# 

# 基本信息

## 主页

http://dev.mysql.com/

当前最新版本5.6.20

## 发行版

每个版本3种类型

\*Standard ：推荐大多数用户下载

\*Max：包含一些还没有正式发布的新特性，用于提升用户的认识和体验

\*Debug：和standard类似，但包含了一些调试信息，会影响性能；

# 安装

# mysql-5.7.18-winx64的安装

解压，并创建data目录

## 修改ini

[client]

port=3306

default-character-set=utf8

[mysqld]

port=3306

character\_set\_server=utf8

basedir=D:/java/mysql-5.7.18-winx64

datadir=D:/java/mysql-5.7.18-winx64/data

#tmpdir=D:/java/mysql-5.7.18-winx64/tmp

#default-storage-engine=INNODB

explicit\_defaults\_for\_timestamp=true

#sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

#max\_connections=200

#skip-grant-tables

[WinMySQLAdmin]

Server=D:/java/mysql-5.7.18-winx64/bin/mysqld.exe

## 注册服务

mysqld install mysql defaults-file="D:\java\mysql-5.7.18-winx64\my.ini"

## 初始化

mysqld --initialize

mysqld --initialize-insecure这个命令不会创建密码

## 启动服务

net start mysql

## 修改密码

**mysqladmin命令修改密码：**

mysqladmin -u root password "123456"

mysqladmin -u root -p password "123456"

**登录修改密码：**

从初始化日志文件中找到初始化密码，.err日志文件中搜索password关键字

mysql -uroot -p

mysql>SET password = PASSWORD('123456');

mysql>FLUSH PRIVILEGES;

[mysqld]条目下加一条命令skip-grant-tables,然后重启mysql，可以免密码扥估

update user set authentication\_string=password("123456") where user="root";

**update表修改密码：**

mysql>use mysql;

mysql>UPDATE user SET authentication\_string =PASSWORD("123456") WHERE user='root';

mysql>FLUSH PRIVILEGES;

## 参考命令

注册mysql服务：mysqld install mysql defaults-file="D:\java\mysql-5.6.36-winx64\my.ini"

初始化mysql ：mysqld --initialize

启动MYSQL服务：net start mysql

停止MYSQL服务：net stop mysql

移出mysql服务：mysqld remove; mysqld -remove; mysqld --remove

# 启动

service mysqld start

service mysqld stop

service mysqld restart

windows

net start mysql

net stop mysql

# 客户端连接

mysql -hip -uroot -ppwd --default-character-set=utf8 //密码前不能有空格

mysql -uadmin -padmin

mysql -uadmin -padmin

# 存储引擎

## MyISAM

数据文件有3个:

\*.frm 表定义

\*.myd 数据文件

\*.myi 索引文件

# 用户操作

## 删除用户

mysql> use mysql

delete from user where user = 'admin';

## 修改密码

mysqladmin -u用户名 -p旧密码 password 新密码

1给root添加密码

mysqladmin -u root -password 123456 //开始root没有密码，所以-p省略。

mysqladmin -u root -poldpwd password 123456 //再次修改秘密

## 设置密码update

mysql> use mysql;

mysql> UPDATE user SET Password=PASSWORD('root') where USER='root';

mysql> flush privileges;

## 创建用户

insert into

mysql.user(Host,User,Password) values("%","admin",password("admin"));

5.7.x以后

INSERT INTO

mysql.user(HOST,USER,authentication\_string) VALUES("%","admin",PASSWORD("admin"));

## 用户授权

grant all privileges on \*.\* to 'admin'@'%' identified by '123qwe!@#';

flush privileges;

# 基本命令

## 查看mysql版本

Select @@Version

## 查看mysql默认编码

show variables like 'character%';

show variables like "character%";

..

## 设置MySQL编码

MySQL(版本是5.1.22)支持多种编码格式，并且可以在多个级别上设置不同的编码格式。

服务器级：在安装MySQL时可以设置服务器的默认编码格式

最简单的修改方法，就是修改mysql的my.ini文件中的字符集键值，

如：

default-character-set = utf8

character\_set\_server = utf8

修改完后，重启mysql的服务，service mysql restart

数据库级：在创建数据库时可以指定数据库的编码格式，如果不指定，则默认继承服务器的编码格式。示例如下：

CREATEDATABASE db\_name DEFAULT CHARACTER SET utf8;

表级：在MySQL中可以对同一个数据库中表指定不同的编码格式，如果不指定将继承数据库的编码格式。示例如下：

CREATETABLE `db\_name`.`tb\_name`(

id VARCHAR(20)NOTNULL,

name VARCHAR(20)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

列级：另外还可以对同一个表中的不同列指定不同的编码格式，示例如下：

CREATETABLE `db\_name`.`tb\_name`(

id varchar(20)NOTNULL,

name varchar(20) CHARACTER SET utf8

);

制定编码方式的一个好处是，增强对中文的处理，可解决表中部分中文字段内容在不同工具下乱码的问题

## SHOW STATUS

<http://blog.haohtml.com/archives/2872>

<http://blog.haohtml.com/archives/4561>

<http://blog.haohtml.com/archives/4176>

## show variables

## 查看设置最大错误数

mysql>show variables like 'max%';

mysql> setglobal max\_error\_count=1000;

或mysql配置文件中添加了max\_connect\_errors=1000

## 解锁用户

mysqladmin flush-hosts;

mysql>flushhosts;

mysqladmin flush-hosts -h172.16.233.126 -uadmin -padmin -P3306;

# mysql架构：

连接层：

服务层：业务逻辑处理层

引擎层：可拔插存储引擎层

存储层：文件存储层

# 支持的存储引擎

|  |
| --- |
| mysql> show engines;  +--------------------+---------+--------------------------------------------------------------+--------------+------+------------+  | Engine | Support | Comment | Transactions | XA | Savepoints |  +--------------------+---------+--------------------------------------------------------------+--------------+------+------------+  | InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign keys | YES | YES | YES |  | MRG\_MYISAM | YES | Collection of identical MyISAM tables | NO | NO | NO |  | MEMORY | YES | Hash based, stored in memory, useful for temporary tables | NO | NO | NO |  | BLACKHOLE | YES | /dev/null storage engine (anything you write to it disappears) | NO | NO | NO |  | MyISAM | YES | MyISAM storage engine | NO | NO | NO |  | CSV | YES | CSV storage engine | NO | NO | NO |  | ARCHIVE | YES | Archive storage engine | NO | NO | NO |  | PERFORMANCE\_SCHEMA | YES | Performance Schema | NO | NO | NO |  | FEDERATED | NO | Federated MySQL storage engine | NULL | NULL | NULL |  +--------------------+---------+--------------------------------------------------------------+--------------+------+------------+  9 rows in set (0.02 sec)  mysql> |

MyISAM:不支持事物，外面，表锁，不适合高并发，文件小，缓冲索引不缓冲数据，关注性能；

InnoDB：支持事物，外面，行锁，适合高并发，文件大，缓冲索引和数据，对内存要求高，关注事物；

# 默认的存储引擎

|  |
| --- |
| mysql> show variables like '%storage\_engine%';  +----------------------------------+--------+  | Variable\_name | Value |  +----------------------------------+--------+  | default\_storage\_engine | InnoDB |  | default\_tmp\_storage\_engine | InnoDB |  | disabled\_storage\_engines | |  | internal\_tmp\_disk\_storage\_engine | InnoDB |  +----------------------------------+--------+  4 rows in set, 1 warning (0.01 sec)  mysql> |

# 字符集

## character set 与 collation

character set是字符集，collation是比对方法；

每个 character set 会对应一定数量的 collation. 查看方法

mysql> show collation;

## 服务器级别

查看

SHOW GLOBAL VARIABLES LIKE 'character\_set\_server';

SHOW GLOBAL VARIABLES LIKE 'collation\_server';

设置

/etc/mysql/my.cnf[mysqld]

character\_set\_server=utf8

collation\_server=utf8\_general\_ci

## 数据库级

查看

SELECT \* FROM information\_schema.schemata WHERE schema\_name = 'rainbowhealthline';

## 表级

查看

SHOW CREATE TABLE course;

显式设置：在创建表时指定

CREATE TABLE utf ( id INT ) DEFAULT CHARSET=utf8 DEFAULT COLLATE=utf8\_bin;

## 列级

查看

SHOW CREATE TABLE course;

显式设置：在创建表时指定

CREATE TABLE Table1(column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1\_german1\_ci);

## 连接级别

查看

SHOW VARIABLES LIKE 'character\_set\_client'; # 服务端使用这个编码来理解客户端发来的statements

SHOW VARIABLES LIKE 'character\_set\_connection' ; # 我还不知道什么意思，等看了mysql源码再说

SHOW VARIABLES LIKE 'character\_set\_results'; # 服务端使用这个编码回送结果集和错误信息

# AliSQL

使用PerconaMysql，存储引擎XtraDB

# 数据文件

.frm文件 存放表结构

.myd文件 存放数据

.myi文件 存放索引

# #日志操作

错误日志： -log-err

查询日志： -log

慢查询日志: -log-slow-queries

更新日志: -log-update

二进制日志： -log-bin

## 是否启用了日志

mysql> show variables like 'log\_%';

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

| Variable\_name | Value |

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

| log\_bin | OFF |

| log\_bin\_basename | |

| log\_bin\_index | |

| log\_bin\_trust\_function\_creators | OFF |

| log\_bin\_use\_v1\_row\_events | OFF |

| log\_builtin\_as\_identified\_by\_password | OFF |

| log\_error | D:\java\mysql-5.7.18-winx64\data\hc-PC.err |

| log\_error\_verbosity | 3 |

| log\_output | FILE |

| log\_queries\_not\_using\_indexes | OFF |

| log\_slave\_updates | OFF |

| log\_slow\_admin\_statements | OFF |

| log\_slow\_slave\_statements | OFF |

| log\_statements\_unsafe\_for\_binlog | ON |

| log\_syslog | ON |

| log\_syslog\_tag | |

| log\_throttle\_queries\_not\_using\_indexes | 0 |

| log\_timestamps | UTC |

| log\_warnings | 2 |

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

19 rows in set, 1 warning (0.00 sec)

## 二进制日志log-bin

[mysqld]

log-bin=d:/mysql/data/mysqlbin 主要用于主从复制，配置了即表示启动

log-err=d:/mysql/data/mysqlerr

## 错误日志log-err

错误日志：默认是关闭的,为了保证简洁和高效

记录严重的警告错误信息，每次启动和关闭的详细信息，类似tomcat的catalina.out

log-err=d:/mysql/data/mysqlerr 配置了即表示启动

## 怎样知道当前的日志

mysql> show master status;

## 显示二进制日志数目

mysql> show master logs;

看二进制日志文件用mysqlbinlog

shell>mysqlbinlog mail-bin.000001

或者shell>mysqlbinlog mail-bin.000001 | tail

## 通用查询日志

#log=/usr/local/mysql/log/alllog.log

#或log=/var/log/mysql/mysql.log

log-output=FILE

--log[=file\_name]或-l [file\_name]选项启动它。

如果没有给定file\_name的值，默认名是host\_name.log

## 错误日志

--log-error

# 慢查询日志

默认不开启慢查询，记录查询的sql语句；

如果不是调优需要，一般不建议启用该参数，会有性能影响；

## 查看是否开启慢查询日志

|  |
| --- |
| **mysql> show variables like '%slow\_query\_log%';**  +---------------------+-------------------------------------------------+  | Variable\_name | Value |  +---------------------+-------------------------------------------------+  | slow\_query\_log | OFF |  | slow\_query\_log\_file | D:\java\mysql-5.7.18-winx64\data\hc-PC-slow.log |  +---------------------+-------------------------------------------------+  2 rows in set, 1 warning (0.01 sec)  mysql> |

## 开启/关闭慢查询日志

**通过命令**

set global slow\_query\_log=1; #临时开启

**通过启动参数**

mysqld.exe --slow-query-log

**通过配置文件**

[mysqld]

sloq\_query\_log = 1

slow\_query\_log\_file = /var/mysql/data/log\_slow.log

缺省文件名是: 主机名-slow.log

## 查询/设置慢查询阈值

**查询**

**mysql> show variables like 'long\_query\_time%';**

**mysql> show global variables like 'long\_query\_time%';**

|  |
| --- |
| +-----------------+-----------+  | Variable\_name | Value |  +-----------------+-----------+  | long\_query\_time | 10.000000 |  +-----------------+-----------+  1 row in set, 1 warning (0.03 sec) |

**命令行**

SET long\_query\_time=10; #设置慢查询时间1s

set global long\_query\_time=10; #重启会重置；

**参数**

long\_query\_time = 10 #10秒

## 制造慢查询示例

select sleep(4);

## 查看慢查询状态/统计

可用于健康检查

#select sleep(4);

**mysql> show status like 'slow%'; #当前会话**

**mysql> show global status like 'slow%'; #全局**

|  |
| --- |
| mysql> show global status like 'slow%';  +---------------------+-------+  | Variable\_name | Value |  +---------------------+-------+  | Slow\_launch\_threads | 0 |  | Slow\_queries | 4 | 慢查询次数  +---------------------+-------+  2 rows in set (0.04 sec)  mysql> show status like 'slow%';  +---------------------+-------+  | Variable\_name | Value |  +---------------------+-------+  | Slow\_launch\_threads | 0 |  | Slow\_queries | 1 |  +---------------------+-------+  2 rows in set (0.01 sec) |

## 慢查询日志分析命令mysqldumpslow

linux环境

mysqldumpslow --help;

以下命令组合more使用防止刷屏

用例：返回记录集最多的10个SQL

mysqldumpslow -s r -t 10 /var/mysql/data/xxx-slow.log

用例：查询访问次数最多的10个SQL

mysqldumpslow -s c -t 10 /var/mysql/data/xxx-slow.log

用例：查询按照时间排序的前10条里面含有左连接的查询语句

mysqldumpslow -s t -t 10 -g”left join” /var/mysql/data/xxx-slow.log

# 全局查询日志

记录所有sql，绝对不要在生产环境使用

**my.cnf**

general\_log=1 #开启

general\_log\_file=/path/logfile #日志路径

log\_output=FILE|TABLE #记录到文件还是表

**cmd**

set global general\_log=1;

set global log\_output='TABLE';

查看结果

select \* from mysql.general\_log;

# 性能下降原因分析

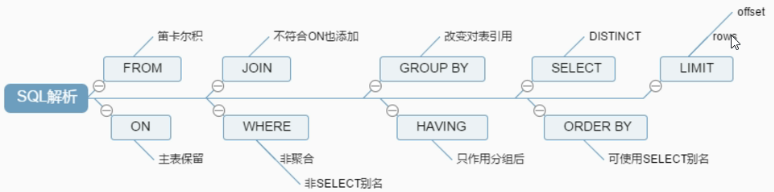
Sql语句烂

索引失效(单值索引，复合索引)

join太多，设计缺陷或变态需求

服务器参数设置不够优化：缓冲，线程数等；

# SQL执行顺序



# 7种JOIN关系

|  |  |  |
| --- | --- | --- |
| 左链接 |  | select \* from A left join B on a.key = b.key |
|  |  | select \* from A left join B on a.key = b.key where b.key is null |
| 右链接 |  | select \* from A right join B on a.key = b.key |
|  |  | select \* from A right join B on a.key = b.key where a.key is null |
| 内联 |  | select \* from A inner join B where a.key = b.key |
| 全链接  MySql不支持 |  | select \* from A full outer join B on a.key = b.key  MySql不支持，可以通过union达到相同效果； |
|  |  | select \* from A full outer join B on a.key = b.key where a.key is null or b.key is null |

# 常用性能分析

查询优化

Mysql常见瓶颈：CPU，IO，通过top, free, iostat, vmstat查看性能状态；

CPU饱和一般发生在数据装入内存或从磁盘读数据

IO瓶颈一般发生在装入数据远大于内存容量的时候

# Explain

## id

序列号，表示查询中执行select字句或操作表的顺序；

值有三种，相同，不同，相同不同共存

相同，表示从上到下顺序加载表

如果是子查询，ID大的先执行

优化：

小表驱动大表

## select\_type

数据读取操作的操作类型；

查询类型主要用于区别普通查询，联合查询，子查询，嵌套查询等的复杂查询；

常见6个值

simple 简单查询

primary

subquery select或where中包含的子查询

derived 在from中包含的子查询被标记为衍生表derived，结果放在临时表中

union union后面的select查询

union result 从union表获取结果

临时表胡增加系统负担，最好能避免；

第二个子查询出现在union之后，标记为union

union包含在from字句中，外出select被标记为derived；

## table

derivedn 衍生，虚表; n表示第n个ID执行出来的虚表；

## type

字面意思是访问类型，实际上代表了性能高低

常见8种值，下列从最好到最差

NULL > system > const > eq\_ref > ref > range > index > ALL

全部

system > const > eq\_ref > ref > fulltext >ref\_or\_null > index\_merge > unique\_subquery > index\_subquery > range > index > ALL

system 系统表，或者只有一条记录的虚表

const 通过唯一索引只匹配一行数据；

eq\_ref 唯一索引扫描，对于每个索引建，表中只有一条记录匹配；

ref 非唯一索引扫描；

fulltext >ref\_or\_null > index\_merge > unique\_subquery > index\_subquery

range 按索引范围检索

index 查询字段全在索引上，全索引扫描Full Index Scan

ALL 全表扫描

## partitions type

## possible\_keys

显示可能被用到的索引，但不一定会被使用；

查询中如果用到覆盖索引，则该索引只出现在key中，比如当type=index时候，

select查询的字段与复合索引顺序与个数都一致，

## key

实际使用的索引

如果为NULL，没有使用或没有索引

## key\_len

索引字段最大可能长度，根据表定于计算；

## ref

显示索引的哪一列被使用了，如果可能的话是一个常数。哪些列或常量被用于查找索引列上的值；

表之间的引用；

## rows

大致估算找到所需要的记录需要扫描的数据行数；

每张表有多少行被优化器查询

## filtered

## Extra

其它重要的信息

### Using filesort:

未使用索引完成排序，尽量不要出现；

说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序读取。mysql中无法利用索引完成的排序操作称为“文件排序”

### Using temporary:

产生了临时表，需要避免；

一般在对结果排序的时候会使用临时表，常见order by，group by。

### Using index:

好事

表示select使用了覆盖索引covering index，避免了访问表数据行；

### Using where:

不用全表扫描，仅通过索引就可以获取所有数据；

### Using join buffer:

### impossible where

where子句的只总是false，不可能获取数据。比如where c1=”a” and c1=”b”

### select table optimized away

在没有group by子句的情况下，基于索引优化min/max操作，不必等到执行间断再进行计算；

查询执行计划生成的阶段即完成优化；

### distinct

优化distinct操作，在找到第一匹配的元组后即停止查找同样的值动作 ；

### no tables

# 索引

mysql索引是B树索引

索引的两大功能：查找和排序；

优势：降低IO成本，通过索引排序降低排序成本，降低CPU消耗；

劣势：占用空间，降低更新表速度(增加)

需要花时间研究建立最优秀的索引或优化查询；

索引数量不宜过多

# 索引操作

## 创建唯一/全文/普通索引

create [unique|fulltext] index idx\_name on tbl\_name(col1[(length)][ASC|DESC],...);

alter table tbl\_name add index [idx\_name](col1,col2);

## 添加主键索引

alter table tbl\_name add primary key (c1,c2);

## 删除索引

drop index idx\_name on tbl\_name;

alter table tbl\_name drop index idx\_name;

## 删除主键

alter table emp drop primary key;

## 显示索引

show index(es) from tbl\_name;

show index from tbl\_name;

show keys from tbl\_name;

desc tbl\_name;

# 索引结构

## BTree索引

## Hash索引

## Full-text索引

## R-Tree索引

# 索引注意事项

**适合创建索引情况**

主键自动创建唯一索引

频繁作为查询条件的字段应该创建索引

查询中与其他表关联的字段，外键关系建立索引

高并发下倾向组合索引

查询中排序的字段通过索引会大大提高排序速度，

分组的字段

**不适合建索引的情况：**

表记录少

数据重复且平均分配

经常增删改的字段不适合创建索引

## 注意事项：

频繁查询的字段；

唯一性太差的字段不适合加索引；

列值频繁变化的字段不适合加索引;

like “%aa”不会用索引；like “aa%”会用索引；

类型不一致不走索引；

or 不走索引？实测两边都是索引的时候,type=rang,或index\_merge,Extra=Using union

or的一方不是索引，则不走索引；

多列索引，要从最左边开始

mysql预估全表扫描比索引快时也不走索引；比如数据少；

# 索引分类

## 主键索引：primary

alter table emp add primary key(empno);

alter table emp drop primary key;

## 唯一索引：unique

create unique index on tablename(name)

索引值必须唯一，但允许null

## 普通索引：index

create index on tablename(name)

### 全文索引：fulltext

FULLTEXT，用于分词,只有MyISAM支持

create fulltext index on tablename(name)

## 复合索引：

create index on tablename(col1,col2)

# 查询表所有的索引

show indexes from table1

# 查看索引的使用情况

SHOW STATUS LIKE "Handler\_read%"

Variable\_name Value

Handler\_read\_first 89

Handler\_read\_key 156 表示使用索引次数

Handler\_read\_last 1

Handler\_read\_next 531

Handler\_read\_prev 49

Handler\_read\_rnd 62

Handler\_read\_rnd\_next 3804792 值大说明效率低

# JOIN索引优化

book (id,card);

class(bookid,card);

phone(phoneid,card);

select \* form book left join class on book.card=class.card left join phone on class.card=phone.card;

小表驱动大表

优先优化内存查询

保证被驱动的表join条件被索引

## 单表

rang类型的字段后面索引失效

如：select \* from t where a=1 and b>1 oeder by c;

create index on t(a,b,c); bad

create index on t(a,c); good

## 两表

左右链接相反建;

left join条件用于确定如何从右表搜索行，左边一定都有，索引应该加在右表上；

right join与left join一样道理，索引应该建立咋左表上；

左右可以互换

select \* from t1 left join t2 on...

select \* from t2 right join t1 on...

## 三表

# 索引优化/失效

1 全值匹配我最爱：按顺序等于条件使用索引；

2 最佳左前缀法则：索引有多列，查询要从最左列开始，不能跳跃中间列(否则部分使用索引)；

3 不在索引上做任何操作(计算，函数，类型转换)

4 范围条件右边的列不会被使用。范围本身也不利用索引解锁，但可以用于排序；

5 尽量使用覆盖索引(索引列与查询列一致)

6 使用不等于(!=, <>)无法利用索引，导致全表扫描

7 is null, is not null 也无法使用索引

8 like通配符开头(‘%xx’)无法使用索引，like%在右边可以，如like “abc%”

覆盖索引可以利用, 索引是name+age,查询SQL可以是select id,name,age where name like “%aaa%”;

9 字符串不加单引号，索引失效

10 少用or，用它来连接是索引失效

select \* from staff where name = ‘June’ or name=’abc’;/不会利用索引；

综上：只有完全匹配和前缀匹配(“avc%”)才能使用索引；

覆盖索引页可以利用

如果必须使用like %abc%,变换； like “abc%”（后续列索引不失效）

当where中的条件与索引顺序不一致时，mysql查询优化器会自动优化排序；

索引c1,c2,c3,c4时：

select \* from t where c1=xx and c2 like xx% and c3=xx; //会利用3个列索引

select \* from t where c1=xx and c2 like %xx and c3=xx; //会利用1个列索引

select \* from t where c1=xx and c2=xx order by c3; //会利用索引排序

select \* from t where c1=xx and c2=xx and c4=xx order by c3; //同上

select \* from t where c1=xx and c2=xx order by c4; //导致内排序，出现Using filesort

select \* from t where c1=xx and c2=xx order by c3, c4; //use index

select \* from t where c1=xx and c2=xx order by c4, c3; //出现Using filesort

select \* from t where c1=xx and c2=xx order by c3, c2; //如果结果集中c2是常量，不会Using filesort

group by：分组之前比排序

select \* from 他where c1=xx and c4=xx group by c2, c3; //use index

select \* from 他where c1=xx and c4=xx group by c3, c2; //Using filesort，Using filesort



# 优化口诀：

全值匹配我最爱，最左前缀要遵守；

带头大哥不能死，中间兄弟不能断；

索引列上别计算，范围之后全失效；

like百分写右边，覆盖索引不写星；

不等空值还有or,索引失效要少用；

字符引号不可丢，SQL高级也不难；

# 查询优化

## 小表驱动大表

select \* from A where id in(select id from B); //B是小表，使用in比较好；

select \* from A where exists(select 1 from B where B.id = A.id); //A是小表，exists较好

exists返回true|false，exist是将单边的in查询变成子查询；

## order by优化

不要产生Using filesort

会使用Using index方式：

order by语句使用索引最左前列

where字句与order by子句条件组合满足最左前列；

order by默认升序；与索引顺序不符合时，也会Using index？？？

order by a asc, b desc; 不能使用索引排序

## fileSort有两种算法

**双路排序**

mysql4.1之前使用的是双路排序，两次磁盘扫面，最终得到数据。

第一次读取行执行和orderby列排序，

第二次然后再扫描已经排好序的列表，按照列表中的值重新读取对应的数据输出；

总结：从磁盘读取排序字段，在bugger进行排序，在从磁盘读取其它字段；

**单路排序**

从磁盘读取查询需要的所有列，按照orderby在buffer中排序，然后扫描排序后的列表进行输出；

把随机IO变成顺序IO。

但需要跟多的空间；

如果一次拿不完效率就比双路还差；sort\_buffer过小，导致多次IO，tmp文件

**优化策略：**

增大sort\_buffer\_size

增大max\_length\_for\_sort\_data

有orderby时不要用select \*，以较少buffer

**为排序使用索引**

## group by优化

实际上是先排序后分组，规则同orderby；

能在where中搞定的不要写having；

# 调优步骤

# 分析，观察，至少跑1天，看看慢SQl

1 开启慢查询日志并捕获慢查询日志

2 explain +慢查询SQL分析

3 show profile查询SQL在mysql服务器里面的执行细节和生命周期情况

4 服务器参数调优

# show profile

是mysql提供的可以用来分析当前会话中语句执行的资源消耗情况。

可用于SQL的调优的测量

这个命令只是在本会话内起作用，即无法分析本会话外的语句。

开启分析功能后，所有本会话中的语句都被分析（甚至包括执行错误的语句），除了SHOW PROFILE和SHOW PROFILES两句本身。

## 查询profiling状态

|  |
| --- |
| mysql> show variables like 'profiling';  +---------------+-------+  | Variable\_name | Value |  +---------------+-------+  | profiling | OFF |  +---------------+-------+  1 row in set, 1 warning (0.01 sec) |

## set profiling设置

set profiling = 1;

set profiling = on;

整常运行SQL

select \* from emp group by id%10 limit 150000;

select \* from emp group by id%20 order by 5;

## show profiles查询

mysql> show profiles; #sessio级别

|  |
| --- |
| +----------+------------+----------------------------------+  | Query\_ID | Duration | Query |  +----------+------------+----------------------------------+  | 1 | 0.02591975 | show variables like 'profiling' |  | 2 | 0.00062875 | show tables |  | 3 | 0.38335550 | select \* from user where age = 2 |  | 4 | 0.36139375 | select \* from user where age = 5 |  | 5 | 0.00023800 | show profiling |  | 6 | 0.00022400 | show global profiles |  +----------+------------+----------------------------------+  6 rows in set, 1 warning (0.00 sec) |

## show profile查询

mysql> **show profile** cpu,block io **for query 11;**

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

| Status | Duration | CPU\_user | CPU\_system | Block\_ops\_in | Block\_ops\_out |

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

| starting | 0.000091 | 0.000000 | 0.000000 | NULL | NULL |

| checking permissions | 0.000010 | 0.000000 | 0.000000 | NULL | NULL |

| Opening tables | 0.000024 | 0.000000 | 0.000000 | NULL | NULL |

| init | 0.000035 | 0.000000 | 0.000000 | NULL | NULL |

| System lock | 0.000013 | 0.000000 | 0.000000 | NULL | NULL |

| optimizing | 0.000012 | 0.000000 | 0.000000 | NULL | NULL |

| statistics | 0.000023 | 0.000000 | 0.000000 | NULL | NULL |

| preparing | 0.000016 | 0.000000 | 0.000000 | NULL | NULL |

| executing | 0.000006 | 0.000000 | 0.000000 | NULL | NULL |

| Sending data | 0.649578 | 0.655204 | 0.000000 | NULL | NULL |

| end | 0.000018 | 0.000000 | 0.000000 | NULL | NULL |

| query end | 0.000014 | 0.000000 | 0.000000 | NULL | NULL |

| closing tables | 0.000025 | 0.000000 | 0.000000 | NULL | NULL |

| freeing items | 0.000420 | 0.000000 | 0.000000 | NULL | NULL |

| cleaning up | 0.000915 | 0.000000 | 0.000000 | NULL | NULL |

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

15 rows in set, 1 warning (0.00 sec)

## show profile参数

mysql> **show profile** type1,type2... **for query 11;**

如果没有指定FOR QUERY，那么输出最近一条语句的信息。

type是可选的，取值范围可以如下：

ALL 显示所有性能信息

CPU 显示用户CPU时间、系统CPU时间

BLOCK IO 显示块IO操作的次数

CONTEXT SWITCHES 显示上下文切换次数，不管是主动还是被动

IPC 显示发送和接收的消息数量

MEMORY [暂未实现]

PAGE FAULTS 显示页错误数量

SOURCE 显示和Source\_function，Source\_file，Source\_line相关的开销信息

SWAPS 显示交换次数相关开销的信息

## show profile结果分析

上述如果出现下列4项内容，表示需要优化了；

converting HEAP to MyISAM 查询结果太大，内存不够用

Creating tmp table 创建临时表

Copying to tmp table on disk 把内训中临时表复制到磁盘

locked

# 锁

表锁，行锁，页锁

## 查看打开的表手否有锁

show open tables;

|  |
| --- |
| mysql> show open tables;  +--------------------+------------------------------------------------------+--------+-------------+  | Database | Table | In\_use | Name\_locked |  +--------------------+------------------------------------------------------+--------+-------------+  | mysql | engine\_cost | 0 | 0 |  | mysql | help\_keyword | 0 | 0 | |

## 对表加表锁

lock table t1 read|write, t2 read|write,...;

示例1：读锁是共享锁，session1 对t1 添加了读锁以后：

session1可以对t1读，不能对t1写，也不能对其它表读写；

session2可以对t1读，不能对t1写(将一直等待)

示例2：写锁是排他锁，session1 对t1 添加了写锁以后：

session1可以对t1读和写，也不能对其它表读写；

session2不能对t1读和写 (将一直等待)

## 解锁

unlock tables;

## 锁统计

mysql> show status like 'table%';

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

| Variable\_name | Value |

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

| Table\_locks\_immediate | 176 |表示可以立即获取锁的查询次数；

| Table\_locks\_waited | 0 |出现表级锁挣用而等待的次数，数值高，表示有锁竞争；

| Table\_open\_cache\_hits | 14 |

| Table\_open\_cache\_misses | 4 |

| Table\_open\_cache\_overflows | 0 |

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

5 rows in set (0.00 sec)

## 行锁

## 行锁升级死锁

update的时候，如果索引失效，行锁会变成表锁；

如：下语句c1是varchar型

update....where c1 = "2000"; //行锁

update....where c1 = 2000; //表锁

## 间隙锁为何

当使用范围条件而不是相等条件检索数据并请求共享或排他锁时，innoBD会给符合条件的所有数据的索引项加锁，对于条件范围内部存在的记录，叫“间隙GAP”；

InnoDB也会对这个间隙加锁，就是间隙锁Next-Key

当query是范围的时候，会锁定整个范围的索引值，某些场景对系统造成很大的隐患；

数据表没有id=2的值，

update t set c1=xxx where id>1 and id<10;

insert t values(2,xxx);上一句未提交时，此句会等待

## 锁定一行

begin

select \* where id=8 for update;

## 锁统计

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

下列统计都是从系统启动开始

|  |
| --- |
| mysql> show status like 'innodb\_row%';  +-------------------------------+-------+  | Variable\_name | Value |  +-------------------------------+-------+  | Innodb\_row\_lock\_current\_waits | 0 |当前正在等待锁的数量  | Innodb\_row\_lock\_time | 0 |锁定总时长  | Innodb\_row\_lock\_time\_avg | 0 |平均时长  | Innodb\_row\_lock\_time\_max | 0 |最大时长  | Innodb\_row\_lock\_waits | 0 |总共等待的次数  | Innodb\_rows\_deleted | 0 |  | Innodb\_rows\_inserted | 0 |  | Innodb\_rows\_read | 8 |  | Innodb\_rows\_updated | 0 |  +-------------------------------+-------+  9 rows in set (0.00 sec) |

当等待次数很高，且每次锁定时间也不小是，就需要分析优化了

show profile

# 事物

## ACID

A Atomicity 原子性

C Consistent 一致性

I Isolation 隔离性

D Durable 持久性

mysql默认隔离级别

## 并发事物带来的问题

更新丢失 Lost Update 最后的更新覆盖了前面事物所做的更新；

脏的 Dirty Read： 事物A读到B已修改但**未提交的数据**

不可重复读 Non-Repeatable Reads 事物A读到B**修改**的数据，事物A两次读不一致，不符合隔离性；

幻读 Phanton Reads 事物A读到B**新增**的数据

## 事物隔离级别

隔离级别------数据一致性--------------------------------------脏读----不可重复读----幻读

未提交读 只能保证不读取物理上损坏的数据 Y Y Y

已提交读 语句级别 N Y Y

可重复读 事务级 N N Y

可序列化 事务级，最高级 N N N

## mysql默认事物隔离级别

可重复读

mysql> show variables like 'tx\_%';

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

| Variable\_name | Value |

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

| tx\_isolation | REPEATABLE-READ |可重复读

| tx\_read\_only | OFF |

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

1. rows in set, 1 warning (0.01 sec)

## 设置客户端隔离级别

**set session transaction isolation level read uncommitted;**

**1.READ-UNCOMMITTED(读取未提交内容)**

**2.READ-COMMITTED（读取提交内容）**

**3.REPEATABLE-READ(可重读)**

**4.SERIERLIZED(可串行化)**

## 自动提交

mysql是自动提交的，可以关闭

set autocommit=0;

# 备份

# 恢复

mysql -uroot-h<ip>-p

mysql> use paas\_account

mysql> source /home/paas/publish/account\_02\_init.sql

# 库操作

查询所有的表

/usr/local/mysql/bin/mysql-hsp50a33124361af.mysql.rds.aliyuncs.com -usp50a33124361af-phhxx\_951753 -t--extended-insert=false --default-character-set=utf8--comments=1 paas\_activity >paas\_activity1.sql

# 导出

<http://www.jb51.net/article/28855.htm>

<http://wenku.baidu.com/view/b2df7ff04693daef5ef73d7d.html>

<http://wenku.baidu.com/view/06b4c371f242336c1eb95e9b.html>

<http://wenku.baidu.com/view/d83cc64ecf84b9d528ea7a2c.html>

<http://wenku.baidu.com/view/4b75e4a70029bd64783e2ca9.html>

mysqldump -hsp50a33124361af.mysql.rds.aliyuncs.com -usp50a33124361af -phhxx\_951753 --databases paas\_account> ./paas\_account.sql

mysqldump--databases paas\_account> ./paas\_account.sql

## mysqldump参数详解

-d, --no-data 不写入表的任何行数据,只表结构

-t不要结构, 只数据

-c, --complete-insert 使用完整的insert语句(用列名字)

--extended-insert=false 使用多条insert语句

--comments=0不要注释

--add-drop-database

--add-drop-table 在每个create语句之前增加一个drop table

--all-databases

--databases

## 1.导出整个数据库

mysqldump -u 用户名 -p 数据库名>导出的文件名

mysqldump -u user\_name -p123456 database\_name>outfile\_name.sql

## 2.导出一个表

mysqldump -u 用户名 -p 数据库名表名>导出的文件名

mysqldump -u user\_name -p database\_nametable\_name>outfile\_name.sql

/usr/local/mysql/bin/mysqldump -uroot -t paas\_storage storage\_store >storage\_store.sql

/usr/local/mysql/bin/mysqldump -uroot -t --extended-insert=false -B paas\_storage --table storage\_store >storage\_store.sql

## 3.导出一个数据库结构

mysqldump -u user\_name -p -d –add-drop-table database\_name>outfile\_name.sql

-d 没有数据 –add-drop-table 在每个create语句之前增加一个drop table

## 3.导出数据

rm -rf \*.sql

/usr/local/mysql/bin/mysqldump -hsp50a33124361af.mysql.rds.aliyuncs.com -usp50a33124361af-phhxx\_951753-t --extended-insert=false --default-character-set=utf8--comments=0–databasespaas\_activity>paas\_activity2.sql

/usr/local/mysql/bin/mysqldump -hsp50a33124361af.mysql.rds.aliyuncs.com -usp50a33124361af-phhxx\_951753 -t--extended-insert=false --default-character-set=utf8--comments=1 paas\_activity >paas\_activity1.sql

# 导入

连接mysql

>mysql--default-character-set=utf8-usp50a33124361af -hsp50a33124361af.mysql.rds.aliyuncs.com -p

mysql> source /data/hc/account\_02\_init\_\_oauth\_scope.sql

## 恢复数据库

mysqldump -uroot -p test\_db>test\_db.sql  //备份

mysql -uroot -p test\_db<test\_db.sql