<https://thinkwon.blog.csdn.net/article/details/104778621>

事务四大特性

原子性：不可分割的操作单元，事务中所有操作，要么全部成功；要么撤回到执行事务之前的状态

一致性：如果在执行事务之前数据库是一致的，那么在执行事务之后数据库也还是一致的；

隔离性：事务操作之间彼此独立和透明互不影响。事务独立运行。这通常使用锁来实现。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。

持久性：事务一旦提交，其结果就是永久的。即便发生系统故障，也能恢复。

MySQL的事务隔离级别

未提交读(Read Uncommitted)：允许脏读，其他事务只要修改了数据，即使未提交，本事务也能看到修改后的数据值。也就是可能读取到其他会话中未提交事务修改的数据

提交读(Read Committed)：只能读取到已经提交的数据。Oracle等多数数据库默认都是该级别 (不重复读)。

可重复读(Repeated Read)：可重复读。无论其他事务是否修改并提交了数据，在这个事务中看到的数据值始终不受其他事务影响。

串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞

MySQL数据库(InnoDB引擎)默认使用可重复读（ Repeatable read)

索引

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B\_TREE。B\_TREE 索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据指针快速寻找数据。

MyISAM引擎使用B+Tree作为索引结构，叶节点的data域存放的是数据记录的地址，即：MyISAM索引文件和数据文件是分离的，MyISAM的索引文件仅仅保存数据记录的地址。MyISAM中索引检索的算法为首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其data域的值，然后以data域的值为地址，读取相应数据记录。MyISAM的索引方式也叫做“非聚集”的。

InnoDB引擎也使用B+Tree作为索引结构，但是InnoDB的数据文件本身就是索引文件，叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。这种索引叫做“聚焦索引”。InnoDB的辅助索引的data域存储相应记录主键的值而不是地址。换句话说，InnoDB的所有辅助索引都引用主键作为data域。聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。InnoDB的索引实现后，不建议使用过长的字段作为主键，因为所有辅助索引都引用主索引，过长的主索引会令辅助索引变得过大。在Innodb中也不建议使用非单调的字段作为主键，因为InnoDB数据文件本身是一颗B+Tree，非单调的主键会造成在插入新记录时数据文件为了维持B+Tree的特性而频繁的分裂调整，十分低效，建议使用自增字段作为主键。

MySQL数据库的四类索引:

　　index ---- 普通索引,数据可以重复，没有任何限制。

　　unique ---- 唯一索引,要求索引列的值必须唯一，但允许有空值；如果是组合索引，那么列值的组合必须唯一。

　　primary key ---- 主键索引,是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值，一般是在创建表的同时创建主键索引。

　　组合索引 ---- 在多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。

　　fulltext ---- 全文索引,是对于大表的文本域：char，varchar，text列才能创建全文索引，主要用于查找文本中的关键字，并不是直接与索引中的值进行比较。fulltext更像是一个搜索引擎，配合match against操作使用，而不是一般的where语句加like。

　　注:全文索引目前只有MyISAM存储引擎支持全文索引，InnoDB引擎5.6以下版本还不支持全文索引

　　所有存储引擎对每个表至少支持16个索引，总索引长度至少为256字节，索引有两种存储类型，包括B型树索引和哈希索引。

　　索引可以提高查询的速度，但是创建和维护索引需要耗费时间，同时也会影响插入的速度，如果需要插入大量的数据时，最好是先删除索引，插入数据后再建立索引。

索引生效条件

假设index（a,b,c）

最左前缀匹配：模糊查询时，使用%匹配时：’a%‘会使用索引，’%a‘不会使用索引

条件中有or，索引不会生效

a and c，a生效，c不生效

b and c，都不生效

a and b > 5 and c,a和b生效，c不生效。

检测索引的效果：

show status like '%handler\_read%'越大越好

sql语句分类：

DDL：数据定义语言（create drop）

DML：数据操作语句（insert update delete）

DQL：数据查询语句（select ）

DCL：数据控制语句，进行授权和权限回收（grant revoke）

TPL：数据事务语句（commit collback savapoint）

数据库三范式：

第一范式：1NF是对属性的原子性约束，要求字段具有原子性，不可再分解；(只要是关系型数据库都满足1NF)

第二范式：2NF是在满足第一范式的前提下，非主键字段不能出现部分依赖主键；解决：消除复合主键就可避免出现部分以来，可增加单列关键字。

第三范式：3NF是在满足第二范式的前提下，非主键字段不能出现传递依赖，比如某个字段a依赖于主键，而一些字段依赖字段a，这就是传递依赖。解决：将一个实体信息的数据放在一个表内实现。

存储引擎 MyISAM和InnoDB、MEMORY区别：

1）InnoDB支持事务，MyISAM不支持。

2）MyISAM适合查询以及插入为主的应用，InnoDB适合频繁修改以及涉及到安全性较高的应用。

3）InnoDB支持外键，MyISAM不支持。

4）从MySQL5.5.5以后，InnoDB是默认引擎。

5）MyISAM支持全文类型索引，而InnoDB不支持全文索引。

6）InnoDB中不保存表的总行数，select count(\*) from table时，InnoDB需要扫描整个表计算有多少行，但MyISAM只需简单读出保存好的总行数即可。注：当count(\*)语句包含where条件时MyISAM也需扫描整个表。

7）对于自增长的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中可以和其他字段一起建立联合索引。

8）清空整个表时，InnoDB是一行一行的删除，效率非常慢。MyISAM则会重建表。MyisAM使用delete语句删除后并不会立刻清理磁盘空间，需要定时清理，命令：OPTIMIZE table dept;

9）InnoDB支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like ‘%lee%’）

10）Myisam创建表生成三个文件：.frm 数据表结构 、 .myd 数据文件 、 .myi 索引文件，Innodb只生成一个 .frm文件，数据存放在ibdata1.log

现在一般都选用InnoDB，主要是MyISAM的全表锁，读写串行问题，并发效率锁表，效率低，MyISAM对于读写密集型应用一般是不会去选用的。

MEMORY是MySQL中一类特殊的存储引擎。它使用存储在内存中的内容来创建表，而且数据全部放在内存中。这些特性与前面的两个很不同。

每个基于MEMORY存储引擎的表实际对应一个磁盘文件。该文件的文件名与表名相同，类型为frm类型。该文件中只存储表的结构。而其数据文件，都是存储在内存中，这样有利于数据的快速处理，提高整个表的效率。值得注意的是，服务器需要有足够的内存来维持MEMORY存储引擎的表的使用。如果不需要了，可以释放内存，甚至删除不需要的表。

MEMORY默认使用哈希索引。速度比使用B型树索引快。当然如果你想用B型树索引，可以在创建索引时指定。

应用场景：

MyISAM不支持事务处理等高级功能，但它提供高速存储和检索，以及全文搜索能力。如果应用中需要执行大量的SELECT查询，那么MyISAM是更好的选择。

InnoDB用于需要事务处理的应用程序，包括ACID事务支持。如果应用中需要执行大量的INSERT或UPDATE操作，则应该使用InnoDB，这样可以提高多用户并发操作的性能。

MEMORY用到的很少，因为它是把数据存到内存中，如果内存出现异常就会影响数据。如果重启或者关机，所有数据都会消失。因此，基于MEMORY的表的生命周期很短，一般是一次性的。

CHAR和VARCHAR的区别：

CHAR和VARCHAR类型在存储和检索方面有所不同

CHAR列长度固定为创建表时声明的长度，长度值范围是1到255

当CHAR值被存储时，它们被用空格填充到特定长度，检索CHAR值时需删除尾随空格。

Mysql中有哪几种锁？

MyISAM支持表锁，InnoDB支持表锁和行锁，默认为行锁

表级锁：开销小，加锁快，不会出现死锁。锁定粒度大，发生锁冲突的概率最高，并发量最低

行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高

共享锁又称读锁（S锁），一个事务获取了共享锁，其他事务可以获取共享锁，不能获取排他锁，其他事务可以进行读操作，不能进行写操作。

排他锁又称写锁（X锁），如果事务T对数据A加上排他锁后，则其他事务不能再对A加任任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。

对于insert、update、delete，InnoDB会自动给涉及的数据加排他锁（X）；

对于一般的Select语句，InnoDB不会加任何锁，事务可以通过以下语句给显示加共享锁或排他锁。

共享锁：SELECT ... LOCK IN SHARE MODE;

排他锁：SELECT ... FOR UPDATE;

存储过程

我们常用的操作数据库语言SQL语句在执行的时候需要要先编译，然后执行，而存储过程（Stored Procedure）是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给定参数（如果该存储过程带有参数）来调用执行它。

一个存储过程是一个可编程的函数，它在数据库中创建并保存。它可以有SQL语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。

优点：

(1).存储过程增强了SQL语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

(2).存储过程允许标准组件是编程。存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。

(3).存储过程能实现较快的执行速度。如果某一操作包含大量的Transaction-SQL代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的Transaction-SQL语句在每次运行时都要进行编译和优化，速度相对要慢一些。

(4).存储过程能过减少网络流量。针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的Transaction-SQL语句被组织程存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大增加了网络流量并降低了网络负载。

(5).存储过程可被作为一种安全机制来充分利用。系统管理员通过执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。

delete、drop、truncate区别

truncate 和 delete只删除数据，不删除表结构 ,drop删除表结构，并且释放所占的空间。

删除数据的速度，drop> truncate > delete

delete属于DML语言，需要事务管理，commit之后才能生效。drop和truncate属于DDL语言，操作立刻生效，不可回滚。

使用场合：

当你不再需要该表时， 用 drop;

当你仍要保留该表，但要删除所有记录时， 用 truncate;

当你要删除部分记录时（always with a where clause), 用 delete.

注意： 对于有主外键关系的表，不能使用truncate而应该使用不带where子句的delete语句，由于truncate不记录在日志中，不能够激活触发器

什么是视图？以及视图的使用场景有哪些？

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

只暴露部分字段给访问者，所以就建一个虚表，就是视图。

查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异.

什么是二进制日志(binlog)

binlog是记录所有数据库表结构变更（例如CREATE、ALTER TABLE…）以及表数据修改（INSERT、UPDATE、DELETE…）的二进制日志。

binlog不会记录SELECT和SHOW这类操作，因为这类操作对数据本身并没有修改，但你可以通过查询通用日志来查看MySQL执行过的所有语句。

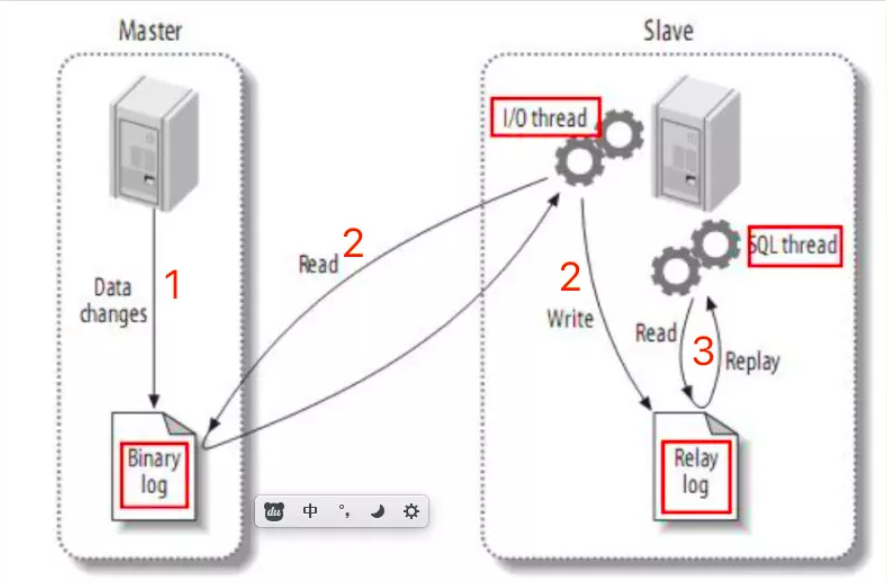
二进制日志包括两类文件：二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为.00000\*）记录数据库所有的DDL和DML(除了数据查询语句)语句事件。

Binlog常见格式



MySQL 的读写分离？MySQL 主从复制原理是啥？

主库将变更写入 binlog 日志，然后从库连接到主库之后，从库有一个 IO 线程，将主库的 binlog 日志拷贝到自己本地，写入一个 relay 中继日志中。接着从库中有一个 SQL 线程会从中继日志读取 binlog，然后执行 binlog 日志中的内容，也就是在自己本地再次执行一遍 SQL，这样就可以保证自己跟主库的数据是一样的。



主服务器上面的任何修改都会通过自己的 I/O tread(I/O 线程)保存在二进制日志 Binary log 里面。

从服务器上面也启动一个 I/O thread，通过配置好的用户名和密码, 连接到主服务器上面请求读取二进制日志，然后把读取到的二进制日志写到本地的一个Realy log（中继日志）里面。

从服务器上面同时开启一个 SQL thread 定时检查 Realy log(这个文件也是二进制的)，如果发现有更新立即把更新的内容在本机的数据库上面执行一遍。

每个从服务器都会收到主服务器二进制日志的全部内容的副本。

从服务器设备负责决定应该执行二进制日志中的哪些语句。

除非另行指定，否则主从二进制日志中的所有事件都在从站上执行。

如果需要，您可以将从服务器配置为仅处理一些特定数据库或表的事件。

而且这里还有另外一个问题，就是如果主库突然宕机，然后恰好数据还没同步到从库，那么有些数据可能在从库上是没有的，有些数据可能就丢失了。

所以 MySQL 实际上在这一块有两个机制，一个是半同步复制，用来解决主库数据丢失问题；一个是并行复制，用来解决主从同步延时问题。

这个所谓半同步复制，也叫 semi-sync 复制，指的就是主库写入 binlog 日志之后，就会将强制此时立即将数据同步到从库，从库将日志写入自己本地的 relay log 之后，接着会返回一个 ack 给主库，主库接收到至少一个从库的 ack 之后才会认为写操作完成了。

所谓并行复制，指的是从库开启多个线程，并行读取 relay log 中不同库的日志，然后并行重放不同库的日志，这是库级别的并行。

<https://www.cnblogs.com/lijiasnong/p/9963905.html>