## 如何避免 sql 注入？

<https://www.cnblogs.com/2016-10-07/p/6785106.html>

1. 采用预编译语句集，它内置了处理SQL注入的能力，只要使用它的setString方法传值即可。
2. 采用正则表达式将包含有 单引号(')，分号(;) 和 注释符号(--)的语句给替换掉来防止SQL注入。

## 数据库的三范式是什么？

<https://www.cnblogs.com/knowledgesea/p/3667395.html>

范式：优化数据数据存储方式的规范。

什么是三大范式：

**第一范式**：当关系模式R的所有属性都不能在分解为更基本的数据单位时，称R是满足第一范式的，简记为1NF。满足第一范式是关系模式规范化的最低要求，否则，将有很多基本操作在这样的关系模式中实现不了。

1、每一列属性都是不可再分的属性值，确保每一列的原子性

2、两列的属性相近或相似或一样，尽量合并属性一样的列，确保不产生冗余数据。

**第二范式**：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF。

每一行的数据只能与其中一列相关，即一行数据只做一件事。只要数据列中出现数据重复，就要把表拆分开来。

**第三范式**：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF.

数据不能存在传递关系，即没个属性都跟主键有直接关系而不是间接关系。像：a-->b-->c 属性之间含有这样的关系，是不符合第三范式的。

比如Student表（学号，姓名，年龄，性别，所在院校，院校地址，院校电话）

这样一个表结构，就存在上述关系。 学号--> 所在院校 --> (院校地址，院校电话)

这样的表结构，我们应该拆开来，如下。

（学号，姓名，年龄，性别，所在院校）--（所在院校，院校地址，院校电话）

最后：

三大范式只是一般设计数据库的基本理念，可以建立冗余较小、结构合理的数据库。如果有特殊情况，当然要特殊对待，数据库设计最重要的是看需求跟性能，**需求>性能>表结构**。所以不能一味的去追求范式建立数据库。

## mysql一张表里面有ID自增主键，当insert了17条记录之后，删除了第15,16,17条记录，再把mysql重启，再insert一条记录，这条记录的ID是18还是15 ？

一般情况下，我们创建的表的类型是InnoDB，如果新增一条记录（不重启mysql的情况下），这条记录的id是18；但是如果重启（文中提到的）MySQL的话，这条记录的ID是15。因为InnoDB表只把自增主键的最大ID记录到内存中，所以重启数据库或者对表OPTIMIZE操作，都会使最大ID丢失。

        但是，如果我们使用表的类型是MylSAM，那么这条记录的ID就是18。因为MylSAM表会把自增主键的最大ID记录到数据文件里面，重启MYSQL后，自增主键的最大ID也不会丢失。

## 如何获取当前数据库版本？

 select version();

MySQL -v

## 说一下 ACID 是什么？

事务是由一组SQL语句组成的逻辑处理单元，事务具有以下4个属性，通常简称为事务的ACID属性。

ACID是：

**Atomic（原子性）  
Consistency（一致性）  
Isolation（隔离性）  
Durability（持久性）**

Atomic（原子性）：指整个数据库事务是不可分割的工作单位。只有使据库中所有的操作执行成功，才算整个事务成功；事务中任何一个SQL语句执行失败，那么已经执行成功的SQL语句也必须撤销，数据库状态应该退回到执行事务前的状态。

Consistency（一致性）：指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。例如对银行转帐事务，不管事务成功还是失败，应该保证事务结束后ACCOUNTS表中Tom和Jack的存款总额为2000元。

Isolation（隔离性）：指的是在并发环境中，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。

Durability（持久性）：指的是只要事务成功结束，它对数据库所做的更新就必须永久保存下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

## char 和 varchar 的区别是什么？

<https://www.cnblogs.com/webph/p/6679815.html>

区别一，定长和变长

char 表示定长，长度固定，varchar表示变长，即长度可变。当所插入的字符串超出它们的长度时，视情况来处理，如果是严格模式，则会拒绝插入并提示错误信息，如果是宽松模式，则会截取然后插入。如果插入的字符串长度小于定义长度时，则会以不同的方式来处理，如char（10），表示存储的是10个字符，无论你插入的是多少，都是10个，如果少于10个，则用空格填满。而varchar（10），小于10个的话，则插入多少个字符就存多少个。

varchar怎么知道所存储字符串的长度呢？实际上，对于varchar字段来说，需要使用一个（如果字符串长度小于255）或两个字节（长度大于255）来存储字符串的长度。但是因为他需要有一个prefix来表示他具体bytes数是多少（因为varchar是变长的，没有这个长度值他不知道如何读取数据）。

区别之二，存储的容量不同

对 char 来说，最多能存放的字符个数 255，和编码无关。

而 varchar 呢，最多能存放 65532 个字符。VARCHAR 的最大有效长度由最大行大小和使用的字符集确定。整体最大长度是 65,532字节

## float 和 double 的区别是什么？

<https://www.cnblogs.com/gulibao/p/5416245.html>

<https://lihaiming.iteye.com/blog/2259943>

<https://zhidao.baidu.com/question/244412924587260364.html?qbl=relate_question_1&word=mysql%D6%D0float%BA%CDdouble%D3%D0%CA%B2%C3%B4%C7%F8%B1%F0>

IEEE754单精度浮点数的有效数字二进制是24位，按十进制来说，是8位；

双精度浮点数的有效数字二进制是53位，按十进制来说，是16 位。

## mysql 的内连接、左连接、右连接有什么区别？

<https://www.cnblogs.com/cdf-opensource-007/p/6517627.html>

内连接指的是把表连接时表与表之间匹配的数据行查询出来，就是两张表之间数据行匹配时，要同时满足ON语句后面的条件才行。

Select \* from table1 inner join table2 on [cond]

左连接和右连接，又叫左外连接和右外连接。

左连接语法：SELECT XXX FROM XXX LEFT OUTER JOIN XXX ON XXX

左连接的意思是，无论是否符合ON语句后面的表连接条件都会把左边那张表的记录全部查询出来，右边的那张表只匹配符合条件的数据行。右连接则与之相反（这里同样OUTER 可以省略）。

## mysql 索引是怎么实现的？

<https://blog.csdn.net/waeceo/article/details/78702584>

MySQL官方对索引的定义为：索引（Index）是帮助MySQL高效获取数据的数据结构。提取句子主干，就可以得到索引的本质：索引是一种数据结构。

myISAM和InnoDB的索引都是由B+树实现的，但具体实现方式不同。

B树和B+树介绍：<https://www.cnblogs.com/vincently/p/4526560.html>

B树，概括来说是一个节点可以拥有多于2个子节点的二叉查找树。与自平衡二叉查找树不同，B-树为系统最优化大块数据的读和写操作。B-tree算法减少定位记录时所经历的中间过程，从而加快存取速度。

2-3查找树**：**<https://www.cnblogs.com/yangecnu/p/Introduce-2-3-Search-Tree.html>

2-3树运行每个节点保存1个或者两个的值。对于普通的2节点(2-node)，他保存1个key和左右两个子树。对应3节点(3-node)，保存两个Key，2-3查找树的定义如下：

1. 要么为空，要么：

2. 对于2节点，该节点保存一个key及对应value，以及两个指向左右节点的节点，左节点也是一个2-3节点，所有的值都比key要小，右节点也是一个2-3节点，所有的值比key要大。

3. 对于3节点，该节点保存两个key及对应value，以及三个指向左中右的节点。左节点也是一个2-3节点，所有的值均比两个key中的最小的key还要小；中间节点也是一个2-3节点，中间节点的key值在两个跟节点key值之间；右节点也是一个2-3节点，节点的所有key值比两个key中的最大的key还要大。

如果中序遍历2-3查找树，就可以得到排好序的序列。在一个完全平衡的2-3查找树中，根节点到每一个为空节点的距离都相同。

B 树可以看作是对2-3查找树的一种扩展，即他允许每个节点有M-1个子节点。

* 根节点至少有两个子节点
* 每个节点有M-1个key，并且以升序排列
* 位于M-1和M key的子节点的值位

下图是一个M=4 阶的B树:



B+树是对B树的一种变形树，它与B树的差异在于：

* 有k个子结点的结点必然有k个关键码；
* 非叶结点仅具有索引作用，跟记录有关的信息均存放在叶结点中。
* 树的所有叶结点构成一个有序链表，可以按照关键码排序的次序遍历全部记录。

B和B+树的区别在于，B+树的非叶子结点只包含导航信息，不包含实际的值，所有的叶子结点和相连的节点使用链表相连，便于区间查找和遍历。

B+ 树的优点在于：

由于B+树在内部节点上不包含数据信息，因此在内存页中能够存放更多的key。 数据存放的更加紧密，具有更好的空间局部性。因此访问叶子节点上关联的数据也具有更好的缓存命中率。

B+树的叶子结点都是相链的，因此对整棵树的便利只需要一次线性遍历叶子结点即可。而且由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

但是B树也有优点，其优点在于，由于B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

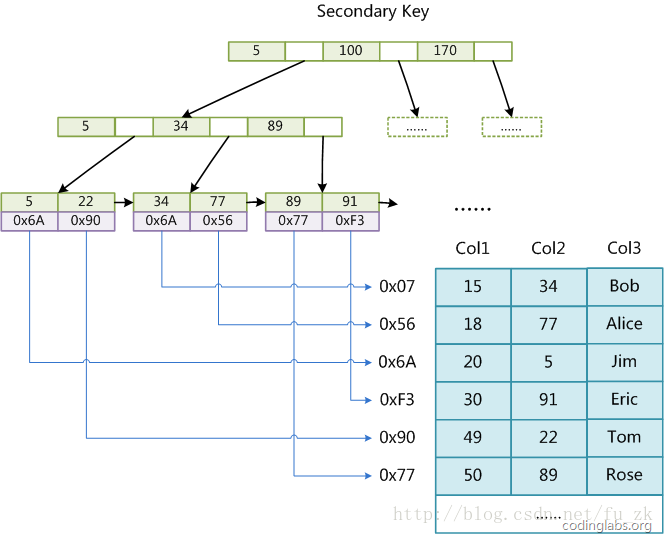
### MySQL索引实现

在MySQL中，索引属于存储引擎级别的概念，不同存储引擎对索引的实现方式是不同的，本文主要讨论MyISAM和InnoDB两个存储引擎([MySQL数据库MyISAM和InnoDB存储引擎的比较](http://www.cnblogs.com/tgycoder/p/5410096.html))的索引实现方式。

#### MyISAM索引实现

MyISAM引擎使用B+Tree作为索引结构，叶结点的data域存放的是数据记录的地址。MyISAM的索引文件仅仅保存数据记录的地址。在MyISAM中，主索引和辅助索引（Secondary key）在结构上没有任何区别，只是主索引要求key是唯一的，而辅助索引的key可以重复。

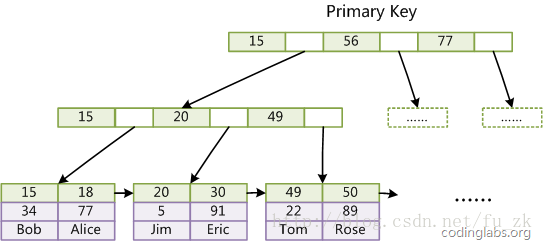
**MyISAM的索引方式也叫做“非聚集”的**，之所以这么称呼是为了与InnoDB的聚集索引区分。



#### InnoDB索引实现

虽然InnoDB也使用B+Tree作为索引结构，但具体实现方式却与MyISAM截然不同。

第一个重大区别是InnoDB的数据文件本身就是索引文件。从上文知道，MyISAM索引文件和数据文件是分离的，索引文件仅保存数据记录的地址。而在InnoDB中，表数据文件本身就是按B+Tree组织的一个索引结构，这棵树的叶结点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。



InnoDB主索引（同时也是数据文件）的示意图，可以看到**叶结点包含了完整的数据记录。**这种索引叫做聚集索引。因为InnoDB的数据文件本身要按主键聚集，所以InnoDB要求表必须有主键（MyISAM可以没有）。

如果没有显式指定，则MySQL系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则MySQL自动为InnoDB表生成一个隐含字段作为主键，这个字段长度为6个字节，类型为长整形。

第二个与MyISAM索引的不同是**InnoDB的辅助索引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。

了解不同存储引擎的索引实现方式对于正确使用和优化索引都非常有帮助，例如知道了InnoDB的索引实现后，就很容易明白为什么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。再例如，用非单调的字段作为主键在InnoDB中不是个好主意，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，而使用自增字段作为主键则是一个很好的选择。

## 怎么验证 mysql 的索引是否满足需求？

<https://blog.csdn.net/howroad/article/details/78227020>

<https://www.cnblogs.com/sand-tiny/p/3977645.html>

explain语句

完全看不懂这帖子他喵的

Explain语句：<https://segmentfault.com/a/1190000008131735>

## 在哪些列上添加索引比较好？

1.比较频繁的作为查询的字段

2.唯一性太差的字段不适合加索引，要找唯一性比较好的

3.更新太频繁的字段不适合做索引

4.不会出现在where中的 不应该建立索引

使用索引需要注意：

⑴只对WHERE和ORDER BY需要查询的字段设置索引，避免无意义的硬盘开销；

⑵组合索引支持前缀索引；

⑶更新表的时候，如增删记录，MySQL会自动更新索引，保持树的平衡；因此更多的索引意味着更多的维护成本

## mysql哪些情况下会应用索引？

<https://www.cnblogs.com/isykw/p/7190270.html>

**索引分四类：**

index ----普通的索引,数据可以重复

fulltext----全文索引，用来对大表的文本域(char，varchar，text)进行索引。语法和普通索引一样。

unique ----唯一索引,唯一索引,要求所有记录都唯一

primary key ----主键索引,也就是在唯一索引的基础上相应的列必须为主键

**like 不能用索引？**

* 尽量减少like，但不是绝对不可用，**”xxxx%” 是可以用到索引的，**

想象一下，你在看一本成语词典，目录是按成语拼音顺序建立，查询需求是，你想找以 “一”字开头的成语（”一%“），和你想找包含一字的成语（“%一%”）

* 除了like，以下操作符也可用到索引：

<，<=，=，>，>=，BETWEEN，IN

<>，not in ，！=则不行

原则：

　　1，单表数据太少，索引反而会影响速度；更新非常频繁的数据不适宜建索引

　　2，where后的条件，order by ,group by 等这样过滤时，后面的字段最好加上索引。根据实际情况，选择PRIMARY KEY、UNIQUE、INDEX等索引，但是不是越多越好，要适度

   　3，联合查询，子查询等多表操作时关连字段要加索引

ps：数据量特别大的时候，最好不要用联合查询，即使你做了索引

## 哪些常见情况不能用索引?

* like “%xxx”
* not in ， ！=
* 对列进行函数运算的情况（如 where md5(password) = “xxxx”）
* WHERE index=1 OR A=10
* 存了数值的字符串类型字段（如手机号），查询时记得不要丢掉值的引号，否则无法用到该字段相关索引，反之则没关系

也即

select \* from test where mobile = 13711112222;

可是无法用到mobile字段的索引的哦（如果mobile是char 或 varchar类型的话）

btw，**千万不要尝试用int来存手机号**（为什么？自己想！要不自己试试）

## 说一下数据库的事务隔离？

<https://www.cnblogs.com/fjdingsd/p/5273008.html>

在MySQL数据库中查看当前事务的隔离级别：

select **@@tx\_isolation**;

## 说一下 mysql 常用的引擎？

<https://www.cnblogs.com/xiaohaillong/p/6079551.html>

看你的mysql现在已提供什么存储引擎:

mysql> show engines;

看你的mysql当前默认的存储引擎:

mysql> show variables like '%storage\_engine%';

你要看某个表用了什么引擎(在显示结果里参数engine后面的就表示该表当前用的存储引擎):

mysql> show create table 表名;

a.Innodb引擎，Innodb引擎提供了对数据库ACID事务的支持。并且还提供了行级锁和外键的约束。它的设计的目标就是处理大数据容量的数据库系统。它本身实际上是基于Mysql后台的完整的系统。Mysql运行的时候，Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是，该引擎是不支持全文搜索的。同时，启动也比较的慢，它是不会保存表的行数的。当进行Select count(\*) from table指令的时候，需要进行扫描全表。所以当需要使用数据库的事务时，该引擎就是首选。由于锁的粒度小，写操作是不会锁定全表的。所以在并发度较高的场景下使用会提升效率的。

b.MyIASM引擎，它是MySql的默认引擎，但不提供事务的支持，也不支持行级锁和外键。因此当执行Insert插入和Update更新语句时，即执行写操作的时候需要锁定这个表。所以会导致效率会降低。不过和Innodb不同的是，MyIASM引擎是保存了表的行数，于是当进行Select count(\*) from table语句时，可以直接的读取已经保存的值而不需要进行扫描全表。所以，如果表的读操作远远多于写操作时，并且不需要事务的支持的。可以将MyIASM作为数据库引擎的首选。

c.大容量的数据集时趋向于选择Innodb。因为它支持事务处理和故障的恢复。Innodb可以利用数据日志来进行数据的恢复。主键的查询在Innodb也是比较快的。

d.大批量的插入语句时（这里是INSERT语句）在MyIASM引擎中执行的比较的快，但是UPDATE语句在Innodb下执行的会比较的快，尤其是在并发量大的时候。

2.两种引擎所使用的索引的数据结构是什么？

答案:都是B+树!

MyIASM引擎，B+树的数据结构中存储的内容实际上是实际数据的地址值。也就是说它的索引和实际数据是分开的，只不过使用索引指向了实际数据。这种索引的模式被称为非聚集索引。

Innodb引擎的索引的数据结构也是B+树，只不过数据结构中存储的都是实际的数据，这种索引有被称为聚集索引。

## 说一下 mysql 的行锁和表锁？

<https://www.cnblogs.com/chenqionghe/p/4845693.html>

* 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，并发度最低。
* 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

### MySQL表级锁的锁模式（MyISAM)

MySQL表级锁有两种模式：表共享锁（Table Read Lock）和表独占写锁（Table Write Lock）。

* 对MyISAM的读操作，不会阻塞其他用户对同一表请求，但会阻塞对同一表的写请求；
* 对MyISAM的写操作，则会阻塞其他用户对同一表的读和写操作；
* MyISAM表的读操作和写操作之间，以及写操作之间是串行的。

当一个线程获得对一个表的写锁后，只有持有锁线程可以对表进行更新操作。其他线程的读、写操作都会等待，直到锁被释放为止。

对ＭyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写请求；对ＭyISAM表的写操作，则会阻塞其他用户对同一表的读和写请求；ＭyISAM表的读和写操作之间，以及写和写操作之间是**串行**的！（**当一线程获得对一个表的写锁后，只有持有锁的线程可以对表进行更新操作。其他线程的读、写操作都会等待，直到锁被释放为止。**）

### InnoDB锁问题

    InnoDB与MyISAM的最大不同有两点：一是支持事务（TRANSACTION）；二是采用了行级锁。

行级锁和表级锁本来就有许多不同之处，另外，事务的引入也带来了一些新问题。

InnoDB实现了以下两种类型的行锁。

* 共享锁（s）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。
* 排他锁（Ｘ）：允许获取排他锁的事务更新数据，阻止其他事务取得相同的数据集共享读锁和排他写锁。

另外，为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB还有两种内部使用的意向锁（Intention Locks），这两种意向锁都是表锁。

意向共享锁（IS）：事务打算给数据行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。

意向排他锁（IX）：事务打算给数据行加排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

## 说一下乐观锁和悲观锁？

　也许大家已经听说过，锁分两种，一个叫 悲观锁，一种称之为 乐观锁。事实上，无论是悲观锁还是乐观锁，都是人们定义出来的概念，是一种解决问题的思想。因此，不仅仅在数据库系统中有乐观锁和悲观锁的概念，像memcache、hibernate、tair等都有类似的概念。比如，在线程并发处理中， Synchronized内置锁 就是悲观锁的一种，也称之为 独占锁，加了synchronized关键字的代码基本上就只能以单线程的形式去执行了，它会导致其他需要该资源的线程挂起，直到前面的线程执行完毕释放所资源；而 乐观锁是一种更高效的机制，它的原理就是每次不加锁去执行某项操作，如果发生冲突则失败并重试，直到成功为止，其实本质上不算锁，所以很多地方也称之为 自旋。

　　在解决数据库的事务并发访问问题时，虽然将事务串形化可以保证数据在多事务并发处理下不存在数据不一致的问题，但串行执行使得数据库的处理性能大幅度地下降，常常是我们接受不了的。所以，一般来说，我们常常结合事务隔离级别和其它并发机制来保证事务的并发，以此来兼顾事务并发的效率与安全性。事实上，大多数数据库的隔离级别都会设置为 Read Committed（只能读取其他事务已提交的数据），然后由应用程序使用乐观锁/悲观锁机制来解决其他事务并发问题，比如不可重复读问题。特别地，乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

　　特别地，乐观锁的理念是：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性；而悲观锁的理念是假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。针对于不同的业务场景，应该选用不同的并发控制方式。所以，不要把乐观并发控制和悲观并发控制狭义的理解为DBMS中的概念，更不要把他们和数据中提供的锁机制（行锁、表锁、排他锁、共享锁）混为一谈。需要指出的是，在DBMS中，悲观锁正是利用数据库本身提供的锁机制来实现的。

1、乐观锁

　　乐观锁，虽然名字中带“锁”，但是乐观锁并不锁住任何东西，而是在提交事务时检查这条记录是否被其他事务进行了修改：如果没有，则提交；否则，进行回滚。相对于悲观锁，在对数据库进行处理的时候，乐观锁并不会使用数据库提供的锁机制。如果并发的可能性并不大，那么乐观锁定策略带来的性能消耗是非常小的。乐观锁采用的实现方式一般是记录数据版本。

　　数据版本是为数据增加的一个版本标识。当读取数据时，将版本标识的值一同读出，数据每更新一次同时对版本标识进行更新。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的版本标识进行比对，如果数据库表当前版本号与第一次取出来的版本标识值相等，则予以更新，否则认为是过期数据。一般地，实现数据版本有两种方式，一种是使用版本号，另一种是使用时间戳。

2、悲观锁

　　悲观锁，正如其名，它指的是对数据被外界修改持保守(悲观)态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现往往依靠数据库提供的锁机制，也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据。悲观并发控制主要用于数据争用激烈的环境，以及发生并发冲突时使用锁保护数据的成本要低于回滚事务的成本的环境中。和乐观锁相比，悲观锁则是一把真正的锁了，它通过SQL语句“select for update”锁住select出的那批数据，这时如果其他事务来更新这批数据时会等待。

　　悲观并发控制实际上是“先取锁再访问”的保守策略，为数据处理的安全提供了保证。但是在效率方面，处理加锁的机制会让数据库产生额外的开销，还有增加产生死锁的机会；另外，在只读型事务处理中由于不会产生冲突，也没必要使用锁，这样做只能增加系统负载；还有会降低了并行性，一个事务如果锁定了某行数据，其他事务就必须等待该事务处理完才可以处理那行数据。

## mysql 问题排查都有哪些手段？

<https://blog.csdn.net/weixin_34220179/article/details/88160163>

explain分析sql语句

## 如何做 mysql 的性能优化？

索引包含了所需的全部值的话，就只select 他们，换言之，**只select 需要用到的字段**，如无必要，可尽量避免select \*

<https://www.cnblogs.com/zhouyusheng/p/8038224.html>

#### 当只要一行数据时使用 LIMIT 1

当你查询表的有些时候，你已经知道结果只会有一条结果，但因为你可能需要去fetch游标，或是你也许会去检查返回的记录数。

在这种情况下，加上 LIMIT 1 可以增加性能。这样一样，MySQL数据库引擎会在找到一条数据后停止搜索，而不是继续往后查少下一条符合记录的数据。

#### 在Join表的时候使用相当类型的例，并将其索引

如果你的应用程序有很多 JOIN 查询，你应该确认两个表中Join的字段是被建过索引的。这样，MySQL内部会启动为你优化Join的SQL语句的机制。

#### 千万不要 ORDER BY RAND()

想打乱返回的数据行？随机挑一个数据？真不知道谁发明了这种用法，但很多新手很喜欢这样用。但你确不了解这样做有多么可怕的性能问题。

如果你真的想把返回的数据行打乱了，你有N种方法可以达到这个目的。这样使用只让你的数据库的性能呈指数级的下降。这里的问题是：MySQL会不得不去执行RAND()函数（很耗CPU时间），而且这是为了每一行记录去记行，然后再对其排序。就算是你用了Limit 1也无济于事（因为要排序）

#### 避免 SELECT \*

从数据库里读出越多的数据，那么查询就会变得越慢。并且，如果你的数据库服务器和WEB服务器是两台独立的服务器的话，这还会增加网络传输的负载。

所以，你应该养成一个需要什么就取什么的好的习惯。

#### 使用 ENUM 而不是 VARCHAR

[ENUM](http://dev.mysql.com/doc/refman/5.0/en/enum.html) 类型是非常快和紧凑的。在实际上，其保存的是 TINYINT，但其外表上显示为字符串。这样一来，用这个字段来做一些选项列表变得相当的完美。

如果你有一个字段，比如“性别”，“国家”，“民族”，“状态”或“部门”，你知道这些字段的取值是有限而且固定的，那么，你应该使用 ENUM 而不是 VARCHAR。

MySQL也有一个“建议”（见第十条）告诉你怎么去重新组织你的表结构。当你有一个 VARCHAR 字段时，这个建议会告诉你把其改成 ENUM 类型。

#### 尽可能的使用 NOT NULL

除非你有一个很特别的原因去使用 NULL 值，你应该总是让你的字段保持 NOT NULL。这看起来好像有点争议，请往下看。

首先，问问你自己“Empty”和“NULL”有多大的区别（如果是INT，那就是0和NULL）？如果你觉得它们之间没有什么区别，那么你就不要使用NULL。（你知道吗？在 Oracle 里，NULL 和 Empty 的字符串是一样的！)

不要以为 NULL 不需要空间，其需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。 当然，这里并不是说你就不能使用NULL了，现实情况是很复杂的，依然会有些情况下，你需要使用NULL值。

#### 固定长度的表会更快

如果表中的所有字段都是“固定长度”的，整个表会被认为是 [“static” 或 “fixed-length”](http://dev.mysql.com/doc/refman/5.1/en/static-format.html)。 例如，表中没有如下类型的字段： VARCHAR，TEXT，BLOB。只要你包括了其中一个这些字段，那么这个表就不是“固定长度静态表”了，这样，MySQL 引擎会用另一种方法来处理。

固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

#### 拆分大的 DELETE 或 INSERT 语句

如果你需要在一个在线的网站上去执行一个大的 DELETE 或 INSERT 查询，你需要非常小心，要避免你的操作让你的整个网站停止相应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。

Apache 会有很多的子进程或线程。所以，其工作起来相当有效率，而我们的服务器也不希望有太多的子进程，线程和数据库链接，这是极大的占服务器资源的事情，尤其是内存。

如果你把你的表锁上一段时间，比如30秒钟，那么对于一个有很高访问量的站点来说，这30秒所积累的访问进程/线程，数据库链接，打开的文件数，可能不仅仅会让你泊WEB服务Crash，还可能会让你的整台服务器马上掛了。

DELETE FROM logs WHERE log\_date <= '2009-11-01' LIMIT 1000

## 用一条SQL 语句 查询出每门课都大于80 分的学生姓名

name   kecheng   fenshu  
张三    语文       81  
张三     数学       75  
李四     语文       76  
李四     数学       90  
王五     语文       81  
王五     数学       100  
王五     英语       90

 select distinct name from table where name not in (select distinct name from table where fenshu<=80)

或者：

select name from table group by name having min(fenshu)>80

## 删除除了自动编号不同, 其他都相同的学生冗余信息

学生表 如下:

自动编号   学号   姓名 课程编号 课程名称 分数  
1        2005001 张三 0001     数学    69  
2        2005002 李四 0001      数学    89  
3        2005001 张三 0001      数学    69

delete tablename where 自动编号 not in(select min( 自动编号) from tablename group by学号, 姓名, 课程编号, 课程名称, 分数)

sql组合查询练习题：

<https://www.cnblogs.com/diffrent/p/8854995.html>

## mysql如何复制表？

1、复制表结构：

CREATE TABLE 新表 SELECT \* FROM 旧表 WHERE 1=2;

或

CREATE TABLE 新表 LIKE 旧表 ;

注意上面两种方式，前一种方式主键类型和自增方式是不会复制过去的，而后一种方式是把旧表的所有字段类型都复制到新表。

2、复制旧表的数据到新表(假设两个表结构一样)

INSERT INTO 新表 SELECT \* FROM 旧表

3、复制旧表的数据到新表(假设两个表结构不一样)

INSERT INTO 新表(字段1,字段2,.......) SELECT 字段1,字段2,...... FROM 旧表

## 触发器的作用？

触发器是一中特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

## 什么是存储过程？用什么来调用？

存储过程是一个预编译的SQL 语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL ，使用存储过程比单纯SQL 语句执行要快。

用CallableStatement来调用存储过程。

## 维护数据库的完整性和一致性，你喜欢用触发器还是自写业务逻辑？为什么？

尽可能使用约束，如check, 主键，外键，非空字段等来约束，这样做效率最高，也最方便。其次是使用触发器，这种方法可以保证，无论什么业务系统访问数据库都可以保证数据的完整新和一致性。最后考虑的是自写业务逻辑，但这样做麻烦，编程复杂，效率低下。

## 请解释 主键 超键 候选键 外键

主 键：

数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

超 键：

在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

候选键：

是最小超键，即没有冗余元素的超键。

外 键：

在一个表中存在的另一个表的主键称此表的外键。

## drop,delete与truncate的区别

drop直接删掉表

truncate删除表中数据，再插入时自增长id又从1开始

delete删除表中数据，可以加where字句

（1） DELETE语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行进行回滚操作。TRUNCATE TABLE 则一次性地从表中删除所有的数据并不把单独的删除操作记录记入日志保存，删除行是不能恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。

（2） 表和索引所占空间。当表被TRUNCATE 后，这个表和索引所占用的空间会恢复到初始大小，而DELETE操作不会减少表或索引所占用的空间。drop语句将表所占用的空间全释放掉。

（3） 一般而言，drop > truncate > delete

（4） 应用范围。TRUNCATE 只能对TABLE；DELETE可以是table和view

（5） TRUNCATE 和DELETE只删除数据，而DROP则删除整个表（结构和数据）。

（6） truncate与不带where的delete ：只删除数据，而不删除表的结构（定义）drop语句将删除表的结构被依赖的约束（constrain),触发器（trigger)索引（index);依赖于该表的存储过程/函数将被保留，但其状态会变为：invalid。

（7） delete语句为DML（data maintain Language),这个操作会被放到 rollback segment中,事务提交后才生效。如果有相应的 tigger,执行的时候将被触发。

（8） truncate、drop是DLL（data define language),操作立即生效，原数据不放到 rollback segment中，不能回滚

（9） 在没有备份情况下，谨慎使用 drop 与 truncate。要删除部分数据行采用delete且注意结合where来约束影响范围。回滚段要足够大。要删除表用drop;若想保留表而将表中数据删除，如果于事务无关，用truncate即可实现。如果和事务有关，或想触发trigger,还是用delete。

（10） Truncate table 表名 速度快,而且效率高,因为:

truncate table 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。DELETE 语句每次删除一行，并在事务日志中为所删除的每行记录一项。TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。

（11） TRUNCATE TABLE 删除表中的所有行，但表结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用 DELETE。如果要删除表定义及其数据，请使用 DROP TABLE 语句。

（12） 对于由 FOREIGN KEY 约束引用的表，不能使用 TRUNCATE TABLE，而应使用不带 WHERE 子句的 DELETE 语句。由于 TRUNCATE TABLE 不记录在日志中，所以它不能激活触发器。

## 存储过程与触发器的区别

触发器与存储过程非常相似，触发器也是SQL语句集，两者唯一的区别是触发器不能用EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发（激活）执行。

触发器是在一个修改了指定表中的数据时执行的存储过程。通常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。

触发器不同于存储过程，触发器主要是通过事件执行触发而被执行的，而存储过程可以通过存储过程名称名字而直接调用。当对某一表进行诸如UPDATE、INSERT、DELETE这些操作时，SQLSERVER就会自动执行触发器所定义的SQL语句，从而确保对数据的处理必须符合这些SQL语句所定义的规则。

## SQL语言的分类

SQL语言共分为四大类：

数据查询语言DQL SELECT子句，FROM子句，WHERE子句组成的查询块

数据操纵语言DML 插入、更新、删除

数据定义语言DDL 用来创建数据库中的各种对象-----表、视图、索引、同义词、聚簇等如：

CREATE TABLE/VIEW/INDEX/SYN/CLUSTER

### 数据控制语言DCL

数据控制语言DCL用来授予或回收访问数据库的某种特权，并控制

数据库操纵事务发生的时间及效果，对数据库实行监视等。如：

1) GRANT：授权。

2) ROLLBACK [WORK] TO [SAVEPOINT]：回退到某一点。

回滚---ROLLBACK

回滚命令使数据库状态回到上次最后提交的状态。其格式为：

SQL>ROLLBACK;

3) COMMIT [WORK]：提交。

在数据库的插入、删除和修改操作时，只有当事务在提交到数据

库时才算完成。在事务提交前，只有操作数据库的这个人才能有权看

到所做的事情，别人只有在最后提交完成后才可以看到。

提交数据有三种类型：显式提交、隐式提交及自动提交。下面分

别说明这三种类型。

(1) 显式提交

用COMMIT命令直接完成的提交为显式提交。其格式为：

SQL>COMMIT；

(2) 隐式提交

用SQL命令间接完成的提交为隐式提交。这些命令是：

ALTER，AUDIT，COMMENT，CONNECT，CREATE，DISCONNECT，DROP，

EXIT，GRANT，NOAUDIT，QUIT，REVOKE，RENAME。

(3) 自动提交

若把AUTOCOMMIT设置为ON，则在插入、修改、删除语句执行后，

系统将自动进行提交，这就是自动提交。其格式为：

SQL>SET AUTOCOMMIT ON；

## myisam与innodb select count(\*)哪个更快，为什么？

myisam更快，因为myisam内部维护了一个计数器，可以直接调取。

varchar(20)和int(20)中的20含义一样吗？

答：不一样，前者表示最多存放20个字符，后者表示最多显示20个字符，但是存储空间还是占4字节存储，存储范围不变；

MySQL主要的索引类型？

普通索引：是最基本的索引，它没有任何限制；

唯一索引：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一；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；

组合索引：指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。使用组合索引时遵循最左前缀原则；

全文索引：主要用来查找文本中的关键字，而不是直接与索引中的值相比较，mysql中MyISAM支持全文索引而InnoDB不支持；

## 使用like 'a%' 、like'%a'、like'%a%'查询时是否会使用索引？

答：'a%'会，其他两个不会；

## 说说使用索引注意事项

索引不会包含有NULL的列，复合索引中只要有一列含有NULL值，那么这一列对于此符合索引就是无效的；

使用短索引，对串列进行索引，如果可以就应该指定一个前缀长度；

短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作；

mysql查询只使用一个索引，因此数据库默认排序可以符合要求的情况下不要使用排序操作，尽量不要包含多个列的排序，如果需要最好给这些列建复合索引；

注意like，上文已经提到；

不要在列上进行运算；

不使用NOT IN 、<>、！=操作，但<,<=，=，>,>=,BETWEEN,IN是可以用到索引的；

索引要建立在经常进行select操作的字段上；

索引要建立在值比较唯一的字段上；

对于那些定义为text、image和bit数据类型的列不应该增加索引；

在where和join中出现的列需要建立索引；

如果where字句的查询条件里使用了函数(如：where DAY(column)=…),mysql将无法使用索引；

在join操作中(需要从多个数据表提取数据时)，mysql只有在主键和外键的数据类型相同时才能使用索引，否则及时建立了索引也不会使用；

说一说什么是外键，优缺点

答：外键指的是外键约束，目的是保持数据一致性，完整性，控制存储在外键表中的数据。使两张表形成关联，外键只能引用外表中列的值；优点：由数据库自身保证数据一致性，完整性，更可靠，因为程序很难100％保证数据的完整性，而用外键即使在数据库服务器当机或者出现其他问题的时候，也能够最大限度的保证数据的一致性和完整性。有主外键的数据库设计可以增加ER图的可读性，这点在数据库设计时非常重要。外键在一定程度上说明的业务逻辑，会使设计周到具体全面。

缺点：可以用触发器或应用程序保证数据的完整性；过分强调或者说使用外键会平添开发难度，导致表过多，更改业务困难，扩展困难等问题；不用外键时数据管理简单，操作方便，性能高（导入导出等操作，在insert, update, delete 数据的时候更快）；

mysql有没有rowid？

答：没有，InnoDB如果没有定义主键，内部会生成一个主键编号rowid ，但是无法查询到。在平时InnoDB建表的时候我们最好自己确定主键，防止每次插入数据前数据库会去生成rowid。

## § 锁机制介绍：行锁、表锁、排他锁、共享锁；

### MyISAM表锁

MySQL的表级锁有两种模式：表共享读锁（Table Read Lock）和表独占写锁（Table Write Lock）。

对MyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写请求；对 MyISAM表的写操作，则会阻塞其他用户对同一表的读和写操作；MyISAM表的读操作与写操作之间，以及写操作之间是串行的！

MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，在执行更新操作 （UPDATE、DELETE、INSERT等）前，会自动给涉及的表加写锁，这个过程并不需要用户干预。

### InnoDB的行锁模式及加锁方法

InnoDB实现了以下两种类型的行锁。

共享锁（s）：又称读锁。允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

排他锁（Ｘ）：又称写锁。允许获取排他锁的事务更新数据，阻止其他事务取得相同的数据集共享读锁和排他写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，直到T释放A上的锁。

对于共享锁大家可能很好理解，就是多个事务只能读数据不能改数据。

对于排他锁大家的理解可能就有些差别，我当初就犯了一个错误，以为排他锁锁住一行数据后，其他事务就不能读取和修改该行数据，其实不是这样的。排他锁指的是一个事务在一行数据加上排他锁后，其他事务不能再在其上加其他的锁。**mysql InnoDB引擎默认的修改数据语句：update,delete,insert都会自动给涉及到的数据加上排他锁**，select语句默认不会加任何锁类型，如果加排他锁可以使用select …for update语句，加共享锁可以使用select … lock in share mode语句。所以加过排他锁的数据行在其他事务种是不能修改数据的，也不能通过for update和lock in share mode锁的方式查询数据，但可以直接通过select …from…查询数据，因为普通查询没有任何锁机制。

另外，为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB还有两种内部使用的意向锁（Intention Locks），这两种意向锁都是表锁。

意向共享锁（IS）：事务打算给数据行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。

意向排他锁（IX）：事务打算给数据行加排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

## § 事务介绍，分布式事务的理解，常见的解决方案有哪些，什么是两阶段提交、三阶段提交；

**如果一个事务调用了不同服务器上的操作，那么它就成为了一个分布式事务。**

<https://www.cnblogs.com/savorboard/p/distributed-system-transaction-consistency.html>

<https://blog.csdn.net/sundehui01/article/details/79169523>

**分布式事务是指会涉及到操作多个数据库的事务**。其实就是将对同一库事务的概念扩大到了对多个库的事务。目的是为了保证分布式系统中的数据一致性。分布式事务处理的关键是必须有一种方法可以知道事务在任何地方所做的所有动作，提交或回滚事务的决定必须产生统一的结果（全部提交或全部回滚）

**二阶段提交(Two-phaseCommit)**是指，在计算机网络以及数据库领域内，为了使基于分布式系统架构下的所有节点在进行事务提交时保持一致性而设计的一种算法(Algorithm)。通常，二阶段提交也被称为是一种协议(Protocol))。在分布式系统中，每个节点虽然可以知晓自己的操作时成功或者失败，却无法知道其他节点的操作的成功或失败。当一个事务跨越多个节点时，为了保持事务的ACID特性，需要引入一个作为协调者的组件来统一掌控所有节点(称作参与者)的操作结果并最终指示这些节点是否要把操作结果进行真正的提交(比如将更新后的数据写入磁盘等等)。因此，二阶段提交的算法思路可以概括为：参与者将操作成败通知协调者，再由协调者根据所有参与者的反馈情报决定各参与者是否要提交操作还是中止操作。

所谓的两个阶段是指：第一阶段：准备阶段(投票阶段)和第二阶段：提交阶段（执行阶段）。

**三阶段提交**有两个改动点：

1、引入超时机制。同时在协调者和参与者中都引入超时机制。

2、在第一阶段和第二阶段中插入一个准备阶段。保证了在最后提交阶段之前各参与节点的状态是一致的。

也就是说，除了引入超时机制之外，3PC把2PC的准备阶段再次一分为二，这样三阶段提交就有CanCommit、PreCommit、DoCommit三个阶段。

### CanCommit阶段

    3PC的CanCommit阶段其实和2PC的准备阶段很像。协调者向参与者发送commit请求，参与者如果可以提交就返回Yes响应，否则返回No响应。

1.事务询问 协调者向参与者发送CanCommit请求。询问是否可以执行事务提交操作。然后开始等待参与者的响应。

2.响应反馈 参与者接到CanCommit请求之后，正常情况下，如果其自身认为可以顺利执行事务，则返回Yes响应，并进入预备状态。否则反馈No

### PreCommit阶段

   协调者根据参与者的反应情况来决定是否可以进行事务的PreCommit操作。根据响应情况，有以下两种可能。

    假如协调者从所有的参与者获得的反馈都是Yes响应，那么就会执行事务的预执行。

1.发送预提交请求 协调者向参与者发送PreCommit请求，并进入Prepared阶段。

2.事务预提交 参与者接收到PreCommit请求后，会执行事务操作，并将undo和redo信息记录到事务日志中。

3.响应反馈 如果参与者成功的执行了事务操作，则返回ACK响应，同时开始等待最终指令。

    假如有任何一个参与者向协调者发送了No响应，或者等待超时之后，协调者都没有接到参与者的响应，那么就执行事务的中断。

1.发送中断请求 协调者向所有参与者发送abort请求。

2.中断事务 参与者收到来自协调者的abort请求之后（或超时之后，仍未收到协调者的请求），执行事务的中断。

### doCommit阶段

    该阶段进行真正的事务提交，也可以分为以下两种情况。

    执行提交

1.发送提交请求 协调接收到参与者发送的ACK响应，那么他将从预提交状态进入到提交状态。并向所有参与者发送doCommit请求。

2.事务提交 参与者接收到doCommit请求之后，执行正式的事务提交。并在完成事务提交之后释放所有事务资源。

3.响应反馈 事务提交完之后，向协调者发送Ack响应。

4.完成事务 协调者接收到所有参与者的ack响应之后，完成事务。

中断事务 协调者没有接收到参与者发送的ACK响应（可能是接受者发送的不是ACK响应，也可能响应超时），那么就会执行中断事务。

1.发送中断请求 协调者向所有参与者发送abort请求

2.事务回滚 参与者接收到abort请求之后，利用其在阶段二记录的undo信息来执行事务的回滚操作，并在完成回滚之后释放所有的事务资源。

3.反馈结果 参与者完成事务回滚之后，向协调者发送ACK消息

4.中断事务 协调者接收到参与者反馈的ACK消息之后，执行事务的中断。

    在doCommit阶段，如果参与者无法及时接收到来自协调者的doCommit或者rebort请求时，会在等待超时之后，会继续进行事务的提交。（其实这个应该是基于概率来决定的，当进入第三阶段时，说明参与者在第二阶段已经收到了PreCommit请求，那么协调者产生PreCommit请求的前提条件是他在第二阶段开始之前，收到所有参与者的CanCommit响应都是Yes。（一旦参与者收到了PreCommit，意味他知道大家其实都同意修改了）所以，一句话概括就是，当进入第三阶段时，由于网络超时等原因，虽然参与者没有收到commit或者abort响应，但是他有理由相信：成功提交的几率很大。 ）

## § MySQL记录binlog的方式主要包括三种模式？每种模式的优缺点是什么？

### Row Level  行模式

日志中会记录每一行数据被修改的形式，然后在slave端再对相同的数据进行修改

**优点：**在row level模式下，bin-log中可以不记录执行的sql语句的上下文相关的信息，仅仅只需要记录那一条被修改。所以rowlevel的日志内容会非常清楚的记录下每一行数据修改的细节。不会出现某些特定的情况下的存储过程或function，以及trigger的调用和触发无法被正确复制的问题

**缺点：**row level，所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，会产生大量的日志内容。

### Statement Level（默认）

每一条会修改数据的sql都会记录到master的bin-log中。slave在复制的时候sql进程会解析成和原来master端执行过的相同的sql来再次执行

优点：statement level下的优点首先就是解决了row level下的缺点，不需要记录每一行数据的变化，减少bin-log日志量，节约IO，提高性能，因为它只需要在Master上锁执行的语句的细节，以及执行语句的上下文的信息。

缺点：由于只记录语句，所以，在statement level下 已经发现了有不少情况会造成MySQL的复制出现问题，主要是修改数据的时候使用了某些定的函数或者功能的时候会出现。

### Mixed 自动模式

在Mixed模式下，MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志格式，也就是在Statement和Row之间选择一种。如果sql语句确实就是update或者delete等修改数据的语句，那么还是会记录所有行的变更。

## § MySQL锁，悲观锁、乐观锁、排它锁、共享锁、表级锁、行级锁；

<https://www.cnblogs.com/leedaily/p/8378779.html>

## § SQL的整个解析、执行过程原理、SQL行转列；

困了，睡觉