**Mysql数据库规划与操作规范**

|  |  |
| --- | --- |
| 作者： |  |
| 创建时间： |  |
| 最新更新时间： |  |
| 版本： | 1.0 |

# 文档控制

## 修改记录

| **日期** | **作者** | **版本** | **修改参考** |
| --- | --- | --- | --- |
|  |  |  |  |
| 2018-01-03 |  | 2．0 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 分发

| **编号** | **名称** | **地点** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 文档格式说明

本文档采用以下格式：

* 命令、代码等指令型内容：

execute command 1

* 文本显示内容

For example

..........

# 目录

[文档控制 1](#_Toc17717)

[3.1 修改记录 1](#_Toc31555)

[3.2 分发 1](#_Toc5539)

[3.3 文档格式说明 1](#_Toc5850)

[4 目录 2](#_Toc20580)

[第一章 Mysql数据库相关概念 4](#_Toc8618)

[1.1 Mysql相关介绍 4](#_Toc21466)

[1.2 特性 4](#_Toc11310)

[1.3 存储引擎 4](#_Toc29618)

[1.4 表空间 5](#_Toc5146)

[1.5 锁 6](#_Toc21692)

[1.5.1 表级锁定 7](#_Toc22838)

[1.5.2 行级锁锁定 7](#_Toc13526)

[1.5.3 死锁 8](#_Toc29248)

[1.6 数据库权限 8](#_Toc26318)

[1.7 日志 10](#_Toc27054)

[1.8 索引 10](#_Toc19106)

[第二章 Mysql数据库设计规范 11](#_Toc2424)

[2.1 Mysql版本选择 11](#_Toc3219)

[2.2 数据库部署方案选择 11](#_Toc22109)

[2.2.1 方案说明与适用场景 11](#_Toc2521)

[2.2.2 方案概要图 12](#_Toc24136)

[2.3 Mysql对操作系统环境要求 17](#_Toc28674)

[2.3.1 Mysql for Linux系统环境要求 17](#_Toc15024)

[2.4 数据库锁设计 18](#_Toc5186)

[2.4.1 表级锁设计 18](#_Toc28461)

[2.4.2 行级锁设计 18](#_Toc10807)

[2.5 表空间设计 19](#_Toc19874)

[2.5.1 共享表空间的设计 19](#_Toc32250)

[2.5.2 独立表空间的设计 19](#_Toc5275)

[2.6 数据库权限控制 20](#_Toc13161)

[第三章 Mysql数据库安装部署操作规范 21](#_Toc25054)

[3.1 部署准备 21](#_Toc23822)

[3.1.1 Linux操作系统 21](#_Toc8767)

[3.1.2 数据库磁盘和文件系统要求 21](#_Toc3868)

[3.1.3 字符集选择 21](#_Toc12584)

[3.2 安装Mysql软件 22](#_Toc30537)

[3.2.1 Mysql相关文件及目录 23](#_Toc11655)

[3.2.2 启动数据库及简单配置 23](#_Toc2200)

[3.3 Mysql主从复制安装部署 24](#_Toc30457)

[3.4 Corosync+pacemaker+mysql+共享存储 24](#_Toc587)

[第四章 数据库运维操作规范 24](#_Toc7524)

[4.1 服务端管理与配置 24](#_Toc5017)

[4.1.1 配置管理 24](#_Toc7785)

[4.2 数据库管理 25](#_Toc10715)

[4.2.2 数据表（table）管理 26](#_Toc26447)

[4.2.3 数据的操作及管理 30](#_Toc9544)

[4.3 管理索引视图 33](#_Toc1874)

[4.3.1 创建索引 33](#_Toc20607)

[4.3.2 CREATE在已存在的表上创建索引 34](#_Toc7562)

[4.3.3 ALTER TABLE在已存在的表上创建索引 34](#_Toc6506)

[4.3.4 查看索引 35](#_Toc9907)

[4.3.5 管理视图 35](#_Toc15945)

[4.4 表空间管理 37](#_Toc12968)

[4.4.1 共享表空间 37](#_Toc30680)

[4.4.2 独立表空间 39](#_Toc1618)

[4.5 日志管理 1](#_Toc5711)

[4.5.1 错误日志： 1](#_Toc11823)

[4.5.2 通用查询日志: 2](#_Toc26534)

[4.5.3 慢查询日志： 2](#_Toc2095)

[4.5.4 二进制日志： 3](#_Toc3901)

[4.5.5 MySQL中日志相关常用的服务器变量说明如表十四所示： 8](#_Toc4552)

[4.6 安全相关管理与配置 11](#_Toc3571)

[4.6.1 帐号与权限管理 11](#_Toc27960)

[4.7 备份与恢复 15](#_Toc30345)

[4.7.1 mysqldump+binlog备份与恢复 15](#_Toc15851)

[4.7.2 数据文件+binlog备份与恢复 16](#_Toc29623)

[4.7.3 数据导入导出 17](#_Toc19629)

[4.8 锁管理 18](#_Toc14379)

[4.8.1 表锁 18](#_Toc28485)

[4.8.2 行锁 19](#_Toc29599)

[4.8.3 死锁 20](#_Toc15551)

[4.9 性能优化 20](#_Toc21893)

[4.9.1 性能优化 20](#_Toc2736)

[第五章 Mysql数据库开发规范 24](#_Toc6377)

[5.1实例设计 24](#_Toc8326)

[5.2表设计 24](#_Toc134)

[5.3索引设计 25](#_Toc4842)

[5.4数据库连接 25](#_Toc16319)

[5.5SQL开发规范 25](#_Toc23470)

[5.6事务使用规范 25](#_Toc30026)

[5.7其他 26](#_Toc8749)

[第六章 常见问题汇总 26](#_Toc10378)

[6.1 mysql DNS反解 26](#_Toc16339)

[6.2错误日志 27](#_Toc18059)

[6.3操作报错 27](#_Toc10650)

[6.4导出数据很快，导入到新库时却很慢 28](#_Toc17395)

# Mysql数据库相关概念

## Mysql相关介绍

MySQL是一个开放源码的小型关联式[数据库管理系统](https://baike.so.com/doc/2035924.html" \t "https://baike.so.com/doc/_blank)，开发者为瑞典MySQL AB公司。目前MySQL被广泛地应用在Internet上的中小型网站中。由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择了MySQL作为网站数据库。

## 特性

与其他的大型数据库例如Oracle、DB2、SQL Server等相比，MySQL自有它的不足之处，如规模小、功能有限(MySQL Cluster的功能和效率都相对比较差)等，但是MySQL数据库具有如下特性；对于一般的个人使用者和中小型企业来说，MySQL提供的功能已经绰绰有余，而且由于MySQL是开放源码软件，因此可以大大降低总体拥有成本。

* 使用C和C++编写，并使用了多种[编译器](https://baike.so.com/doc/6133192.html" \t "https://baike.so.com/doc/_blank)进行测试，保证[源代码](https://baike.so.com/doc/5395725.html" \t "https://baike.so.com/doc/_blank)的可移植性
* 支持[AIX](https://baike.so.com/doc/3676093.html" \t "https://baike.so.com/doc/_blank)、[FreeBSD](https://baike.so.com/doc/1845399.html" \t "https://baike.so.com/doc/_blank)、HP-UX、Linux、Mac OS、NovellNetware、OpenBSD、OS/2 Wrap、Solaris、[Windows](https://baike.so.com/doc/5211242.html" \t "https://baike.so.com/doc/_blank)等多种操作系统
* 为多种[编程语言](https://baike.so.com/doc/538266.html" \t "https://baike.so.com/doc/_blank)提供了API。这些编程语言包括C、C++、[Python](https://baike.so.com/doc/1790119.html" \t "https://baike.so.com/doc/_blank)、[Java](https://baike.so.com/doc/2886868.html" \t "https://baike.so.com/doc/_blank)、[Perl](https://baike.so.com/doc/5106225.html" \t "https://baike.so.com/doc/_blank)、PHP、Eiffel、[Ruby](https://baike.so.com/doc/4990463.html" \t "https://baike.so.com/doc/_blank)和Tcl等。
* 支持[多线程](https://baike.so.com/doc/1712669.html" \t "https://baike.so.com/doc/_blank)，充分利用CPU[资源](https://baike.so.com/doc/3867650.html" \t "https://baike.so.com/doc/_blank)
* 优化的SQL查询算法，有效地提高查询速度
* 既能够作为一个单独的[应用程序](https://baike.so.com/doc/3417785.html" \t "https://baike.so.com/doc/_blank)应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中。
* 提供多语言支持，常见的编码如中文的GB 2312、BIG5，日文的Shift\_JIS等都可以用作数据表名和数据列名。
* 提供TCP/IP、ODBC和[JDBC](https://baike.so.com/doc/2417290.html" \t "https://baike.so.com/doc/_blank)等多种数据库连接途径。
* 提供用于管理、检查、优化数据库操作的管理工具。
* 支持大型的数据库。可以处理拥有上千万条记录的大型数据库。
* 支持多种存储引擎。

## 存储引擎

在mysql中配置有许多不同的存储引擎，用户可以灵活的选择适用于服务器、数据库和表格的存储引擎，以便获得最大的性能和最大的灵活性。这也是 mysql不同于其他大型数据库以及mysql为何如此受欢迎的主要因素。一些大型的数据库仅采用了一种存储引擎，以一种尺码满足所有需求也就意味着会牺牲一些性能。

表一说明了mysql常用引擎之间的差异对比，对根据不同场景选择不同的存储引擎具有参考作用

|  |  |  |  |
| --- | --- | --- | --- |
| 特点 | MyiSAM | InnoDB | MEMORY |
| 存储限制 | 有 | 64TB | 有 |
| 事物安全 | 无 | 支持 |  |
| 锁机制 | 表锁 | 行锁 | 表锁 |
| B树索引 | 支持 | 支持 | 支持 |
| 哈希索引 |  |  | 支持 |
| 全文索引 | 支持 |  |  |
| 集群索引 |  | 支持 |  |
| 数据缓存 |  | 支持 | 支持 |
| 索引缓存 | 支持 | 支持 | 支持 |
| 数据可压缩 | 支持 |  |  |
| 空间使用 | 低 | 高 | N/A |
| 内存使用 | 低 | 高 | 中等 |
| 批量插入的速度 | 高 | 低 | 高 |
| 支持外键 |  | 支持 |  |
| 使用建议 | 由于该存储引擎不支持事务、也不支持外键，所以访问速度较快。 | 由于该存储引擎在事务上具有优势，即支持具有提交、回滚及崩溃恢复能力等事务特性，所以比MyISAM存储引擎占用更多的磁盘空间。 因此当需要频繁的更新、删除操作，同时还对事务的完整性要求较高，需要实现并发控制，建议选择 | 存储数据的位置是内存（缓存空间里），因此访问速度最快，但是安全上没有保障。适合于需要快速的访问或临时表 |
| 使用场景 | 尽量少使用myisam，myisam对cpu，内存利用率不高，并发支持不好,表级别锁，发生锁冲突的概率较高,相对并发度也低。当对事务完整性没有要求并以访问为主的应用适合使用该存储引擎 | 在绝大对数情况场景中建议使用该引擎，尤其是OLTP,在mysql 5.5及以后的版本中，默认的存储引擎是InnoDB  行级别锁，发生锁冲突的概率较低，相对并发也高。 | 适用于数据安全要求不高的环境中，如：临时计数等 |

表格一：Mysql数据库引擎相关差异

## 表空间

Innodb这种引擎，与MYISAM引擎的区别很大。特别是它的数据存储格式等；对于Innodb的数据结构，首先要解决两个概念性的问题: 共享表空间以及独占表空间，下面将列举一个表来描述表空间。

共享表空间以及独占表空间都是针对数据的存储方式而言的，具体说明请参考表二

|  |  |  |  |
| --- | --- | --- | --- |
| 表空间 | 概念 | 优点 | 缺点 |
| 共享表空间 | 某一个数据库的所有的表数据，索引文件全部放在一个文件中，默认这个共享表空间的文件路径在data目录下。 默认的文件名为:ibdata1初始化为10M。 | 可以放表空间分成多个文件存放到各个磁盘上（表空间文件大小不受表大小的限制，如一个表可以分布在不同步的文件上）。数据和文件放在一起方便管理。 | 所有的数据和索引存放到一个文件中意味着将有一个很常大的文件，虽然可以把一个大文件分成多个小文件，但是多个表及索引在表空间中混合存储，这样对于一个表做了大量删除操作后表空间中将会有大量的空隙，特别是对于统计分析，日值系统这类应用最不适合用共享表空间。 |
| 独立表空间 | 每一个表都将会生成以独立的文件方式来进行存储，每一个表都有一个.frm表描述文件，还有一个.ibd文件。 其中这个文件包括了单独一个表的数据内容以及索引内容，默认情况下它的存储位置也是在表的位置之中 | 每个表都有自己独立的表空间；每个表的数据和索引都会存储在各个独立的表空间中；可以实现 单表 在不同的数据进行迁移；表空间可以回收（除了drop table操作，表空不能自己回收）；drop table 操作自动回收表空间，如果对统计分析或是日值表，删除大量数据后可以通过 ：alter table tablename engin=innodb进行回缩不用的空间；对于使用inodb-plugin的innodb使用truncate table会使用空间收缩。；对于使用独立表空间，不管怎么删除 ，表空间的碎片都不会太严重。 | 单表增加过大，如超过100G。对于单表增长过大的问题，如果使用共享表空间可以把文件分开，但有同样有一个问题，如果访问的范围过大同样会访问多个文件，一样会比较慢。对于独立表空间也有一个解决办法是：使用分区表，也可以把那个大的表空间移动到别的空间上然后做一个连接。其实从性能上出发，当一个表超过100个G有可能响应也是较慢了，对于独立表空间还容易发现问题早做处理。 |

表格二：MySQL表空间参考表

## 锁

锁是计算机协调多个进程或线程并发访问某一资源的机制。在数据库中，除传统的计算资源（如CPU、RAM、I/O等）的争用以外，数据也是一种供许多用户共享的资源。如何保证数据并发访问的一致性、有效性是所有数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性能的一个重要因素。从这个角度来说，锁对数据库而言显得尤其重要，也更加复杂。

总的来说，MySQL各存储引擎使用了三种类型（级别）的锁定机制：行级锁定，页级锁定和表级锁定，具体可参考表三说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 锁类型 | 特点 | 适用场景 | 对应存储引擎 |
| 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。 | 表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用，如Web应用 | MyISAM，MEMORY，CSV等一些非事务性存储引擎。 |
| 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 | 行级锁则更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统 | 使用行级锁定的主要是InnoDB存储引擎。 |
| 页面锁 | 开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。 | .... | 使用页级锁定的主要是BerkeleyDB存储引擎。 |

表格三：MySQL中的锁相关特性

### 表级锁定

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）。锁模式的兼容性如表四所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 请求锁模式           是否兼容  当前锁模式 | None | 读锁 | 写锁 |
| 读锁 | 是 | 是 | 否 |
| 写锁 | 是 | 否 | 否 |

表格四：MySQL中的表锁兼容性

可见，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！根据如表20-2所示的例子可以知道，当一个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。

### 行级锁锁定

InnoDB的锁定机制和Oracle数据库有不少相似之处。InnoDB的行级锁定同样分为两种类型，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这两种。

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，则只能等待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。而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的表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。所以，可以说InnoDB的锁定模式实际上可以分为四种：共享锁（S），排他锁（X），意向共享锁（IS）和意向排他锁（IX），我们可以通过表五来总结上面这四种所的共存逻辑关系：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 请求锁模式     是否兼容  当前锁模式 | X | IX | S | IS |
| X | 冲突 | 冲突 | 冲突 | 冲突 |
| IX | 冲突 | 兼容 | 冲突 | 兼容 |
| S | 冲突 | 冲突 | 兼容 | 兼容 |
| IS | 冲突 | 兼容 | 兼容 | 兼容 |

表格五：MySQL中的行锁模式兼容性

如果一个事务请求的锁模式与当前的锁兼容，InnoDB就将请求的锁授予该事务；反之，如果两者不兼容，该事务就要等待锁释放。  
 意向锁是InnoDB自动加的，不需用户干预。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁；事务可以通过以下语句显示给记录集加共享锁或排他锁。

### 死锁

MyISAM表锁是deadlock free的，这是因为MyISAM总是一次获得所需的全部锁，要么全部满足，要么等待，因此不会出现死锁。在InnoDB中，除单个SQL组成的事务外，锁是逐步获得的，这就决定了在InnoDB中发生死锁是可能的。

两个事务都需要获得对方持有的排他锁才能继续完成事务，这种循环锁等待就是典型的死锁；发生死锁后，InnoDB一般都能自动检测到，并使一个事务释放锁并回退，另一个事务获得锁，继续完成事务。但在涉及外部锁，或涉及表锁的情况下，InnoDB并不能完全自动检测到死锁，这需要通过设置锁等待超时参数 innodb\_lock\_wait\_timeout来解决。需要说明的是，这个参数并不是只用来解决死锁问题，在并发访问比较高的情况下，如果大量事务因无法立即获得所需的锁而挂起，会占用大量计算机资源，造成严重性能问题，甚至拖跨数据库。我们通过设置合适的锁等待超时阈值，可以避免这种情况发生。

## 数据库权限

MySQL的权限体系在实现上比较简单，相关权限信息主要存储在mysql.User、mysql.db、mysql.Host、mysql\_table\_priv和mysql.column\_priv几个表中。由于权限信息数据量比较小，而且访问又比较频繁，所以MySQL在启动时就会将所有的权限信息都Load到内存中保存在几个特定的结构中，所以才有了我们手动修改了权限相关的表后，都需要通过执行"FLUSH PRIVILEGES" 命令重新加载MySQL的权限信息。我们也可以通过GRANT,REVOKE或者DROP USER命令所做的修改权限后也会同时更新到内存结构中的权限信息。

表六描述了mysql数据库相关的权限列表：

|  |  |  |
| --- | --- | --- |
| 权限 | 级别 | 说明 |
| [ALL [PRIVILEGES]](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_all) | Server administration | 某级别的所有权限 |
| [ALTER](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_alter) | Tables | 更改表，比如添加字段、索引等 |
| [ALTER ROUTINE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_alter-routine) | Stored routines | 更改存储过程权限 |
| [CREATE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create) | Databases, tables, or indexes | 创建数据库、表或索引权限 |
| [CREATE ROUTINE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create-routine) | Stored routines | 创建存储过程权限 |
| [CREATE TABLESPACE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create-tablespace) | Server administration | 创建表空间 |
| [CREATE TEMPORARY TABLES](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create-temporary-tables) | Tables | 创建临时表 |
| [CREATE USER](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create-user) | Server administration | 创建用户 |
| [CREATE VIEW](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_create-view) | Views | 创建视图 |
| [DELETE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_delete) | Tables | 删除表中数据 |
| [DROP](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_drop) | Databases, tables, or views | 删除数据库、表或视图 |
| [EVENT](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_event) | Databases | 事件与定时任务管理权限 |
| [EXECUTE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_execute) | Stored routines | 执行存储过程 |
| [FILE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_file) | File access on server host | 操作系统文件读写权限 |
| [GRANT OPTION](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_grant-option) | Databases, tables, or stored routines | 将本帐号拥有的权限grant或revoke |
| [INDEX](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_index) | Tables | 索引操作权限 |
| [INSERT](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_insert) | Tables or columns | 插入数据 |
| [LOCK TABLES](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_lock-tables) | Databases | 锁表 |
| [PROCESS](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_process) | Server administration | 进程管理权限 |
| [RELOAD](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_reload) | Server administration | 执行flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, reload等命令的权限 |
| [SELECT](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_select) | Tables or columns | 查询权限 |
| [SHOW DATABASES](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_show-databases) | Server administration |  |
| [SHOW VIEW](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_show-view) | Views |  |
| [SHUTDOWN](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_shutdown) | Server administration | 关闭MySQL数据库 |
| [SUPER](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_super) | Server administration | 管理权限，包括其他部分权限 |
| [TRIGGER](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_trigger) | Tables | 触发器权限 |
| [UPDATE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_update) | Tables or columns | 更新权限 |
| [USAGE](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\security.html" \l "priv_usage) | Server administration | 相当于无任何权限，仅供部分特殊命令使用 |

表格六：MySQL数据库操作权限列表

## 日志

Mysql日志的作用是当数据库遭到意外损害，可以通过日志文件来查询出错原因，利用日志文件进行数据恢复。

可参考表七对mysql日志类型及功能：

|  |  |
| --- | --- |
| 日志文件 | 记入文件中的信息类型 |
| 错误日志 | 记录 MySQL 服务器启动、关闭及运行错误等信息 |
| 查询日志 | 记录建立的客户端连接和执行的语句。 |
| 二进制日志 | 又称binlog日志，以二进制文件的方式记录数据库中除 SELECT 以外的操作，主要用于复制和即时点恢复。 |
| 慢日志 | 记录所有执行时间超过long\_query\_time秒的所有查询或不使用索引的查询。 |
| 事务日志 | 或称redo日志，记录Innodb事务相关的如事务执行时间、检查点等  产生的日志。 |

表格七：MySQL日志类型及功能简介

## 索引

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

相关具体说明可参考表八：

|  |  |
| --- | --- |
| 索引名称 | 概念 |
| 普通索引[INDEX] | 最基本的索引，它没有任何限制，MyIASM中默认的BTREE类型的索引，也是我们大多数情况下用到的索引。 |
| 唯一索引[UNIQUE INDEX] | 与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似。 |
| 全文索引[FULLTEXT INDEX] | FULLTEXT索引仅可用于 MyISAM 表,可增快查询速度，不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。 |
| 单列索引、多列索引 | 多个单列索引与单个多列索引的查询效果不同，因为执行查询时，MySQL只能使用一个索引，会从多个索引中选择一个限制最为严格的索引。 |
| 组合索引（最左前缀） | 平时用的SQL查询语句一般都有比较多的限制条件，所以为了进一步榨取MySQL的效率，就要考虑建立组合索引。 |

表格八：MySQL索引类型

# Mysql数据库设计规范

## Mysql版本选择

当前（2017年6月）推荐使用的MySQL版本为5.6及5.7版本，其最新稳定版本为5.7（2015年发布），5.5及以下的版本建议不再使用在新建系统上,鉴于5.6版本与5.7版本在稳定性上差别不大，且5.7版本在性能提升及新特性方面的优势，推荐使用5.7版本。

* 安全性提升：默认root用户随机密码、默认采用SSL加密连接，用户密码过期控制及锁定解锁等用户相关安全管理操作的增加。
* 新增sys schema，其中包含一系列性能相关的视图，为DBA和开发者更为方便地提供了诊断及优化相关信息。
* 新增对JSON的支持，便于某些应用存储JSON类的信息；
* 优化器改善，包括可对执行中的SQL做explain以及更为强大的hint功能。
* 主备复制功能及性能改善。

## 数据库部署方案选择

针对mysql架构方案，目前有三种主流方案：一、mysql针对读写分离主备复制方案；二、corosync+pacemaker+mysql+共享存储，RHCL7推荐mysql高可用方案；三、mysql-cluster mysql官方集群方案。

### 方案说明与适用场景

方案选型可参考表九：

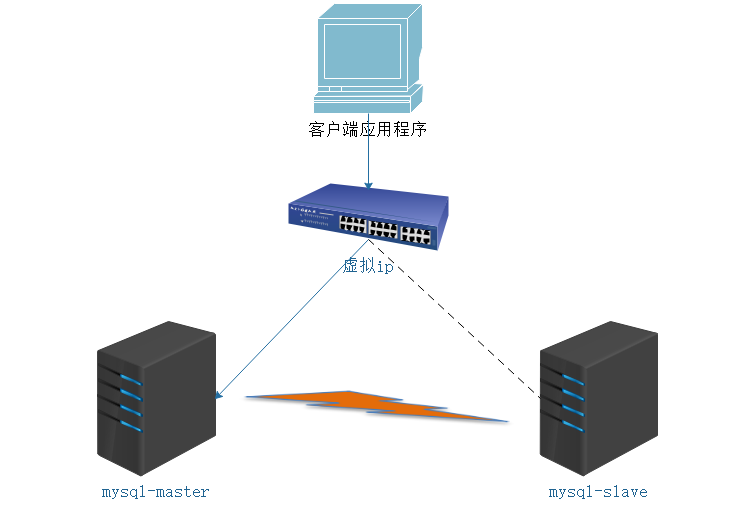
| 方案 | 方案说明 | 适用场景 |
| --- | --- | --- |
| Mysql主备复制 | 数据库运行于两台主机，分为一个主，一个备。应用连接至主服务器进行读写操作。备服务器仅可读，因此也可将只读应用连至备服务器，分担主服务器压力。发生主服务器宕机，网络故障时，可以整合keepalived高可用，由keepalived负责将虚拟ip切换至新主服务器即之前的备服务器，应用可继续访问。 | Mysql读写分离使用该方案 |
| Corosync+pacemaker+mysql+共享存储 | 数据库日常运行于一台主机，但实例目录与数据库目录均位于一块数据盘，且均同时挂载于主备机。当corosync监测主mysql服务器设备故障Pacemaker负责故障转移，将mysql切换至备机运行，此过程仅涉及网络虚拟IP切换及共享数据资源切换，不涉及其他共享资源切换。 | 红帽官方推荐方案，此方案代替了RHCL6的RHCS高可用方案。也是目前mysql高可用主流方案。适用于对数据一致性要求较高的mysql集群。 |
| Mysql-cluster | 集群由5个节点组成，两个sql节点：用来访问 Cluster数据的节点。也就是Mysql服务；两个数据节点：用于保存 Cluster的数据。数据节点的数目与副本的数目相关，是片段的倍数。；管理服务器：负责管理MySQL Cluster内的其他节点，如提供配置数据、启动并停止节点、运行备份等 | 该方案不支持外键，运维复杂，数据备份恢复难度大，网络文件系统和SAN设备是不推荐或不支持的，适用于对数据要求可用度高且对外键没要求的应用场景。 |

表格九：不同场景不同方案描述说明

### 方案概要图

1. Mysql主备

|  |  |
| --- | --- |
| 资源描述 | 说明 |
| 服务器 | 2台/一主多备需多台 |
| 虚拟ip | 1个 |
| 是否需要共享存储 | 否 |



Mysql5.7 Gtid+主备半同步复制提升了半复制性能，对于mysql主备+keepalived高可用方案，使用mysql5.7异步复制为基础，搭建基于Gtid+主备半同步复制集群，Master等待至少一个Slave应答就可以提交，降低主备数据不一致的风险，需要在低延时的网络中使用，当半同步复制发生超时会暂时关闭半同步复制，转而使用异步复制。当master dump线程发送完一个事务的所有事件之后，如果在设定时间内，收到了从库的响应，则主从又重新恢复为半同步复制，保证了主备复制的不间断；通过keepalived监控脚本可以防止极小机率由于网络故障引起的脑裂；该方案极小几率会存在个别数据无法复制的情况。

工作原理：

GTID事物是全局唯一性的，且一个事务对应一个GTID，在master更新数据时，会在事务前产生GTID，一同记录到binlog日志中，slave端的i/o 线程将变更的binlog，写入到本地的relay log中，然后Slave端的sql线程从relay log中获取GTID，然后对比slave端的binlog是否有记录，如果有记录，说明该GTID的事务已经执行，slave会忽略，如果没有记录，slave就会从relay log中执行该GTID的事务，并记录到binlog。

**说明：**

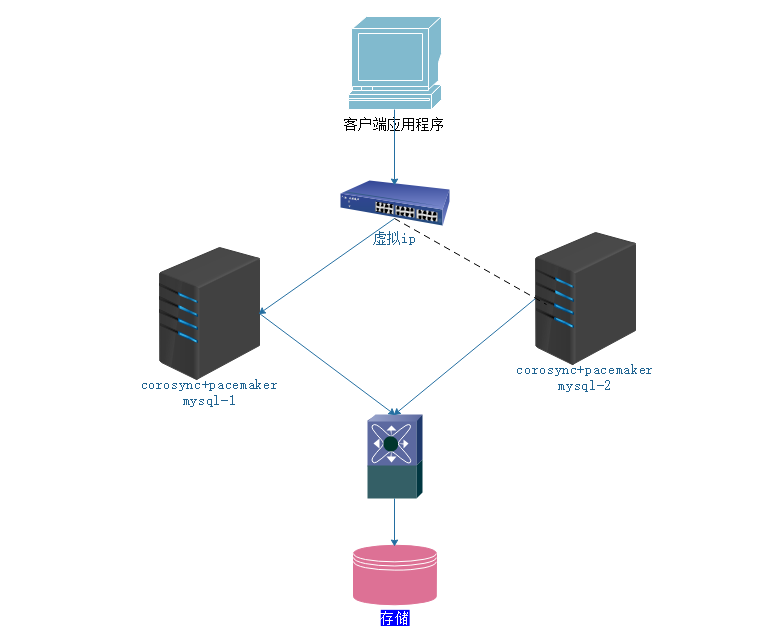
* 通过MySQL自身的日志复制功能进行数据备份。
* 节点数量并不固定，可以根据需要进行配置，可实现较为灵活的主从架构如一主多从、多主、多级复制、延时复制等。
* 需要使用外部方法实现master状态检测及主从切换。
* 进行主从切换时，需要使用外部方法实现客户端对新master的重定向，如虚拟IP或应用中间件重定向等。

**特点：**

|  |  |
| --- | --- |
| 打分项 | 满足程度 |
| 复杂性 | 中 |
| 可扩展性 | 中 |
| 支持读写分离 | 是 |
| 数据丢失场景 | 日志同步级别配置较低时master发生崩溃 |
| 数据完全不可恢复场景 | 所有节点数据均不能提供访问 |
| 数据备份实时性 | 高 |
| 是否支持HA | 是 |
| 引入额外风险概率 | 低 |
| 在线数据不一致风险(不包括故障恢复) | 低 |
| 性能 | 中 |
| 适合场景 | 读写分离主从复制 |

1. Corosync+pacemaker+mysql+共享存储

|  |  |
| --- | --- |
| 资源描述 | 说明 |
| 服务器 | 2台 |
| 虚拟ip | 1个 |
| 是否需要共享存储 | 是 |
| 是否需要fence设备 | 是 |



Pacemaker+Corosync是RHCL7红帽官方推荐高可用方案，也是目前流行的高可用集群的套件， 在整个Mysql5.7高可用集群中，corosync实现了两台mysql服务器之间的心跳信息传输功能，也称心跳监测，整个运行模式就是corosync不断在网络中监测各个指定的设备是否正常响应，Pacemaker是这个集群的管理工具，用于资源转移，如果一旦由corosync监测到设备（资源）发生故障，就由资源转移功能进行应用的切换，以继续提供服务；当两个节点出现分歧时，由第3方的仲裁者决定听谁的。这个仲裁者，可能是一个锁服务，一个共享盘或者其它, 当不能确定某个节点的状态时，通过fencing把对方干掉，确保共享资源被完全释放，前提是必须要有可靠的fence设备，该策略可以防止极小机率引起的脑裂。

工作原理：

两边同时部署数据库服务，数据库的datadir或basedir与datadir存放于共享存储内，当虚拟ip位于哪台主机，那么哪台主机提供应用服务，corosync心跳监测资源设备是否正常提供服务，当发生资源故障，那么由pacemaker进行应用切换，将mysql切换至备机运行，此过程仅涉及网络虚拟IP切换与共享数据资源切换，不涉及其他共享资源切换。

**说明：**

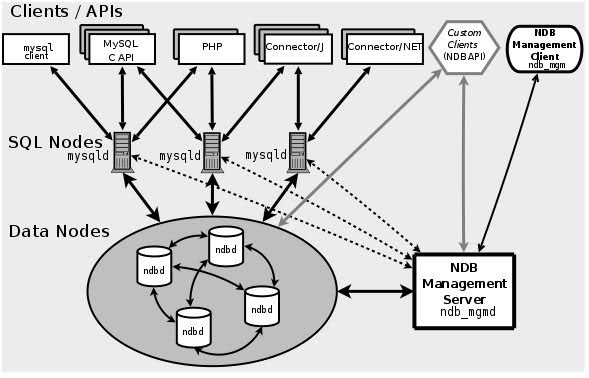
* Corosync是心跳检测，判断Mysql服务器是否正常运行；
* Pacemaker是一个集群管理器，用于mysql资源转移。
* 整个运行模式就是corosync不断的在网络中检测各个指定的设备是否能够正常响应，如果一旦发生设备故障，就由资源转移功能进行应用的切换，以继续提供服务。

**特点：**

|  |  |
| --- | --- |
| 打分项 | 满足程度 |
| 复杂性 | 高 |
| 可扩展性 | 中 |
| 支持扩展负载均衡 | 否 |
| 脑裂几率 | 几乎没有 |
| 在线数据不一致风险(不包括故障恢复) | 无 |
| 是否支持HA | 是 |
| 数据丢失场景 | 极低 |
| 操作难度 | 中 |
| 性能 | 高 |
| 适合场景 | 适用于对数据一致性要求较高的mysql集群。 |

1. Mysql-cluster

|  |  |
| --- | --- |
| 资源描述 | 说明 |
| 服务器 | 一共5台（可扩展） |
| SQL节点 | 2个 |
| Data节点 | 2个 |
| Management管理节点 | 1个 |
| 是否需要共享存储 | 否 |
| 是否需要虚拟ip | 否 |



Mysql-Cluster是Mysql官方集群方案，是一种分布式设计，目标是要达到没有任何单点故障点。因此，任何组成部分都应该拥有自己的内存和磁盘。具备负载均衡高可用，高吞吐量和低延迟,可扩展性强，支持在线扩容，节点组内主从同步采用的是同步复制，来保证组内节点数据的强一致性，通过这种冗余设计，数据的可用度可以达到99.999%

**说明：**

* Management Server管理节点，用来实现整个集群的管理，理论上一般只启动一个，而且宕机也不影响 cluster 的服务，这个进程只在cluster 启动以及节点加入集群时起作用， 所以这个节点不是很需要冗余，理论上通过一台服务器提供服务就可以了。
* Data Nodes数据库节点，用来存储数据，可以和管理节点(MGM) , 用户端节点(API) 可以处在 不同的机器上，也可以在同一个机器上面，集群中至少要有一个DB节点，2个以上 时就能实现集群的高可用保证，DB节点增加时，集群的处理速度会变慢。
* SQL Nodes客户端节点，通过他实现 cluster DB 的访问，这个节点也就是普通的 mysqld 进程， 需要在配置文件中配置ndbcluster 指令打开 NDB Cluster storage engine 存储引擎，增加 API 节点会提高整个集群的并发访问速度和整体的吞吐量，该节点 可以部署在Web应用服务器上，也可以部署在专用的服务器上，也可以和DB部署在同一台服务器上。

**特点：**

|  |  |
| --- | --- |
| 打分项 | 满足程度 |
| 复杂性 | 很高 |
| 可扩展性 | 高 |
| 支持扩展负载均衡 | 是 |
| 脑裂几率 | 无 |
| 在线数据不一致风险(不包括故障恢复) | 无 |
| 是否支持HA | 是 |
| 数据丢失场景 | 无 |
| 操作难度 | 高 |
| 性能 | 高 |
| 网络性能要求 | 极高 |
| 适合场景 | 适用于对数据要求可用度高且对外键没要求的应用场景。 |

## Mysql对操作系统环境要求

### Mysql for Linux系统环境要求

由于目前JXXX主推Redhat6.6 Redhat7.2 故只论述这两个版本的操作系统

大多数MySQL都部署在linux系统上，所以操作系统的一些参数也会影响到MySQL性能，需要对linux内核进行适当调整，可参考表格十：

| 建议内核参数设置 | 配置 mysql数据服务器的内核参数 |
| --- | --- |
| net.ipv4.tcp\_fin\_timeout = 30 | #TIME\_WAIT超时时间，默认是60s |
| net.ipv4.tcp\_tw\_reuse = 1 | #1表示开启复用，允许TIME\_WAIT socket重新用于新的TCP连接，0表示关闭 |
| net.ipv4.tcp\_tw\_recycle = 1 | #1表示开启TIME\_WAIT socket快速回收，0表示关闭 |
| net.ipv4.tcp\_max\_tw\_buckets = 4096 | #系统保持TIME\_WAIT socket最大数量，如果超出这个数，系统将随机清除一些TIME\_WAIT并打印警告信息 |
| net.ipv4.tcp\_max\_syn\_backlog = 4096 | #进入SYN队列最大长度，加大队列长度可容纳更多的等待连接 |

表格十：建议内核参数设置

在linux系统中，如果进程打开的文件句柄数量超过系统默认值1024，就会提示“too many files open”信息，所以要调整打开文件句柄限制，参考表十一：

| 建议内核参数设置 |
| --- |
| \* soft nofile 65535 |
| \* hard nofile 65535 |

表十一：文件句柄参数设置

## 数据库锁设计

### 2.4.1 表级锁设计

对于MyISAM存储引擎，虽然使用表级锁定在锁定实现的过程中比实现行级锁定或者页级锁所带来的附加成本都要小，锁定本身所消耗的资源也是最少。但是由于锁定的颗粒度比较到，所以造成锁定资源的争用情况也会比其他的锁定级别都要多，从而在较大程度上会降低并发处理能力。所以，在设计MyISAM存储引擎锁定问题的时候，最关键的就是如何让其提高并发度。由于锁定级别是不可能改变的了，所以我们首先需要尽可能让锁定的时间变短，然后就是让可能并发进行的操作尽可能的并发。

1. 缩短锁定时间

* 尽两减少大的复杂Query，将复杂Query分拆成几个小的Query分布进行
* 尽可能的建立足够高效的索引，让数据检索更迅速；
* 尽量让MyISAM存储引擎的表只存放必要的信息，控制字段类型；
* 利用合适的机会优化MyISAM表数据文件。

1. 分离能并行的操作

* MyISAM的存储引擎具有ConcurrentInsert（并发插入）的特性，MyISAM存储引擎有一个控制是否打开Concurrent Insert功能的参数选项：concurrent\_insert，可以设置为0，1或者2。

1. 合理利用读写优先级

* MYSQL的表级锁定在默认情况下是写优先级大于读。所以，如果系统是一个以读为主，而且要优先保证查询性能的话，可以通过设置系统参数选项low\_priority\_updates=1，将写的优先级设置为比读低，即告诉MYSQL尽量先处理读请求；如果系统须要有限保证数据写入的性能的话，则不用设置low\_priority\_updates参数了。
* 另外一个方面还可以通过开启Key Cache，用来缓存索引来提高读取速度，如果这样还不觉得快的话，还可以通过Query Cache功能来直接缓存Query的结果集。当然，还可以合理利用第三方案Cache软件，如Memcached，来缓存数据，提升系统性能。

1. 读写分离

* 将原数据表进行复制作为slave端，这些数据只用于用户的select操作，因为这些数据是没有经过updatet和delete过的，所以它查询起来非常快，而且也不会出现写锁资源争用的情况；而原数据用于除select以外的其它操作。这样经过读写分离后的数据操作起来速度会更快。当然在建立数据表的同时你可以对表进行适当的分区，经过分区后的数据表，所有数据会分成几块放在不同的区域中，这就好像分成几个硬盘一样，查询的时候首先会将要查的目标数据定位在某个区中进行查询，从而节省了IO与数据检索量，扫描起来就会更快，从而提高查询速度

### 2.4.2 行级锁设计

* 尽可能让所有的数据检索都通过索引来完成，从而避免InnoDB因为无法通过索引键加锁而升级为表级锁定；
* 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；
* 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；
* 尽量控制事务的大小，减少锁定的资源量和锁定时间长度；
* 在业务环境允许的情况下，尽量使用较低级别的事务隔离，以减少MySQL因为实现事务隔离级别所带来的附加成本。

降低死锁概率

由于Innodb的行级锁定和事务性，所以肯定会产生死锁

* 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；
* 在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁产生概率；
* 对于非常容易产生死锁的业务部分，可以尝试使用升级锁定颗粒度，通过表级锁定来减少死锁产生的概率；

## 表空间设计

### 共享表空间的设计

1. innodb共享表空间文件是ibdata1,并且初始设置10M空间，当表空间不足时自动扩张

这样就会导致一个问题，当越来越多的数据增加的时候，ibdata也会持续膨胀，有的达到几十G，上百G

那么，，当前存储数据的磁盘分区满的时候，要怎么样去扩展数据空间呢？

多路径设置：

* 放在不同的磁盘，可以平均磁盘负载，提高数据库性能。

1. mysql数据库会生成空间碎片了，这些空间碎片对于我们来讲影响不大但如果空间碎片多了会导致mysql查询缓存

对于小表的话直接用ALTER TABLE table\_name ；回收表空间，对于大表就不能直接采用这种方式，因为会造成长时间的锁表。可以采用新建表转移数据，然后删除旧表的形式，然后再重命名表。

### 独立表空间的设计

innodb独占表空间:  每一个表都将会生成以独立的文件方式来进行存储，每一个表都有一个.frm表描述文件，还有一个.ibd文件，这个文件包括了单独一个表的数据内容以及索引内容。

* 单表增加过大，当单表占用空间过大时，存储空间不足，只能从操作系统层面思考解决方法；
* 对于每个表设置自已独立的表空间。
* Drop table操作自动回收表空间，如果对于统计分析或是日值表，删除大量数据后可以通过:alter table TableName engine=innodb;回缩不用的空间。
* 对于使用innodb-plugin的Innodb引擎使用turncate table使空间收缩。

## 字符集选择

创建数据库及表时设置字符集，避免出现中文乱码

通常情况下，为了避免mysql中文乱码，选择UTF-8（1208）字符集，除特殊要求外尽量不要选择较偏门的字符集。

**注意：**

在创建数据库时没有指定字符集，如果出现中文乱码可以通过修改/etc/my.cnf配置文件进行字符集修改：

[mysql.server]

default-character-set = utf8

[client]

default-character-set = utf8

该方法需要重启数据库才能生效，但是生产环境往往是不允许这样的操作，故在创建数据库之前强制要求字符集选择UTF-8（1208）字符集。如果配置文件修改了还乱码，那剩下问题就一定在connection连接层上。

解决方法是在发送查询前执行SET NAMES 'utf8';它相当于下面的三句指令：

SET character\_set\_client = utf8;

SET character\_set\_results = utf8;

## 数据库权限控制

MySQL权限管理使用访问控制列表(ACL)方式，控制粒度最细可达到列级。把MySQL帐号分为三种，如表十二所示：

|  |  |
| --- | --- |
| 账号类型 | 建议 |
| 管理员帐号 | 即root，可授予all on \*.\* with grant option.不允许远程登录。 |
| 应用帐号 | 尽量根据应用系统的功能模块划分使用不同的应用帐号，结合MySQL登录认证来限制应用帐号仅在某些网段的IP上进行连接。如：财务模块可单独设立应用帐号fin，对应财务数据库findb拥有ALL权限，同时限制该帐号仅可从应用服务器网段发起连接： grant all on findb.\* to fin@'10.0.0.%'; |
| 普通用户帐号 | 普通用户帐号应首先划分只读帐号与读写帐号，遵循“最小权限”原则，不对帐号授予任何除必须之外的权限。限制普通帐号仅能在部分网段进行连接，对于可能暴露于公网的数据库更应严格执行连接地址限制。 |

表十二：MySQL账号设计表格

**注意：**

对于以下权限的grant操作，应该特别加以重视：

* 慎用grant all形式授权，除对root帐号外禁止使用\*.\*作为权限对象。
* FILE权限，拥有FILE权限的数据库用户，可以将数据库所在服务器上的全局可读文件及数据目录文件读取到MySQL中，进而使用select命令读取到该文件的内容。如：MySQL帐号someone拥有file权限，则someone可以将服务器文件系统中的如/etc/XXX.conf文件读取到MySQL数据库中，进而获取到XXX.conf文件的内容。对于必须使用FILE权限的数据库，应设置secure\_file\_priv选项来限制其使用范围。
* GRANT\_OPTION权限，拥有GRANT\_OPTION权限的帐号可以将其拥有的其他权限授予他人，除管理用的帐号外，不建议对任何帐号赋予GRANT\_OPTION。
* PROCESS权限，有用PROCESS权限的帐号可查看当前执行的语句，这些语句包括正在执行的修改或设置密码的语句。
* 对于mysql schema的权限不应授予非管理用的帐号，比如对mysql.user的数据操作可直接修改整个帐号体系。
* 禁止创建“匿名”帐号（即不指定username的帐号）。

**建议：**

* 规范账号命名规则。
* 账号的授权域精确到主机名(或IP)。
* 账号相关操作，不建议直接操作user表。
* 定期整理账号，清理没有人使用的账号。
* 周期性更新密码。
* 建议账号命名中包含读写等标识，如后缀为\_w表示可写账号。
* 建议不同业务模块使用不同账号。

# Mysql数据库安装部署操作规范

## 部署准备

### Linux操作系统

本文档的安装部署部分适用于使用rpm包管理方式在RHEL 6.6操作系统上安装MySQL 5.7版本。对同类操作系统有参考意义，如：ORACLE LINUX/CENTOS/FEDORA等。

|  |  |
| --- | --- |
| 操作系统版本 | 内核版本 |
| Redhat6.6 |  |
| Redhat7.2 |  |

### 数据库磁盘和文件系统要求

磁盘要求

使用磁盘条带化技术来提高IO性能：根据数据的重要性及数据库层高可用方案的不同，可对重要性低或高可用方案完备的数据库使用RAID 0级别的条带化，对要求高可靠的数据库使用RAID 0+1级别的条带化，对写频繁的数据库建议不使用RAID N(如RAID 5/6等需要做奇偶校验的方式)条带化。

文件系统要求

xfs和 ext4 在性能上并没有非常突出的优点. 对于支持mysql, 都足够用了,不建议使用ext3文件系统，ext3文件系统有太多严重的限制，例如inode只有一个互斥变量，并且fsync（）时会刷新所有脏块等。

## 安装Mysql软件

可通过yum源或直接下载rpm包的方式安装，任选其一：

**通过yum源安装：**

##配置yum repo：

cat > /etc/yum.repos.d/mysql-community.repo<<\eofrepo

[mysql57-community]

name=MySQL 5.7 Community Server

baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/

enabled=1

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql

eofrepo

##执行安装：

sudo yum install mysql-community-server

**下载rpm包安装：**

##将以下rpm包下载到服务器上

http://dev.mysql.com/get/mysql-community-client-5.7.18-1.el6.x86\_64.rpm

http://dev.mysql.com/get/mysql-community-common-5.7.18-1.el6.x86\_64.rpm

http://dev.mysql.com/get/mysql-community-libs-5.7.18-1.el6.x86\_64.rpm

http://dev.mysql.com/get/mysql-community-libs-compat-5.7.18-1.el6.x86\_64.rpm

<http://dev.mysql.com/get/mysql-community-server-5.7.18-1.el6.x86_64.rpm>

##安装：

yum install ./mysql-community\*rpm

mysql-community-server的依赖中包含了mysql-community-client、mysql-community-common以及mysql-community-libs；

此安装会默认创建操作系统上的mysql:mysql用户及组；

### Mysql相关文件及目录

Mysql安装相关目录如表十三所示：

| **Files or Resources** | **Location** |
| --- | --- |
| Client programs and scripts | /usr/bin |
| **[mysqld](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\programs.html" \l "mysqld" \o "4.3.1 mysqld — The MySQL Server)** server | /usr/sbin |
| Configuration file | /etc/my.cnf |
| Data directory | /var/lib/mysql |
| Error log file | For RHEL, Oracle Linux, CentOS or Fedora platforms: /var/log/mysqld.log  For SLES: /var/log/mysql/mysqld.log |
| Value of [secure\_file\_priv](file:///D:\\Users\\Young\\Desktop\\refman-5.7-en.html-chapter\\server-administration.html" \l "sysvar_secure_file_priv) | /var/lib/mysql-files |
| System V init script | For RHEL, Oracle Linux, CentOS or Fedora platforms: /etc/init.d/mysqld  For SLES: /etc/init.d/mysql |
| Systemd service | For RHEL, Oracle Linux, CentOS or Fedora platforms: mysqld  For SLES: mysql |
| Pid file | /var/run/mysql/mysqld.pid |
| Socket | /var/lib/mysql/mysql.sock |
| Keyring directory | /var/lib/mysql-keyring |
| Unix manual pages | /usr/share/man |

表十三：MySQL相关文件及路径（rpm版本）

### 启动数据库及简单配置

修改配置文件：

默认配置文件为/etc/my.cnf

##根据需要修改mysql的数据目录，该目录在启动前应不存在，由启动初始化创建：

datadir=/some\_other\_datadir

启动mysqld

service mysqld start

初次启动mysqld时，会执行以下操作：

* 初始化mysql：包括创建相关的目录文件、配置合适的文件权限、创建相关的数据库表等；
* [默认创建一个root@localhost的数据库帐号，并生成一个随机密码，该密码会记录在/var/log/mysqld.log中。](mailto:默认创建一个root@localhost的数据库帐号，并生成一个随机密码，该密码会记录在/var/log/mysqld.log中。)

若出现启动失败，可在日志文件中查找原因，日志包括：/var/log/mysqld.log和数据目录的`hostname`.err/`hostname`.log等。

## Mysql主从复制安装部署

由于篇幅过长参考**Mysql无数据丢失主从半同步复制安装部署.docx**

## Corosync+pacemaker+mysql+共享存储

# 数据库运维操作规范

## 服务端管理与配置

### 配置管理

#### 建议配置

建议的/etc/my.cnf配置文件内容如下：

[mysqld]

datadir=/mysql

socket=/var/lib/mysql/mysql.sock

symbolic-links=0

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

validate\_password\_check\_user\_name=ON

log-bin=mysql-bin

server-id=1

secure\_file\_priv = ""

#### 拆分配置

如果需要拆分配置文件，则将相同配置写入主配置文件中，不同的配置放入子配置文件中，并在主配置文件中使用!include命令引用。同时子配置文件推荐以section段进行命名。示例如下：

cat etc/my.cnf

[mysqld]

!include ${MYSQL\_BASEDIR}/etc/mysqld.cnf

cat etc/mysqld.cnf

[mysqld]

server-id=2000

## 数据库管理

### ****创建数据库****

CREATE DATABASE 的语法：

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name

[create\_specification [, create\_specification] ...]

create\_specification:

[DEFAULT] CHARACTER SET charset\_name

| [DEFAULT] COLLATE collation\_name

#### create 创建数据库

MySQL

create database firstDB;

#### Create创建数据库指定字符集

CREATE DATABASE db\_name DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

#### show 查看所有数据库

MySQL

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| firstDB |

| mysql |

| performance\_schema |

+--------------------+

4 rows in set (0.00 sec)

### ****数据表（table）管理****

#### alter 修改数据库编码

**alter 命令修改数据库编码：**

默认创建的数据库默认不支持中文字符，如果我们需要它支持中文字符，则将它的编码设置为utf8格式：

MySQL

mysql> ALTER DATABASE testDB CHARACTER SET UTF8;

Query OK, 1 row affected (0.00 sec)

#### use 使用数据库

MySQL

mysql> use firstDB;

Database changed

#### 查看当前使用的数据库

MySQL

mysql> select database();

+------------+

| database() |

+------------+

| firstdb |

+------------+

1 row in set (0.00 sec)

#### drop 删除数据库

MySQL

mysql> drop database firstDB;

Query OK, 0 rows affected (0.00 sec)

我们首先创建一个数据库，提供我们往后的使用：

MySQL

mysql> create database testDB;

Query OK, 1 row affected (0.00 sec)

创建后记得用use命令进入（使用）数据库，不然后面的操作都会不成功的。

#### create 创建表

MySQL

mysql> create table PEOPLE (

-> ID int AUTO\_INCREMENT PRIMARY KEY,

-> NAME varchar(20) not null,

-> AGE int not null,

-> BIRTHDAY datetime);

Query OK, 0 rows affected (0.01 sec)

#### show 显示表

显示当前数据库所有的数据表

MySQL

mysql> show tables;

+------------------+

| Tables\_in\_testdb |

+------------------+

| PEOPLE |

+------------------+

1 row in set (0.00 sec)

#### desc 查看表结构

MySQL

mysql> desc PEOPLE

-> ;

+----------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| NAME | varchar(20) | NO | | NULL | |

| AGE | int(11) | NO | | NULL | |

| BIRTHDAY | datetime | YES | | NULL | |

+----------+-------------+------+-----+---------+----------------+

4 rows in set (0.01 sec)

#### alter 修改表结构（增、删、改）

默认创建的表不支持中文字符，所以需将表编码设置为utf8：

MySQL

mysql> ALTER TABLE KEYCHAIN CONVERT TO CHARACTER SET UTF8;

Query OK, 1 row affected (0.02 sec)

Records: 1 Duplicates: 0 Warnings: 0

#### A lter 在表中添加列（字段）

MySQL

mysql> alter table PEOPLE add star BOOL;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

**提示：**

在MySQL里，布尔类型会自动转换为tinyint(1)类型。

我们不妨使用desc去查看一下PEOPLE表结构：

MySQL

mysql> desc PEOPLE;

+----------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| NAME | varchar(20) | NO | | NULL | |

| AGE | int(11) | NO | | NULL | |

| BIRTHDAY | datetime | YES | | NULL | |

| star | tinyint(1) | YES | | NULL | |

+----------+-------------+------+-----+---------+----------------+

5 rows in set (0.00 sec)

#### alter 修改表（列）字段

MySQL

mysql> alter table PEOPLE MODIFY star int;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

也可以指定 int(n) 的长度，比如 int(2)。

我们再次使用desc查看PEOPLE表结构：

MySQL

mysql> desc PEOPLE;

+----------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| NAME | varchar(20) | NO | | NULL | |

| AGE | int(11) | NO | | NULL | |

| BIRTHDAY | datetime | YES | | NULL | |

| star | int(11) | YES | | NULL | |

+----------+-------------+------+-----+---------+----------------+

5 rows in set (0.00 sec)

#### delete 删除表（列）字段

MySQL

mysql> alter table PEOPLE DROP column star;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

删除后，再次查看PEOPLE表结构：

MySQL

mysql> desc PEOPLE;

+----------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| NAME | varchar(20) | NO | | NULL | |

| AGE | int(11) | NO | | NULL | |

| BIRTHDAY | datetime | YES | | NULL | |

+----------+-------------+------+-----+---------+----------------+

4 rows in set (0.00 sec)

删除字段成功，现在我们已经不能看到star的字段了。

#### rename 重命名表名

MySQL

mysql> RENAME TABLE PEOPLE TO NEW\_PEOPLE;

Query OK, 0 rows affected (0.00 sec)

#### null or not null

修改表字段允许为空或不允许为空：

MySQL

mysql> ALTER TABLE PEOPLE MODIFY AGE INT(3) NULL;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

把 PEOPLE 表的 AGE 字段设置成“允许为空”，即插入记录时这个字段可以不录入。否则相反。

它的格式为：ALTER TABLE <TALBE\_NAME> MODIFY <CLOUMN> <NULL ‘OR’ NOT NULL>

#### create 利用已有数据创建新表

MySQL

mysql> create table newTable select \* from PEOPLE;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

我们查看一下目前数据库存在的表：

MySQL

mysql> show tables;

+------------------+

| Tables\_in\_testdb |

+------------------+

| PEOPLE |

| newTable |

+------------------+

2 rows in set (0.00 sec)

### ****数据的操作及管理****

数据表的基本操作，包含增、删、改、查数据。

以下命令均在PEOPLE表上操作。

#### 增加数据（增）

PEOPLE表目前是没有数据的，它是空的数据表，我们现在先添加一些数据。

insert into 命令添加数据：

MySQL

mysql> insert into PEOPLE VALUES (null, 'Anny', 22, '1992-05-22');

Query OK, 1 row affected (0.00 sec)

使用select命令查看表，现在我们查看PEOPLE数据表的数据：

MySQL

mysql> select \* from PEOPLE;

+----+------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

+----+------+-----+---------------------+

1 row in set (0.00 sec)

数据表现在有一条数据。

我们多添加几条数据，如：

MySQL

mysql> select \* from PEOPLE;

+----+--------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+--------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

| 2 | Garvey | 23 | 1991-05-22 00:00:00 |

| 3 | Lisa | 25 | 1989-05-22 00:00:00 |

| 4 | Nick | 24 | 1990-05-22 00:00:00 |

| 5 | Rick | 24 | 1991-05-22 00:00:00 |

+----+--------+-----+---------------------+

5 rows in set (0.00 sec)

#### 删除数据（删）

delete 命令删除数据：

MySQL

mysql> delete from PEOPLE where name = 'Lisa';

Query OK, 1 row affected (0.01 sec)

再次查询PEOPLE表：

MySQL

mysql> select \* from PEOPLE;

+----+--------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+--------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

| 2 | Garvey | 23 | 1991-05-22 00:00:00 |

| 4 | Nick | 24 | 1990-05-22 00:00:00 |

| 5 | Rick | 24 | 1991-05-22 00:00:00 |

+----+--------+-----+---------------------+

4 rows in set (0.00 sec)

已经看不到名为“Lisa”的数据了。

#### 修改数据（改）

update 命令修改数据：

MySQL

mysql> update PEOPLE set name='Calvin' where name = 'Garvey';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

查询PEOPLE表内容：

MySQL

mysql> select \* from PEOPLE;

+----+--------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+--------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

| 2 | Calvin | 23 | 1991-05-22 00:00:00 |

| 4 | Nick | 24 | 1990-05-22 00:00:00 |

| 5 | Rick | 24 | 1991-05-22 00:00:00 |

+----+--------+-----+---------------------+

4 rows in set (0.00 sec)

名为“Garvey”的记录已经修改为“Calvin”。

3.4 查询数据（查）

select 命令查询数据，最简单的就是查询表的所有数据，也就是我们最初使用到的那条命令：

MySQL

mysql> select \* from PEOPLE;

+----+--------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+--------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

| 2 | Calvin | 23 | 1991-05-22 00:00:00 |

| 4 | Nick | 24 | 1990-05-22 00:00:00 |

| 5 | Rick | 24 | 1991-05-22 00:00:00 |

+----+--------+-----+---------------------+

4 rows in set (0.00 sec)

格式：select \* from <表名>，\*代表所有字段。

查询数据时也可指定显示的（列）字段：

MySQL

mysql> select NAME, AGE, BIRTHDAY from PEOPLE;

+--------+-----+---------------------+

| NAME | AGE | BIRTHDAY |

+--------+-----+---------------------+

| Anny | 22 | 1992-05-22 00:00:00 |

| Calvin | 23 | 1991-05-22 00:00:00 |

| Nick | 24 | 1990-05-22 00:00:00 |

| Rick | 24 | 1991-05-22 00:00:00 |

+--------+-----+---------------------+

4 rows in set (0.00 sec)

格式：select <字段名,字段名,…> from <表名>。

select查询命令还有很多的高级用法，比如用来查找不重复（distinct）的数据，使数据 按条件排序（order by），按查询条件显示数据（where）等等。

## 管理索引视图

### ****创建索引****

索引在MySQL中也叫做“键”，是存储引擎用于快速找到记录的一种数据结构。索引对于良好的性能非常关键，尤其是当表中的数据量越来越大时，索引对于性能的影响愈发重要；索引优化应该是对查询性能优化最有效的手段，索引能够轻易将查询性能提高好几个数量级；索引相当于字典的音序表，如果要查某个字，如果不使用音序表，则需要从几百页中逐页去查。

**语法**

CREATE TABLE 表名 (  
字段名1 数据类型 [完整性约束条件…],  
字段名2 数据类型 [完整性约束条件…],  
[UNIQUE | FULLTEXT | SPATIAL ] INDEX | KEY  
[索引名] (字段名[(长度)] [ASC |DESC])   
);

#### 创建普通索引

CREATE TABLE department10 (  
dept\_id INT,  
dept\_name VARCHAR(30) ,  
comment VARCHAR(50),  
INDEX index\_dept\_name (dept\_name)  
);

#### 创建唯一索引

CREATE TABLE department11 (  
dept\_id INT,  
dept\_name VARCHAR(30) ,  
comment VARCHAR(50),  
UNIQUE INDEX index\_dept\_name (dept\_name)  
);

#### 创建全文索引

CREATE TABLE department12 (  
dept\_id INT,  
dept\_name VARCHAR(30) ,  
comment VARCHAR(50),  
log text,  
FULLTEXT INDEX index\_log (log)  
);

#### 创建多列索引

CREATE TABLE department13 (  
dept\_id INT,  
dept\_name VARCHAR(30) ,  
comment VARCHAR(50),  
INDEX index\_dept\_name\_comment (dept\_name, comment)  
);

### ****CREATE创建索引****

**语法：**

CREATE [UNIQUE | FULLTEXT | SPATIAL ] INDEX 索引名

ON 表名 (字段名[(长度)] [ASC |DESC]) ;

#### 创建普通索引

CREATE INDEX index\_dept\_name ON department (dept\_name);

#### 创建唯一索引

CREATE UNIQUE INDEX index\_dept\_name ON department (dept\_name);

#### 创建全文索引

CREATE FULLTEXT INDEX index\_dept\_name ON department (dept\_name);

#### 创建多列索引

CREATE INDEX index\_dept\_name\_ comment ON department (dept\_name, comment);

### ALTER 创建索引

**语法：**

ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL ] INDEX

索引名 (字段名[(长度)] [ASC |DESC]) ;

#### 创建普通索引

ALTER TABLE department ADD INDEX index\_dept\_name (dept\_name);

#### 创建唯一索引

ALTER TABLE department ADD UNIQUE INDEX index\_dept\_name (dept\_name);

#### 创建全文索引

ALTER TABLE department ADD FULLTEXT INDEX index\_dept\_name (dept\_name);

#### 创建多列索引

ALTER TABLE department ADD INDEX index\_dept\_name\_comment (dept\_name, comment);

### 查看索引

SHOW CRETAE TABLE 表名\G

#### 删除索引

show create table employee6;

DROP INDEX 索引名 ON 表名;

### ****管理视图****

#### 创建视图

视图是从数据库里导出一个或多个表的虚拟表，是用来方便用户对数据的操作。

MySQL

mysql> CREATE VIEW PEOPLE\_VIEW (

-> NAME, AGE)

-> AS SELECT NAME, AGE FROM PEOPLE;

#### 查看视图。

MySQL

EOPLE PEOPLE.AGE PEOPLE.BIRTHDAY PEOPLE.ID PEOPLE.NAME

mysql> SELECT \* FROM PEOPLE\_VIEW

-> ;

+--------+-----+

| NAME | AGE |

+--------+-----+

| Anny | 22 |

| Calvin | 23 |

| Nick | 24 |

| Rick | 24 |

+--------+-----+

4 rows in set (0.00 sec)

* DESC 命令查看视图的结构。

MySQL

mysql> DESC PEOPLE\_VIEW;

+-------+---------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+---------+------+-----+---------+-------+

| ID | int(11) | NO | | 0 | |

+-------+---------+------+-----+---------+-------+

1 row in set (0.01 sec)

#### 替换视图

创建或替换原有视图。

MySQL

mysql> CREATE OR REPLACE VIEW PEOPLE\_VIEW(PEOPLE\_ID,PEOPLE\_NAME,PEOPLE\_AGE) AS SELECT ID,NAME,AGE FROM PEOPLE;

创建或替换后查看视图。

MySQL

mysql> SELECT \* FROM PEOPLE\_VIEW;

+-----------+-------------+------------+

| PEOPLE\_ID | PEOPLE\_NAME | PEOPLE\_AGE |

+-----------+-------------+------------+

| 1 | Anny | 22 |

| 2 | Calvin | 23 |

| 4 | Nick | 24 |

| 5 | Rick | 24 |

+-----------+-------------+------------+

4 rows in set (0.00 sec)

#### 操作视图

当视图数据有变化时（增、删、改），真实的表数据也会随着改变。也就是说，对视图的操作就是对表的数据，所以我们可以把视图当作表。

例：往视图插入一条数据。

MySQL

mysql> INSERT INTO PEOPLE\_VIEW VALUES(NULL, 'Kerry', '33');

Query OK, 1 row affected (0.00 sec)

插入数据成功后查看视图。

MySQL

mysql> SELECT \* FROM PEOPLE\_VIEW ;

+-----------+-------------+------------+

| PEOPLE\_ID | PEOPLE\_NAME | PEOPLE\_AGE |

+-----------+-------------+------------+

| 1 | Anny | 22 |

| 2 | Calvin | 23 |

| 4 | Nick | 24 |

| 5 | Rick | 24 |

| 6 | Kerry | 33 |

+-----------+-------------+------------+

5 rows in set (0.00 sec)

可以在视图上看到我们刚刚插入的数据，现在我们就来验证一下真实的表是否也会作出变化。

MySQL

mysql> SELECT \* FROM PEOPLE;

+----+--------+-----+---------------------+

| ID | NAME | AGE | BIRTHDAY |

+----+--------+-----+---------------------+

| 1 | Anny | 22 | 1992-05-22 00:00:00 |

| 2 | Calvin | 23 | 1991-05-22 00:00:00 |

| 4 | Nick | 24 | 1990-05-22 00:00:00 |

| 5 | Rick | 24 | 1991-05-22 00:00:00 |

| 6 | Kerry | 33 | NULL |

+----+--------+-----+---------------------+

5 rows in set (0.00 sec)

可见，真实的表数据也已经有所改变，刚刚往视图里插入的那一条数据存在于真实表中，真理便是：对视图的操作就是对表的数据。

#### 删除视图

MySQL

mysql> DROP VIEW PEOPLE\_VIEW;

Query OK, 0 rows affected (0.00 sec)

## 表空间管理

**INNODB 对于表的存储有两种形式 ：**

* 一种是共享表空间，及多张表放在一个文件中。
* 一种是独立表空间，每个表都有独立的数据文件。

### 共享表空间

#### 共享表空间配置

查看当前共享表空间

mysql> show variables like '%innodb\_data\_file\_path%' ;

+-----------------------+------------------------+

| Variable\_name | Value |

+-----------------------+------------------------+

| innodb\_data\_file\_path | ibdata1:12M:autoextend |

+-----------------------+------------------------+

1 row in set (0.00 sec)

[root@localhost mysql]# du -h ibdata1

12M ibdata1

#### 增加共享表空间

[root@localhost mysql]# cat /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

innodb\_data\_file\_path=ibdata1:12M;ibdata2:24M:autoextend

#### 查看变更后配置

mysql> show variables like '%innodb\_data\_file\_path%' ;

+-----------------------+------------------------------------+

| Variable\_name | Value |

+-----------------------+------------------------------------+

| innodb\_data\_file\_path | ibdata1:12M;ibdata2:24M:autoextend |

+-----------------------+------------------------------------+

1 row in set (0.00 sec)

[root@localhost mysql]# ll

total 135200

-rw-rw---- 1 mysql mysql 56 Nov 29 09:45 auto.cnf

drwx------ 2 mysql mysql 4096 Nov 29 16:17 dao

-rw-rw---- 1 mysql mysql 12582912 Nov 29 16:58 ibdata1

-rw-rw---- 1 mysql mysql 25165824 Nov 29 16:58 ibdata2

-rw-rw---- 1 mysql mysql 50331648 Nov 29 16:58 ib\_logfile0

-rw-rw---- 1 mysql mysql 50331648 Nov 29 08:16 ib\_logfile1

-rw-rw---- 1 mysql mysql 5 Nov 29 16:58 localhost.localdomain.pid

drwx------ 2 mysql mysql 4096 Nov 29 08:16 mysql

srwxrwxrwx 1 mysql mysql 0 Nov 29 16:58 mysql.sock

drwx------ 2 mysql mysql 4096 Nov 29 08:16 performance\_schema

-rw-r--r-- 1 root root 117 Nov 29 08:16 RPM\_UPGRADE\_HISTORY

-rw-r--r-- 1 mysql mysql 117 Nov 29 08:16 RPM\_UPGRADE\_MARKER-LAST

drwx------ 2 mysql mysql 4096 Nov 29 16:05 test

[root@localhost mysql]# du -h ibdata\*

12M ibdata1

24M ibdata2

### 独立表空间

mysql> show variables like '%innodb\_file\_per\_table%' ;

+-----------------------+-------+

| Variable\_name | Value |

+-----------------------+-------+

| innodb\_file\_per\_table | ON |

+-----------------------+-------+

1 row in set (0.00 sec)

#### 查看目录

[root@localhost dao]# ll

total 4

-rw-rw---- 1 mysql mysql 65 Nov 29 16:04 db.opt

[root@localhost dao]# ls

db.opt

#### 共享表空间回收

推荐使用重置表存储引擎的方法

alter table table\_name engine=innodb

重置期间所有写会话会被阻塞，读会话不受影响，重置消耗时间与已删除数据无关，仅与剩下数据呈线性相关，数据与时间消耗比例约为100万：30秒

## 

## 日志管理

### ****错误日志：****

错误日志主要记录如下几种日志：

* 服务器启动和关闭过程中的信息
* 服务器运行过程中的错误信息
* 事件调度器运行一个时间是产生的信息
* 在从服务器上启动从服务器进程是产生的信息

**错误日志定义：**

可以用--log-error[=file\_name]选项来指定mysqld保存错误日志文件的位置。如果没有给定file\_name 值，mysqld使用错误日志名host\_name.err 并在数据目录中写入日志文件。在mysql5.5.7之后：服务器关闭此项功能。只能使用重命名原来的错误日志文件，手动冲洗日志创建一个新的：方式如下：

mv stu18.magedu.com.err  stu18.magedu.com.err.old

mysqladmin flush-logs

如果你执行FLUSH LOGS，错误日志用-old重新命名后缀并且mysqld创建一个新的空日志文件。(如果未给出--log-error选项，则不会重新命名）。

* **查看当前错误日志配置：**

mysql> SHOW GLOBAL VARIABLES LIKE '%log\_error%';

+---------------+---------------------------------------+

| Variable\_name  | Value                                                  |

+---------------+---------------------------------------+

| log\_error          | /data/mysql/localhost.localdomain.err |

+---------------+---------------------------------------+

1 row in set (0.45 sec)

* **是否记录警告日志：**

mysql> SHOW GLOBAL VARIABLES LIKE '%log\_warnings%';

+---------------+-------+

| Variable\_name | Value   |

+---------------+-------+

| log\_warnings    | 1         |

+---------------+-------+

1 row in set (0.00 sec)

### ****通用查询日志:****

* 启动开关：general\_log={ON|OFF}
* 日志文件变量：general\_log\_file[=/PATH/TO/file]
* 全局日志开关：log={ON|OFF}   该开关打开后，所有日志都会被启用
* 记录类型：log\_output={TABLE|FILE|NONE}

因此，要启用通用查询日志，需要至少配置general\_log=ON，log\_output={TABLE|FILE}。而general\_log\_file如果没有指定，默认名是host\_name.log。

看看上述几个值的默认配置：

mysql> SHOW GLOBAL VARIABLES LIKE '%general\_log%';

+------------------+---------------------------+  
| Variable\_name      | Value                                |  
+------------------+---------------------------+  
| general\_log           | OFF                                  |  
| general\_log\_file     | /data/mysql/localhost.log |  
+------------------+---------------------------+

mysql> SHOW GLOBAL VARIABLES LIKE '%log\_output%';  
+---------------+-------+  
| Variable\_name  | Value  |  
+---------------+-------+  
| log\_output       | FILE     |  
+---------------+-------+

### ****慢查询日志：****

MySQL如果启用了slow\_query\_log=ON选项，就会记录执行时间超过long\_query\_time的查询（初使表锁定的时间不算 作执行时间）。日志记录文件为slow\_query\_log\_file[=file\_name]，如果没有给出file\_name值， 默认为主机名，后缀为-slow.log。如果给出了文件名，但不是绝对路径名，文件则写入数据目录。

* **默认与慢查询相关变量：**

mysql> SHOW GLOBAL VARIABLES LIKE '%slow\_query\_log%';

+----------------------------+--------------------------------+

| Variable\_name                     | Value                                        |

+----------------------------+--------------------------------+

| slow\_query\_log                    | OFF                                          |

| slow\_query\_log\_file              | /data/mysql/localhost-slow.log |

+----------------------------+--------------------------------+

服务器参数设定方法与通用查询日志相同，不做解释。

默认没有启用慢查询，为了服务器调优，建议开启。

mysql> SET GLOBAL slow\_query\_log=ON;

Query OK, 0 rows affected (1.45 sec)

如果要长久生效，则需要在配置文件中定义。

* **那么多久算是慢呢？**

如果查询时长超过long\_query\_time的定义值（默认10秒），即为慢查询：

mysql> SHOW GLOBAL VARIABLES LIKE 'long\_query\_time';

 +-----------------+-----------+

| Variable\_name    | Value         |

+-----------------+-----------+

| long\_query\_time | 10.000000 |

+-----------------+-----------+

### ****二进制日志：****

* **二进制日志启动开关：log-bin [= file\_name]**

在5.6及以上版本一定要手动指定。5.6以下版本默认file\_name为$datadir/mysqld-binlog

二进制日志用于记录所有更改数据的语句。主要用于复制和即时点恢复。

查看二进制日志的工具为：mysqlbinlog

二进制日志包含了所有更新了数据或者已经潜在更新了数据（例如，没有匹配任何行的一个DELETE）的所有语句。语句以“事件”的形式保存，它描述数据更改。二进制日志还包含关于每个更新数据库的语句的执行时间信息。它不包含没有修改任何数据的语句。

二进制日志的主要目的是在数据库存在故障时，恢复时能够最大可能地更新数据库（即时点恢复），因为二进制日志包含备份后进行的所有更新。二进制日志还用于在主复制服务器上记录所有将发送给从服务器的语句。

**那么二进制日志是记录执行的语句还是执行后的结果数据呢？**

1. **第一种情况：**

加入一个表有10万行数据，而现在要执行一个如下语句将amount字段的值全部在原来的基础上增加1000：

UPDATE sales.january SET amount=amount+1000；

此时如果要记录执行后的结果数据的话，日志会非常大。

因此在这种情况下应记录执行语句。这种方式就是基于语句的二进制日志。

1. **第二种情况：**

如果向某个字段插入的是当前的时间呢？如下：

INSERT INTO tb SET Birthdate=CURRENT\_TIME();

此时就不能记录语句了，因为不同时间执行的结果是不一样的。这是应该记录这一行的值，这种就是基于行（row）的二进制日志。

在有些情况，可能会结合两种方式来记录，这种叫做混合方式的二进制日志。

* **二进制日志记录时间：**

默认情况下，并不是每次写入时都将二进制日志与硬盘同步。因此如果操作系统或机器(不仅仅是MySQL服务器)崩溃，有可能二进制日志中最后的语句 丢失了。要想防止这种情况，你可以使用sync\_binlog全局变量(1是最安全的值，但也是最慢的)，使二进制日志在每N次二进制日志写入后与硬盘同步。

对非事务表的更新执行完毕后立即保存到二进制日志中。对于事务表，例如BDB或InnoDB表，所有更改表的更新(UPDATE、DELETE或 INSERT) 被缓存起来，直到服务器接收到COMMIT语句。在该点，执行完COMMIT之前，mysqld将整个事务写入二进制日志。当处理事务的线程启动时，它为 缓冲查询分配binlog\_cache\_size大小的内存。如果语句大于该值，线程则打开临时文件来保存事务。线程结束后临时文件被删除。

* **二进制日志的管理：**
* 日志的滚动：

在my.cnf中设定max\_binlog\_size = 200M，表示限制二进制日志最大尺寸为200M，超过200M后进行滚动。MySQL的滚动方式与其他日志不太一样，滚动时会创建一个新的编号大1的日志用于记录最新的日志，而原日志名字不会被改变。

每次重启MySQL服务，日志都会自动滚动一次。

另外如果需要手动滚动，则使用命令：

mysql> FLUSH LOGS;

* 日志的查看：

**查看有哪些二进制日志文件：**

mysql> SHOW BINARY LOGS;

+----------------------+-----------+  
| Log\_name                   | File\_size    |  
+----------------------+-----------+  
| mysqld-binlog.000001 |       143   |  
| mysqld-binlog.000002 |       120   |  
+----------------------+-----------+

**查看当前正在使用的是哪一个二进制日志文件：**

mysql> SHOW MASTER STATUS;

+----------------------+----------+--------------+------------------+-------------------+

| File                             | Position   | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+----------------------+----------+--------------+------------------+-------------------+

| mysqld-binlog.000002 |      120   |                       |                            |                              |

+----------------------+----------+--------------+------------------+-------------------+

* 做个操作后再次查看：

mysql> use jiaowu;

Database changed

mysql> INSERT INTO students (Name,Age) VALUES ('stu1',24);

Query OK, 1 row affected (0.07 sec)

mysql> SHOW MASTER STATUS;

+----------------------+----------+--------------+------------------+-------------------+

| File                             | Position   | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+----------------------+----------+--------------+------------------+-------------------+

| mysqld-binlog.000002 |      394   |                      |                             |                              |

+----------------------+----------+--------------+------------------+-------------------+

1 row in set (0.00 sec)

可以看到Position（位置）已经改变。

* **查看二进制日志内容：**

mysql> SHOW BINLOG EVENTS IN 'mysqld-binlog.000002';

+----------------------+-----+-------------+-----------+-------------+------------------------------------------------------------------+

| Log\_name             | Pos | Event\_type  | Server\_id | End\_log\_pos | Info                                                             |

+----------------------+-----+-------------+-----------+-------------+------------------------------------------------------------------+

| mysqld-binlog.000002 |   4 | Format\_desc |         1 |         120 | Server ver: 5.7.-log, Binlog ver: 4                            |

| mysqld-binlog.000002 | 120 | Query       |         1 |         203 | BEGIN                                                            |

| mysqld-binlog.000002 | 203 | Intvar      |         1 |         235 | INSERT\_ID=11                                                     |

| mysqld-binlog.000002 | 235 | Query       |         1 |         363 | use `jiaowu`; INSERT INTO students (Name,Age) VALUES ('stu1',24) |

| mysqld-binlog.000002 | 363 | Xid         |         1 |         394 | COMMIT /\* xid=13 \*/                                              |

+----------------------+-----+-------------+-----------+-------------+------------------------------------------------------------------+

5 rows in set (0.01 sec)

* 该语句还可以加上Position（位置），指定显示从哪个Position（位置）开始：

mysql> SHOW BINLOG EVENTS IN 'mysqld-binlog.000002' FROM 203;

+----------------------+-----+------------+-----------+-------------+------------------------------------------------------------------+

| Log\_name             | Pos | Event\_type | Server\_id | End\_log\_pos | Info                                                             |

+----------------------+-----+------------+-----------+-------------+------------------------------------------------------------------+

| mysqld-binlog.000002 | 203 | Intvar     |         1 |         235 | INSERT\_ID=11                                                     |

| mysqld-binlog.000002 | 235 | Query      |         1 |         363 | use `jiaowu`; INSERT INTO students (Name,Age) VALUES ('stu1',24) |

| mysqld-binlog.000002 | 363 | Xid        |         1 |         394 | COMMIT /\* xid=13 \*/                                              |

+----------------------+-----+------------+-----------+-------------+------------------------------------------------------------------+

3 rows in set (0.00 sec)

* **使用命令mysqlbinlog查看二进制日志内容：**

**基本语法：**

mysqlbinlog [options] log-files

* **常用options（选项）：**

--start-position     ：开始位置  
--stop-position     ：结束位置

--start-datetime 'yyyy-mm-dd hh:mm:ss'  ：开始时间  
--stop-datetime 'yyyy-mm-dd hh:mm:ss'  ：结束时间

****示例：****

[root@localhost mysql]# mysqlbinlog --start-datetime '2017-09-26 23:30:00' mysqld-binlog.000002

[root@localhost mysql]# mysqlbinlog --start-position 203 --stop-position 389  mysqld-binlog.000002

* **使用二进制日志还原数据：**

使用mysqlbinlog读取需要的日志内容，使用标准输入重定向到一个sql文件，然后在mysql服务器上导入即可，

**如下：**

mysqlbinlog mysqld-binlog.000002 >/root/temp\_date.sql

* **删除二进制日志文件：**

二进制日志文件不能直接删除的，如果使用rm等命令直接删除日志文件，可能导致数据库的崩溃。

* **必须使用命令PURGE删除日志，语法如下：**

PURGE { BINARY | MASTER } LOGS  
{ TO 'log\_name' | BEFORE datetime\_expr }

**示例：**

查看当前所有日志文件

[root@localhost mysql]# ll mysqld-binlog.0\*

-rw-rw----. 1 mysql mysql 445 9月  22 15:13 mysqld-binlog.000002

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000003

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000004

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000005

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000006

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000007

-rw-rw----. 1 mysql mysql 120 9月  22 15:13 mysqld-binlog.000008

* 删除mysqld-binlog.000006以前的日志文件：

mysql> PURGE BINARY LOGS TO 'mysqld-binlog.000006';

Query OK, 0 rows affected (0.02 sec)

##查看删除情况：

ll mysqld-binlog.0\*

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000006

-rw-rw----. 1 mysql mysql 171 9月  22 15:13 mysqld-binlog.000007

-rw-rw----. 1 mysql mysql 120 9月  22 15:13 mysqld-binlog.000008

mysql> SHOW BINARY LOGS;

+----------------------+-----------+

| Log\_name                   | File\_size    |

+----------------------+-----------+

| mysqld-binlog.000006 |       171   |

| mysqld-binlog.000007 |       171   |

| mysqld-binlog.000008 |       120   |

+----------------------+-----------+

3 rows in set (0.00 sec)

### ****MySQL中日志相关常用的服务器变量说明如表十四所示：****

|  |  |
| --- | --- |
| 语法 | 说明 |
| expire\_logs\_days={0..99} | 设定二进制日志的过期天数，超出此天数的二进制日志文件将被自动删除。默认为0，表示不启用过期自动删除功能。如果启用此功能，自动删除工作通常发生在MySQL启动时或FLUSH日志时。作用范围为全局，可用于配置文件，属动态变量。 |
| general\_log={ON|OFF} | 设定是否启用查询日志，默认值为取决于在启动mysqld时是否使用了--general\_log选项。如若启用此项，其输出位置则由 --log\_output选项进行定义，如果log\_output的值设定为NONE，即使用启用查询日志，其也不会记录任何日志信息。作用范围为全局， 可用于配置文件，属动态变量。 |
| general\_log\_file=FILE\_NAME | 查询日志的日志文件名称，默认为“hostname.log"。作用范围为全局，可用于配置文件，属动态变量。 |
| binlog-format={ROW|STATEMENT|MIXED} | 指定二进制日志的类型，默认为STATEMENT，建议更改为MIXED。如果设定了二进制日志的格式，却没有启用二进制日志，则MySQL启动时会产生警告日志信息并记录于错误日志中。作用范围为全局或会话，可用于配置文件，且属于动态变量。 |
| log={YES|NO} | 是否启用记录所有语句的日志信息于一般查询日志(general query log)中，默认通常为OFF。MySQL 5.6已经弃用此选项。 |
| log-bin={YES|NO} | 是否启用二进制日志，如果为mysqld设定了--log-bin选项，则其值为ON，否则则为OFF。其仅用于显示是否启用了二进制日志，并不反应log-bin的设定值。作用范围为全局级别，属非动态变量。 |
| log\_bin\_trust\_function\_creators={TRUE|FALSE} | 此参数仅在启用二进制日志时有效，用于控制创建存储函数时如果会导致不安全的事件记录二进制日志条件下是否禁止创建存储函数。默认值为0，表示除非 用户除了CREATE ROUTING或ALTER ROUTINE权限外还有SUPER权限，否则将禁止创建或修改存储函数，同时，还要求在创建函数时必需为之使用DETERMINISTIC属性，再不然 就是附带READS SQL DATA或NO SQL属性。设置其值为1时则不启用这些限制。作用范围为全局级别，可用于配置文件，属动态变量。 |
| log\_error=/PATH/TO/ERROR\_LOG\_FILENAME | 定义错误日志文件。作用范围为全局或会话级别，可用于配置文件，属非动态变量。 |
| log\_output={TABLE|FILE|NONE} | 定义一般查询日志和慢查询日志的保存方式，可以是TABLE、FILE、NONE，也可以是TABLE及FILE的组合(用逗号隔开)，默认为 TABLE。如果组合中出现了NONE，那么其它设定都将失效，同时，无论是否启用日志功能，也不会记录任何相关的日志信息。作用范围为全局级别，可用于 配置文件，属动态变量。 |
| log\_query\_not\_using\_indexes={ON|OFF} | 设定是否将没有使用索引的查询操作记录到慢查询日志。作用范围为全局级别，可用于配置文件，属动态变量。 |
| log\_slave\_updates | 用于设定复制场景中的从服务器是否将从主服务器收到的更新操作记录进本机的二进制日志中。本参数设定的生效需要在从服务器上启用二进制日志功能。 |
| log\_slow\_queries={YES|NO} | 是否记录慢查询日志。慢查询是指查询的执行时间超出long\_query\_time参数所设定时长的事件。MySQL 5.6将此参数修改为了slow\_query\_log。作用范围为全局级别，可用于配置文件，属动态变量。 |
| log\_warnings=# | 设定是否将警告信息记录进错误日志。默认设定为1，表示启用；可以将其设置为0以禁用；而其值为大于1的数值时表示将新发起连接时产生的“失败的连接”和“拒绝访问”类的错误信息也记录进错误日志。 |
| long\_query\_time=# | 设定区别慢查询与一般查询的语句执行时间长度。这里的语句执行时长为实际的执行时间，而非在CPU上的执行时长，因此，负载较重的服务器上更容易产 生慢查询。其最小值为0，默认值为10，单位是秒钟。它也支持毫秒级的解析度。作用范围为全局或会话级别，可用于配置文件，属动态变量。 |
| max\_binlog\_cache\_size{4096 .. 18446744073709547520} | 二进定日志缓存空间大小，仅应用于事务缓存，其上限由max\_binlog\_stmt\_cache\_size决定。作用范围为全局级别，可用于配置文件，属动态变量。 |
| max\_binlog\_size={4096 .. 1073741824} | 设定二进制日志文件上限，单位为字节，最小值为4K，最大值为1G，默认为1G。某事务所产生的日志信息只能写入一个二进制日志文件，因此，实际上的二进制日志文件可能大于这个指定的上限。作用范围为全局级别，可用于配置文件，属动态变量。 |
| max\_relay\_log\_size={4096..1073741824} | 设定从服务器上中继日志的体积上限，到达此限度时其会自动进行中继日志滚动。此参数值为0时，mysqld将使用max\_binlog\_size参数同时为二进制日志和中继日志设定日志文件体积上限。作用范围为全局级别，可用于配置文件，属动态变量。 |
| innodb\_log\_buffer\_size={262144 .. 4294967295} | 设定InnoDB用于辅助完成日志文件写操作的日志缓冲区大小，单位是字节，默认为8MB。较大的事务可以借助于更大的日志缓冲区来避免在事务完成 之前将日志缓冲区的数据写入日志文件，以减少I/O操作进而提升系统性能。因此，在有着较大事务的应用场景中，建议为此变量设定一个更大的值。作用范围为 全局级别，可用于选项文件，属非动态变量。 |
| innodb\_log\_file\_size={108576 .. 4294967295} | 设定InnoDB用于辅助完成日志文件写操作的日志缓冲区大小，单位是字节，默认为8MB。较大的事务可以借助于更大的日志缓冲区来避免在事务完成 之前将日志缓冲区的数据写入日志文件，以减少I/O操作进而提升系统性能。因此，在有着较大事务的应用场景中，建议为此变量设定一个更大的值。作用范围为 全局级别，可用于选项文件，属非动态变量。 |
| innodb\_log\_files\_in\_group={2 .. 100} | 设定日志组中日志文件的个数。InnoDB以循环的方式使用这些日志文件。默认值为2。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| innodb\_log\_group\_home\_dir=/PATH/TO/DIR | 设定InnoDB重做日志文件的存储目录。在缺省使用InnoDB日志相关的所有变量时，其默认会在数据目录中创建两个大小为5MB的名为ib\_logfile0和ib\_logfile1的日志文件。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| innodb\_support\_xa={TRUE|FLASE} | 存储引擎事务在存储引擎内部被赋予了ACID属性，分布式(XA)事务是一种高层次的事务，它利用“准备”然后“提交”(prepare- then-commit)两段式的方式将ACID属性扩展到存储引擎外部，甚至是数据库外部。然而，“准备”阶段会导致额外的磁盘刷写操作。XA需要事务 协调员，它会通知所有的参与者准备提交事务(阶段1)。当协调员从所有参与者那里收到“就绪”信息时，它会指示所有参与者进行真正的“提交”操作。 此变量正是用于定义InnoDB是否支持两段式提交的分布式事务，默认为启用。事实上，所有启用了二进制日志的并支持多个线程同时向二进制日志写入数据的 MySQL服务器都需要启用分布式事务，否则，多个线程对二进制日志的写入操作可能会以与原始次序不同的方式完成，这将会在基于二进制日志的恢复操作中或 者是从服务器上创建出不同原始数据的结果。因此，除了仅有一个线程可以改变数据以外的其它应用场景都不应该禁用此功能。而在仅有一个线程可以修改数据的应 用中，禁用此功能是安全的并可以提升InnoDB表的性能。作用范围为全局和会话级别，可用于选项文件，属动态变量。 |
| relay\_log=file\_name | 设定中继日志的文件名称，默认为host\_name-relay-bin。也可以使用绝对路径，以指定非数据目录来存储中继日志。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| relay\_log\_index=file\_name | 设定中继日志的索引文件名，默认为为数据目录中的host\_name-relay-bin.index。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| relay-log-info-file=file\_name | 设定中继服务用于记录中继信息的文件，默认为数据目录中的relay-log.info。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| relay\_log\_purge={ON|OFF} | 设定对不再需要的中继日志是否自动进行清理。默认值为ON。作用范围为全局级别，可用于选项文件，属动态变量。 |
| relay\_log\_space\_limit=# | 设定用于存储所有中继日志文件的可用空间大小。默认为0，表示不限定。最大值取决于系统平台位数。作用范围为全局级别，可用于选项文件，属非动态变量。 |
| slow\_query\_log={ON|OFF} | 设定是否启用慢查询日志。0或OFF表示禁用，1或ON表示启用。日志信息的输出位置取决于log\_output变量的定义，如果其值为NONE，则即便slow\_query\_log为ON，也不会记录任何慢查询信息。作用范围为全局级别，可用于选项文件，属动态变量。 |
| slow\_query\_log\_file=/PATH/TO/SOMEFILE | 设定慢查询日志文件的名称。默认为hostname-slow.log，但可以通过--slow\_query\_log\_file选项修改。作用范围为全局级别，可用于选项文件，属动态变量。 |
| sql\_log\_bin={ON|OFF} | 用于控制二进制日志信息是否记录进日志文件。默认为ON，表示启用记录功能。用户可以在会话级别修改此变量的值，但其必须具有SUPER权限。作用范围为全局和会话级别，属动态变量。 |
| sql\_log\_off={ON|OFF} | 用于控制是否禁止将一般查询日志类信息记录进查询日志文件。默认为OFF，表示不禁止记录功能。用户可以在会话级别修改此变量的值，但其必须具有SUPER权限。作用范围为全局和会话级别，属动态变量。 |
| sync\_binlog=# | 设定多久同步一次二进制日志至磁盘文件中，0表示不同步，任何正数值都表示对二进制每多少次写操作之后同步一次。当autocommit的值为1时，每条语句的执行都会引起二进制日志同步，否则，每个事务的提交会引起二进制日志同步。 建议设置为1。 |

表十四：mysql日志相关说明

## 安全相关管理与配置

### 帐号与权限管理

**帐号与权限管理的原则及注意事项参考本文档：2.6数据库权限控制。**

#### 帐号管理

帐号相关：

* 帐号由username和host组成，其中username是大小写敏感的；
* 同一username在不同host上视作不同帐号，在使用中不指定host的情况下，视同username@'%'（%即任意host）；
* 创建帐号时必须附带密码，否则该帐号将可无密码登录；

##### 帐号操作

**创建帐号：**

##以下为创建finley帐号，其密码为some\_pass

##之所以在创建了finley@%之外，仍需创建finley@localhost，是为了避免存在''@localhost这样的匿名帐号的情况影响finley@localhost的登录

CREATE USER 'finley'@'localhost' IDENTIFIED BY 'some\_pass';

CREATE USER 'finley'@'%' IDENTIFIED BY 'some\_pass';

**删除帐号：**

DROP USER 'jeffrey'@'localhost';

**帐号锁定解锁：**

#锁定帐号：

alter user finley@'%' account lock;

#解锁帐号:

alter user finley@'%' account unlock;

##### 密码操作

**修改密码：**

alter user finley@'%' identified by 'New\_pass';

**密码过期：**

设置密码过期后，过期的帐号必须重新设置密码才可以继续操作，但由于MySQL目前并无记录历史密码的方式，所以用户可以简单地通过“重置”（即设置为原来同样的密码）来解除过期，意义不大，此处仅记录以供参考。

##设置某帐号为密码过期：

ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;

##设置全局密码过期天数：

SET GLOBAL default\_password\_lifetime = 180;

##或在配置文件中添加

default\_password\_lifetime=180

##针对特定帐号设置：

ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;

ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;

ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;

**密码策略设置：**

SHOW VARIABLES LIKE 'validate\_password%';

+--------------------------------------+--------+

| Variable\_name | Value |

+--------------------------------------+--------+

| validate\_password\_check\_user\_name | OFF |

| validate\_password\_dictionary\_file | |

| validate\_password\_length | 8 |

| validate\_password\_mixed\_case\_count | 1 |

| validate\_password\_number\_count | 1 |

| validate\_password\_policy | MEDIUM |

| validate\_password\_special\_char\_count | 1 |

+--------------------------------------+--------+

以上变量可通过在MySQL中使用SET GLOBAL xxx=yyy的形式设置（重启后失效），或在/etc/my.cnf中设定：

##在MySQL中设置全局变量形式修改策略：

##设置密码长度最小值：

set global validate\_password\_length=10;

##或将相关的变量写入在/etc/my.cnf的[mysqld]下

**变量解释如表十五所示：**

|  |  |  |
| --- | --- | --- |
| 变量名 | 变量类型及范围 | 解释 |
| validate\_password\_check\_user\_name | ON/OFF，默认为OFF | 验证密码与帐号是否相同 |
| validate\_password\_dictionary\_file | 文件路径，默认为空 | 弱密码字典文件 |
| validate\_password\_length | int，默认为8 | 密码最小长度 |
| validate\_password\_mixed\_case\_count | int，默认为1 | 密码中最少须含多少个小写字母与大写字母 假如设置为1，则须同时有1个小写与1个大写字母 |
| validate\_password\_number\_count | int，默认为1 | 密码中最少须含数字多少 |
| validate\_password\_policy | 0/1/2/LOW/MEDIUM/HIGH，默认为MEDIUM | 密码策略检查级别： 0或LOW,仅检查密码长度； 1或 MEDIUM，检查包括：长度、大小写字母、数字及特殊字符 2或HIGH，检查包括：MEDIUM全部检查项+字典 |
| validate\_password\_special\_char\_count | int，默认为1 | 密码中最少须含特殊字符多少 |

表十五：MYSQL相关变量解释

#### 权限管理

对某表拥有create及drop操作权限，则同时拥有对该表所在database的create及drop操作

##### 权限操作

**GRANT操作：**

使用GRANT ALL：

##将crm\_db数据库所有权限授予crm@%:

grant all on crm\_db.\* to crm@'%';

##将crm\_db.table\_1表的所有权限授予someone@'%'：

grant all on crm\_db.table\_1 to someone@'%';

使用WITH GRANT OPTION:

grant select on some\_db.some\_table to user\_a@'%' WITH GRANT OPTION;

以上授权语句将赋予user\_a在某表上的select权限，且user\_a可将该表的select权限授予其他帐号（或revoke其他帐号的select权限）。

**flush privileges**

在执行授权操作后，需要使用flush privileges语句来使权限立即生效。

flush privileges;

**REVOKE操作：**

revoke操作与grant操作相似，仅需将grant ... to ...改为revoke ... from ...

revoke select on some\_db.some\_table from user\_a@'%';

**查询权限：**

查询某用户的权限：

show grants for user\_a@'%';

## 备份与恢复

主要备份方式有以下三种：

|  |  |  |
| --- | --- | --- |
| 方式 | 实时度 | 适用场景 |
| 主从复制 | 高 | 可靠性要求高，恢复过程简单快捷 |
| mysqldump+binlog | 低 | 在线增量备份,可实现全备后任意时间点的恢复 |
| 数据文件备份+binlog | 低 | 离线增量备份,可实现全备后任意时间点的恢复 |

* 主从复制的备份方式见本文档3.4章节
* 后两种方式均为全量备份为基础，结合binlog实现增量备份，区别在于使用mysqldump或数据文件级别的全量备份
* mysqldump方式备份可在线执行，但恢复数据时耗时较长
* 数据文件备份方式需要关闭数据库后对数据文件进行备份，但恢复时耗时较短
* 一般生产环境建议使用mysqldump+binlog形式
* binlog与datafile应存放在不同的磁盘上，binlog磁盘应采取必要的冗余措施（如RAID）

### mysqldump+binlog备份与恢复

**创建全量备份**

mysqldump --single-transaction --flush-logs \

--master-data=2 --all-databases > backup\_full.sql

* --single-transaction表示该导出操作视为一个事务，为保证数据一致性，尽量使用该参数；
* --flush-logs会在导出前执行一次binlog切换，方便后续的增量备份；
* --master-data=2表示将在sql文件中记录当前binlog内部的entry位置，留作参考；
* --all-databases表示将执行全库备份。

**增量备份**

在创建全量备份之时起，只需定期将数据库binlog文件进行磁带或远程备份，即可实现增量备份的需要。

从全量备份backup\_full.sql中可以找到以下信息：

-- Position to start replication or point-in-time recovery from

-- CHANGE MASTER TO MASTER\_LOG\_FILE='gbichot2-bin.000007',MASTER\_LOG\_POS=4;

该信息由全备时的--master-data=2参数生成，表示全备执行之后的所有增量操作记录在名为gbichot2-bin.000007及其后的binlog中。

* 以上述示例为准，则全量备份文件为backup\_full.sql，增量备份文件为gbichot2-bin.000007及其后的所有binlog

**恢复操作**

从备份中获取：

全量备份文件backup\_full.sql

gbichot2-bin.000007及其后的所有binlog

传送到恢复目标机器，在恢复目标库中首先恢复全量备份：

mysql < backup\_full.sql

将增量的binlog应用到数据库中：

##将binlog处理为sql文件：

mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 ... > tmpfile.sql

##若是由于人为误操作（如drop database等）造成故障而需要恢复的，可以将tmpfile.sql文件中的破坏操作删除或做相应处理

##执行tmpfile.sql

mysql < tmpfile.sql

**任意时间点恢复操作**

任意时间点恢复的关键为mysqlbinlog处理binlog时增加时间点参数，使用--stop-datetime='2017-12-25 11:25:56'形式即可恢复到指定时间点

mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 --stop-datetime='2017-12-25 11:25:56'

| mysql

### 数据文件+binlog备份与恢复

此备份恢复方式与上一节中的**增量操作**一致，以下仅演示数据文件备份与恢复的**全量操作**部分。

执行mysql的clean shutdown

show variables like 'innodb\_fast\_shutdown'

SET GLOBAL innodb\_fast\_shutdown=0;

shutdown;

备份datadir目录（若日志和数据文件或其他相关文件存放在datadir之外的目录，则应同时备份这些目录）

datadir=/var/lib/mysql

cd $datadir

tar cvf /tmp/mysql\_datadir.tar \*

备份/etc/my.cnf

**执行全量恢复**

在另一台机器上安装mysql，将mysql\_datadir.tar解压到数据文件目录中，使用备份的my.cnf启动数据库即可

### 数据导入导出

**设置导出目录相关权限：**

##首先对于mysql，应在设置文件/etc/my.cnf中将secure\_file\_priv变量设置为空，以解除对于导出导入目录的限制

##在/etc/my.cnf中添加：

[mysqld]

secure\_file\_priv = ""

**使用mysqldump导出数据：**

mysqldump db1 table1 > db1.table1.sql

生成的sql文件可用于直接在另一个mysql数据库中执行以完成导入：

mysql < db1.table1.sql

**使用mysqldump --tab将建表语句与数据分开导出：**

##导出db1数据库中的table1

mysqldump --tab=/tmp --fields-terminated-by=',' \

--fields-optionally-enclosed-by='"' db1 table1

##mysqldump格式为

Usage: mysqldump [OPTIONS] database [tables]

OR mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]

OR mysqldump [OPTIONS] --all-databases [OPTIONS]

* 其中--tab指定导出目录
* --fields-terminated-by 指定分隔符
* --fields-optionally-enclosed-by 指定非数字型字段使用什么符号包含
* mysqldump --tab导出的文件分为两种，针对每张表（假设为table1）,会生成table1.sql和table1.txt两个文件，分别为建表语句与数据

**导入数据mysqldump --tab导出的数据**

导入数据使用mysqlimport工具，使用与导出时一致的选项

##先执行建表语句:

mysql < table1.sql

##导入数据：

mysqlimport --fields-terminated-by=',' \

--fields-optionally-enclosed-by='"' table1.txt

## 锁管理

### 表锁

查询表级锁争用情况

mysql> show status like 'table%';

+----------------------------+---------+

| Variable\_name | Value |

+----------------------------+---------+

| Table\_locks\_immediate | 100 |

| Table\_locks\_waited | 11 |

+----------------------------+---------+

这里有两个状态变量记录MySQL内部表级锁定的情况，两个变量说明如下：

* Table\_locks\_immediate：产生表级锁定的次数；
* Table\_locks\_waited：出现表级锁定争用而发生等待的次数；

两个状态值都是从系统启动后开始记录，出现一次对应的事件则数量加1。如果这里的Table\_locks\_waited状态值比较高，那么说明系统中表级锁定争用现象比较严重，就需要进一步分析为什么会有较多的锁定资源争用了。

**并发插入**

MyISAM存储引擎有一个[系统](https://www.2cto.com/os/" \t "https://www.2cto.com/database/201712/_blank)变量concurrent\_insert，专门用以控制其并发插入的行为，其值分别可以为0、1或2。

|  |  |
| --- | --- |
| 值 | 操作说明 |
| 0 | 不允许并发插入。 |
| 1 | 如果MyISAM表中没有空洞（即表的中间没有被删除的行），MyISAM允许在一个进程读表的同时，另一个进程从表尾插入记录。这也是MySQL的默认设置。 |
| 2 | 无论MyISAM表中有没有空洞，都允许在表尾并发插入记录。 |

### 行锁

获取InonoD行锁争用情况

mysql> show status like 'innodb\_row\_lock%';

+-------------------------------+-------+

| Variable\_name | Value |

+-------------------------------+-------+

| Innodb\_row\_lock\_current\_waits | 0 |

| Innodb\_row\_lock\_time | 0 |

| Innodb\_row\_lock\_time\_avg | 0 |

| Innodb\_row\_lock\_time\_max | 0 |

| Innodb\_row\_lock\_waits | 0 |

+-------------------------------+-------+

rows in set (0.00 sec)

如果发现争用比较严重，如Innodb\_row\_lock\_waits和Innodb\_row\_lock\_time\_avg的值比较高，还可以通过设置InnoDB Monitors来进一步观察发生锁冲突的表、数据行等，并分析锁争用的原因。

用户可以通过设计和SQL调整等措施减少锁冲突和死锁，包括：

* 尽量使用较低的隔离级别
* 精心设计索引，并尽量使用索引访问数据，使加锁更精确，从而减少锁冲突的机会。
* 选择合理的事务大小，小事务发生锁冲突的几率也更小。
* 给记录集显示加锁时，最好一次性请求足够级别的锁。比如要修改数据的话，最好直接申请排他锁，而不是先申请共享锁，修改时再请求排他锁，这样容易产生死锁。
* 不同的程序访问一组表时，应尽量约定以相同的顺序访问各表，对一个表而言，尽可能以固定的顺序存取表中的行。这样可以大减少死锁的机会。
* 尽量用相等条件访问数据，这样可以避免间隙锁对并发插入的影响。
* 不要申请超过实际需要的锁级别；除非必须，查询时不要显示加锁。
* 对于一些特定的事务，可以使用表锁来提高处理速度或减少死锁的可能。

### 死锁

发生死锁后，InnoDB一般都能自动检测到，并使一个事务释放锁并退回，另一个事务获得锁，继续完成事务。但在涉及外部锁，或涉及锁的情况下，InnoDB并不能完全自动检测到死锁，这需要通过设置锁等待超时参数innodb\_lock\_wait\_timeout来解决。需要说明的是，这个参数并不是只用来解决死锁问题，在并发访问比较高的情况下，如果大量事务因无法立即获取所需的锁而挂起，会占用大量计算机资源，造成严重性能问题，甚至拖垮数据库。我们通过设置合适的锁等待超时阈值，可以避免这种情况发生。

通常来说，死锁都是应用设计的问题，通过调整业务流程、数据库对象设计、事务大小、以及访问数据库的SQL语句，绝大部分都可以避免。

下面就通过实例来介绍几种死锁的常用方法：

* 在应用中，如果不同的程序会并发存取多个表，应尽量约定以相同的顺序为访问表，这样可以大大降低产生死锁的机会。如果两个session访问两个表的顺序不同，发生死锁的机会就非常高！但如果以相同的顺序来访问，死锁就可能避免。
* 在程序以批量方式处理数据的时候，如果事先对数据排序，保证每个线程按固定的顺序来处理记录，也可以大大降低死锁的可能。
* 在事务中，如果要更新记录，应该直接申请足够级别的锁，即排他锁，而不应该先申请共享锁，更新时再申请排他锁，甚至死锁。
* 在REPEATEABLE-READ隔离级别下，如果两个线程同时对相同条件记录用SELECT...ROR UPDATE加排他锁，在没有符合该记录情况下，两个线程都会加锁成功。程序发现记录尚不存在，就试图插入一条新记录，如果两个线程都这么做，就会出现死锁。这种情况下，将隔离级别改成READ COMMITTED，就可以避免问题。
* 当隔离级别为READ COMMITED时，如果两个线程都先执行SELECT...FOR UPDATE，判断是否存在符合条件的记录，如果没有，就插入记录。此时，只有一个线程能插入成功，另一个线程会出现锁等待，当第１个线程提交后，第２个线程会因主键重出错，但虽然这个线程出错了，却会获得一个排他锁！这时如果有第３个线程又来申请排他锁，也会出现死锁。对于这种情况，可以直接做插入操作，然后再捕获主键重异常，或者在遇到主键重错误时，总是执行ROLLBACK释放获得的排他锁。

## 性能优化

### 性能优化

#### 公共参数优化

Mysql数据库公共参数优化如表十六所示：

|  |  |
| --- | --- |
| 参数 | 说明 |
| max\_connections | 同时处理最大连接数，推荐设置最大连接数是上限连接数的80%左右 |
| sort\_buffer\_size | 查询排序时缓冲区大小，只对order by和group by起作用，可增大此值为16M |
| open\_files\_limit | 打开文件数限制，如果show global status like 'open\_files'查看的值等于或者大于open\_files\_limit值时，程序会无法连接数据库或卡死 |

表十六：mysql公共参数优化

#### MyiSAM引擎优化

MyiSAM引擎参数主要需要注意几个选项，如表十七所示：

|  |  |  |
| --- | --- | --- |
| 参数 | 含义 | 备注 |
| key\_buffer\_size = 16M | 索引缓存区大小 | 一般设置物理内存的30-40% |
| read\_buffer\_size = 128K | 读操作缓冲区大小 | 推荐设置16M或32M |
| query\_cache\_type = ON | 打开查询缓存功能 |  |
| query\_cache\_limit = 1M | 查询缓存限制 | 只有1M以下查询结果才会被缓存，以免结果数据较大把缓存池覆盖 |
| query\_cache\_size = 16M | 查看缓冲区大小 | 用于缓存SELECT查询结果，下一次有同样SELECT查询将直接从缓存池返回结果，可适当成倍增加此值 |

表十七：MyiSAM引擎优化参数

#### InnoDB引擎优化

MySQL 5.7 提供了更加合适的默认值，一般情况下只要调整如表十八几个选项就可以了，其余参数根据实际情况再进行配置。

把innodb相关的参数显示出来，接下来我们对关键参数进行优化。

mysql> show variables like "%innodb%";

|  |  |  |
| --- | --- | --- |
| 参数 | 含义 | 备注 |
| innodb\_status\_file | 周期性输出状态文件 | 默认值即可 |
| innodb\_buffer\_pool\_size  对于32G内存数据库服务器建议：innodb\_buffer\_pool\_size=8G | 数据及索引的缓冲区大小 | 这个是Innodb最重要的参数，主要作用是缓存innodb表的索引，数据，插入数据时的缓冲，默认值为128M。  如果是一个专用DB服务器，那么它可以占到内存的70%-80%。  并不是设置的越大越好。设置的过大，会导致system的swap空间被占用，导致操作系统变慢，从而减低sql查询的效率。  如果你的数据比较小，那么可分配是你的数据大小+10%左右做为这个参数的值。例如：数据大小为50M,那么给这个值分配innodb\_buffer\_pool\_size＝64M就够了  设置方法：在my.cnf文件里：  innodb\_buffer\_pool\_size=4G |
| innodb\_additional  \_mem\_pool\_size | 内存池耗尽后每次新分配内存大小 | 日志中会出现告警信息，频繁出现说明内存配置不当 |
| innodb\_log\_file\_size  对于32G内存数据库服务器建议：  innodb\_log\_file\_size=256M | 每个日志文件的大小 | 这个参数指定在一个日志组中，每个log的大小。innodb的logfile就是事务日志，用来在mysql crash后的恢复.所以设置合理的大小对于mysql的性能非常重要，直接影响数据库的写入速度，事务大小，异常重启后的恢复。在mysql5.5以前innodb的logfile最大设置为4GB,在5.6以后的版本中logfile最大的可以设为512GB。一般取256M可以兼顾性能和recovery的速度。  设置方法：在my.cnf文件里：  innodb\_log\_file\_size=256M |
| innodb\_log\_files  \_in\_group | 日志组中日志文件的数目 | 默认值即可 |
| innodb\_log\_buffer\_size | 事务在内存中的缓冲，也就是日志缓冲区的大小， | 默认设置即可，具有大量事务的可以考虑设置为16M。 |
| innodb\_flush\_log\_at\_trx\_commit | 控制事务的提交方式,也就是控制log的刷新到磁盘的方式。 | 这个参数只有3个值（0，1，2）.默认为1，性能更高的可以设置为0或是2，这样可以适当的减少磁盘IO（但会丢失一秒钟的事务。），游戏库的MySQL建议设置为0。主库请不要更改了。  其中：  0：log buffer中的数据将以每秒一次的频率写入到log file中，且同时会进行文件系统到磁盘的同步操作，但是每个事务的commit并不会触发任何log buffer 到log file的刷新或者文件系统到磁盘的刷新操作；  1：（默认为1）在每次事务提交的时候将logbuffer 中的数据都会写入到log file，同时也会触发文件系统到磁盘的同步；  2：事务提交会触发log buffer 到log file的刷新，但并不会触发磁盘文件系统到磁盘的同步。此外，每秒会有一次文件系统到磁盘同步操作。  说明：  这个参数的设置对Ｉｎｎｏｄｂ的性能有很大的影响，所以在这里给多说明一下。  当这个值为1时：innodb 的事务LOG在每次提交后写入日值文件，并对日值做刷新到磁盘。这个可以做到不丢任何一个事务。  当这个值为2时：在每个提交，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新,在对日志文件的刷新在值为2的情况也每秒发生一次。但需要注意的是，由于进程调用方面的问题，并不能保证每秒１００％的发生。从而在性能上是最快的。但操作系统崩溃或掉电才会删除最后一秒的事务。  当这个值为0时：日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新，但是在一个事务提交不做任何操作。mysqld进程的崩溃会删除崩溃前最后一秒的事务。  从以上分析，当这个值不为１时，可以取得较好的性能，但遇到异常会有损失，所以需要根据自已的情况去衡量。 |
| innodb\_flush\_method | 这个参数控制着innodb数据文件及redo log的打开、刷写模式。 | 有三个值：fdatasync(默认)，O\_DSYNC，O\_DIRECT  默认是fdatasync，调用fsync()去刷数据文件与redo log的buffer  为O\_DSYNC时，innodb会使用O\_SYNC方式打开和刷写redo log,使用fsync()刷写数据文件  为O\_DIRECT时，innodb使用O\_DIRECT打开数据文件，使用fsync()刷写数据文件跟redo log。  在类unix操作系统中，文件的打开方式为O\_DIRECT会最小化缓冲对io的影响，该文件的io是直接在用户空间的buffer上操作的，并且io操作是同步的，因此不管是read()系统调用还是write()系统调用，数据都保证是从磁盘上读取的 |
| innodb\_buffer\_pool\_instances = 1 | 缓冲池实例个数 | 推荐设置4个或8个 |
| innodb\_file\_per\_table = OFF | 默认是共享表空间 | 共享表空间idbdata文件不断增大，影响一定的I/O性能。推荐开启独立表空间模式，每个表的索引和数据都存在自己独立的表空间中，可以实现单表在不同数据库中移动。 |

表十八：Innodb引擎优化参数

#### MySQL Server优化

在内存足够大的情况下，可调低交换空间使用概率甚至完全禁用交换空间。因为部分系统在内存未耗尽的情况下也可能使用交换空间，进而导致性能问题。

##在内存大于等于4G的情况下，可完全禁用页面交换：

sysctl vm.swappiness=0

echo "vm.swappiness=0" >> /etc/sysctl.conf

* 使用磁盘条带化技术来提高IO性能：根据数据的重要性及数据库层高可用方案的不同，可对重要性低或高可用方案完备的数据库使用RAID 0级别的条带化，对要求高可靠的数据库使用RAID 0+1级别的条带化，对写频繁的数据库建议不使用RAID N(如RAID 5/6等需要做奇偶校验的方式)条带化。
* 数据文件与日志文件可存放在不同磁盘来提高IO性能。
* 存放数据文件与日志文件的文件系统可使用“noatime”选项来挂载，因为记录数据文件的访问时间其实毫无意义。

##假设数据文件存放在/data文件系统

##则将/etc/fstab中/data所在行中添加“noatime”选项：

/dev/sda2 /data ext4 defaults,noatime 0 0

##fstab文件修改后重新挂载文件系统：

mount -a

# Mysql数据库开发规范

## 5.1实例设计

* 应规定数据库名称命名方式，库名不应过长(如32字节内)。
* 应当首先考虑分表策略而非分库策略。
* 创建数据库时显式指定字符集。

## 5.2表设计

* 应规定表名及列名命名方式，表名及列名不应过长。
* 表名及列名中不能包含保留字符和特殊字符。
* 表名推荐包含业务系统及功能标识。
* 建表时显式指定字符集。
* 建表时显示指定存储引擎，无特殊目的推荐使用InnoDB。
* 生产环境使用非InnoDB、MyISAM、Memory存储引擎时须由DBA审核通过。
* Memory引擎不与其他引擎共享同一个实例。
* 不推荐使用enum、set、blob、text等类型。
* 表自增列建议使用bigint。
* 表示状态等少值类型的字段(如性别)推荐使用tinyint或者smallint。
* 业务活跃的大表中应当包含创建时间和修改时间。
* 表中所有字段都尽量使用NOT NULL属性，由业务规定默认值。

## 5.3索引设计

* InnoDB所有表都必须包含主键，且使用单列做为主键。
* 使用自增序列作为主键可以极大提高性能。
* 主键不应当被更新。可以删除。
* 使用B-tree类型索引。生产环境使用其他类型索引时应当由DBA审核通过。
* 索引键长度不应超过64KB。
* 单个表内的索引数量不应过多。
* 少值类型字段不应当使用B-tree索引。
* 索引覆盖可以极大提高性能。

## 5.4数据库连接

* 应用程序需要有连接超时、空闲超时和失败重连机制。失败重连需要有一定时间间隔。
* 系统应当具备高可用和负载均衡机制。
* 应用程序错误日志中应当包含对应数据库地址以及原始错误码，用于故障诊断。
* 使用连接池的应用程序必须配置初始、最大、最小连接数以及超时时间和连接回收策略。
* 应用程序使用的客户端及中间件需要与服务端兼容。

## 5.5SQL开发规范

* SELECT语句显式指定字段形成，不使用\*。
* SELECT语句使用UNION ALL代替UNION。子句控制在5个内。
* 确定只需要一条记录是使用LIMIT 1。
* INSERT语句显式指定字段名称及值。不使用values。
* 关联查询时推荐使用别名，引用时使用“别名.字段名”方式。
* 生产系统中的关联查询控制在三个以内。
* 生产系统中不推荐使用外关联。
* 使用数据量较小的表作为关联查询中的驱动表。
* 使用顺序关联的方式改写嵌套子查询。
* DML类型语句中禁止使用关联查询。
* 不使用force类功能。

## 5.6事务使用规范

* 事务操作行数控制在2000以内。
* WHERE IN语句参数控制在2000以内。
* 对有auto\_increment属性的表进行写入操作时，并发控制在200内。
* SQL级别、事务级别、存储引擎在主从库上要一致。
* 对于事务不敏感的查询放到从库上进行。
* 前段程序禁止使用set类型语句。

## 5.7其他

* 更新删除语句需要包含WHERE语句。
* 生产库中的大表禁止表扫描。
* 生产库中数据dump等操作推荐使用从库。
* WHERE语句中LIKE参数不使用全模糊匹配，或者至少包含其他可使用索引的字段。
* 不在索引列上使用函数查找。
* 尽量不使用存储过程及触发器。
* 生产库上执行的操作的中间结果及最终结果不要超过1MB。
* 应用中不对系统库进行写操作。
* 生产库中不进行实时汇总等统计计算。

# 常见问题汇总

## 6.1 mysql DNS反解

**现象：**

错误日志有类似警告：

120119 16:26:04 [Warning] IP address '192.168.1.10' could not be resolved: Name or service not known

120119 16:26:04 [Warning] IP address '192.168.1.14' could not be resolved: Name or service not known

120119 16:26:04 [Warning] IP address '192.168.1.17' could not be resolved: Name or service not known

通过show processlist发现大量类似如下的连接：

|592|unauthenticated user|192.168.1.10:35320|NULL|Connect| |login|NULL|

|593|unauthenticated user|192.168.1.14:35321|NULL|Connect| |login|NULL|

|594|unauthenticated user|192.168.1.17:35322|NULL|Connect| |login|NULL|

**解决办法：**

修改配置文件添加并需要重启：

[mysqld]

skip-name-resolve

#skip-name-resolve 参数的作用：不再进行反解析(ip不反解成域名)，这样可以加快数据库的反应时间。

## 6.2错误日志

**现象：**

错误日志如下提示：

Error: Can’t create a new thread (errno 12)

数据库服务器问题，数据库操作无法创建新线程。一般是有以下3个方面的原因：

* MySQL 线程开得太多。
* 服务器系统内存溢出。
* 环境软件损坏或系统损坏。

**解决办法：**

* 进入 phpmyadmin 的 mysql 数据库中的 user 表，对数据库的用户进行编辑，修改 max\_connections 的值。适当的改小一点。
* 联系服务器管理员检查服务器的内存和系统是否正常，如果服务器内存紧张，请检查一下哪些进程消耗了服务器的内存，同时考虑是否增加服务器的内存来提高整个系统的负载能力。
* mysql版本更改为稳定版本
* 优化网站程序的sql等等

## 6.3操作报错

**现象：**

操作提示报错：

ERROR 1010 (HY000): Error dropping database

mysql> drop database xjtrace;

ERROR 1010 (HY000): Error dropping database (can't rmdir './xjtrace/

**解决办法：**

在做数据库删除时出现这种提示，其原因是在database下面含有自己放进去的文件，譬如.txt文件或.sql文件等，只要进去把这个文件删了在执行。

## 6.4导出数据很快，导入到新库时却很慢

**现象**

MySQL导出的SQL语句在导入时有可能会非常非常慢，经历过导入仅400万条记录，竟用了近2个小时。在导出时合理使用几个参数，可以大大加快导入的速度。

* -e 使用包括几个VALUES列表的多行语法;
* –max\_allowed\_packet=XXX 客户端/服务器之间通信的缓存区的最大大小;
* –net\_buffer\_length=XXX TCP/IP和套接字通信缓冲区大小,创建长度达net\_buffer\_length的行

**注意：**

max\_allowed\_packet和net\_buffer\_length不能比目标数据库的配置数值大，否则可能出错。

首先确定目标库的参数值

mysql> show variables like 'max\_allowed\_packet';

+--------------------+---------+

| Variable\_name | Value |

+--------------------+---------+

| max\_allowed\_packet | 1048576 |

+--------------------+---------+

1 row in set (0.00 sec)

mysql> show variables like 'net\_buffer\_length';

+-------------------+-------+

| Variable\_name | Value |

+-------------------+-------+

| net\_buffer\_length | 16384 |

+-------------------+-------+

1 row in set (0.00 sec)

**解决办法：**

根据参数值书写mysqldump命令，如：

mysql>mysqldump -uroot -p 数据库名 -e –max\_allowed\_packet=1048576 -net\_buffer\_length=16384 > SQL文件

**例如：**

mysql> mysqldump -uroot -p limm -e --max\_allowed\_packet=1048576 --net\_buffer\_length=16384 >xjtrace\_data\_`date +%F`.sql

#之前2小时才能导入的sql现在几十秒就可以完成了。