MySQL5 入门培训

v1.0

王年传

2011-08-08

提纲

- **MySQL基础**
  - 1.基础知识
  - 2.引擎选择
  - 3.安装部署
  - 4.基础操作(增/删/改/查/建)
  - 5.数据库开发(存储过程,触发器,事件调度)
  - 6.常用命令
  - 7.常用工具
  - 8.数据库巡检

1. 基础知识

- **主流数据库产品介绍.**
  - 磁盘数据库: Oracle, DB2, Sybase, SQL Server, Informix, 国产库; MySQL(开源), PostgreSQL(开源)
  - 内存数据库: Timesten, Altibase, 其它开源产品
  - NoSQL数据库: MongoDB, Redis, CouchDB, Cassandra, Memcached...
- **MySQL是一个多线程,多用户,开源的SQL数据库服务器, 它采用双许可方式, 用户可根据GPL条款将其作为开放源码产品使用, 或从MySQL所属公司购买标准的商业许可证.**
  - 1979年: TcX公司Monty(MySQL创始人)开发名为Unireg报表工具
  - 199605: MySQL1.0发布
  - 199610: MySQL3.11.1发布, 接下两年MySQL3.22发布
  - 1999-2000: MySQL AB公司成立, MySQL3.23发布
  - 200004: MyISAM(取代ISAM)引擎发布, 引入主从机制, 支持全文检索
  - 200110: MySQL4.0 Alpha版发布(首次引入InnoDB引擎)
  - 200303: MySQL4.0 稳定版发布(InnoDB成熟, 引进Query Cache, Slave改为双线程)
  - 200312: MySQL5.0 Alpha版发布(支持视图,游标,储过程,分布式事务)
  - 200410: MySQL4.1 稳定版发布(支持子查询,Unicode,预处理语句)
  - 200510: MySQL5.0 稳定版发布, Oracle收购Innobase Oy公司
  - 200511: MySQL5.1 Alpha版发布(支持表分区,行复制,时间调度,引擎和插件API标准化)
  - 200801: Sun公司收购MySQL AB
  - 200904: Oracle公司收购Sun
  - 200911: MySQL5.1 稳定版发布
  - 201012: MySQL5.5 稳定版发布(引入半复制,InnoDB性能提升,使用跨平台的cmake编译)

2. 引擎选择

- **MySQL支持数个存储引擎作为对不同表的类型的处理器.**
  - MyISAM: 管理非事务表. 它提供高速存储和检索, 以及全文搜索能力. MyISAM在所有MySQL配置里被支持, 它是MySQL5.5以前版本默认的存储引擎.
  - MEMORY: 管理非事务表. 它提供"内存中"表, 默认包含在MySQL中.
  - MERGE: 管理非事务表. 它允许集合将被处理同样的MyISAM表作为一个单独的表. 就像MyISAM一样, MEMORY和MERGE存储引擎处理非事务表. 默认包含在MySQL中.
  - InnoDB和BDB: 提供事务安全表. BDB被包含在为支持它的操作系统发布的MySQL-Max二进制分发版里. InnoDB也默认被包括在所有二进制分发版里.它是MySQL5.5版本默认的存储引擎.
  - EXAMPLE: "存根"引擎, 它不做什么. 可以用这个引擎创建表, 但没有数据被存储于其中或从其中检索. 它演示说明如何开始编写新存储引擎.
  - NDB Cluster: 是被MySQL Cluster用来实现分割到多台计算机上的表的存储引擎. 它在MySQL-Max5.1二进制分发版里提供. 当前只被Linux,Solaris和Mac OS X支持.
  - ARCHIVE: 被用来无索引的, 非常小的覆盖存储的大量数据, 不支持update/delete.
  - CSV: 把数据以逗号分隔的格式存储在文本文件中.
  - BLACKHOLE: 接受但不存储数据, 并且检索总是返回一个空集.
  - FEDERATED: 把数据存在远程数据库中. 类似Oracle中的DBLINK.

3. 安装部署： 版本选择

- **MySQL版本: 如果首次使用, 建议使用最新稳定版**
  - MySQL官方区分社区版和企业版, 一般使用社区版, 因为它免费
  - 如果使用MyISAM引擎, 推荐5.1最新稳定版(5.1.58)
  - 如果使用InnoDB引擎, 推荐5.5最新稳定版(5.5.15)
  - 也可以选择使用其它存储引擎: XtraDB, Maria
  - 还可以选择使用其它MySQL: MariaDB分支, Percona Server衍生版, Drizzle精简版等等
- **操作系统: RHEL5**
  - 架构: x86_64, 无内存限制
  - 编译器: gcc4.1.2版本以上, glibc2.3以上版本
- **软件下载:** http://mirrors.sohu.com/mysql/MySQL-5.1/
  - tar.gz源码包(推荐使用): mysql-5.1.58.tar.gz
  - tar.gz二进制包(区分操作系统类型和架构): mysql-5.1.58-linux-x86_64-glibc23.tar.gz
  - rpm包(区分操作系统版本和架构): MySQL-server-community-5.1.58-1.rhel5.x86_64.rpm
  - src.rpm包(区分操作系统版本): MySQL-community-5.1.58-1.rhel5.src.rpm

```
# cat /etc/issue | head -1
Red Hat Enterprise Linux Server release 5.4 (Tikanga)
# uname -m
x86_64
# rpm -q gcc glibc glibc-common gcc-c++
```

3. 安装部署： 规划

- **安装建议: 同一主机可安装多个MySQL实例, 便于升级和测试**
  - 定义独立的MySQL实例名, 自定义变量MYSQL_SID代表实例名
  - 定义独立的数据库文件和日志文件目录(目录名中含MySQL实例名)
  - 其它路径或变量依赖于MYSQL_SID变量

```
# vi ~/.bash_profile
MYSQL_SID=MYDB51
MYSQL_HOME=/usr/local/mysql51
export MYSQL_TCP_PORT=3351
export LD_RUN_PATH=$MYSQL_HOME/lib
export PATH=.:$MYSQL_HOME/bin:$PATH
export MYSQL_DATA=/u01/mydata/${MYSQL_SID}
export MYSQL_LOG=/u01/mylog/${MYSQL_SID}
export MYSQL_UNIX_PORT=${MYSQL_LOG}/mysql51.sock
# . ~/.bash_profile
```

3. 安装部署： 编译

1) 增加用户和组

```
# groupadd dba
# useradd -g dba mysql
```

2) 解压安装包

```
# tar zxvf mysql-5.1.58.tar.gz && cd mysql-5.1.58
```

3) 设置**gcc**参数**:** 使用**3**级优化模式**,** 可提高性能

```
# CC=gcc CFLAGS="-O3 -mpentiumpro -fomit-frame-pointer"
# CXX=g++ CXXFLAGS="-O3 -mpentiumpro -fomit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti"
```

4) 编译安装**:** 根据需要定制**,** 静态线程方式编译**,** 会比二进制版本性能更高

```
# ./configure --prefix=/usr/local/mysql51
--localstatedir=/u01/mydata/${MYSQL_SID} \
--enable-thread-safe-client --enable-assembler --enable-profiling --enable-local-infile \
--with-charset=utf8 --with-collation=utf8_general_ci --with-extra-charsets=all \
--with-unix-socket-path=/u01/mylog/${MYSQL_SID}/mysql51.sock --with-tcp-port=3351 \
--with-mysqld-user=mysql --without-debug \
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static \
--without-docs --without-embedded-server --with-pthread --with-big-tables \
--with-plugins=archive,blackhole,csv,federated,heap,innobase,innodb_plugin,partition
# gmake -j 8
# gmake install
```

3. 安装部署： 默认参数文件

mysqld读取默认参数文件的顺序是**/etc/my.cnf, /etc/mysql/my.cnf, ${MYSQL_HOME}/etc/my.cnf, ~/.my.cnf.**
鉴于一台服务器上可能会安装多个**mysqld,** 建议使用**${MYSQL_HOME}/etc/my.cnf**作为参数文件**.**

1) 确认默认文件是否存在
ll /etc/my.cnf /etc/mysql/my.cnf ~/.my.cnf

2) 删除可能存在的默认参数文件**,** 防止被**mysqld**读取
mv /etc/my.cnf /etc/my.cnf.enc
mv /etc/mysql/my.cnf /etc/mysql/my.cnf.enc
mv ~/.my.cnf ~/.my.cnf.enc

3) 查看系统自带参数文件**:** 其中的一些默认参数取决于编译时的参数设定**(**如端口**,socket**等**)**
ls -1 $MYSQL_HOME/share/mysql/*.cnf
/usr/local/mysql51/support-files my-huge.cnf
/usr/local/mysql51/support-files/my-innodb-heavy-4G.cnf
/usr/local/mysql51/support-files/my-large.cnf
/usr/local/mysql51/support-files/my-medium.cnf
/usr/local/mysql51/support-files/my-small.cnf
其中my-huge.cnf是给1-2G内存, 以MyISAM引擎为主的数据库设置;
其中my-innodb-heavy-4G.cnf是给4G内存且以InnoDB引擎为主的数据库设置的;
以上几种一般用于测试环境是没有问题的, 但如果用于线上环境, 强烈自定义一份.

3. 安装部署： 重要参数说明

本例假设使用InnoDB作为MySQL默认引擎.

mkdir -p $MYSQL_HOME/etc

cp $MYSQL_HOME/support-files/my-innodb-heavy-4G.cnf $MYSQL_HOME/etc/my.cnf

vi $MYSQL_HOME/etc/my.cnf

[mysql]段主要参数说明:

port = 3351 数据库访问端口

socket = /u01/mylog/MYDB51/mysql51.sock 数据库访问socket

default-character-set = gbk 客户端连接字符集

[mysqld]段主要参数说明:

default-storage-engine = InnoDB 设置默认引擎

query_cache_size = 256M 查询缓存大小

table_open_cache = 2048 表句柄缓存数量

max_connections = 256 数据库最大连接

key_buffer_size = 4G MyISAM索引缓存大小, 如果不使用InnoDB情况下, 建议为物理内存的1/3左右

read_buffer_size = 2M read_rnd_buffer_size = 4M sort_buffer_size = 8M join_buffer_size = 4M 每个线程内存配置

event_scheduler = 1 启用事件调度

lower_case_table_names = 1 表名大小写不敏感

innodb_buffer_pool_size = 8G InnoDB数据缓存大小,会缓存数据和索引, 建议为物理内存的2/3~3/4左右

innodb_log_file_size = 256M redo日志文件大小

注意: 如果很少使用MyISAM表，那么也保留16-32MB的 key_buffer_size以适应给予磁盘的临时表索引所需.

3. 安装部署： 启动MySQL

1) 建立数据库文件主目录并设置权限

```
# mkdir -p /u01/mydata/${MYSQL_SID}
# chown -R mysql.dba /u01/mydata/${MYSQL_SID}
# chown -R root.dba $MYSQL_HOME
```

2) 创建授权表**:** 建立默认用户和数据字典

```
# cd $MYSQL_HOME
# ./bin/mysql_install_db
```

3) 启动**MySQL**实例

```
# $MYSQL_HOME/bin/mysqld_safe --user=mysql &
```

4) 设置自动自动

```
# cp $MYSQL_HOME/share/mysql/mysql.server /etc/init.d/mysqld51
# chmod 751 /etc/init.d/mysqld51
# chkconfig --level 35 mysqld51 on
# service mysqld51 restart
```

5) 基本安全设置**:** 用于线上环境的数据库应该去掉匿名用户和**test**库**,**并设置**root**密码**.**

```
# mysql_secure_installation
```

提示: 新装的MySQL默认root密码为空, 设置时有提示则输入y.

4. 基本操作：连接, 帮助, 密码, 用户, 变量

1) 连接和退出

```
# mysql -uroot -p                                  //连接到本机MySQL
# mysql -h110.110.110.110 -uroot -piamdba          //连接到远程MySQL, -u与root可以不用加空格, 其它也一样.
mysql> exit                                        //或输入quit, 或Ctrl-D都可以退出
```

2) 查看帮助

```
mysql> ?                                           //查看mysql客户端帮助
mysql> ? contents                                  //查看所有的帮助分类
mysql> ? tables                                    //查看某个具体的帮助分类, 如tables
mysql> ? create database                           //查看某个具体的命令帮助, 如create database
```

3) 修改密码

```
# mysqladmin -uroot -piamdba password newiamdba
```

也可以直接修改mysql.user表的密码字段:

```
mysql> update user set password=password('newiamdba') where mysql.user='username';
mysql> flush privileges;                           //强制刷新内存授权表, 否则用的还是缓冲中的密码
```

4) 新增用户

增加一个用户admin密码为oss, 可以从任何地方连接服务器的一个完全的超级用户:

```
mysql> grant all privileges on *.* to admin@"%" identified by 'oss' with grant option;
```

增加一个用户test2密码为abc, 让它只可以在192.168.41.42上登录, 并可以对数据库db_test进行操作:

```
mysql> grant select,insert,update,delete on db_test.* to test2@192.168.41.42 identified by "abc";
```

5) 自定义变量和系统变量

```
mysql> set @var='test'; select @var;
mysql> select @@innodb_buffer_pool_size;
```

4. 基本操作：一个完整例子

```
mysql> drop database if exists db_school;          //如果存在db_school则删除
mysql> create database db_school;                  //建立库db_school
mysql> use db_school;                              //打开库db_school
mysql> create table t_teacher(
    id int(3) auto_increment not null primary key,
    name char(10) not null,
    address varchar(50) default '北京',
    year date) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

注: 将ID设为长度为3的数字字段int(3), 并让它每个记录自动加一(auto_increment), 并不能为空not null, 而且让它成为主键primary key.

```
mysql> insert into t_teacher values(1,'encle','北京一中','1976-10-10');   //插入数据, 默认是自动提交
mysql> insert into t_teacher values(2,'jack','北京2中','1975-10-10');
mysql> select * from t_teacher;                                           //查询数据
mysql> rename table t_teacher to t_teacher_new;                           //给表改名
mysql> alter table t_teacher_new change id newid varchar(5) not null;    //修改字段属性
mysql> alter table t_teacher_new add leave_timedatetime not null after address;   //在表的address后新增字段
mysql> desc t_teacher_new;                                                //显示表结构
mysql> select * from t_teacher_new;
mysql> show create table t_teacher_new;                                   //显示新表的完整建表脚本
```

4. 基本操作： select, 导出导入

1) 基本select操作

```
mysql> select version(),current_date();          //显示版本和当前日期
mysql> select user(),now();                       //显示当前用户和当前时间
mysql> select (20+5)*4;                           //做算术
mysql> select * from mysql.user limit 3;          //返回前3行
mysql> select * from mysql.user limit 0,3;        //返回前3行
mysql> select * from mysql.user limit 3,5;        //返回从第4行开始的前5行, 即4-8行
```

第一个参数指定返回的第一行在所有数据中的位置, 从0开始(注意不是1), 第二个参数指定最多返回行数.

2) 导入导出文本数据

```
# vi /tmp/test.txt
3    rose    北京二中    1976-10-10
4    mike    北京一中    1975-12-23
```

导入：

```
mysql> use db_school
mysql> load data local infile "/tmp/test.txt" into table t_teacher_new;
mysql> select * from t_teacher_new;
```

导出:

```
mysql> select * from t_teacher_new into outfile '/tmp/test_new.txt';
mysql> exit
# cat /tmp/test_new.txt
```

4. 基本操作：外部脚本, 操作系统命令

1) 执行批量SQL或存储过程脚本

```
# vi /tmp/test.sql
use db_test;
create table ttttt (a int);
# mysql -uroot -p
mysql> source /tmp/test.sql
Database changed
Query OK, 0 rows affected (0.57 sec)
mysql> select * from ttttt;
Empty set (0.00 sec)
mysql> exit
```

2) 执行的操作系统命令

mysql客户端(非windows版)提供了一个system命令, 只要在system后接要执行的操作系统命令即可:

```
mysql> system uname -a
Linux new.linekong.com 2.6.18-164.el5PAE #1 SMP Tue Aug 18 15:59:11 EDT 2009 i686 i686 i386 GNU/Linux
```

5. 数据库开发： 存储过程简单例子

1) 新建存储过程

```
mysql> create procedure sp1() create table t_test10(c1 date);
mysql> create procedure sp2() select * from t_test10;
mysql> create procedure sp3() drop table t_test10;
```

2) 显示存储过程基本信息

```
mysql>  show procedure status;
```

3) 显示存储过程代码信息

```
mysql> show create procedure sp1;
```

4) 调用存储过程

```
mysql> call sp1;
mysql> call sp2;
mysql> call sp3;
```

5) 删除存储过程

```
mysql> drop procedure sp1;
mysql> drop procedure sp2;
mysql> drop procedure sp3;
```

5. 数据库开发： 存储过程复杂例子

1) 有输入参数的存储过程

```
mysql> create procedure sp5 (p INT) SET @x=p;
mysql> call sp5(12345);
mysql> SELECT @x;
```

2) 有输出参数的存储过程

```
mysql> create procedure sp6 (OUT p INT) SET p=-5;
mysql> call sp6(@y);
mysql> SELECT @y;
```

3) 详细分析过程体

由于MySQL默认把分号当作结束符, 需临时更改结束符为别的符号, 刷完后, 再把结束符置为分号. 如下:

```
mysql> delimiter //
mysql> create procedure sp7()
BEGIN
SET @a=5;
SET @b=5;
INSERT INTO t VALUES(@a);
SELECT s1 * @a FROM t WHERE s1 >= @b;
END;
//
mysql> delimiter ;
```

5. 数据库开发： 触发器

1) 语法

CREATE [DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW trigger_stmt

2) 参数说明**:**

DEFINER: 指定触发器的创建者, 默认为登录mysqld服务器的账号信息;
trigger_name: 触发器的名称, 要符合mysql对待数据库对象命名的规范;
trigger_time: 触发表上的触发器语句体执行的时间, 行更新前还是行更新后, 2个选项值BEFOR或AFTER;
tbl_name: 指定触发器是对应那一个数据库对象;
trigger_stmt: 为触发器内部可执行的语句体;

3) 示例**:** 记录所有用户对表**t_car,** 进行**UPDATE**操作修改数据的行为进行记录

```
mysql> DELIMITER $$
mysql> CREATE TRIGGER tri_t_car BEFORE UPDATE ON t_car FOR EACH ROW
BEGIN
IF NEW.pay<>OLD.pay THEN
      INSERT INTO t_record(username,client_ip,update_Before,update_After,gmt_create)
      VALUES(SUBSTRING_INDEX(USER(),'@',1),SUBSTRING_INDEX(USER(),'@',-
      1),OLD.pay,NEW.pay,NOW());
END IF;
END $$
mysql> DELIMITER ;
```

5. 数据库开发： 事件调度

1) 前提: 确认EVENT可用

mysql> Select @@event_scheduler;

mysql> SET GLOBAL event_scheduler = ON;

在参数文件[mysqld]段的下面加入如下行:

event_scheduler=1

2) 简单例子: 每秒插入一条记录到数据表

mysql> USE db_test;

mysql> CREATE TABLE aaa (timeline TIMESTAMP);

mysql> CREATE EVENT e_test_insert ON SCHEDULE EVERY 1 SECOND DO INSERT INTO test.aaa
VALUES (CURRENT_TIMESTAMP);

mysql> SELECT * FROM aaa;

3) 复杂例子

每天定时清空test表:

mysql> CREATE EVENT e_test ON SCHEDULE EVERY 1 DAY DO TRUNCATE TABLE test.aaa;

5天后开启每天定时清空test表, 一个月后停止执行:

mysql> CREATE EVENT e_test ON SCHEDULE EVERY 1 DAY STARTS CURRENT_TIMESTAMP +
INTERVAL 5 DAY ENDS CURRENT_TIMESTAMP + INTERVAL 1 MONTH DO TRUNCATE TABLE
test.aaa;

2007年7月20日12点整清空test表:

mysql> CREATE EVENT e_test ON SCHEDULE AT TIMESTAMP "2007-07-20 12:00:00" DO TRUNCATE
TABLE test.aaa;

6. 常用命令： show系列

1) show variables命令: 获取会话参数和全局参数

```
mysql> show variables like '%size';
mysql> show global variables like '%size';
```

2) show index命令: 获取表的索引信息

```
mysql> help show index
mysql> show indexes from dw_task_status from dogwars;
```

3) show processlist命令: 显示哪些线程正在运行

```
mysql> show processlist;                    //Info列中只显示前100个字符
mysql> show full processlist;               //Info列中显示全部字符
```

关注state列. 只要有一个线程保持同一个状态好几秒中, 那么可能是有问题发生了, 需要检查一下.

4) show open tables命令: 查看现在打开了哪些表(不包括临时表)

```
mysql> help show open tables;
mysql> show open tables from dogwars;       //from过滤, 只输出某个数据库下打开的表
mysql> show open tables like '%users%';     //like过滤, 只输出含有某些字符的表
mysql> show open tables where In_use > 0;   //where过滤, 只输出表上线程数大于0的表
```

5) show profile命令: 实时记录并分析某session中所有sql语句的性能

```
mysql> show variables like 'have_profiling';   //确信profiling特性是可用的
mysql> SET profiling=1;                        //在本session开启profiling,会对系统性能有点影响
mysql> xxx                                     //执行一些sql操作
mysql> SHOW PROFILES;                          //把sql语句和执行时间记录下来
mysql> SHOW PROFILE FOR QUERY 1;               //查看第1个sql各状态的执行时间
```

6. 常用命令： 优化表和解释SQL

1) optimize table命令:

对于MyISAM或Memory表, 如果已经删除了表的一大部分, 或者如果已经对含有可变长度行的表(含有VARCHAR, BLOB或TEXT列的表)进行了很多更改, 则应使用OPTIMIZE TABLE来优化表物理文件.

mysql> help optimize table

2) explain命令: 解释SQL如何呗处理的, 提供有关表如何联合和以什么次序等信息.

explain tbl_name 或 explain [extend] select_statement

前者可以得出一个表的字段结构等等, 后者主要是给出相关的一些索引信息.

mysql> explain select * from dw_users; //获得基本解释信息

mysql> explain extend select * from dw_users; //获得扩展解释信息

mysql> show warnings; //查询相应的优化信息

输出字段含义:

id: select查询的序列号

select_type: select查询的类型, 主要是区别普通查询和联合查询,子查询之类的复杂查询.

table: 输出的行所引用的表.

type: 联合查询所使用的类型. 显示的是访问类型, 是较为重要的一个指标, 结果值从好到坏依次是 system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL, 一般来说, 得保证查询至少达到range级别, 最好能达到ref.

possible_keys: 指出能使用哪个索引在该表中找到行. 如果是空的, 没有相关的索引.

key: 显示实际决定使用的键. 如果没有索引被选择, 键是NULL.

rows: 这个数表示要遍历多少数据才能找到.

Extra: 该列包含解决查询的详细信息.

7. 常用工具： MySQL自带

1) mysqldumpslow: 通过分析慢查询日志来分析SQL

首先打开慢查询功能:

vi $MYSQL_HOME/etc/my.cnf

[mysqld]

long_query_time = 2

slow_query_log_file = /u01/mylog/WEBSITE/slow_query.lo

log-queries-not-using-indexes

其中: long_query_time=2表示查询超过2秒才记录; slow_query_log_file为慢查询日志存放的位置; log-queries-not-using-indexes表示记录下那些没有使用索引的查询(无论执行时间多长).

输出记录次数最多的10条SQL语句:

mysqldumpslow -s c -t 10 /u01/mydata/slow_query.log

输出返回记录集最多的10个查询:

mysqldumpslow -s r -t 10 /u01/mydata/slow_query.log

输出按照时间排序的前10条里面含有左连接的查询:

mysqldumpslow -s t -t 10 -g "left join" /u01/mydata/slow_query.log

2) mysqlbinlog: 从二进制日志找出历史DML语句(insert/update/delete)

mysqlbinlog mysql-bin.000019 |grep -i -E "^(update|insert|delete)" > bin.dml.sql　　//解析二进制日志

tail -n 100000 bin.dml.sql > bin.tail_dml.sql　　//找出最近的10w条sql来分析

sed -i -r 's/[0-9]{1,}/D/g' bin.tail_dml.sql　　//把所有的数字替换成D, 这样可以统计分表上面的sql

排序, 找出执行次数最多的sql语句, 按照降序排列, 取top 50的sql来分析:

sort bin.tail_dml.sql | uniq -c | sort -n -r | head -n 50 > bin.sorted_top_50.sql

7. 常用工具：第三方

1) phpMyAdmin

phpMyAdmin是使用PHP编写的, 以网页方式管理MySQL数据库的一个开源管理工具. 需要先安装Apache/PHP/MySQL.

官方网站: http://www.phpmyadmin.net

2) Maatkit

Maatkit是一个Perl写的MySQL开源管理工具, 根据调查全球大约有70%多的MySQL管理员使用这个工具来管理MySQL. 需要Perl,DBI,DBD::mysql相关模块的支持.

参考文档: http://www.sbear.cn/archives/tag/maatkit

3) tuning-primer.sh(推荐)

tuning-primer.sh有点类似Oracle中的ADDM. 和mysqlreport一样也支持.my.cnf, 相比mysqlreport更进一步, 除了报表还进一步提供了修改建议, 在终端上按照问题重要程度分别用黄色/红色字符标记问题.

下载地址: http://www.day32.com/MySQL/tuning-primer.sh

同类工具: MySQLTuner, mysqlreport, mytop

4) HeidiSQL(推荐)

HeidiSQL 是一个功能非常强大的 MySQL客户端软件, 采用 Delphi 开发, 支持Windows操作系统.

体积小, 速度流畅, 提供非常方便的监控和管理功能.

下载地址: http://www.heidisql.com/download.php

同类工具: Workbench(官方), Toad for mysql, SQL Front, SQLyog, Navicat

8. 数据库巡检

1)通过工具来巡检

如phpMyAdmin(一个基于php的MySQL管理平台).

2) 通过命令来巡检

```
mysql> status                                  //了解数据库的版本和平台以及字符集等相关信息
mysql> show engines;                           //了解数据库中支持哪些存储引擎
mysql> show plugins;                           //查下插件情况
mysql> show variables like 'have_ndbcluster';  //搞清楚这个环境是单机还是集群
mysql> show master status;                     //是否配置了REPLICATION
mysql> show slave status \G;
mysql> show variables like 'log%';             //查看日志模式, 慢查询日志和ERR日志存放位置
mysql> show triggers;                          //查看有哪些触发器
mysql> show procedure status;                  //查看有哪些存储过程
mysql> show variables like 'have_part%';       //是否支持分区
```

如果支持哪些使用了分区表:

```
mysql> select TABLE_NAME from information_schema.PARTITIONS where PARTITION_NAME is not null;
```

有多少用户拥有超级权限:

```
mysql> select * from information_schema.USER_PRIVILEGES where PRIVILEGE_TYPE='SUPER';
```

是否有密码为空(ROOT密码默认为空):

```
mysql> select host,User,Password from mysql.user where Password='';
mysql> delete from mysql.user where Password='';flush PRIVILEGES;       //密码为空马上删除
mysql> show processlist;                       //查看当前有多少并发
```

提纲

- **MySQL高级**
  - 9.分区技术
  - 10.复杂SQL(左连接/右连接/等值连接)
  - 11.性能优化
  - 12.备份恢复
  - 13.主从设置(Replication)
  - 14.读写分离和切分(Sharding)
  - 15.高可用性(HA)
  - 16.自动化监控

9. 分区技术： Range

MySQL的分区技术不同与之前的分表技术, 它与水平分表有点类似, 但是它是在逻辑层进行的水平分表, 对与应用程序而言它还是一张表.

MySQL5.1有分区类型:

RANGE分区: 基于属于一个给定连续区间的列值, 把多行分配给分区;

LIST分区: 类似于按RANGE分区, 区别在于LIST分区是基于列值匹配某个值来进行选择;

HASH分区: 基于用户定义的表达式的返回值来进行选择的分区, 该表达式使用将要插入到表中的这些行的列值进行计算. 这个函数可以包含MySQL中有效的,产生非负整数值的任何表达式;

KEY分区: 类似于按HASH分区, 区别在于KEY分区只支持计算一列或多列, 且MySQL服务器提供其自身的哈希函数.

```
mysql> CREATE TABLE employees (
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     hired DATE NOT NULL DEFAULT '1970-01-01',
    ->     separated DATE NOT NULL DEFAULT '9999-12-31',
    ->     job_code INT NOT NULL,
    ->     store_id INT NOT NULL
    -> )
    -> PARTITION BY RANGE (store_id) (
    ->     PARTITION p0 VALUES LESS THAN (6),
    ->     PARTITION p1 VALUES LESS THAN (11),
    ->     PARTITION p2 VALUES LESS THAN (16),
    ->     PARTITION p3 VALUES LESS THAN (21)
    -> ) ;
```

9. 分区技术：List & Hash

```
mysql> CREATE TABLE employees (
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     hired DATE NOT NULL DEFAULT '1970-01-01',
    ->     separated DATE NOT NULL DEFAULT '9999-12-31',
    ->     job_code INT,
    ->     store_id INT
    -> )
    -> PARTITION BY LIST(store_id)
    -> (
    ->     PARTITION pNorth VALUES IN (3,5,6,9,17),
    ->     PARTITION pEast VALUES IN (1,2,10,11,19,20),
    ->     PARTITION pWest VALUES IN (4,12,13,14,18),
    ->     PARTITION pCentral VALUES IN (7,8,15,16)
    -> );
```

```
mysql> CREATE TABLE employees (
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     hired DATE NOT NULL DEFAULT '1970-01-01',
    ->     separated DATE NOT NULL DEFAULT '9999-12-31',
    ->     job_code INT,
    ->     store_id INT
    -> )
    -> PARTITION BY HASH(YEAR(hired))
    -> PARTITIONS 4
    -> ;
```

10. 复杂SQL： 左连接

SELECT L.columnname……,R.* columnname…..

FROM left_table L LEFT JOIN right_table R

ON L.columnname_join=R.columnname_join AND R.columnname=XXX

WHERE L.columnname=XXX…..

ON字句连接条件, 用于把2表中等值的记录连接在一起, 但是不影响记录集的数量. 若是表left_table中的某记录, 无法在表right_table找到对应的记录, 则此记录依然显示在记录集中, 只是表right_table需要在查询显示的列的值用NULL替代;

ON字句连接条件中表right_table.columnname=XXX用于控制right_table表是否有符合要求的列值, 还是用NULL替换的方式显示在查询列中, 不影响记录集的数量;

WHERE字句控制记录是否符合查询要求, 不符合则过滤掉;

总结:

ON字句控制right_table的列值符合显示, 还是不符合就用NULL替换, 不影响最终符合查询要求的记录集;
WHERE字句是控制那些记录是显示在最终的记录集中.

10. 复杂SQL：右连接

SELECT L.columnname……,R.* columnname…..

FROM left_table L RIGHT JOIN right_table R

ON L.columnname_join=R.columnname_join AND L.columnname=XXX

WHERE R.columnname=XXX…..

ON字句连接条件, 用于把2表中等值的记录连接在一起, 若是表right_table中的某记录, 无法在表left_table找到对应的记录, 则表left_able需要在查询显示的列的值用NULL替代;

ON字句连接条件中表left_table.columnname=XXX用于控制left_table表是否有符合要求的列值, 还是用NULL替换的方式显示在查询列表中;

WHERE字句控制记录是否符合查询要求, 不符合则过滤掉;

总结:

ON字句控制left_table的列值符合显示, 还是不符合而用NULL替换掉, 不影响最终符合查询要求的记录集;
WHERE字句是控制那些记录是显示在最终的记录集中. 会发现LEFT JOIN和RIGHT JOIN是类似的, 只是以连接关键字左边还是右边表为准匹配

10. 复杂SQL： 等值连接

SELECT L.columnname……,R.* columnname…..

FROM left_table L [INNER] JOIN right_table R

ON L.columnname_join=R.columnname_join

WHERE L.columnname=XXX….. AND R.columnname=XXX….

或者

SELECT L.columnname……,R.* columnname…..

FROM left_table L, right_table R

WHERE L.columnname_join=R.columnname_join

AND L.columnname=XXX….. AND R.columnname=XXX….

ON字句连接条件, 不再与左连接或右连接的功效一样, 除了作为2表记录匹配的条件外, 还会起到过滤记录的作用, 若left_table中记录无法在right_table中找到对应的记录, 则会被过滤掉;

WHERE字句, 不管是涉及表left_table或表right_table上的限制条件, 还是涉及2表连接的条件, 都会对记录集起到过滤作用, 把不符合要求的记录刷选掉;

11. 性能优化：举例一

1) 为搜索字段建立索引

如果在表中有某字段经常用来做搜索, 那么请为其建立索引吧. 但记住别滥用!

2) 尽量使用绑定变量

MySQL默认开启了查询缓存, 这是被MySQL的数据库引擎处理的. 当有很多相同的查询被执行了多次的时候, 这些查询结果会被放到一个缓存中, 这样, 后续的相同的查询就不用操作表而直接访问缓存结果了.

//查询缓存不开启:

```
$r = mysql_query("SELECT username FROM user WHERE signup_date >= CURDATE()");
```

//开启查询缓存:

```
$today = date("Y-m-d");
$r = mysql_query("SELECT username FROM user WHERE signup_date >= '$today'");
```

像 NOW() ,CURDATE()和 RAND() 或其它此类的函数不会开启查询缓存, 因为其返回值是易变的.

3) 当只要一行数据时使用 LIMIT 1

有时需要去fetch游标, 或是检查返回的记录数. 在已经知道结果只会有一条情况下, 加上 LIMIT 1 可以增加性能. 因为MySQL引擎会在找到一条数据后停止搜索, 而不是继续往后查少下一条符合记录的数据.

//没有效率的:

```
$r = mysql_query("SELECT * FROM user WHERE country = 'China'");
if (mysql_num_rows($r) > 0) {... }
```

//有效率的:

```
$r = mysql_query("SELECT 1 FROM user WHERE country = 'China' LIMIT 1");
 if (mysql_num_rows($r) > 0) {... }
```

11. 性能优化：举例二

4) 在Join表的时候使用相当类型的例, 并将其索引

如果有很多 JOIN 查询, 应该确认两个表中Join的字段是被建过索引的. 而且这些被用来Join的字段, 应该是相同的类型的. 例如: 如果要把 DECIMAL字段和一个 INT 字段Join在一起, 就无法使用它们的索引; 对于那些STRING类型, 还需要有相同的字符集才行(两个表的字符集有可能不一样).

```
$r = mysql_query("SELECT company_name FROM users  LEFT JOIN companies ON (users.state = companies.state) WHERE users.id = $user_id");
```

5) 千万不要 ORDER BY RAND()

MySQL会不得不去执行RAND()函数(很耗CPU时间), 而且这是为了每一行记录去记行, 然后再对其排序. 就算是你用了Limit 1也无济于事(因为要排序), 如果真想把返回的数据行打乱了, 有N种方法.

```
//千万不要这样做:
$r = mysql_query("SELECT username FROM user ORDER BY RAND() LIMIT 1");
//这要会更好:
$r = mysql_query("SELECT count(*) FROM user");
$d = mysql_fetch_row($r); $rand = mt_rand(0,$d[0] - 1);
$r = mysql_query("SELECT username FROM user LIMIT $rand, 1");
```

11. 性能优化：举例三

6) 避免 SELECT *

从数据库里读出越多的数据, 那么查询就会变得越慢. 并且, 如果数据库服务器和WEB服务器是两台独立主机的话, 这还会增加网络传输的负载. 所以, 应该养成一个需要什么就取什么的好的习惯.

//不推荐:

```
$r = mysql_query("SELECT * FROM user WHERE user_id = 1");
$d = mysql_fetch_assoc($r);
echo "Welcome {$d['username']}";
```

//推荐:

```
$r = mysql_query("SELECT username FROM user WHERE user_id = 1");
$d = mysql_fetch_assoc($r);
echo "Welcome {$d['username']}";
```

12. 备份恢复： 逻辑备份

1) mysqldump概述

mysqldump [OPTIONS] database [tables]

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

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

默认options会依次从/etc/my.cnf,/etc/mysql/my.cnf,$MYSQL_HOME/etc/my.cnf,~/.my.cnf文件读取.

注意, 如果运行mysqldump没有--quick或--opt选项, mysqldump将在导出结果前装载整个结果集到内存中.

MyISAM中为了保持数据一致性需要在备份之前对进行备份的数据库加读锁操作flush table with read lock;

InnoDB则可以在mysqladmin命令中加入-single-transaction选项, 生成一个快照以保证数据备份期间的一致性.

2) 备份全库示例

```
# mysqldump -uu_test --master-data=2 --flush-logs --hex-blob --routines --triggers --all_databases >
    /u01/backup/dump_all.sql 2>&1 &
```

注: 设置--master-data后会自动加上--lock-all-tables, 在备份文件22行包含change manster语句.

3) 备份某些库示例

```
# mysqldump -uu_test --master-data=2 --flush-logs --hex-blob --routines --triggers --database db_test db_test2 >
    /u01/backup/dump_2db.sql 2>&1 &
```

4) 备份某些表示例

```
# mysqldump -uu_test --hex-blob db_test t_test1 t_test2 > /u01/backup/dump_tables.sql 2>&1 &
```

12. 备份恢复： 逻辑恢复

1) 设置密码环境变量**,** 这样后续执行**mysql**时不用显示设置密码

export MYSQL_PWD=xxx

2) 恢复某个数据库数据

mysql -uu_test db_test < /u01/backup/db_test.sql

3) 恢复全部数据库数据

mysql -uu_test < /u01/backup/db_all.sql

13. 主从设置

Replication是一个异步的复制过程, 从一个**MySQL instace(**称之**Master)**复制到另一个**MySQL**实例**(**称之**Slave). 在Master**与**Slave**之间的实现整个复制过程主要由三个线程来完成, 其中两个线程**(SQL**线程和**IO**线程**)**在**Slave,** 另外一个线程**(IO**线程**)**在**Master.**

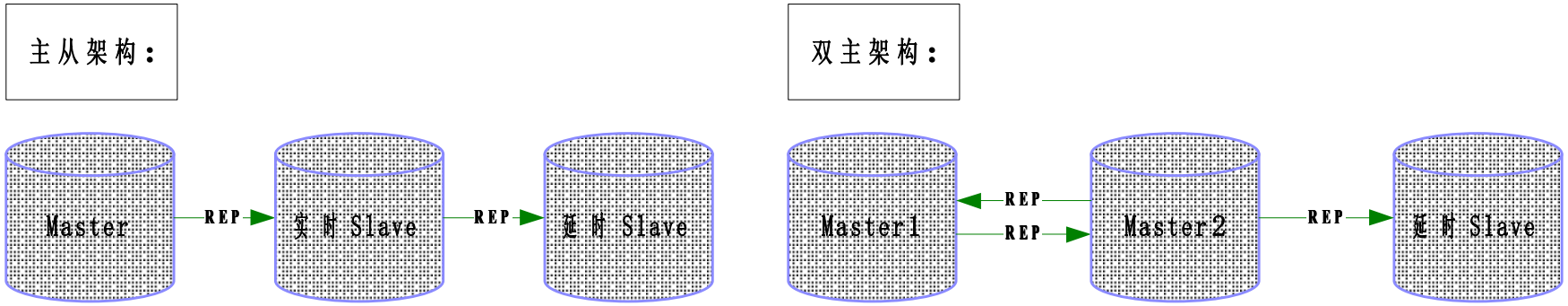
Replication的常用架构有以下几种**:**

常规复制架构(Master-Slave)

级联复制架构(Master-Slave-Slaves)

双主复制架构(Master-Master)

双主级联复制架构(Master-Master-Slaves)



14. 读写分离和切分

1) 业务逻辑层实现

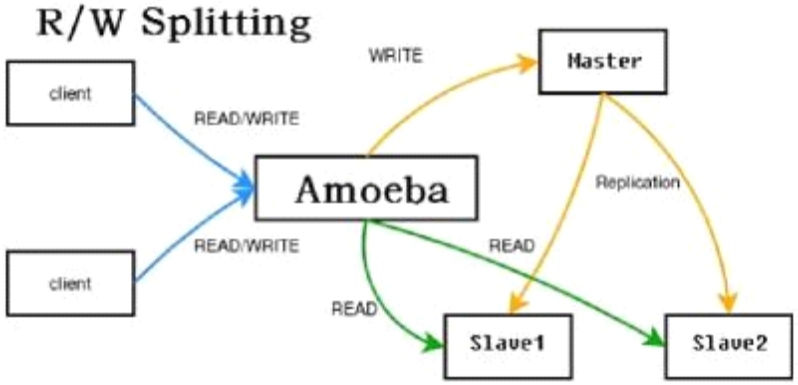
开发人员在程序里实现, 但工作量大, 维护成本高.

2) MySQL Proxy

MySQL Proxy是处在MySQL数据库客户和服务端之间的程序, 支持嵌入性脚本语言Lua. 可以用来分析,监控和变换(transform)通信数据, 它支持负载平衡和故障转移处理,查询分析和日志,查询重写(query rewriting),执行shell命令等功能. 其更强大的一项功能是实现"读写分离(Read/Write Splitting)".

3) Amoema for MySQL(推荐)

Amoeba(变形虫)项目, 致力于MySQL的分布式数据库前端代理层, 它主要在应用层访问MySQL的时候充当SQL路由功能. 在Client和DB Server(s)之间, 对客户端透明. 通过Amoeba能够完成多数据源的高可用,负载均衡,数据切片的功能, 目前Amoeba已在很多企业的生产线上面使用.



Amoema通过xml文件配置, 非常简单. 它可以在不同主机上启动多个, 并且做同样的配置来进行水平扩展, 以分担压力和提升可用性, 可以将Amoeba和MySQL装在同一台主机, 也可以装在不同的主机上, Amoeba本身不做数据缓存, 所以对于内存消耗很少, 主要是CPU占用.

15. 高可用性(HA)

1) Mysql Cluster集群

MySQL官方负载均衡产品, 通过NDB引擎实现, 支持在线增加节点来扩容.

2) Keepalived双机(简单)

Keepalived是一个基于VRRP协议来实现的高可用(HA)方案, 可以利用其来避免单点故障. 至少会有2台服务器运行Keepalived, 一台为主服务器(MASTER), 一台为备份服务器(BACKUP), 但是对外表现为一个虚拟IP, 主服务器会发送特定的消息给备份服务器, 当备份服务器收不到这个消息的时候, 即主服务器宕机的时候, 备份服务器就会接管虚拟IP, 继续提供服务, 从而保证了高可用性. 可以应用到任何HA的应用, 包括web服务,app服务等, 结合LVS或HAProxy还能实现负载均衡.

3) DRBD+Heartbeat负载均衡(应用广泛)

DRBD实现数据镜像, Hearbeat实现主从自动切换. DRBD是一种块设备, 可以被用于高可用环境之中. 它类似于一个网络RAID1功能 .

4) LVS+Heartbeat负载均衡

LVS实现负载均衡, Hearbeat实现主从自动切换. LVS 是LINUX VIRTUL SERVER的简称, 核心就是通过一组服务器来进行负载均衡. 通过前端的负载调度器, 无缝地将网络请求调度到真实服务器上, 从而使得服务器集群的结构对客户是透明的. 客户程序不受服务器集群的影响不需作任何修改.

5) MMM高可用(Google)

在Master-Master-Slaves或Master-Master架构的基础上, 通过MMM实现Failover和读写分离. MMM环境至少包括2台数据库(双Master)和一台监控主机, 可以有多个Slave.

16. 自动化监控

1) 人工编写脚本

开发成本和维护成本高.

2) Nagios(推荐)

插件丰富;
可自编写插件(Perl或shell);
邮件实时报警;
短信报警(运营商手机邮箱);
生成性能趋势图(PNP插件).
MySQL监控插件: check_mysql_health
MySQL监控自定义插件: check_mysql_remote

3) Cacti

性能趋势动态分析;
结合Nagios的完美解决方案.

5x8工作制DBA支持多个7x24数据库系统支持:

完善的高可用架构 和 自动的监控机制 是必需的.

Monitoring

- Tactical Overview
- Host Detail
- Service Detail
- Hostgroup Overview
- Servicegroup Overview
- Status Map
- 3-D Status Map
- Service Problems
- Host Problems
- Network Outages
- Comments
- Downtime
- Process Info
- Performance Info
- Scheduling Queue

Reporting

- Trends
- Availability
- Alert Histogram
- Alert History
- Alert Summary
- Notifications
- Event Log

Configuration

- View Config

Host	Service	Status	Last Check	Duration	Attempt	Status Information
157-lk-home	MYSQL:Connected threads	OK	2011-08-11 15:36:55	43d 0h 6m 58s	1/1	OK - threads_connected is 2
	MYSQL:Connected time	OK	2011-08-11 15:37:13	21d 3h 40m 18s	1/3	OK - 0.02 seconds to connect as u_test
	MYSQL:Slave IO thread	OK	2011-08-11 15:38:54	16d 5h 10m 2s	1/3	OK - Slave io is running
	MYSQL:Slave SQL thread	OK	2011-08-11 15:38:52	43d 0h 6m 11s	1/3	OK - Slave sql is running
	MYSQL:Sync behind master	OK	2011-08-11 15:36:56	12d 13h 32m 40s	1/3	OK - Slave is 0 seconds behind master
164-dbserver4	MYSQL3350:Connected threads	OK	2011-08-11 15:36:56	42d 4h 58m 50s	1/1	OK - threads_connected is 2
	MYSQL3350:Connected time	OK	2011-08-11 15:36:56	7d 5h 38m 36s	1/3	OK - 0.03 seconds to connect as u_test
	MYSQL3350:Slave IO thread	OK	2011-08-11 15:38:54	37d 0h 7m 31s	1/3	OK - Slave io is running
	MYSQL3351:Connected threads	OK	2011-08-11 15:34:54	43d 0h 0m 24s	1/1	OK - threads_connected is 2
	MYSQL3351:Connected time	OK	2011-08-11 15:36:56	7d 5h 38m 36s	1/3	OK - 0.02 seconds to connect as u_test
	MYSQL3351:Slave IO thread	OK	2011-08-11 15:38:54	27d 4h 6m 52s	1/3	OK - Slave io is running
177-new	MYSQL:Aborted connections per sec	OK	2011-08-11 15:39:32	10d 2h 51m 11s	1/3	OK - 0.02 aborted connections/sec
	MYSQL:Aborted connections per sec for client died	OK	2011-08-11 15:35:44	43d 0h 0m 7s	1/3	OK - 0.00 aborted (client died) connections/sec
	MYSQL:Cached threads	OK	2011-08-11 15:35:54	16d 5h 37m 54s	1/3	OK - 126 cached threads
	MYSQL:Connected threads	OK	2011-08-11 15:35:44	10d 2h 50m 0s	1/1	OK - threads_connected is 6
	MYSQL:Connected time	OK	2011-08-11 15:39:32	10d 2h 41m 54s	1/3	OK - 0.02 seconds to connect as u_test
	MYSQL:Created threads per sec	OK	2011-08-11 15:35:44	10d 2h 52m 11s	1/3	OK - 0.00 threads created/sec
	MYSQL:InnoDB buffer pool hitrate	OK	2011-08-11 15:38:19	16d 5h 37m 55s	1/12	OK - innodb buffer pool hitrate at 100.00%
	MYSQL:InnoDB buffer pool waits for clean page available	OK	2011-08-11 15:39:32	16d 5h 37m 54s	1/3	OK - 0 innodb buffer pool waits in 300 seconds (0.0000/sec)
	MYSQL:InnoDB log waits for a too small log buffer	OK	2011-08-11 15:37:13	16d 5h 35m 44s	1/3	OK - 0 innodb log waits in 300 seconds (0.0000/sec)
	MYSQL:MyISAM key cache hitrate	OK	2011-08-11 15:39:32	16d 5h 4m 6s	1/12	OK - myisam keycache hitrate at 100.00%

~完~

Blog: http://hi.baidu.com/edeed/blog

Weibo: http://weibo.com/edeed