**数据库**

**概述**

数据库（DB , DataBase）:依照某种数据模型进行组织并存放到存储器的数据集合。

数据库管理系统（DBMS , DataBase Management System）:用来操纵和管理数据库的大型服务软件。

数据库系统（DBS , DataBase System）:即 DB+DBMS ,指带有数据库并整合了数据库管理软件的计算机系统。

主流数据库服务软件：

NoSQL:Redis Mongodb

RDBMS:Oracle（跨平台） MySQL(跨平台) MariaDB(开源) DB2(IBM) SQL SERVER(微软) Sybase（Sybase） PostgreSQL（加州大学伯克利分校）

**MySQL 数据库**

特点：

– 适用于中小规模、关系型数据库系统

– 支持 Linux/Unix 、 Windows 等多种操作系统

– 使用 C 和 C++ 编写,可移植性强

– 通过 API 支持 Python/Java/Perl/PHP 等语言

典型应用环境：

– LAMP 平台,与 Apache HTTP Server 组合

– LNMP 平台,与 Nginx 组合





数据存储流程:

连接数据库服务器（命令行/API/图形工具）--建库--建表--插入记录--断开连接

操作指令类型:

- MySQL指令:环境切换、看状态、退出等控制

- SQL指令:数据库定义 / 查询 / 操纵 / 授权语句

DDL数据定义语言 (create alter drop)

DML 数据操作语言 ( insert update delete )

DCL 数据控制语言 (grant revoke)

DTL 数据事物语言 (commit rollback savepoint )

MySQL 基本操作：

数据库的命名规则：

– 可以使用数字 / 字母 / 下划线,但不能纯数字

– 区分大小写,具有唯一性

– 不可使用指令关键字、特殊字符

MySQL数据类型：

数值类型：



整数型:

– 使用 UNSIGNED 修饰时,对应的字段只保存正数

– 数值不够指定宽度时,在左边填空格补位

– 使用关键字 ZEROFILL 时,填 0 代替空格补位

>create table t2(id int(3) unsigned zerofill);

>insert into t2 values(2);

>select \* from t2;

id

002

浮点型:

– 定义格式: float( 总宽度 , 小数位数 )

– 当字段值与类型不匹配时,字段值作为 0 处理

– 数值超出范围时,仅保存最大 / 最小值

>create table t3(pay float(5,2) );

字符类型：

• 定长：char(字符数) – 最大长度255字符,不够指定字符数时在右边用空格补齐

• 变长：varchar(字符数) – 按数据实际大小分配存储空间

• 大文本类型：text/blob– 字符数大于65535存储时使用

日期时间类型:



时间函数：

无需库、表，可直接调用，使用SELECT指令输出函数结果



枚举类型：

单选，ENUM – 格式：enum(值1, 值2, 值N)

多选，SET – 格式：set(值1, 值2, 值N)

约束条件

Null 允许为空，默认设置 ； NOT NULL 不允许为空 ；Key 索引类型 ； Default 设置默认值，缺省为NULL。

修改表结构

用法：>ALTER TABLE 表名 执行动作（Add添加字段；Modify修改字段类型；Change修改字段名；Drop删除字段；Rename 修改表名）

修改字段类型：

>ALTER TABLE 表名 modify 字段名 类型(宽度) 约束条件; 可加AFTER 字段名; 或者 FIRST;

修改字段名（当新字段名后跟新类型和约束条件时，也可修改字段类型）：

>ALTER TABLE 表名 change 源字段名 新字段名 类型(宽度) 约束条件；  
删除字段：

>ALTER TABLE 表名 drop 字段名;

修改表名：

>ALTER TABLE 表名 Rename 新表名；

MySQL键值

索引：对记录集的多个字段进行排序的方法。

索引结构:Btree、B+tree 、hash、fulltext

hash索引：采用一定的哈希算法，把键值换算成新的哈希值，检索时只需一次哈希算法即可立刻定位到相应的位置，速度非常快。适用于键值唯一的情况，不适用于范围查询，不支持利用索引排序及多表联合查询的左匹配规则。

fulltext全文索引：目前只有MyISAM引擎支持。其可以在CREATE TABLE ，ALTER TABLE ，CREATE INDEX 使用，不过目前只有 CHAR、VARCHAR ，TEXT 列上可以创建全文索引。值得一提的是，在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用CREATE INDEX创建FULLTEXT索引，要比先为一张表建立FULLTEXT然后再将数据写入的速度快很多。

B-Tree 索引：MySQL数据库中使用最为频繁的索引类型，除了 Archive 存储引擎之外的其他所有的存储引擎都支持 B-Tree 索引。mysql中innodb和myisam引擎中的B-tree索引使用的是B+tree（即每一个叶子节点都包含指向下一个叶子节点的指针，从而方便叶子节点的范围遍历，并且除叶子节点外其他节点只存储键值和指针）。

索引优点：

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；

可以加快数据的检索速度。

索引缺点：

当对表的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度；

索引需要占物理空间。

键值类型：

INDEX：普通索引 字段的KEY标志是MUL

UNIQUE：唯一索引

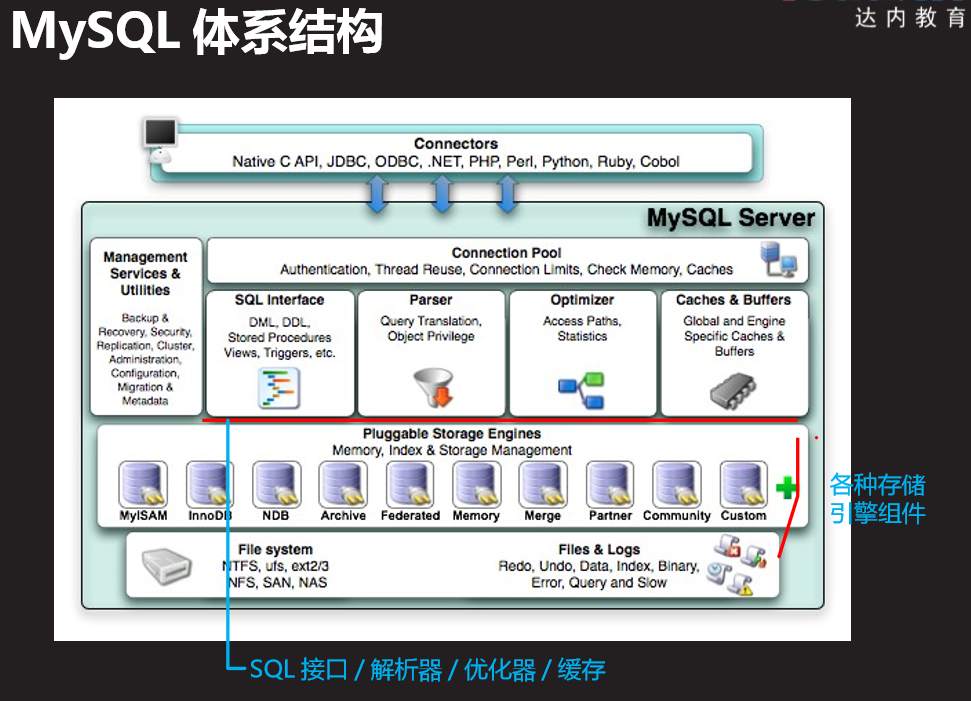
FULLTEXT：全文索引

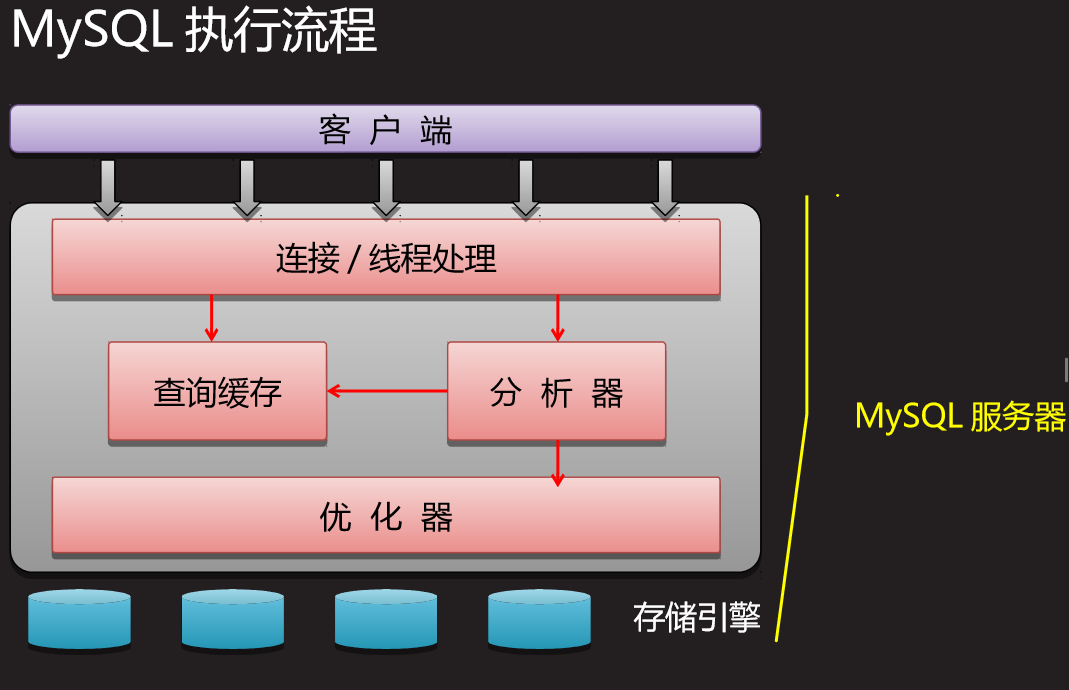
PRIMARY KEY:主键 一个表中只能有一个primary key字段, 对应的字段值不允许有重复，不允许赋NULL值，若多个字段作为PRIMARY KEY，称为复合主键，必须一起创建。字段的KEY标志是PRI ；常与 AUTO\_INCREMENT连用 ；常把表中能够唯一标识记录的字段设置为主键字段 [记录编号字段]。

FOREIGN KEY:外键 表的存储引擎必须是innodb, 字段类型要一致, 被参照字段必须要是索引类型的一种(primary key)。

数据库管理

MySQL工作原理：





1. connectors 与其他编程语言中的sql 语句进行交互，如php、java等。

2. Management Serveices & Utilities（系统管理和控制工具）

3. Connection Pool (连接池) 管理缓冲用户连接，线程处理等需要缓存的需求。

4. SQL Interface (SQL接口) 接受用户的SQL命令，并且返回用户需要查询的结果。比如select from就是调用SQL Interface。

5. Parser（解析器） SQL命令传递到解析器的时候会被解析器验证和解析。

主要功能：

a . 将SQL语句分解成数据结构，并将这个结构传递到后续步骤，后面SQL语句的传递和处理就是基于这个结构。

b. 在分解构成中遇到错误，说明这个sql语句不合理，语句不会继续执行。

6. Optimizer (查询优化器) SQL语句在查询之前会使用查询优化器对查询进行优化(产生多种执行计划,最终数据库会选择最优化的方案去执行,尽快返会结果) 他使用的是“选取-投影-联接”策略进行查询。

用一个例子就可以理解： select uid,name from user where gender = 1;

这个select 查询先根据where 语句进行选取，而不是先将表全部查询出来以后再进行gender过滤

这个select查询先根据uid和name进行属性投影，而不是将属性全部取出以后再进行过滤

将这两个查询条件联接起来生成最终查询结果.

7. Cache和Buffer (查询缓存) 如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数据。这个缓存机制是由一系列小缓存组成的。比如表缓存，记录缓存，key缓存，权限缓存等。

8.Engine (存储引擎) 存储引擎是MySql中具体的与文件打交道的子系统。也是Mysql最具有特色的一个地方。Mysql的存储引擎是插件式的。它根据MySql AB公司提供的文件访问层的一个抽象接口来定制一种文件访问机制（这种访问机制就叫存储引擎）。

SQL 语句执行过程：

数据库通常不会被直接使用，而是由其他编程语言通过SQL语句调用mysql，由mysql处理并返回执行结果。那么Mysql接受到SQL语句后，又是如何处理的呢？

首先程序的请求会通过mysql的connectors与其进行交互，请求到处后，会暂时存放在连接池（connection pool)中并由处理器（Management Serveices & Utilities）管理。当该请求从等待队列进入到处理队列，管理器会将该请求丢给SQL接口（SQL Interface）。SQL接口接收到请求后，它会将请求进行hash处理并与缓存中的结果进行对比，如果完全匹配则通过缓存直接返回处理结果；否则，需要完整的走一趟流程：

(1)由SQL接口丢给后面的解释器（Parser），上面已经说到，解释器会判断SQL语句正确与否，若正确则将其转化为数据结构。

(2)解释器处理完，便来到后面的优化器（Optimizer），它会产生多种执行计划,最终数据库会选择最优化的方案去执行,尽快返会结果。

(3)确定最优执行计划后，SQL语句此时便可以交由存储引擎（Engine）处理，存储引擎将会到后端的存储设备中取得相应的数据，并原路返回给程序。

注意：

(1)如何缓存查询数据？

存储引擎处理完数据，并将其返回给程序的同时，它还会将一份数据保留在缓存中，以便更快速的处理下一次相同的请求。具体情况是，mysql会将查询的语句、执行结果等进行hash，并保留在cache中，等待下次查询。

(2)buffer与cache的区别？

从上面的图可以看到，缓存那里实际上有buffer和cache两个，那它们之间是否有什么不同呢？简单的说就是，buffer是写缓存，cache是读缓存。

(3)如何判断缓存中是否已缓存需要的数据

在第一次查询后，mysql便将查询语句以及查询结果进行hash处理并保留在缓存中，SQL查询到达之后，对其进行同样的hash处理后，将两个hash值进行对照，如果一样，则命中，从缓存中返回查询结果；否则，需要整个流程走一遍。

MySQL存储引擎

作为可插拔式的组件提供：

MySQL服务软件自带的功能程序，处理表的处理器

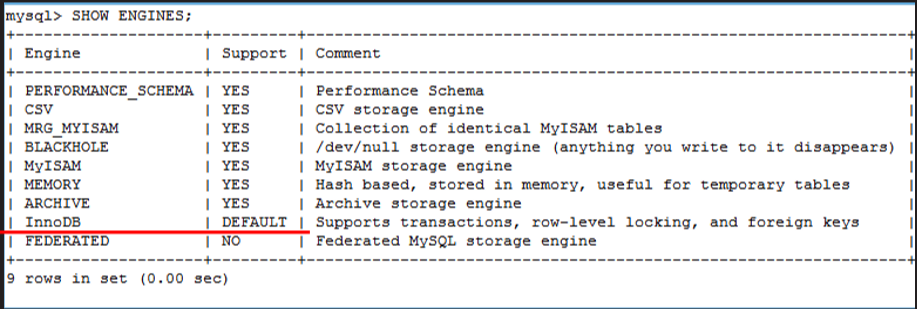
不同的存储引擎有不同的功能和数据存储方式

默认的存储引擎：

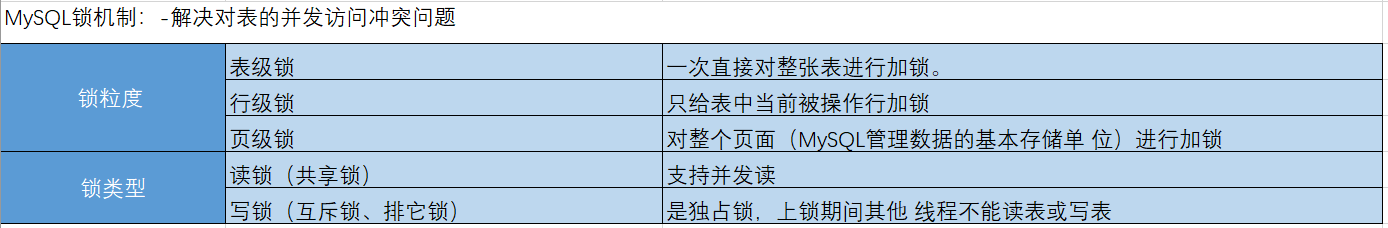
MySQL 5.0/5.1 ---> MyISAM

MySQL 5.5/5.6 ---> InnoDB

列出可用的存储引擎类型： >SHOW ENGINES; 或 >SHOW ENGINES\G;







事务Transaction

事务定义：对数据库服务的访问过程（连接数据库服务器 操作数据 断开连接）；一个最小的不可再分的工作单元；通常一个事务对应一个完整的业务,一个完整的业务需要批量的DML(insert、update、delete)语句共同联合完成,事务只和DML语句有关，或者说DML语句才有事务。

事务开启标志：任何一条DML语句(insert、update、delete)执行，标志事务的开启

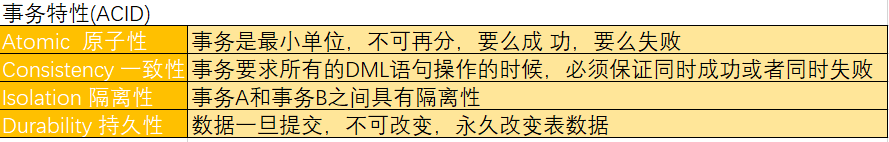
事务结束标志：

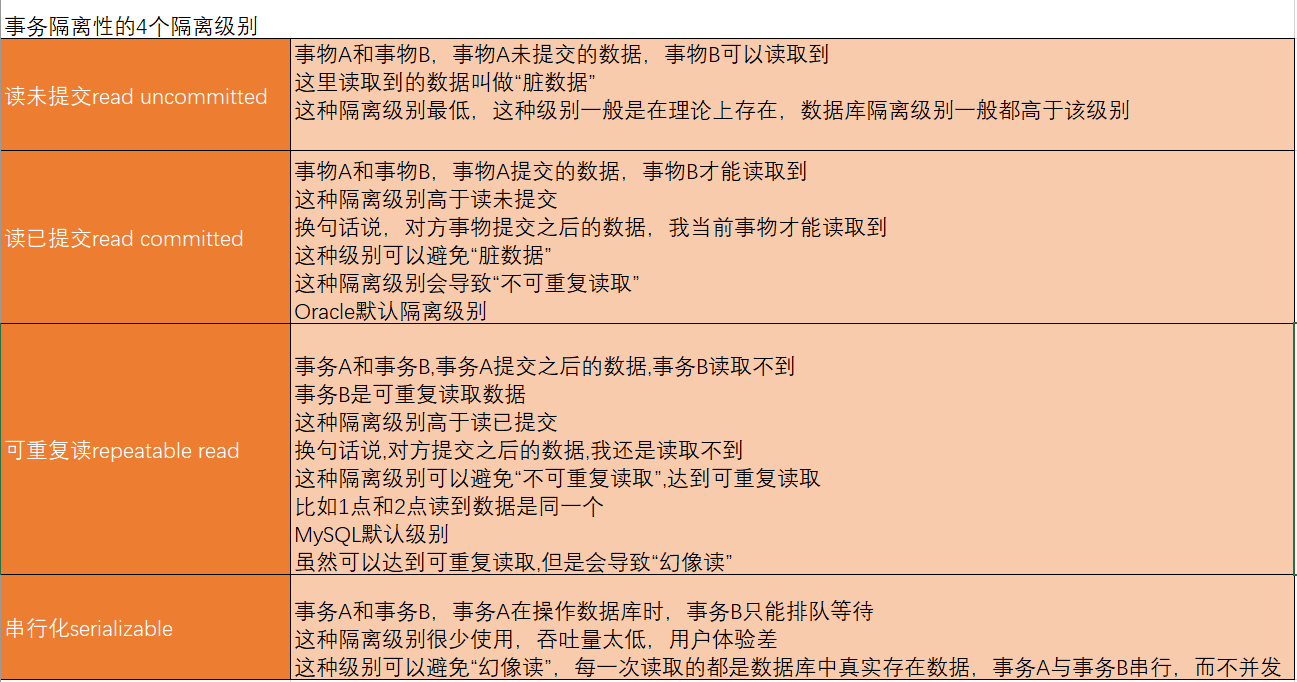
提交：成功的结束，将所有的DML语句操作历史记录和底层硬盘数据来一次同步

回滚：失败的结束，将所有的DML语句操作历史记录全部清空

事物与数据库底层数据：

在事物进行过程中，未结束之前，DML语句是不会更改底层数据，只是将历史操作记录一下，在内存中完成记录。只有在事物结束的时候，而且是成功的结束的时候，才会修改底层硬盘文件中的数据。

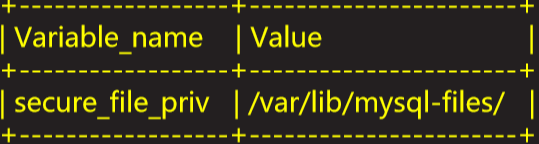




数据导入导出

设置搜索路径：

> show variables like "secure\_file\_priv"; //查看默认使用目录及目录是否存在



# mkdir /myload ; chown mysql /myload

# vim /etc/my.cnf //修改搜索路径

[mysqld]

secure\_file\_priv="/myload"

数据导入：

定义：把系统文件的内容存储到数据库服务器的表里。

语法：LOAD DATA INFILE “目录名/文件名” INTO TABLE 表名 FIELDS TERMINATED BY “分隔符” LINES TERMINATED BY “\n”;

eg. >create table userdb.user(name char(50),password char(1), uid int(2),gid int(2), comment varchar(100),homedir char(60),shell char(50),index(name));

> load data infile "/myload/user.txt" into table userdb.user fields terminated by ":" lines terminated by "\n";

> alter table userdb.user add id int(2) zerofill primary key auto\_increment first;

> select id ,name,uid from userdb.user limit 3;

注意事项：

– 字段分隔符要与文件内的一致

– 指定导入文件的绝对路径

– 导入数据的表字段类型要与文件字段匹配

– 禁用SElinux

* 导出的目录所属者要为mysql

数据导出:

语法：SELECT查询.. .. INTO OUTFILE “目录名/文件名” FIELDS TERMINATED BY “分隔符” LINES TERMINATED BY “\n”;

eg. > select \* from userdb.user where uid<100 into outfile "/ myload/user2.txt";

注意事项：

– 导出的内容由SQL查询语句决定

– 禁用SElinux

* 导入的目录所属者要为mysql，并要写的权限

管理表记录（增加，更新，删除）

增加表记录：

给所有字段赋值 – INSERT INTO 表名 VALUES (字段1值，.. ..，字段N值)， (字段1值，.. ..，字段N值)， (字段1值，.. ..，字段N值)， .. .. ;

给指定字段赋值 – INSERT INTO 表名 (字段1,.. ..,字段N) VALUES (字段1值，字段2值，字段N值)，(字段1值，字段2值，字段N值)， .. .. ;

注意事项

– 字段值要与字段类型相匹配

– 对于字符类型的字段，要用双或单引号括起来

– 依次给所有字段赋值时，字段名可以省略

– 只给一部分字段赋值时，必须明确写出对应的字段名称

查询表记录：

格式1 – SELECT 字段1, .. .., 字段N FROM 表名;

格式2 – SELECT 字段1, .. .., 字段N FROM 表名 WHERE 条件表达式;

注意事项

– 使用 \* 可匹配所有字段

– 指定表名时，可采用 库名.表名 的形式

更新表记录：

更新表内的所有记录 – UPDATE 表名 SET 字段1=字段1值, 字段2=字段2值,

字段N=字段N值；

只更新符合条件的部分记录 – UPDATE 表名 SET 字段1=字段1值, 字段2=字段2值, 字段N=字段N值; WHERE 条件表达式;

注意事项

– 字段值要与字段类型相匹配

– 对于字符类型的字段，要用双或单引号括起来

– 若不使用WHERE限定条件，会更新所有记录

– 限定条件时，只更新匹配条件的记录

删除表记录：

删除所有的表记录 – DELETE FROM 表名;

仅删除符合条件的记录 – DELETE FROM 表名 WHERE 条件表达式;

匹配条件

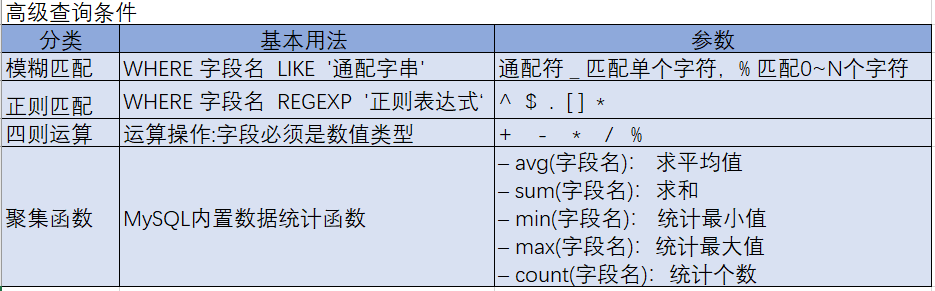


>select username,uid,gid from usertab where uid is null and gid is null;

> update usertab set uid=3000,gid=3000 where username="lucy";

>select id from usertab where name="yaya" and uid is not null;

>update usertab set username=null where id=2;



模糊匹配：

>select username from usertab where username like '\_ \_ \_ \_';

>select username from usertab where username like 'a\_ \_t';

>insert into usertab(username)values("a");

>select username from usertab where username like 'a%';

>select username from usertab where username like '\_%\_';

正则匹配：

>select username from usertab where username regexp '[0-9]';

>select username from usertab where username regexp '^[0-9]';

>select username from usertab where username regexp '[0-9]$';

>select username from usertab where username regexp 'a.\*t';

>select username from usertab where username regexp '^a.\*t$';

>select username,uid from usertab where uid regexp '..';

>select username,uid from usertab where uid regexp '^..$';

四则运算：

>update usertab set uid=uid+1 where id <=10;

>select username ,uid,gid, uid+gid as zh from usertab where username="mysql";

>select username,uid,gid, uid+gid as zh,(uid+gid)/2 as pjz from usertab

where username="mysql";

> select username , age , 2018-age s\_year from usertab where username="root";

聚集函数：

>select max(uid) from usertab;

>select sum(uid) from usertab;

>select min(uid) from usertab;

>select avg(uid) from usertab;

>select count(id) from usertab;

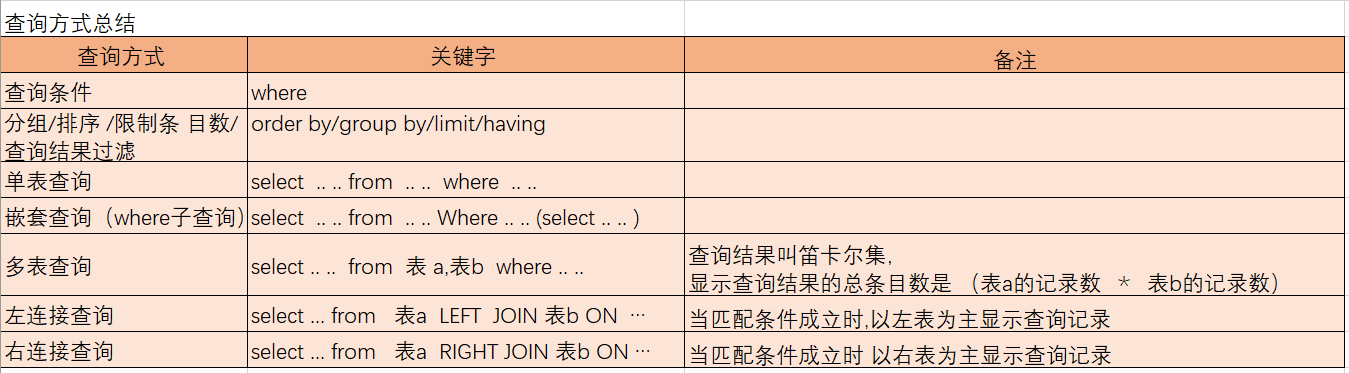
>select count(username) from usertab where shell="/bin/bash";

去重显示：

>select distinct shell from usertab;

>select distinct shell from usertab where uid >10 and uid<=100;





复制表

• 将源表a复制为新表b – CREATE TABLE b SELECT \* FROM a;

• 将指定的查询结果复制为新表c – CREATE TABLE c SQL查询语句;

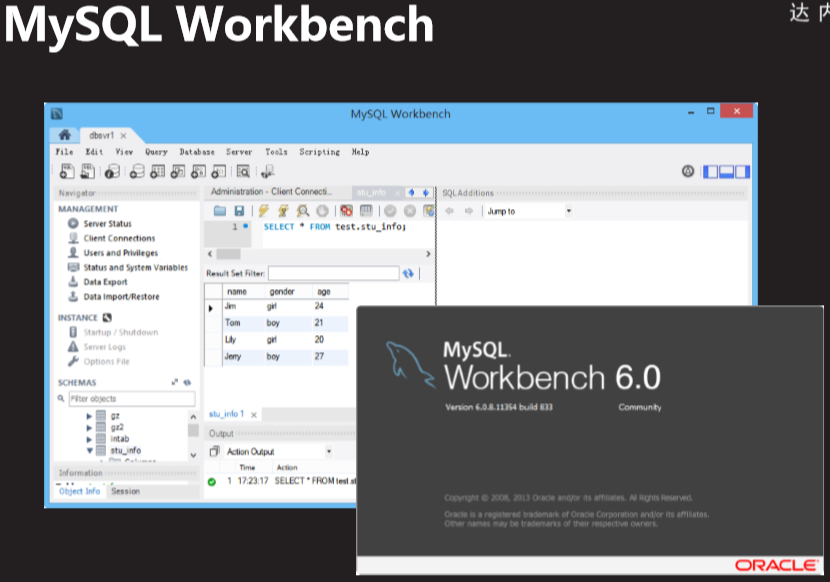
• 复制源表d的结构到新表e – CREATE TABLE e SELECT \* FROM d WHERE FALSE;

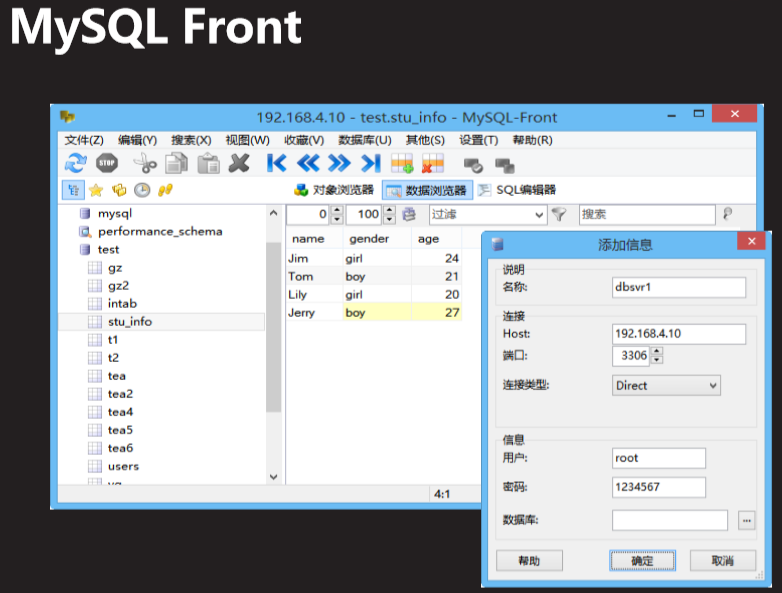
• 将源表f的名称改为g – ALTER TABLE f RENAME TO g;

注意：复制的内容由sql查询命令决定，不会复制源表字段的键值给新表。

MySQL管理方式









PhpMyAdmin

• 基本思路

1. 安装httpd、mysql、php-mysql及相关包

2. 启动httpd服务程序

3. 解压phpMyAdmin包，部署到网站目录

4. 配置config.inc.php，指定MySQL主机地址

5. 创建授权用户

6. 浏览器访问、登录使用

#yum -y install httpd php php-mysql

#tar -zxf phpMyAdmin-2.11.11-all-languages.tar.gz -C /var/www/html/

#cd /var/www/html/

#mv phpMyAdmin-2.11.11-all-languages/ phpmyadmin

#chown -R apache:apache phpmyadmin/

#cp phpmyadmin/config.sample.inc.php phpmyadmin/config.inc.php

# sed -n '17p;31p' /var/www/html/phpmyadmin/config.inc.php

$cfg['blowfish\_secret'] = 'plj123';

$cfg['Servers'][$i]['host'] = 'localhost';

>create database bbsdb;grant all on bbsdb.\* to admin@”localhos t” identified by “123456”;

#systemctl start httpd ; firefox <http://localhost/phpmyadmin>

用户授权及权限撤销

密码恢复及设置

恢复MySQL管理密码:

• 密码忘了怎么办？

1. 停止MySQL服务程序

2. 跳过授权表启动MySQL服务程序 skip-grant-tables ---->写入/etc/my.cnf

3. 重设root密码（更新user表记录）

4. 以正常方式重启MySQL服务程序

# vim /etc/my.cnf

[mysqld]

......

skip-grant-tables

# systemctl restart mysqld

# mysql

> update mysql.user set authentication\_string=password("888888") where user="root"

and host="localhost";

> flush privileges;

# vim /etc/my.cnf

[mysqld]

......

#skip-grant-tables //此处#表示注释

# systemctl restart mysqld

# mysql -uroot -p888888

设置管理员密码：

• 查看模块、修改配置文件、重启服务

#find / -name "validate\_password.so" /usr/lib64/mysql/plugin/debug/validate\_password.so /usr/lib64/mysql/plugin/validate\_password.so

# vim /etc/my.cnf

[mysqld]

plugin-load=validate\_password.so //加载模块

validate-password=FORCE\_PLUS\_PERMANENT //永久启用模块 validate\_password\_policy=0 //只验证密码长度

validate\_password\_length=6 //指定密码长度

# systemctl restart mysqld

• 在Shell命令行修改登陆密码

需要验证旧密码，不适用于跳过授权表启动的情况

# mysqladmin -hlocalhost -uroot -p password "新密码"

Enter password: //输入旧密码

创建授权

MySQL授权库和表信息

• 授权库 mysql ，主要的几个表

– user表，存储授权用户的访问权限

– db表，存储授权用户对数据库的访问权限

– tables\_priv表，存储授权用户对表的访问权限

– columns\_priv表，存储授权用户对字段的访问权限

GRANT 配置授权

• 语法 – GRANT 权限列表.. .. ON 库名.表名 TO 用户名@'客户端地址' IDENTIFIED BY '密码' [ WITH GRANT OPTION ];

• 注意事项

– 当库名.表名 为 \*.\* 时，匹配所有库所有表

– 授权设置存放在 mysql 库的 user 表

• 权限列表

– all：匹配所有权限

– SELECT,UPDATE,INSERT .. ..

– SELECT,UPDATE (字段1, .. .. , 字段N)

• 客户端地址

– %：匹配所有主机

– 192.168.1.%：匹配指定的一个网段

– 192.168.1.1：匹配指定IP地址的单个主机

– %.tarena.com：匹配一个DNS区域

– svr1.tarena.com：匹配指定域名的单个主机

查看用户授权

• 用户查看自己的权限 – SHOW GRANTS;

• 管理员可查看其他用户的权限 – SHOW GRANTS FOR 用户名@'客户端地址';

重设用户密码

• 授权用户修改自己的密码 – SET PASSWORD=PASSWORD('新密码');

• 管理员可重设其他用户的密码 – SET PASSWORD FOR 用户名@'客户端地址'=PASSWORD('新密码');

撤销授权

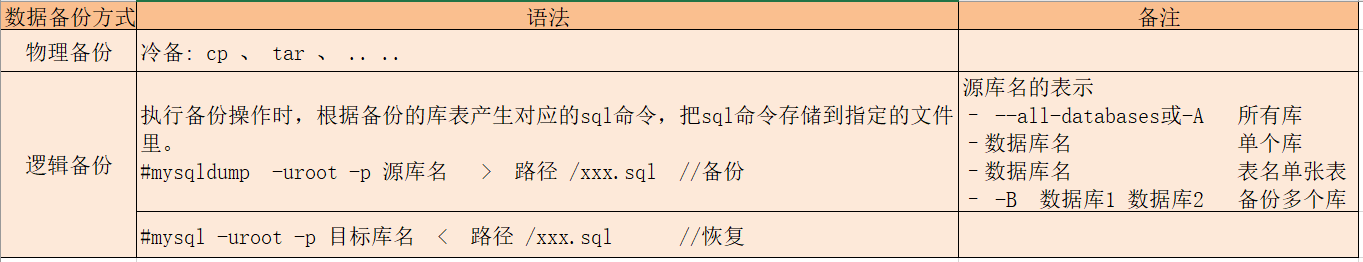
撤销用户权限

• 基本用法

– REVOKE 权限列表 ON 库名.表名

– FROM用户名@'客户端地址;

数据库备份



备份的类型

•热备份：读写不受影响（MyISAM不支持热备，InnoDB支持热备）

•温备份：仅可以执行读操作

•冷备份：离线备份，读写操作均中止

数据备份策略:

• 完全备份 – 备份所有数据(一台服务器、一个库、一张表)

• 增量备份 – 备份自上一次备份(包含完全备份、差异备份、增量备份)之后有变化的数据

• 差异备份 – 备份自上一次完全备份之后有变化的数据

完全备份的缺点

•数据量大时，备份和恢复数据都受磁盘I/O

•备份和恢复数据会给表加写锁

•使用完全备份文件恢复数据，只能把数据恢复到备份时的状态。完全备份后新写入的数据无法恢复

备份的目标：

•数据库数据，每个表空间单独存放

•二进制日志，需要和数据分开存储

• InnoDB的事务日志

•存储过程、存储函数、触发器或事件调度器等

•服务器的配置文件：/etc/my.cnf



生产环境下对数据做备份的方法：

•crond 执行备份脚本（shell python go ruby perl ）

•搭建MySQL主从结构实现数据的自动备份

使用脚本对数据做备份要考虑的问题：

•备份策略： 完全+差异 完全+增量

•备份时间：数据库服务访问量少时

•备份频率：根据数据产生量，决定备份频率，一般周多

•备份文件的命名：库名-日期.sql 日期\_xx.sql

•备份文件的存储设置（存储空间）：准备独立的存储设备存储备份文件，lv + raid

•如何执行备份：使用周期性计划任务执行本机脚本

备份方案

1. cp + tar == 物理冷备

将数据目录打包压缩备份，需要停服务，不推荐

1. lvm快照 + binlog == 几乎物理热备 + 增量备份

备份：需要将数据库目录存放到lvm逻辑卷上

3、mysqldump + InnoDB + binlog= 完全逻辑热备 + 增量备份

4、Xtrabackup + InnoDB == 完全热备 + 增量备份

5、使用Xtrabackup实现单表备份

实时增量备份 / 恢复



MySQL 主从同步

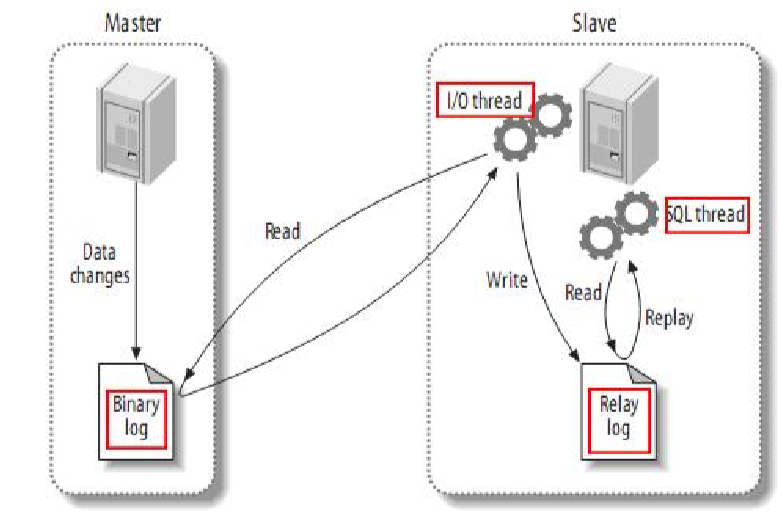
主从同步原理

• Master启用 binlog 日志，记录数据更改操作

• Slave 运行2个线程

– Slave\_IO：复制master主机 binlog日志文件里的S QL到本机的relay-log文件里。

– Slave\_SQL：执行本机relay-log文件里的SQL语句 ，重现Master的数据操作



Slave\_IO 线程没有运行

– 报错：Slave\_IO\_Running:

•原因分析

– 连接不上 master数据库服务器

• 解决办法

– 检查物理连接（ping）、检查授权用户

– 检查是否有防火墙规则 （service iptables stop）

– 关闭SElinux (setenforce 0)

– 或是binlog日志文件指定错误（日志名或pos节点）

– stop slave ; change master to 选项=值 ; start slave ;

Slave\_SQL 线程没有运行 – 报错：Slave\_SQL\_Running: No

• 原因分析

– 执行本机中继日志里的sql命令时，sql命令使用的库/表或记录在本机不存在。

• 解决办法

– 1、stop slave;

– 2、创建或恢复需要用到的库或表

– 3、start slave;

构建主从同步思路：

1. 确保数据相同 – 从库必须要有主库上的数据（生产中在线热备；对采用MyISAM的库，可离线备份）

2. 配置主服务器 – 启用binlog日志及设置格式，设置server\_id, 授权用 户

3. 配置从服务器 – 设置server\_id，指定主数据库服务器信息

4. 测试配置 – 客户端连接主库，写入的数据，在连接从库的时候也能够访问到。

MySQL主从同步结构模式：

• 基本应用 – 单向复制：主 --> 从

• 扩展应用

– 链式复制：主 --> 从 --> 从

– 双向复制：主 <--> 从

– 放射式复制：从 <-- 主 --> 从

从

MySQL主从同步复制模式：

• 异步复制（Asynchronous replication）

– 主库在执行完客户端提交的事务后会立即将结果返给 客户端，并不关心从库是否已经接收并处理。

• 全同步复制（Fully synchronous replication）

– 当主库执行完一个事务，所有的从库都执行了该事务 才返回给客户端。

• 半同步复制（Semisynchronous replication）

– 介于异步复制和全同步复制之间，主库在执行完客户 端提交的事务后不是立刻返回给客户端，而是等待至 少一个从库接收到并写到relay log中才返回给客户端。

配置永久半同步 /etc/my.cnf 的[mysqld]下方

主： plugin-load=rpl\_semi\_sync\_master=semisync\_master.so rpl\_semi\_sync\_master\_enabled=1

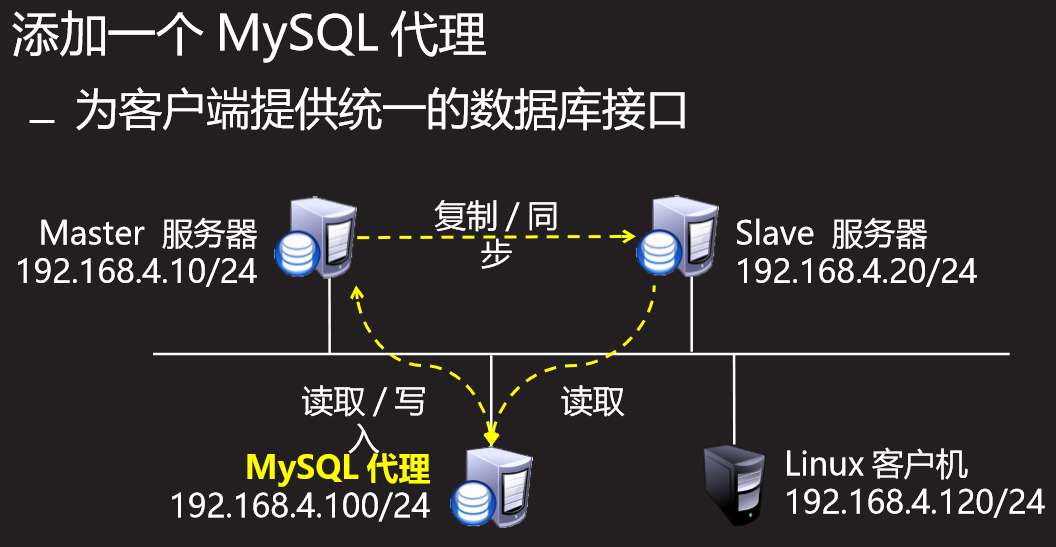
从： plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so rpl\_semi\_sync\_slave\_enabled=1

在有的高可用架构下，master和slave需同时启动，以便在切换后能继续使用半同步复制 plugin-load = "rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semis ync\_slave.so“

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enabled = 1

**MySQL读写分离**



读写分离的原理

• 多台MySQL服务器

– 分别提供读、写服务，均衡流量

– 通过主从复制保持数据一致性

• 由MySQL代理面向客户端

– 收到SQL写请求时，交给服务器A处理

– 收到SQL读请求时，交给服务器B处理

– 具体区分策略由服务设置

构建读写分离的思路

1. 已搭建好MySQL主从复制 – 其中Slave为只读
2. 添加一台MySQL代理服务器 – 部署/启用 maxscale或mycat
3. 客户端通过代理主机访问MySQL数据库 – 访问代理服务器

Maxscale配置：/etc/maxscale.cnf

[server1] //定义数据库服务器主机名

type=server

address=192.168.4.10 //master主机ip地址

port=3306

protocol=MySQLBackend

[server2] //定义数据库服务器

type=server

address=192.168.4.20 //slave主机ip地址

port=3306

protocol=MySQLBackend

[MySQL Monitor] //定义要监视的数据库服务器

type=monitor

module=mysqlmon

servers=server1,server2 //定义的主、从数据库服务器主机名

user=scalemon //用户名

passwd=111111 //密码

monitor\_interval=10000

[Read-Write Service] //定义实现读写分离的数据库服务器

type=service

router=readwritesplit

servers=server1,server2 //定义的主、从数据库服务器主机名

user=maxscale //用户名

passwd=111111 //密码

max\_slave\_connections=100%

**MySQL 多实例**

• 多实例：在一台物理主机上运行多个数据库服务

• 多实例优点： – 节约运维成本 – 提高硬件利用率

• 配置多实例步骤

– 安装支持多实例服务的软件包 mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz（源码）

– 修改主配置文件

– 根据配置文件做相应设置

– 初始化授权库

– 启动服务

– 客户端访问

主配置文件/etc/my.cnf

– 每个实例要有独立的数据库目录和监听端口号

– 每个实例要有独立的实例名称和独立的sock文件

[mysqld\_multi] //启用多实例

mysqld = /usr/local/mysql/bin/mysqld\_safe //指定进程文件的路径

mysqladmin = /usr/local/mysql/bin/mysqladmin //指定管理命令路径

user = root //指定调用进程的用户

[mysqlX] //实例进程名称 ,X表示实例名称,如 [mysql2]

port = 3307 //端口号

datadir = /data3307 //数据库目录 ，要手动创建

socket =/data3307/mysql.sock //指定sock文件的路径和名称

pid-file = /data3307/mysqld.pid //进程pid号文件位置

log-error = /data3307/mysqld.err //错误日志位置

**MySQL 性能调优**

结合MySQL工作原理

• 提高MySQL系统的性能、响应速度

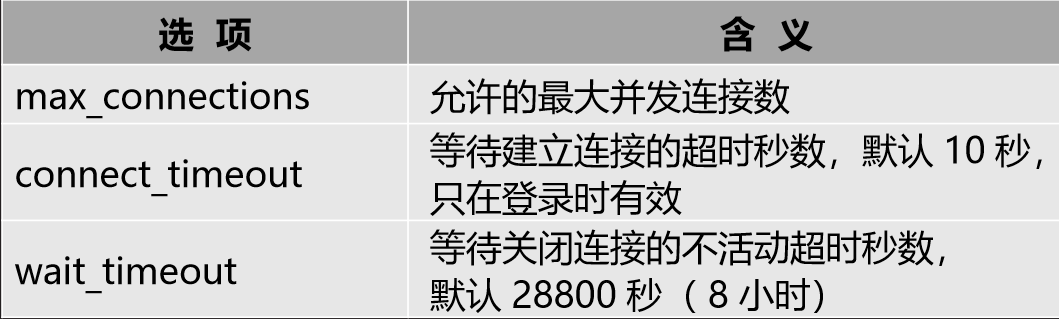
– 替换有问题的硬件（CPU/磁盘/内存等）

– 服务程序的运行参数调整

– 对SQL查询进行优化

并发及连接控制

• 连接数、连接超时



• 当前已使用的连接数/默认的最大连接数 理想比率 <= 85%

缓存参数控制

• 缓冲区、线程数量、开表数量



• key\_buffer\_size=8M，当 Key\_reads / Key\_read\_requests 较低时，可适当加大此缓存值

• sort\_buffer\_size=256K – 增大此值可提高ORDER和GROUP的速度

• 查看表记录读取缓存read\_buffer\_size – 此缓存值影响SQL查询的响应速度

• 查看可重用线程数 >show variables like “thread\_cache\_size”;

查看当前的线程重用状态>show global status like “threads\_%”;

Cached connected created running

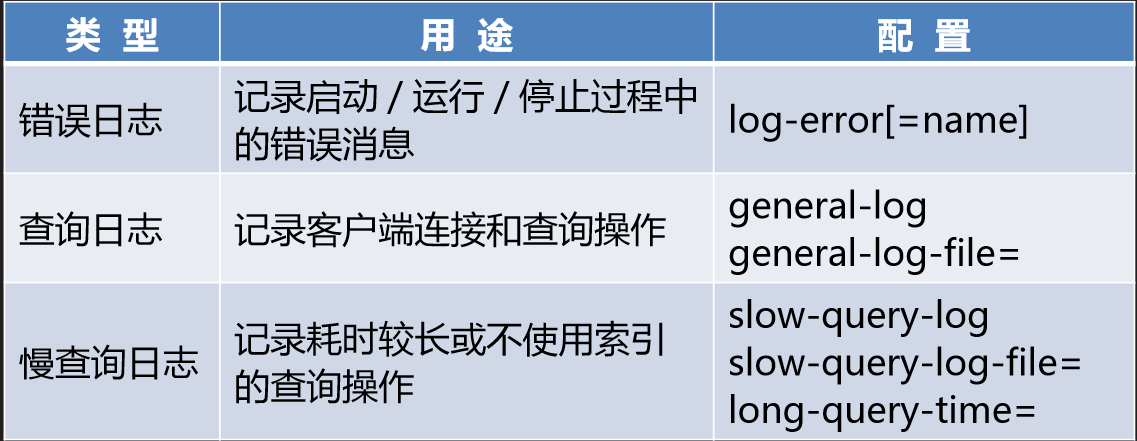
• 查看已打开、打开过多少个表 >show global status like “open%tables”;

查看可缓存多少个打开的表 >show variables like “table\_open\_cache”;

理想比率 <= 95%

SQL查询优化

MySQL日志类型



优化SQL查询-记录慢查询

]# vim /etc/my.cnf

[mysqld]

.. ..

slow\_query\_log=1 //启用慢查询

slow\_query\_log\_file=mysql-slow.log //指定慢查询日志文件

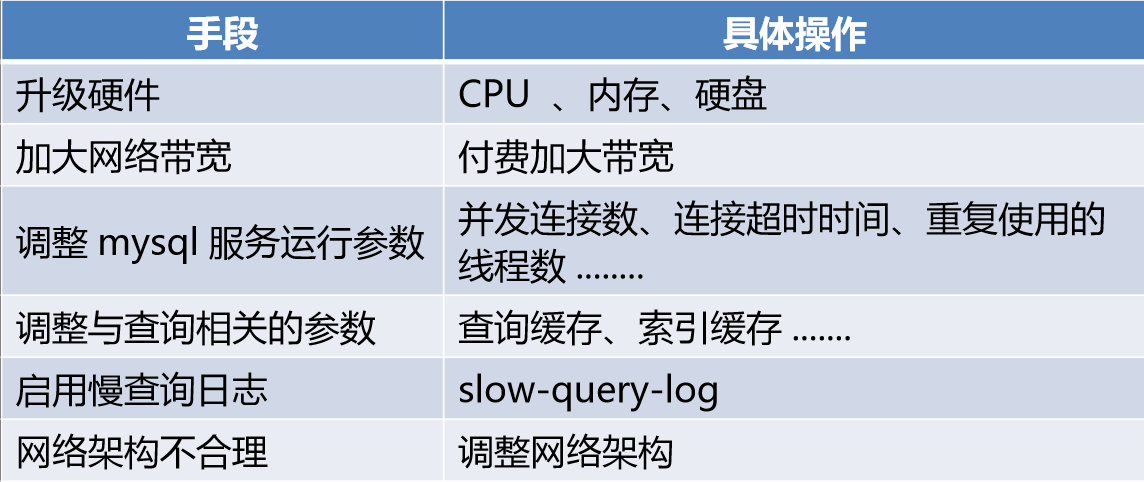
long\_query\_time=5 //超过指定秒数（默认10 秒）的查询才被记录

log\_queries\_not\_using\_indexes=1 //记录未使用索引的查询

关于查询的缓存

• 查看当前的查询缓存统计>show global status like “qcache%”;

调优思路总结



MHA（Master High Availability）

架构的介绍：在mysql故障切换过程中，能达到0-30秒之内自动完成数据库的切换操作，并且在故障切换过程中，能够最大程度的保障数据的一致性，从而实现真正意义的高可用。

目前MHA支持一主多从的架构，要求一个复制的集群至少有三台服务器，一台是master一台是备用master另外一台是slave。

目前淘宝针对一个复制集群3台服务器进行改造TMHA支持一主一从取消中间备用master。

MHA工作原理：

1从宕机的master保存二进制日志事件（binlog events）

2识别含有最新更新的slave

3应用差异的中继日志（relay log）到其他的slave

4应用从master保存的二进制日志事件（binlog events）

5提升一个slave为新的master

6使其他的slave连接新的master进行复制

MHA切换流程：

1读取配置验证所有slave指向主库

2检查切换状态

3找新主

4使得新主与老主的数据一致

5关闭老主

6使得reset slave与新主一致

7重指新主

8激活新主

**MySQL视图**

视图(View)简介：

– 虚拟表

– 内容与真实的表相似，包含一系列带有名称的列和行 数据。

– 视图并不在数据库中以存储的数据的形式存在。

– 行和列的数据来自定义视图时查询所引用的基表，并且在具体引用视图时动态生成。

– 更新视图的数据，就是更新基表的数据

– 更新基表数据，视图的数据也会跟着改变

视图优点：

• 简单

– 使用视图的用户完全不需要关心视图中的数据是通过什么查询得到的。

– 视图中的数据对用户来说已经是过滤好的符合条件的结果集。

• 安全

– 用户只能看到视图中的数据。

• 数据独立

– 一旦视图的结构确定了，可以屏蔽表结构变化对用户 的影响。

使用视图的限制：

• 不能在视图上创建索引

• 在视图的FROM子句中不能使用子查询

• 以下情形中的视图是不可更新的

– 包含以下关键字的SQL语句：聚合函数(SUM、MIN 、MAX、COUNT等)、DISTINCT、GROUP BY 、HAVING、UNION或UNION ALL

– 常量视图

– JOIN

– FROM一个不能更新的视图

– WHERE子句的子查询引用了FROM子句中的表

– 使用了临时表，视图是不可更新

**MySQL存储过程**

存储过程：

– 数据库中保存的一系列sql命令的集合

– 编写存储过程时，可以使用变量、条件判断、流程控 制等

– 存储过程，就是MySQL中的脚本

存储过程优点：

– 提高性能

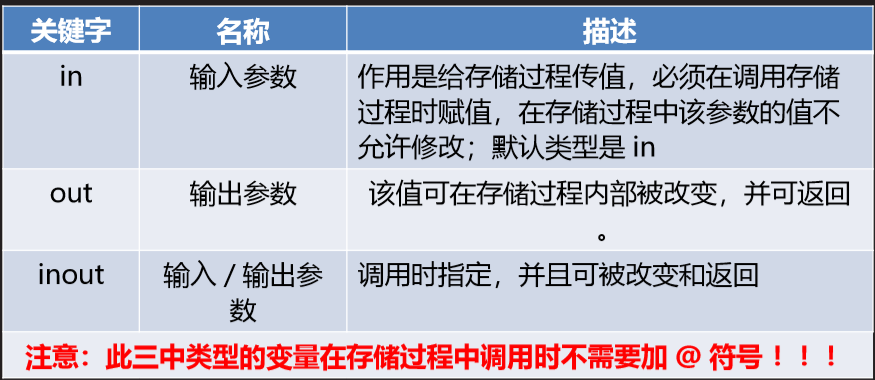
– 可减轻网络负担

– 可以防止对表的直接访问

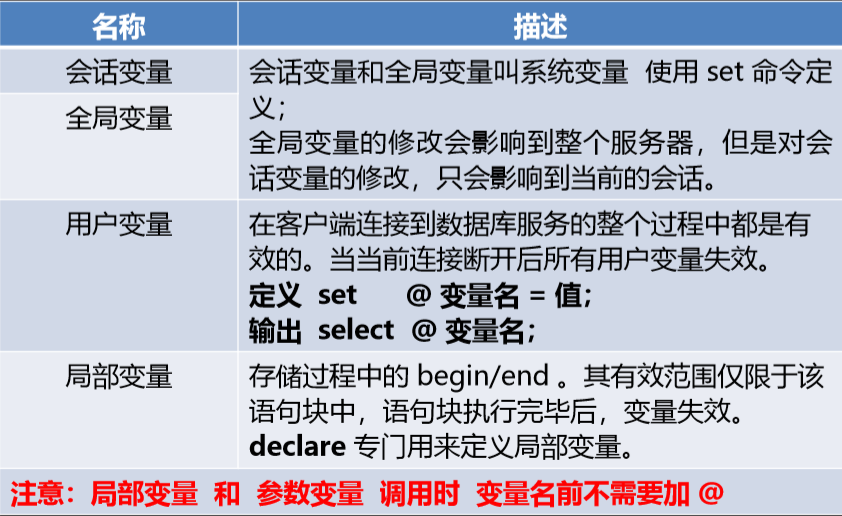
– 避免重复的sql操作

参数类型：

>Create procedure 名称(类型 参数名 数据类型 ，类型 参数名 数据类型 ）；



变量类型 • 变量的种类:全局变量\会话变量\用户变量\局部变量



**分库分表**

• 定义：

– 通过某种特定条件，将存放在一个数据库(主机)中的 数据，分散存放到多个数据库(主机)中。

– 已达到分散单台设备负载的效果，即分库分表

– 数据的切分根据其切分规则的类型，分为2种切分模 式

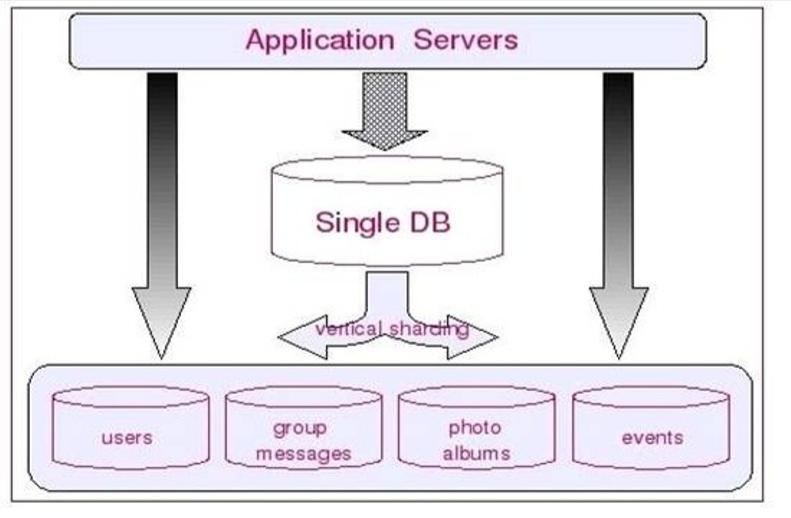
– 垂直分割(纵向) 和 水平分割(横向)

垂直分割

• 纵向切分

– 把单一的表，拆分成多个表，并分散到不同的数据库 (主机)上。

– 一个数据库由多个表构成，每个表对应不同的业务， 可以按照业务对表进行分类，将其分布到不同的数据 库(主机)上，实现专库专用，让不同的库(主机)分 担不同的业务。

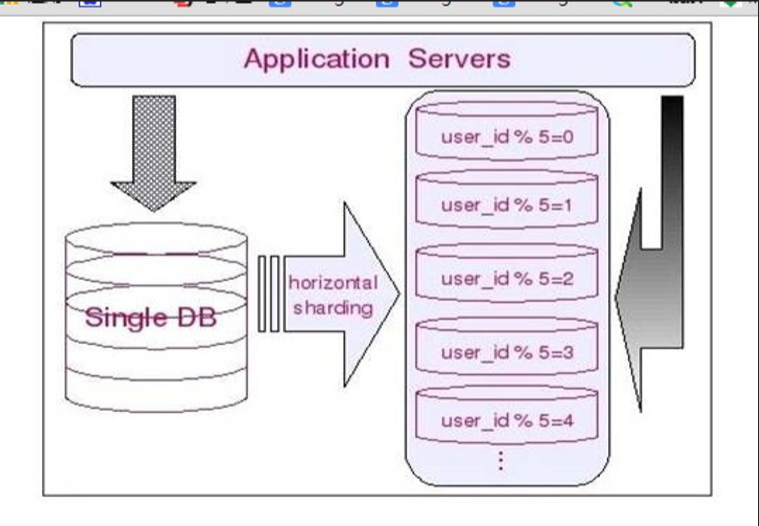


水平分割

• 横向切分

– 按照表中某个字段的某种规则，把向表中写入的记录 分散到多个库(主机)中。

– 简单来说，就是按照数据行切分，将表中的某些行存 储到指定的数据库 (主机) 中。



Mycat是基于Java的分布式数据库系统中间层，为 高并发下的分布式提供解决方案

– 支持JDBC形式连接

– 支持MySQL、Oracle、Sqlserver、Mongodb等

– 提供数据读写分离服务

– 可以实现数据库服务器的高可用

– 提供数据分片服务

– 基于阿里巴巴Cobar进行研发的开源软件

– 适合数据大量写入数据的存储需求

分片规则：

• mycat服务提供10种分片规则。

– 1枚举法 sharding-by-intfile

– 2固定分片 rule1

– 3范围约定 auto-sharding-long

– 4求模法 mod-long

– 5日期列分区法 sharding-by-date

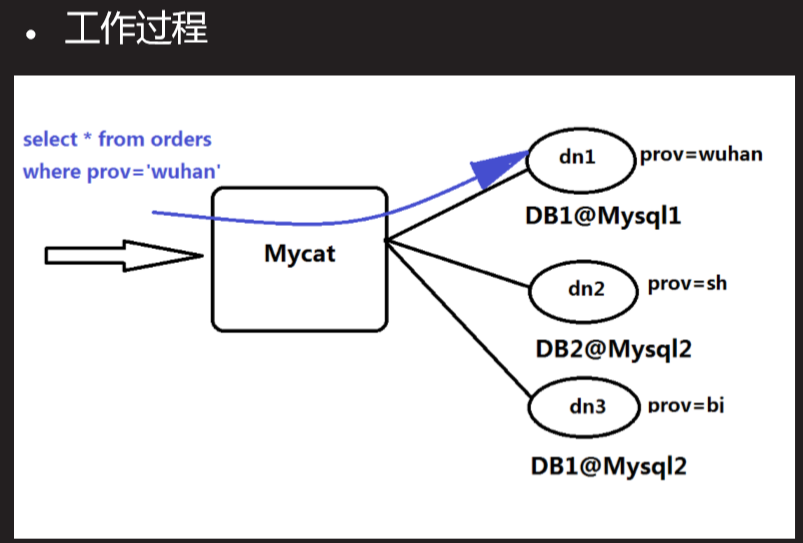
– 6通配取模 sharding-by-pattern

– 7ASCII码求模通配 sharding-by-prefixpattern

– 8编程指定 sharding-by-substring

– 9字符串拆分hash解析 sharding-by-stringhash

– 10一致性hash sharding-by-murmur



当Mycat收到一个SQL时

– 会先解析这个SQL查找涉及到的表，然后看此表的定 义

– 如果有分片规则，则获取到SQL里分片字段的值，并 匹配分片函数，得到该QL对应的分片列表

– 然后将SQL发往这些分片去执行，最后收集和处理所 有分片返回的结果数据，并输出到客户端

以 select \* from Orders where prov=?语句为例，查到prov=wuhan， 按照分片函数，wuhan返回dn1，于是SQL就发给了MySQL1，去 取DB1上的查询结果，并返回给用户。 如果上述SQL改为elect \* from Orders where prov in (‘wuhan’,‘beijing’)，那么，SQL就会发给ySQL1与MySQL2去执行， 然后结果集合并后输出给用户。但通常业务中我们的SQL会有Order By 以及Limit翻页语法，此时就涉及到结果集在Mycat端的二次处理。