则未必。而且「不可能有能做到100%的持久性保证的策略」否则还需要备份做什么。



### 什么是脏读、不可重复读、幻读

#### **脏读**

在事务A修改数据之后提交数据之前，这时另一个事务B来读取数据，如果不加控制，事务B读取到A修改过数据，之后A又对数据做了修改再提交，则B读到的数据是脏数据，此过程称为脏读Dirty Read。



#### **不可重复读**

一个事务内在读取某些数据后的某个时间，再次读取以前读过的数据，却发现其读出的数据已经发生了变更、或者某些记录已经被删除了。



#### **幻读**

事务A在按查询条件读取某个范围的记录时，事务B又在该范围内插入了新的满足条件的记录，当事务A再次按条件查询记录时，会产生新的满足条件的记录（幻行 Phantom Row）



#### **不可重复读与幻读有什么区别？**

不可重复读的重点是修改：在同一事务中，同样的条件，第一次读的数据和第二次读的「数据不一样」。（因为中间有其他事务提交了修改）

幻读的重点在于新增或者删除：在同一事务中，同样的条件，第一次和第二次读出来的「记录数不一样」。（因为中间有其他事务提交了插入/删除）

### 四个隔离级别知道吗？解决了什么问题

SQL实现了四个标准的隔离级别，每一种级别都规定了一个事务中所做的修改，哪些在事务内和事务间是可见的，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。



各个隔离级别可以不同程度的解决脏读、不可重复读、幻读。隔离级别各有所长，没有完美的解决方案，脱离业务场景谈具体实施都是耍流氓。



### MySQL中哪些存储引擎支持事务？

MySQL中InnoDB和NDB Cluster存储引擎提供了事务处理能力，以及其他支持事务的第三引擎。

### 什么是自动提交？

MySQL默认采用自动提交AUTOCOMMIT模式。也就是说，如果不是显式地开始一个事务，则每个查询都被当作一个事务执行提交操作。

对于MyISAM或者内存表这些事务型的表，修改AUTOCOMMIT不会有任何影响。对这类表来说，没有COMMIT或者ROLLBACK的概念，也可以说是相当于一直处于AUTOCOMMIT启用的模式。

### 在事务中可以混合使用存储引擎吗？

尽量不要在同一个事务中使用多种存储引擎，MySQL服务器层不管理事务，事务是由下层的存储引擎实现的。

如果在事务中混合使用了事务型和非事务型的表（例如InnoDB和MyISAM表）,在正常提交的情况下不会有什么问题。

但如果该事务需要回滚，非事务型的表上的变更就无法撤销，这会导致数据库处于不一致的状态，这种情况很难修复，事务的最终结果将无法确定。所以，为每张表选择合适的存储引擎非常重要。

### MySQL存储引擎类型有哪些？

最常用的存储引擎是InnoDB引擎和MyISAM存储引擎，InnoDB是MySQL的默认事务引擎。

查看数据库表当前支持的引擎 ：



### InnoDB存储引擎的特点和应用场景？

InnoDB是MySQL的默认「事务引擎」，被设置用来处理大量短期（short-lived）事务，短期事务大部分情况是正常提交的，很少会回滚。

#### **特点**

采用多版本并发控制（MVCC，MultiVersion Concurrency Control）来支持高并发。并且实现了四个标准的隔离级别，通过间隙锁next-key locking策略防止幻读的出现。

引擎的表基于聚簇索引建立，聚簇索引对主键查询有很高的性能。不过它的二级索引secondary index非主键索引中必须包含主键列，所以如果主键列很大的话，其他的所有索引都会很大。因此，若表上的索引较多的话，主键应当尽可能的小。另外InnoDB的存储格式是平台独立。

InnoDB做了很多优化，比如：磁盘读取数据方式采用的可预测性预读、自动在内存中创建hash索引以加速读操作的自适应哈希索引（adaptive hash index)，以及能够加速插入操作的插入缓冲区（insert buffer)等。

InnoDB通过一些机制和工具支持真正的热备份，MySQL的其他存储引擎不支持热备份，要获取一致性视图需要停止对所有表的写入，而在读写混合场景中，停止写入可能也意味着停止读取。

#### **MyISAM存储引擎的特点和应用场景？**

MyISAM是MySQL 5.1及之前的版本的默认的存储引擎。MyISAM提供了大量的特性，包括全文索引、压缩、空间函数（GIS)等，但MyISAM不「支持事务和行级锁」，对于只读数据，或者表比较小、可以容忍修复操作，依然可以使用它。

##### **特性**

MyISAM「不支持行级锁而是对整张表加锁」。读取时会对需要读到的所有表加共享锁，写入时则对表加排它锁。但在表有读取操作的同时，也可以往表中插入新的记录，这被称为并发插入。

MyISAM表可以手工或者自动执行检查和修复操作。但是和事务恢复以及崩溃恢复不同，可能导致一些「数据丢失」，而且修复操作是非常慢的。

对于MyISAM表，即使是BLOB和TEXT等长字段，也可以基于其前500个字符创建索引，MyISAM也支持「全文索引」，这是一种基于分词创建的索引，可以支持复杂的查询。

如果指定了DELAY\_KEY\_WRITE选项，在每次修改执行完成时，不会立即将修改的索引数据写入磁盘，而是会写到内存中的键缓冲区，只有在清理键缓冲区或者关闭表的时候才会将对应的索引块写入磁盘。这种方式可以极大的提升写入性能，但是在数据库或者主机崩溃时会造成「索引损坏」，需要执行修复操作。

#### **InnoDB与MyISAM对比**

说了这么多估计看一眼也没记住，给你一张表，简单罗列两种引擎的主要区别，如下图。



#### **其他存储引擎**

MySQL还支持其他一些存储引擎，比如memory引擎、NDB集群引擎、CSV引擎，由于这些引擎没有上述InnoDB 和MyISAM 常用，这里不作介绍，感兴趣可以去翻MySQL文档了解。这里同样给出官方链接：https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html