MYSQL性能调优与架构设计

**MYSQL设计原则:**简单、高效、可靠

# MYSQL物理文件组成

## 日志文件：

**错误日志(Error Log):启动参数--log-error[=file\_name]**错误日志记录了MYSQL Server运行过程中所有较为严重的警告和错误信息，已经MYSQL每次启动和关闭的详细信息。在默认情况下，系统记录错误日志的功能是关闭的，错误信息被输出到标准错误输出，如果开启系统记录错误日志的功能，需要在启动是开始-log-error选项

**二进制日志(Binary Log):启动参数--log-bin[=file\_name]**二进制日志，MYSQL会将所有修改数据库的query以二进制形式记录到日志文件中。当然日志中并不仅限于query语句这么简单，还包括每一条query所执行的时间，所消耗的资源，以及修改的事务信息，所以binlog是事务安全的

**更新日志(update log):**更新日志是MYSQL在较老的版本上使用的，其功能和binlog基本类似，只不过不是以二进制格式来记录而是以简单的文件格式记录内容。自从MYSQL增加了binlog功能之后，就很少使用更新日志了。从版本5.0开始，MYSQL已经不再支持更新日志了

**查询日志(query log): 启动参数- --log[=file\_name]**查询日志记录MYSQL中所有的query，由于记录了所有的query，包括所有的select，体积比较大，开启后对性能也有较大的影响，所有建议谨慎使用该功能。一般只用于跟踪某些特殊的sql性能问题才会短暂打开功能。默认的查询日志文件名为hostname.log

**慢查询日志(slow query log): 启动参数--log-slow-queries[=file\_name]**慢查询日志中记录的是执行时间较长的query，也急救室我们常说的slow query，默认文件名为hostname-slow.log，默认目录也是数据目录

**Innodb的在线redo日志(innodb redo log):**Innodb是一个事务安全的存储引擎，其事务安全性就是通过在线redo日志和记录在表空间中的undo信息来保证的。Redo日志红记录了Innodb所做的所有物理变更和事务信息。通过redo日志和undo信息，Innodb保证了再任何情况下的事务安全性。Innodb的redo日志同样默认存放在数据目录下，可以通过innodb\_log\_group\_home\_dir来更改设置日志的存放位置，通过innodb\_log\_files\_in\_group设置日志的数量

## 数据文件:

在MYSQL中每一个数据库都在定义好(或者默认)的数据目录下存在一个以数据库名字命名的文件加，用来存放该数据库中各种表数据文件。不同的MYSQL存储引擎有各自不同的数据库文件，存放位置也有区别。多数存储引擎的数据文件都存放在和MYISAM数据文件位置相同的目录下，但是没给数据库的扩展名却各不一样如MYISAM用”.MYD”做完扩展名，Innodb用”.ibd”，Archive用”.arc”，CVS用”.csv”等待

**“.frm”文件：**与表相关的元数据(meta)信息都存放在”.frm”文件中，包括表结构的定义信息等。不论是什么存储引擎，每一个表都会有一个以表名命名的”.frm”文件。所有的”.frm”文件都存放在所属数据库的文件夹下面

**“.MYD”文件:** “.MYD”文件是MYISAM存储引擎专用，存放MYISAM表的数据。每一个MYISAM表都会有一个”.MYD”文件与之对应，同样存放所属数据库文件下，和”.frm”文件在一起

**“.MYI”文件:**”.MYI”文件也是属于MYISAM存储引擎的，主要存放MYISAM表的索引相关信息。对应MYISAM存储来说。可以被cache的内容主要就是来源与”.MYI”文件中，每一个MYISAM表对应一个”.MYI”文件，存放位置和”.frm”以及”.MYD”一样

**“.ibd”文件和ibdata文件:**这两种文件都是存放Innodb数据的文件，之所以有两种文件来存放Innodb的数据(包括索引)，是因为Innodb的数据存储方式能够通过配置来决定是使用共享表空间存放数据，还是独立表空间存放数据。独立表空间存储方式使用”.ibd”文件来存放数据，切每一个表一个”.ibd”文件，文件存放和MYISAM数据相同的位置

如果选择共享存储表空间来存放数据，则会使用ibdata文件来存放，所以表共同使用一个(或者多个，可自行配置)ibdata文件。Ibdata文件可以通过innodb\_data\_home\_dir和innodb\_data\_file\_path两个参数共同配置组成，innodb\_data\_home\_dir配置数据存放的总目录，而innodb\_data\_file\_path配置每一个文件的名称。当然可以不配置innodb\_data\_home\_dir而直接在innodb\_data\_file\_path中可以一次配置多个ibdata文件。文件可以是指定大小，也可以是自动扩展的，但是Innodb限制了仅仅只有最后一个ibdata文件能够配置成自动扩展类型。当我们需要添加新的ibdata文件的时候，只能添加在innodb\_data\_file\_path配置的最后，而且必须重启MYSQL才能完成ibdata的添加工作

如果我们使用独表空间存储方式的话，就不会有这样的问题，但是如果使用裸设备的话。每个表一个裸设备。可能造成裸设备数量非常大，而且不同容易控制大学，实现比较困难，而共享表空间却不会有这个问题，容易控制裸设备数量。推荐使用独享表空间存储方式

## Replication相关文件:

**Master.info文件:**master.info文件存在Slave端的数据目录下，里面存放了该Slave的Master端的相关信息，包括Master的主机地址，连接用户，连接密码，连接端口，当前日志位置，已经读取到的日志位置等信息

**Relay log和relay log index:**mysql-relay-bin.xxxxxn文件用于存放Slave端的I/O线程从Master端所读取到的Binary log信息，然后由Slave端的SQL线程从relay log中读取并解析相应的日志信息，转化成Master所执行的SQL语句，然后再Slave端应用

mysql-relay-bin.index文件的功能类似于mysql-bin.index，同样是记录日志的存放位置的绝对路径，只不过他所记录的不算Binary Log，而是Relay Log

**relay-log.info文件:**类似于master.info，它存放通过Slave的I/O线程写入本地的relay log的相关信息。供Slave端的SQL线程以及某些管理操作随时能够获取当前复制的相关信息

# 逻辑模块组成

MYSQL可以看出是二层架构，第一层我们通常叫做SQL Layer，在MYSQL数据库系统处理底层数据之前的所有工作都是在这一层完成的，包括权限判断，SQL解析，执行计划优化，query cache的处理等等;第二层就是存储引擎层，我们通常叫做Storage Engine Layer，也就是底层数据存取操作实现部分，由多种存引擎共同组成。



## SQL Layer中包含多个子模块:

1. 初始化模块

初始化模块就是在MYSQL Server启动的时候，对整个系统做各种各样的初始化操作，比如buffer，cache结构的初始化和内存空间的申请，各种系统变量的初始化设定，各种存储引擎的初始化设置，等等

1. 核心API

核心API模块主要是为了提供一些需求非常高效的底层操作功能的优化实现，包括各种底层数据结构的实现，特殊算法的实现，字符串处理，数字处理等，小文件I/O，格式化输出，已经最重要的内存管理部分。核心API模块的所有源代码都集中在mysys和syrings文件夹下面。

1. 网络交换模块

底层网络交换模块抽象出底层网络交互所使用的接口API，底层实现网络数据的接收与发送，以方便其他各个模块调用，已经对这一部分的维护。

1. Client & Server交互协议模块

任何C/S结构的软件系统，都肯定会有自己独有的信息交换协议，MYSQL也不例外。MYSQL的Client & Server交互协议模块部分，实现了客户端与MYSQL交换过程中的所有协议。当然这些协议都是建立在现有的OS和网络协议之上。如TCP/IP以及Unix Socket

1. 用户模块

用户模块所实现的功能，主要包括用户的登录连接权限控制和用户的授权管理。

1. 访问控制模块

访问控制模块实时监控客人的每一个动作，给不同的客人以不同的权限。访问控制模块实现的功能就是根据用户模块中各用户的授权信息，以及数据库自身特有的各种约束，来控制用户对数据的访问。用户模块和服务控制模块两者结合起来，组成了MYSQL整个数据库系统的权限安全管理的功能

1. 连接管理、连接线程和线程管理

连接管理模块负责监听对MYSQL Server的各种请求，接收连接请求，转发所有连接请求到线程管理模块。每一个连接上MYSQL Server的客户端请求都会被分配(或创建)一个连接线程为其单独服务。而连接线程的主要工作就是负责MYSQL Server与客户端的通信，接收客户端的命令请求，传递Server端的结果信息等。线程管理模块负责管理维护这些连接线程。包括线程的创建，线程的cache等

1. Query解析和转发模块

在MYSQL中我们习惯将所有Client端发给Server端的命令都称为query，在MYSQL Server里面，连接线程接收到客户端的应该Query后，会直接将该query传递给专门负责将各种Query进行分类然后转发给各个对应的处理模块，这个模块就是query解析和转发模块。其主要工作就是讲query语句进行语义和语法的分析，然后按照不同的操作类型进行分类，然后做出针对性的转发

1. Query Cache模块

Query Cache模块在MYSQL中是一个非常重要的模块，他的主要功能是将客户端提交给MYSQL的Select类query请求的返回结果集cache到内存中，与该query的应该hash值做一个对应。该Query所取数据的基表发生任何数据的变化之后，MYSQL会自动使该query的Cache失效。在读写比例非常搞的应用系统中，Query Cache对性能的提高是非常显著的。当然它对内存的消耗也是非常大的

1. Query优化器模块

Query优化器，就是优化客户端请求的query，根据客户端请求的query语句，和数据库中的一些统计信息，在一系列算法的基础上进行分析，得出一个最优的策略，告诉后面的程序如何取得这个query语句的结果

1. 表变更管理模块

表变更管理模块主要是负责完成以下DML和DDL的query，如:update、delete、insert、create table、alter table等语句的处理

1. 表维护模块

表的状态检查、错误修复，已经优化和分析等工作都是表维护模块需要做的事情

1. 系统状态管理模块

系统状态管理模块负责在客户端请求系统的世界，将各种状态数据返回给用户，像DBA常用的各种show status命令，show variables命令等，所得到的结果都是有这个模块返回的

1. 表管理器

每一个MYSQL的表都有一个表的定义文件，也就是\*.frm文件。表管理器的工作主要就是维护这些文件，已经一个cache，该cahce中的主要内容是各个表的结构信息。此外它还在维护table基本的锁管理

1. 日志记录模块

日志记录模块主要负责整个系统级别的逻辑层的日志的记录，包括error log，binary log，slow query log等

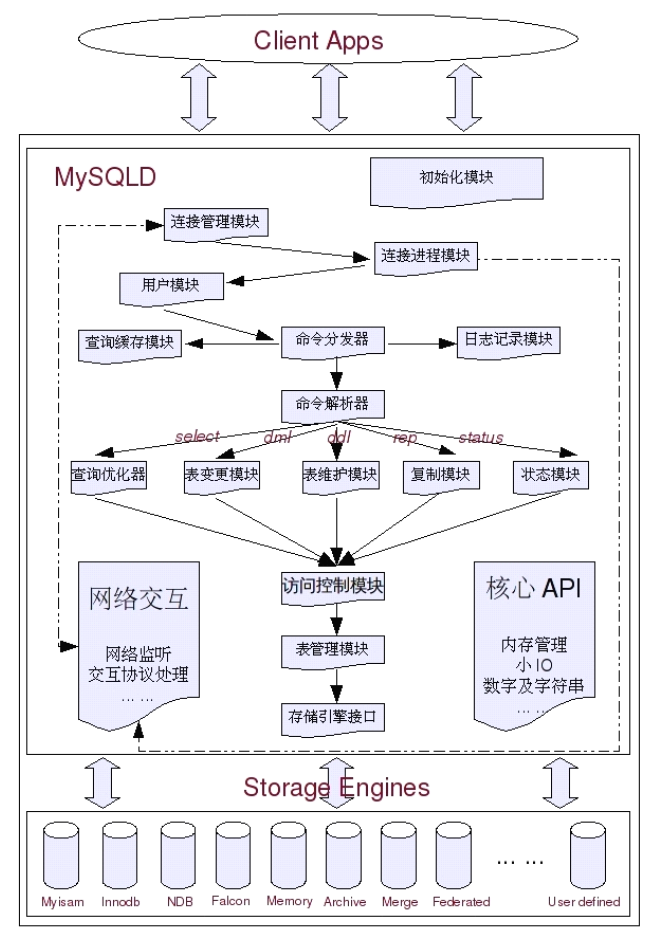
1. 复制模块

复制模块又可分为Master模块和Slave模块两部分，Master模块主要负责在Replication环境中读取Mater端的binary日志，已经与Slave端的I/O线程交互等工作。Slave模块比Master请求和接受binary日志，并写入本地relay log的I/O线程。另外一个负责从relay log中读取相关日志事件，然后解析成可以在Slave端正确执行并得到和Master端完全相同的结果的命令再交给Slave执行的SQL线程

1. 存储引擎接口模块

存储引擎接口模块可以说是MYSQL数据库中最有特色的一点。目前各种数据库产品中，基本上只有MYSQL可以实现其底层数据存储引擎的插件式管理。这个模块实际上只是一个抽象类，但是因为它成功地将各种数据处理高度抽象化，才成就了今天MYSQL可插拔存储引擎的特色





# 存储引擎

## MYISAM存储引擎

MYISAM支持三种类型的索引:

1、B-Tree索引

B-Tree就是所有的索引节点都按照balance tree的数据结构来存储，所有的索引数据节点都在叶节点

2、R-Tree索引

R-Tree索引的存储方式和b-tree索引有一些去吧，主要设计用于为存储空间和多维数据的字段做索引，目前的MYSQL版本，也仅支持geometry类型的字段作索引

3、Full-text索引

Full-text索引就是全文索引，他的存储结构也是b-tree。主要是为了解决在我们需要用like查询的低效问题

MYSQL在遇到如下情况的时候可能会出现表文件损坏

1. 当mysqld正在做写操作的时候被kill掉或者其他情况造成异常终止
2. 主机Crash
3. 硬盘硬件故障
4. MYISAM存储引擎中的Bug

MYISAM存储引擎的某个表出错之后，仅影响到该表，而不会影响到其他表，更不会影响到其他的数据库。如果我们的数据正在进行过程中发现某个MYISAM表出现问题了，则可以在线通过check table命令来尝试效验他，并可以通过repair table命令来尝试修复。在数据库关闭状态先，我们也可以通过myisamchk工具对数据库中某个(或某些)表进行检测或者修复。不过强烈建议不到万不得已不要轻易对表进行修复操作，修复之前尽量做好可能的备份工作，以免带来不必要的后果

另外MYISAM存储引擎的表理论上是可以被多个数据库实例同时使用操作的，但是不论是我们都不建议这样做，建议尽量不要在多个mysqld之间共享MYISAM存储文件

## Innodb存储引擎

**Innodb功能特点：**

1. 支持事务安装
2. 数据多版本读取

Innodb在事务支持的同时，为了保证数据的一致性以及并发时候的性能，通过对undo信息，实现了数据的多版本读取

1. 锁定机制的改进

Innodb改变了MYISAM的锁机制，实现了行锁。虽然Innodb的行锁机制的实现是通过索引来完成的，但毕竟在数据库中99%的SQL语句都是要使用索引来做检索数据的。所以。行锁定机制无疑为Innodb在承受高并发压力的环境下增强了不小的竞争力

1. 实现外键

Innodb实现了外键引用这一数据库的重要特性，使在数据库端控制部分数据的完整性成为可能。虽然很多数据库系统调优专家都不建议这样做，但是对于不少用户来说在数据库端加入外键控制可能仍然是成本最低的选择

**Innodb的物理结构分为两大部分:**

1. 数据文件(表数据和索引数据)

**存放数据表中的数据和所有的索引数据，包括主键和其他普通索引。**在Innodb中，存在了表空间(tablespace)这样一个概念，但是他和Oracle的表空间又有较大的不同。

**虽然我们可以自行设定使用共享表空间还是独享表空间来存放我们的表，但是共享表空间都是必须存在的，因为Innodb的undo信息和其他一些元数据信息都是存放在共享表空间里面的。**共享表空间的数据文件是可以设置为固定大小和可自动扩展大小两种形式的，自动扩展形式的文件可以设置文件的最大大小和每次扩展量。在创建自动扩展的数据文件的时候，**建议大家最好加上最大尺寸的属性。**一个原因是文件系统本身是有一定大小限制的(但是Innodb并不知道)，还有一个原因就是自身维护的方便。另外，Innodb不仅可以使用文件系统，还可以使用原始块设备，也就是我们常说的裸设备

当我们的文件表空间快要用完的时候，我们必须要为其增加数据文件，当然只有共享表空间有次操作。共享表空间增加数据文件的操作比较简单，只需要在innodb\_data\_file\_path参数后安装标准格式设置好文件路径和相关属性即可，不过需要注意的，就是**Innodb在创建新数据文件的时候是不会创建目录的，如果指定目录不存在，则会报错并无法启动。另外一个较为令人头疼的就是Innodb在给共享表空间增加数据文件之后，必须要重启数据库系统才能生效，如果是使用裸设备，还需要有两次重启**

1. 日志文件

Innodb的日志文件和Oracle的redo日志比较类似，同样可以设置多个日志组(最少2个)，同意采用轮询策略来顺序写入，甚至在老版本中还有和Oracle一样的日志归档特性、如果你的数据库中有创建了Innodb的表，那么千万别全部删除innodb的日志文件，因为很可能就好让你的数据库crash，无法启动，或者是丢失数据

由于**Innodb是事务安全的存储引擎**，所以系统Crash对他来说并不能造成非常严重的损失，由于有redo日志的存在，有checkpoint机制的保护，Innodb完全可以通过redo日志将数据库Crash时刻已经完成但还没来得及将数据写入磁盘的事务恢复，也能够将所有部分完成并写入磁盘的未完成事务回滚并将数据还原

**Innodb不仅在功能特性方面和MYISAM存储引擎有较大区别，在配置上面也是单独处理的**、在MYSQL启动参数文件设置中，Innodb的所有参数基本上都带有前缀”innodb\_”,不论是innodb数据和日志相关，还是其他一些性能，事务等待相关的参数都是一样。和所有Innodb相关的系统变量一样，所有的Innodb相关的系统状态值也同样全部以”innodb\_”前缀。当然，我们也完全可以仅仅通过一个参数(skip-innodb)来屏蔽MYSQL中的Innodb存储引擎，这样即使我们再按照编译的时候将Innodb存储引擎安装进去了，使用者也无法创建Innodb的表

## NDB Cluster存储引擎

NDB存储引擎也叫NDB Cluster存储引擎，只要用于MYSQL Cluster分布式集群环境，Cluster是MYSQL从5.0版本才开始提供的新功能

MYSQL Cluster环境由以下三部分组成:

1. **负责管理各个节点的Manage节点主机**

管理节点负责整个Cluster集群中各个节点的管理工作，包括集群的配置，启动关闭各节点，以及实施数据的备份恢复等。管理节点会获取整个Cluster环境中各节点的状态和错误信息，并且将各Cluster集群中各个节点的信息反馈给整个集群中其他的所有节点。由于管理节点上保存在整个Cluster环境的配置，同时担任了集群各节点的基本沟通工作，所以他必须是最先被启动的节点

1. **SQL层的SQL服务节点，MYSQL Server**

主要负责实现一个数据库在存储层之上的所有事情，比如连接管理，query优化和响应，cache管理等等，只有存储层的工作交给了NDB数据节点去处理了。也就是说MYSQL Cluster环境中SQL节点，可以被认为是一个不需要提供任何存储引擎的MYSQK服务器，因为他的存储引擎有Cluster环境中的NDB节点来担任，所有SQL层各MYSQL服务器的启动与普通的MYSQL启动有一定的区别，必须要填ndbcluster项，可以添加再去爱my.cnf配置文件中，也可以通过启动命令来指定

1. **Storage层的NDB数据节点**

NDB是一个内存式存储引擎也是就是说，他会将所有的数据和索引数据都load到内存中，但也会将数据持久化到存储设备上。不过最新版本，已经支持用户自己选择数据可以不全部load到内存中了.

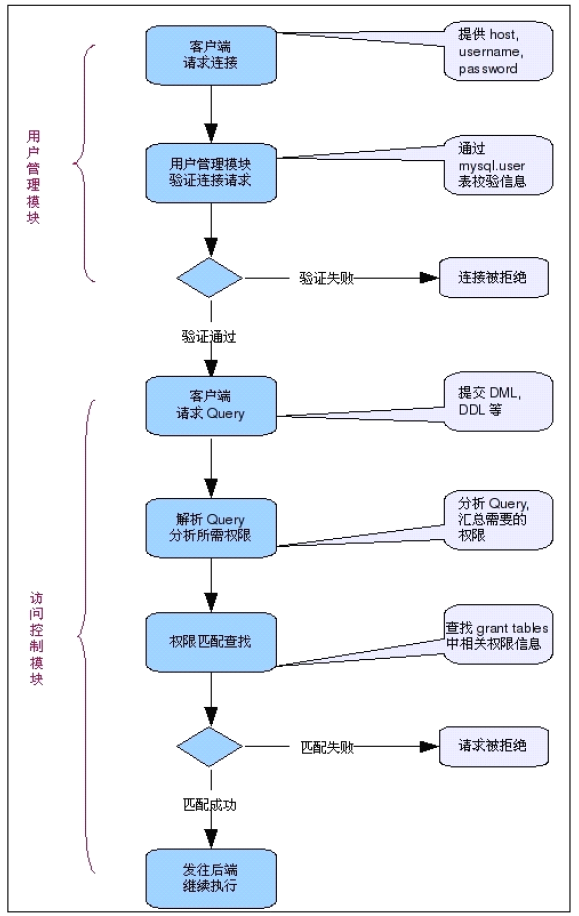
NDB节点主要是实现底层数据存储的功能，保存Cluster的数据，每一个NDB节点保存完整数据的一部分(或者一份完整的数据，视节点数目和配置而定)，在MYSQL Cluster里面叫做一个fragment。而每一个fragment，正常情况来将都会在其他的主机上面有一份(或多分)完全相同多分镜像存在。这些都是通过配置来完成的，所以只要配置得当，MYSQL Cluster在存储层不会出现单点的问题。一般来说，NDB节点被组织成一个一个的NDB Group，一个NDB Group实际上就是一组存有完全相同的物理数据的NDB节点群

# MYSQL安全管理

数据库系统安全相关因素：

1. 外网物理
2. 主机
3. 数据库
4. 代码

MYSQL权限系统



# MYSQL备份与恢复

**逻辑备份：**数据库逻辑备份就是备份软件按照我们最初所设计的逻辑关系，以数据库的逻辑结构对象为单位，将数据库中的数据按照预定义的逻辑关联格式一条一条生成相关的文本文件，以达到备份的目的

**常用的逻辑备份:**逻辑备份可以说是最简单，也是目前小型系统最常使用的备份方式。在MYSQL中我们常用的逻辑备份主要就是两种，一种是将数据生成可以完全重现当前数据库中数据的INSERT语句，另外一种就是讲数据通过逻辑备份软件，将我们数据库表数据以特定分隔符进行分割后记录在文本文件中

**物理备份:**数据库物理备份就是对数据库的物理对象所做的备份。数据库的物理对象主要由数据库的物理数据文件、日志文件以及配置文件等组成、在MYSQL数据库中，除了MYSQL系统共有的一些日志文件和系统表的数据文件之外，每一种存储引擎自己还会有不太一样的物理对象。

# MYSQL性能优化

需求和架构及业务实现优化：55%

Query语句优化：30%

数据库自身的优化：15%

数据库应用系统的优化，实际上是一个需要多方面配合，多方面优化的才能产生根本新改善的事情。简单来说，可以通过**数据库应用系统的性能优化;商业需求合理化，系统架构最优化，逻辑实现精简化，硬件设施理性化**

# MYSQL数据库锁机制

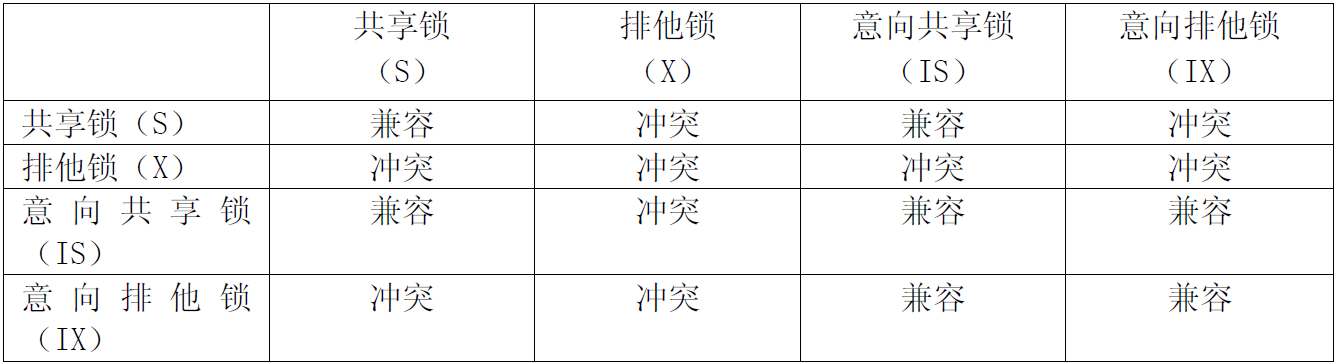
为了保证数据的一致完整性，任何一个数据库都存在锁定机制。锁定机制的优劣直接影响到一个数据库的并发处理能力和性能，所以锁定机制的实现也就成为了各种数据库的核心技术之一

Innodb的行级锁同样分为两种类型。。**共享锁**和**排他锁**，而在锁定机制的实现过程中为了让行级锁定和表级锁定共存，Innodb也同样使用了意向锁(表级锁定)的概念，也就有了**意向共享锁**和**意向排他锁**

当一个事务需要给自己需要的某个资源加锁的时候，如果遇到一个共享锁正锁定着自己需要的资源的时候，自己可以再加一个共享锁，不过不能加排他锁。但是，如果遇到自己需要锁定的资源已经被一个排他锁占有之后，则只能等到该锁定释放资源之后自己才能获取锁定资源并添加自己的锁定

意向锁的作用就是当一个事务在需要获取资源锁定的时候，如果遇到自己需要的资源已经被排他锁占用的时候，该事务可以需要锁定行上面添加一个合适的意向锁。如果自己需要一个共享锁，那么就在表上面添加一个意向共享锁。而如果自己需要的是某行(或者某些行)上面添加一个排他锁的话，则先在表上面添加一个意向排他锁。意向共享锁可以同时并存多个，但是意向排他锁同时只能有一个存在

Innodb锁模式:**共享锁(S)，排他锁(X)，意向共享锁(IS)和意向排他锁(IX)**



Oracle锁定数据是通过需要锁定的某行记录所在的物理block上的事务槽上表级锁定信息，而Innodb的锁定则是通过在指定数据记录的第一个索引键之前和最后一个索引键之后的空域空间上标记锁定信息而实现的。Innodb的这种锁定实现发生被称为”NEXT-KEY locking”(间隙锁)，因为query系过程中通过范围查找的话，他会锁定整个范围内所有的索引键值，即使这个键值并不存储

间隙锁有一个比较致命的弱点，就是当锁定一个范围键值之后，即使某个不存在的键值也会被无辜的锁定，而造成在锁定的时候无法插入锁定键值范围内的任何数据。在某些场景下着看你会对性能造成很大的危害。而Innodb给出的解释是为了阻止幻读的出现，所有选择使用间隙锁来实现锁定

其他性能隐患:

1. 当Query无法利用索引的时候，Innodb会放弃使用行级别锁定而改用表级别的锁定，造成比并发性能的降低
2. 当Query使用的索引并不包含所过滤条件的时候，数据检索使用到的索引键所只想的数据可能有部分并不属于该Query的结果集的行列，但是也会被锁定，因为间隙锁锁定的是一个范围，而不是具体的索引键
3. 当Query在使用索引定位数据的时候，如果使用的索引键一样但访问的数据不同的时候(索引只是过滤条件的一部分)，一样会被锁定

**死锁**

在Innodb的事务管理和锁定机制中，有专门检测死锁的机制，会在系统产生死锁之后的很短时间内就检测到该死锁的存在。当Innodb检测到系统中产生了死锁之后，Innodb会通过相应的判断来选择产生死锁的两个事务中较小的事务来回滚，而让另外一个较大的事务成功完成

**合理利用锁机制优化MYSQL**

**MYISAM表锁优化建议**

1. 缩短锁定时间
   1. 尽量减少大的复杂Query，将复杂Query分拆成几个小的Query分布进行
   2. 尽可能的建立足够高效的索引，让数据检索更迅速
   3. 尽量让MYISAM存储引擎的表只存放必要的信息，控制字段类型
   4. 利用合适的机会优化MYISAM表数据文件
2. 分离能并行的操作
3. 合理利用读写优先级

**Innodb行锁优化建议**

1. 尽可能让所有的数据检索都是通过索引来完成，从而避免Innodb因为无法通过索引加锁而升级为表锁定
2. 合理设计索引，让Innodb在索引键上面加锁的时候尽可能准确，尽可能的缩小锁定范围，避免造成不必要的锁定而影响其他query的执行
3. 尽可能减少基于范围的数据检索过滤条件，避免因为间隙锁带来的负面影响而锁定了不该锁定的记录
4. 尽量控制事务的大小，减少锁定的资源量和锁定时间长度
5. 在业务环节允许的情况下，尽量使用低级别的事务隔离，以减少MYSQL因为事务隔离级别所带来的附加成本

**减少死锁概率的建议**

1. 类似业务模块中，尽可能按照相同的访问顺序来访问，防止产生死锁
2. 在同一事务中，尽可能做到一次锁定所需要的的所有资源，减少死锁产生概率
3. 对非常容易产生死锁的业务部分，可以尝试使用升级锁定粒度，通过表级锁定来减少死锁产生的概率

# Query语句优化

1. 优化更需要优化的Query
2. 定位优化对象的性能瓶颈
3. 明确优化目标
4. 从Explain入手
5. 多使用profile
6. 永远用小结果驱动大的结果集
7. 尽可能在索引中完成排序
8. 只取出自己需要的Columns
9. 仅仅使用最有效的过滤条件
10. 尽可能避免复杂的Join和子查询

# 选择索引优化建议：

1. 对于单键索引，尽量选择针对当前Query过滤性更好的索引
2. 在选择组合索引的时候，当前Query中过滤性最好的字段在索引字段顺序中排列越靠前越好
3. 在选择组合索引的时候，尽量选择可以能够包含当前Query的where字句中更多字段的索引
4. 尽可能通过分析统计信息和调整Query的写法来达到选择合适索引的目的而减少通过使用Hint人为控制索引的选择，因为这会使后期的维护成本增加，同时增加维护所带来的潜在风险

# MYSQL索引限制

1. MYISAM存储引擎索引长度总和不能超过1000字节
2. BLOB和TETX类型的列只能创建前缀索引
3. MYSQL目前不支持函数索引
4. 使用不等于(!=或者<>)的时候MYSQL无法使用索引
5. 过滤字段使用了函数运算后(如abs(column))，MYSQL无法使用索引
6. Join语句中Join条件字段类型不一致的时候MYSQL无法使用索引
7. 使用like操作的时候如果条件以通配符开始(%....)MYSQL无法使用索引
8. 使用非等值查询的时候MYSQL无法使用Hash索引

# MYSQL常用优化

1. **网络连接与连接线程**

**Max\_conecctions:**整个MSYQL允许的最大连接数，主要影响整个MYSQL应用的并发处理能力。一般来说500-800左右是一个比较合适的参考值

**Max\_user\_conecctions:**每个用户允许的最大连接数，对于普通的应用来说，完全没有做太多的限制，可以可能放开一些

**Net\_buffer\_length:**网络包传输中，传输消息之前的net buffer初始化大小。这个参数主要可能影响的是网络传输的效率，系统默认大小为16KB，一般来说可以满足大多数场景，当然如果我们的查询都是非常小，每次网络传输量都很小，而且系统内存又比较紧缺的情况下，可以适当将该值降低到8KB

**Max\_allowed\_packet:**在网络传输中，一次传消息数量的最大值。当我们的消息传输量大于net\_buffer\_length的设置时，MYSQL会自动增大net buffer的大小，直到缓冲区大小达到max\_allowed\_packet所设置的值。系统默认值为1MB，最大值是1GB，必须设定为1024的倍数，单位为字节

**Back\_log:**在MYSQL的连接请求等待队列中允许存放的最大连接请求数。系统默认值为50，最大可以设置为65535。当我们增大back\_log的设置的时候，同时还需要注意OS级别多无聊监听队列的限制。

**Thread\_cache\_size:**Thread Cache池中应该存放的连接线程数。当系统最初启动的时候，并不会马上就创建Thread\_cache\_size所设置数目的连接线程存放在Thread Cache池中，而是所在连接线程的创建及使用，慢慢将用完的连接线程存入其中。当存放的连接线程达到Thread\_cache\_size值之后，MYSQL就不会再继续保存用完的连接线程了。在短连接的应用系统中Thread\_cache\_size的值应该设置的相对大一些，不应该小于应用系统对数据库的实际并发请求数

**Thread\_stack:**每个连接线程被创建的时候，MYSQL给他分配的内存大小。使用系统默认值(192KB)基本上可以所有的普通应用环境。如果该值设置太小，会影响MYSQL连接线程能够处理客户端请求的Query内容的大小，以及用户创建的Procedures和Functions等

1. **Innodb缓存相关优化**

Innodb\_buffer\_pool\_size的合理设置，Innodb存储引擎机制和MYISAM的最大区别就在于Innodb不仅仅缓存索引，同时还会缓存实际的数据。所以，完全相同的数据库，使用Innodb存储引擎可以使用更多的内存来缓存数据库相关的信息，当然前提是要有足够的物理内存

