# MySQL学习笔记

## **0.mysql安装**

### 1.D:\ProgramFiles\mysql-5.7.19-winx64下新建data/和my.ini

my.ini:

[mysql]

# 设置mysql客户端默认字符集

default-character-set=utf8

[mysqld]

#设置3306端口

port = 3306

# 设置mysql的安装目录

basedir=D:\ProgramFiles\mysql-5.7.19-winx64

# 设置mysql数据库的数据的存放目录

datadir=D:\ProgramFiles\mysql-5.7.19-winx64\data

# 允许最大连接数

max\_connections=200

# 服务端使用的字符集默认为8比特编码的latin1字符集

character-set-server=utf8

# 创建新表时将使用的默认存储引擎

default-storage-engine=INNODB

### 2.管理员打开cmd

### 3.mysqld -install //安装服务

### 4.mysqld --initialize //初始化

### 5.net start mysql //启动服务

### 6.zwj-PC.error 里查找temporary password 找到随机生成的密钥。

如：root@localhost: puw?edJEU5/S。puw?edJEU5/S就是密钥

### 7.修改密码：set password = password('zwj123456');

alter user 'root'@'localhost' password expire never;

flush privileges;

### 8.添加用户：create user 'zwj'@'localhost' identified by 'zwj153';

zwj153是密码，zwj是用户名

localhost是指该用户只能在本地登录。改为%表示一台电脑上都可以登录。

set password for 'username'@'host' = password ('newpassword');

### 9. create database zwj\_test;

### 10.授权： grant select,insert,update on zwj\_test.\* to 'zwj'@'localhost';

把数据库zwj\_test下所有表授权给zwj,让它可以select,insert,update（如果授权所有行为设为all）

all privileges包含以下权限：

select,

insert,

update,

delete,

create,

drop,

references,

index,

alter,

create temporary tables,

lock tables,

execute,

create view,

show view,

create routine,

alter routine,

event,

trigger

### 11.撤销授权：revoke select,insert,update on zwj\_test.\* form 'zwj'@'localhost';

### 12.删除用户：drop user 'username'@'host';

### 13.查看权限：show grants for zwj@localhost;

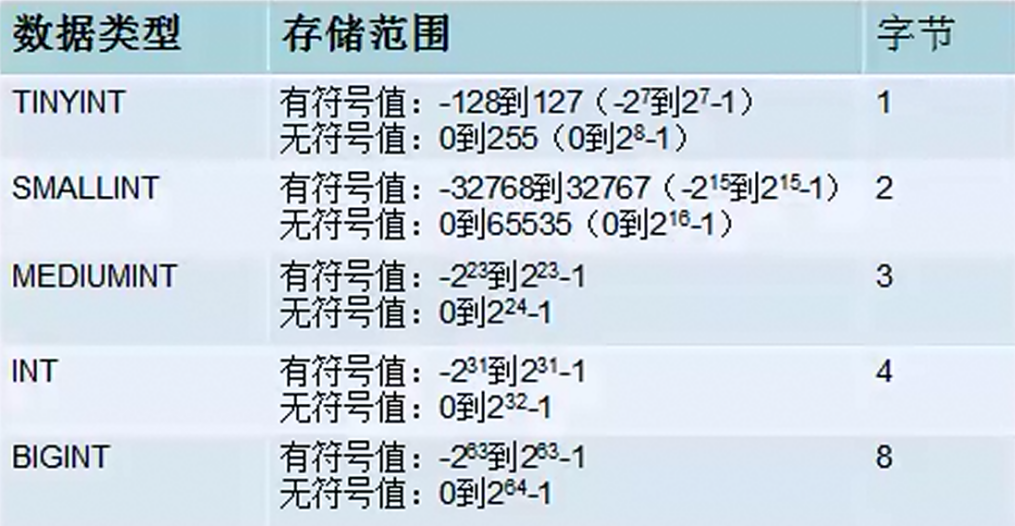
GRANT USAGE:mysql usage权限就是空权限，默认create user的权限，只能连库，啥也不能干

### 14.切换用户：exit退出,再登录。

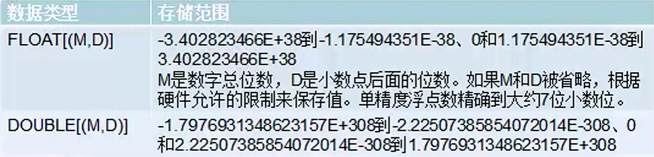
### 15.安装navicat for mysql

## **1.mysql基础**

### **数值类型**

Tinyint 是可以替代布尔类型，mysql没有布尔类型。

### 2.**浮点型**



### 3.**日期和时间类型**



### 4.**字符串类型**



varchar与char的区别：

CHAR的长度是固定的，而VARCHAR2的长度是可以变化的， 比如，存储字符串“abc"，对于CHAR (10)，表示你存储的字符将占10个字节(包括7个空字符)，而同样的VARCHAR2 (10)则只占用3个字节的长度，10只是最大值，当你存储的字符小于10时，按实际长度存储。

CHAR的效率比VARCHAR2的效率稍高。

VARCHAR2虽然比CHAR节省空间，但是如果一个VARCHAR2列**经常被修改**，而且每次被修改的数据的长度不同，这会引起‘行迁移’(Row Migration)现象，而这造成多余的I/O，是[数据库设计](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AE%BE%E8%AE%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9ujmdPj9-PvDsuj6zmWn10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHm4PHDLnH0" \t "https://zhidao.baidu.com/question/_blank)和调整中要尽力避免的，在这种情况下用CHAR代替VARCHAR2会更好一些。

### 5.主键，外键

**主键**(primary key)能够唯一标识表中某一行的属性或属性组。主键常常与外键构成 参照，完整性约束，防止出现数据不一致。主键可以保证记录的唯一和主键域非空, 数 据库管理系统对于主键自动生成唯一索引，所以主键也是一个特殊的索引。

**定义主键：**primary key（id）,在create表时声明。

**外键**（foreign key） 是用于建立和加强两个表数据之间的链接的一列或多列。外键约 束主要用来维护两个表之间数据的一致性。简言之，**表的外键就是另一表的主键**，外键 将两表联系起来。

**定义外键：**foregin key (parent\_id) references parent(id) on update **cascade**on delete **restrict**.

在create表时声明,parent\_id是子表声明的，parent是父表，id是父表主键。on update cascade表示父则可以更新parent记录时,child中的相应记录也会被更新;on delete restrict表明删除parent数据不影响子表。cascade(跟随外键改动), restrict(限制外表中的外键改动)。

MySql建立外键需要满足三个条件：  
 1.两个表必须是InnoDB数据引擎  
 2.使用在外键关系的域必须为索引型(Index)  
 3.使用在外键关系的域必须与数据类型相似

### 6.别名

为表取别名select \* from department d WHERE d.d\_id=1001;

为字段取别名select id as d\_id FROM department;//as可省

### 7.MySQL数据引擎

a.  **ISAM:**最老的引擎， 执行读取操作的速度很快，而且不占用大量的内存和存储资源。 不足，它不支持事务处理，也不能够容错。

b.  **MyISAM:**ISAM的mysql扩张， 继承了ISAM的优点，还增加了一些功能，如 索引， 字段管理功能，表格锁定的机制等。并强调了快速读取操作，所以web开发大多数用此 引擎。缺点和ISAM一样。而且为了优化多个并发的读写操作和锁定的机制，它必须经 常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间。

c.  **HEAP:** 它允许只驻留在内存里的临时表格，驻留在内存里让HEAP要比ISAM和 MYISAM都快，但是，它的表格存储在内存里，所以不及时保存的话，会造成数据丢失。

d.  **InnoDB，** 支持事务处理（transaction process）和外来键。但是速度比前者慢， 一般用于处理巨大数据量时的最大性能设计。

**更改数据库引擎**：alter table table\_name engine=innodb;

**查看表的引擎状态**：show table status from zwj\_test where name = 'hat\_area';

### 8.启动mysql

Net start mysql,net stop mysql(重启就执行两条命令)

### 9.schema

Mysql schema等价于数据库

### 10..frm

.frm文件是用来保存每个数据表的元数据(meta)信息，包括表结构的定义等，.frm文件跟数据库存储引擎无关，也就是任何存储引擎的数据表都必须有.frm文件，命名方式为数据表名.frm，如user.frm. .frm文件可以用来在数据库崩溃时恢复表结构。

## sql命令

### 1.数据库

Show databaes;

use xxx;

create database xxx;

drop database xxx;

### 2.表

Create table test\_table(number int auto\_increment,id int,name varchar(20),info varchar(50),primary key(number));

Show tables;

Drop table test\_table;

show columns from test\_table;//显示表的结构

Show table status like ‘rename\_table’;//显示表信息

check table rename\_table;//检测表损坏情况

show status like 'open%tables';//显示被打开的表数

show status like 'threads%';//显示线程情况

show status like 'created\_tmp%';//临时表使用

### 3.插入数据

insert into test\_table(id,name,info) values (20150118,'zwj','我是张为金');

### 4.更改数据

update test\_table set name='zwjj',info='i am zwjj';

### 5.删除数据

delete from test\_table where id=20150018;

### 6.修改表结构

alter table test\_table drop info;//删除字段info

alter table test\_table add info char(30);//添加字段info

alter table test\_table modify info char(35);//改变字段info

alter table test\_table rename to rename\_table;//修改表名

alter table rename\_table alter info set default 'i am null';//修改默认值

### 7.where语句，having语句

select id,name,info from test\_table where number=1;

select id,name,info from rename\_table where number>1 and id=20150118;

* where 子句的作用是在对查询结果进行分组前，将不符合where条件的行去掉，即在分组之前过滤数据，where条件中不能包含聚组函数，使用where条件过滤出特定的行。
* having 子句的作用是筛选满足条件的组，即在分组之后过滤数据，条件中经常包含聚组函数，使用having 条件过滤出特定的组，也可以使用多个分组标准进行分组。

### 8.and,or,not,in,between,exists

**and**检索匹配满足所有给定条件的行

**or**检索匹配满足给定任一条件的行

#在处理OR操作符之前，AND操作符会被优先处理

**in**指定条件范围

select id,name from rename\_table where number in(1,3,’c’);.

**not**否定它之后所跟的任何条件

**Between A and B** 检索A-B的范围的值。//Between给的范围，in给的具体的值。

select \* from table where time between '2008-08-08' and '2009-01-01'

**exists**:exists的返回结果是bool型，只有true或者false

Select \* from table t where exists(select a\_id from edu\_table e where e.a\_id=t.id\_);

### 9.union语句

union:联合的意思，即把两次或多次查询结果合并起来。两次查询的列数必须一致。

select info from rename\_table where id=20150118 union select info from rename\_table where name='zwjjj';//重复的结果会去除

Union all:UNION ALL命令和UNION命令几乎是等效的，不过UNION ALL命令会列出重复的值。

### 10.order by 排序

select \* from rename\_table order by id Desc;

Desc降序 asc升序

### 11.groupby与聚合函数

group by根据一个或多个列对结果集进行分组。

常见的聚合函数如下表：

| 函数 | 作用 | 支持性 |
| --- | --- | --- |
| sum(列名) | 求和 |  |
| max(列名) | 最大值 |  |
| min(列名) | 最小值 |  |
| avg(列名) | 平均值 |  |
| first(列名) | 第一条记录 | 仅Access支持 |
| last(列名) | 最后一条记录 | 仅Access支持 |
| count(列名) | 统计记录数 | 注意和count(\*)的区别 |

select id from rename\_table group by id;

select min(id) from rename\_table;

select id,count(id) from rename\_table group by id;

select id,count(id) from rename\_table group by id having count(id) > 1;

### 12.大于，小于，等于

eq相等   ne、neq不相等，   gt大于， lt小于 gte、ge大于等于   lte、le 小于等于   not非   mod求模

<> 与!=都是不等于的意思

使用&lt;

### 13.join查询

* Inner join（内连接,或等值连接）：获取两个表中字段匹配关系的记录。
* Left join（左连接）：获取左表所有记录，即使右表没有对应匹配的记录。
* Right join（右连接）： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。

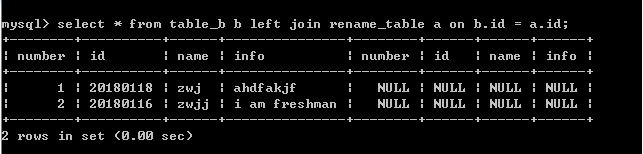
mysql不支持Full join,不过可以通过UNION 关键字来合并 LEFT JOIN 与 RIGHT JOIN来模拟FULL join.

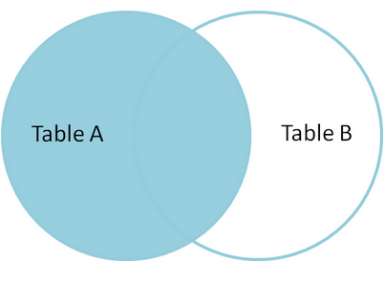
内连接，也叫等值连接，inner join产生同时符合A和B的一组数据：

select \* from table\_b b inner join rename\_table a on a.number = b.number;

左连接从左表(A)产生一套完整的记录,与匹配的记录(右表(B)) .如果没有匹配,右侧将包含null：

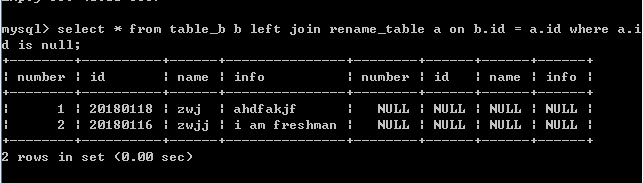
select \* from table\_b b left join rename\_table a on b.id = a.id;

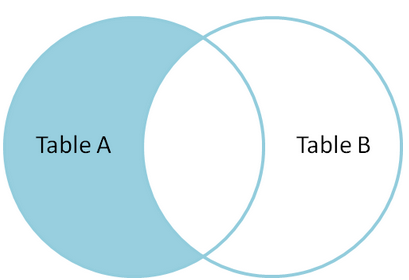




如果想只从左表(A)中产生一套记录，但不包含右表(B)的记录，可以通过设置where语句来执行：

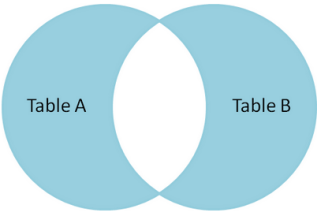
select \* from table\_b b left join rename\_table a on b.id = a.id where a.id is null;





求差集：

SELECT \* FROM A LEFT JOIN B ON A.name = B.name WHERE B.id IS NULL union SELECT \* FROM A right JOIN B ON A.name = B.name WHERE A.id IS NULL;



cross join：交叉连接，得到的结果是两个表的乘积，即[笛卡尔积](http://baike.baidu.com/view/348542.htm?fromtitle=%E7%AC%9B%E5%8D%A1%E5%B0%94%E7%A7%AF&fromid=1434391&type=syn)

使用场景：如果A表示某学校学生的集合，B表示该学校所有课程的集合，则A与B的笛卡尔积表示所有可能的选课情况。

mysql> select \* from A cross join B;

### 14.双主键模式

复合主键 就是指你表的主键含有一个以上的字段组成

create table two\_key(number int,id int,name char(20),info longtext,primary key(number,id));

如果我们设置了唯一索引，那么在插入重复数据时，SQL语句将无法执行成功,并抛出错。

insert ignore into two\_key(number,id,name,info) values(1,20181111,'hyc','i am hu');

**insert ignore into** 会**忽略数据库中已经存在的数据**，如果数据库没有数据，就插入新的数据，如果有数据的话就跳过这条数据。

### 15.处理重复元素

统计重复元素：select name,count(name) from table\_b group by name;

删除重复数据：

变更主键：先删除主键alter table table\_b drop primary key;

再添加主键alter table table\_b add primary key(name);

### 16.unique唯一索引

UNIQUE 约束唯一标识数据库表中的每条记录。 每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束。

create table two\_p\_key(number int,id int ,name char(20),info text,unique(

number),unique(id));

删除：ALTER TABLE Test DROP INDEX name;

更改：CREATE UNIQUE INDEX `psw` on Test(password);

### 17.导入,导出数据

source d:/province.sql；

mysqldump -u zwj -p zwj\_test hat\_area>d:\dump.txt

### 18.like语句与模糊查询

SQL模式

1，%：表示任意个或多个字符。

2，\_：表示任意**单个**字符。

select \* from hat\_area where area like '%青%';

select \* from hat\_area where area like '\_青\_';

正则模式

select \* from hat\_area where area regexp '^青';//以青开头

like查询是不区分大小写的。要区分大小写like的后面加个binary就可以。

也可以在建表时，加以标识。

### 19.limit分页

Select \* from rename\_table limit 5,10;// 检索记录行 6-15

//Select \* from rename\_table limit 5,-1;// 检索记录行 6之后，mysql不支持会报错

Select \* from rename\_table limit 5;// 检索记录前5行

Select \* from rename\_table limit (20,10);// 检索记录第20行向后10行

### 20.using

using相当于join on中的on.

### 21.distinct关键字

distinct这个关键字来过滤重复的记录，但是实际中我们往往用distinct来返回不重复字段的条数（count(distinct id)）,其原因是distinct只能返回他的目标字段，而无法返回其他字段。

## **3.索引**

MySQL索引的建立对于MySQL的高效运行是很重要的，索引可以大大提高MySQL的检索速度。

索引分单列索引和组合索引。单列索引，即一个索引只包含单个列，一个表可以有多个单列索引，但这不是组合索引。组合索引，即一个索引包含多个列。

创建索引时，你需要确保该索引是应用在 SQL 查询语句的条件(一般作为 WHERE 子句的条件)。

建立索引会占用磁盘空间的索引文件。

### **建立索引**

1. PRIMARY  KEY（主键索引）  
    mysql>ALTER  TABLE  `table\_name`  ADD  PRIMARY  KEY (  `column`  )   
   2.UNIQUE(唯一索引)  
         mysql>ALTER  TABLE  `table\_name`  ADD  UNIQUE (  
   `column` )   
   3.INDEX(普通索引)  
    mysql>ALTER  TABLE  `table\_name`  ADD  INDEX index\_name (  `column`  )

**alter table hat\_area add index index\_area(area);**4.FULLTEXT(全文索引)  
 mysql>ALTER  TABLE  `table\_name`  ADD  FULLTEXT ( `column` )

MySql自带的全文索引只能对英文进行全文检索，目前无法对中文进行全文检索。

**alter table hat\_area add fulltext ft\_area(area);//ft\_area索引名**

**select \* from hat\_area where match(area) against('青河县' in natural language mode);**

检索方式

1）、自然语言检索：in natural language mode

2）、布尔检索：in boolean mode

全文检索的最小许可字符：show variables like 'ft\_min\_word\_len';

5.多列索引  
 mysql>ALTER  TABLE  `table\_name`  ADD  INDEX index\_name (  `column1`,  `column2`,  `column3`  )

### **Key关键字**

KEY通常是INDEX同义词。

### **UUID**

UUID很好用的一点是这样设计出来的id永远不会是重复的。

将id变为uuid:update rename\_table set id = UUID(); //需要保证长度为char36

本身 UUID 是 32 位，因为 MySQL 生成的 UUID 有四个中划线，所以在 utf8 字符集里，长度为 36 位.

alter table rename\_table add uuid char(36);

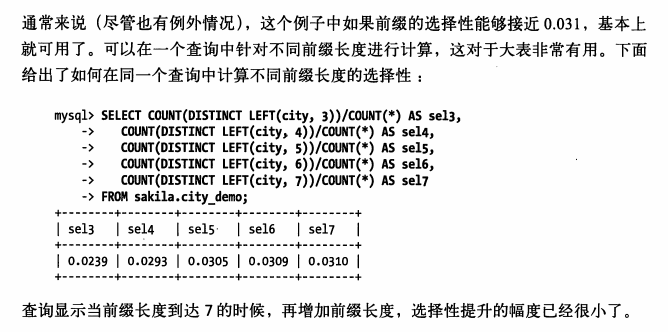
update rename\_table set uuid = uuid();

### **二级索引**

以InnoDB来说，**每个InnoDB表具有一个特殊的索引称为聚集索引**。如果您的表上定义有主键，该主键索引是聚集索引。如果你不定义为您的表的主键时，MySQL取第一个唯一索引（unique）而且只含非空列（NOT NULL）作为主键，InnoDB使用它作为聚集索引。如果没有这样的列，InnoDB就自己产生一个这样的ID值，它有六个字节，而且是隐藏的，使其作为聚簇索引。

### **前缀索引**

**创建前缀索引：alter table hat\_area add key (area(2));**

**查找最佳前缀：**

select count(\*) as cnt,left(area,1) as pref from hat\_area group by pref order by cnt desc limit 10;

**覆盖索引**

**索引包含了所有要查询的字段的值**

### **删除索引：**

drop index index\_area on hat\_area;

### MySQL KEY:

PRI主键约束；

UNI唯一约束；

MUL可以重复。

### **查看索引**

mysql> show index from tblname;

mysql> show keys from tblname;

· Table

表的名称。

· Non\_unique

如果索引不能包括重复词，则为0。如果可以，则为1。

· Key\_name

索引的名称。

· Seq\_in\_index

索引中的列序列号，从1开始。

· Column\_name

列名称。

· Collation

列以什么方式存储在索引中。在MySQL中，有值‘A’（升序）或NULL（无分类）。

· Cardinality

索引中唯一值的数目的估计值。通过运行ANALYZE TABLE或myisamchk -a可以更新。基数根据被存储为整数的统计数据来计数，所以即使对于小型表，该值也没有必要是精确的。基数越大，当进行联合时，MySQL使用该索引的机 会就越大。

· Sub\_part

如果列只是被**部分地编入索引**，则为被编入索引的字符的数目。如果整列被编入索引，则为NULL。

· Packed

指示关键字如何被压缩。如果没有被压缩，则为NULL。

· Null

如果列含有NULL，则含有YES。如果没有，则该列含有NO。

· Index\_type

用过的索引方法（BTREE, FULLTEXT, HASH, RTREE）。

## 4.node连接mysql

Npm install mysql;

mysql.createConnection(options)

|  |  |
| --- | --- |
| user | 新的用户 (默认为早前的一个). |
| password | 新用户的新密码 (默认为早前的一个). |
| charset | 新字符集 (默认为早前的一个). |
| database | 新数据库名称 (默认为早前的一个). |

|  |  |
| --- | --- |
| createConnection | 用于创建链接的函数. (Default: mysql.createConnection) |
| waitForConnections | 决定当没有连接池或者链接数打到最大值时pool的行为. 为true时链接会被放入队列中在可用是调用，为false时会立即返回error. (Default: true) |
| connectionLimit | 最大连接数. (Default: 10) |
| queueLimit | 连接池中连接请求的烈的最大长度，超过这个长度就会报错，值为0时没有限制. (Default: 0) |
| canRetry | 值为true时，允许连接失败时重试(Default: true) |
| removeNodeErrorCount | 当连接失败时 errorCount 值会增加. 当errorCount 值大于 removeNodeErrorCount 将会从PoolCluster中删除一个节点. (Default: 5) |
| defaultSelector | 默认选择器. (Default: RR) |
| RR | 循环. (Round-Robin) |
| RANDOM | 通过随机函数选择节点. |
| ORDER | 无条件地选择第一个可用节点. |

**var** pool = mysql.createPool({  
 host:'localhost',  
 user:'zwj',  
 port:3306,  
 password:"zwj153",  
 database:'zwj\_test',  
 connectionLimit:1000,//不能超过mysql最大连接数。默认20  
 queueLimit:0, //队伍中等待连接的最大数量，0为不限制。  
 waitForConnections:**true** //决定当没有连接池或者链接数打到最大值时pool的行为. 为true时链接会被放入队列中在可用是调用，为false时会立即返回error. (Default: true)  
});  
  
**var** i=0;  
**var** start = **new** Date().getTime();  
router.get('/',**function** (req,res,next) {  
 **var** sql = 'show tables';  
 pool.getConnection(**function** (error,connection) {  
 connection.query(sql,**function** (error,result,feild) {  
 **if**(error) **throw** error;  
 console.log('i->:',i++);  
 //console.log(feild);  
 res.send(result);  
 **var** end = **new** Date().getTime();  
 console.log("end - start:",end-start);  
 console.log("pool 连接数",pool.\_allConnections.length);  
 connection.release();//释放链接  
 });  
 })  
});

forever是一个简单的命令式nodejs的守护进程，能够启动，停止，重启App应用。

>npm install forever -g

>forever start ./bin/www

warn: --minUptime not set. Defaulting to: 1000ms

warn: --spinSleepTime not set. Your script will exit if it does not stay up f

or at least 1000ms

info: Forever processing file: ./bin/www

启动是成功的

>forever list查看服务状态

>forever stop ./bin/www停止服务

PM2是一个带有负载均衡功能的Node应用的进程管理器.

>npm install pm2 -g

>pm2 start ./bin/www

>pm2 list

>pm2 monit #监视所有进程

> pm2 logs #显示所有进程日志  
> pm2 stop all #停止所有进程  
> pm2 restart all #重启所有进程

Pm2比forever更强

## 5.mysql性能

### 1.show profile

set profiling = 1;//服务器执行的语句都会测量其耗费时间

show profiles;//显示上条查询花费时间

show profile for query 1;//显示了查询的更精确响应时间

### 2.explain查看分析结果

使用explain语句去查看分析结果:

**type=const**表示通过索引一次就找到了；

**key=primary**的话，表示使用了主键；

**type=all**,表示为全表扫描；

**key=null**表示没用到索引。type=ref,因为这时认为是多个匹配行，在联合查询中，一般为REF。

|  |  |
| --- | --- |
| 项 | 说明 |
| id | MySQL Query Optimizer 选定的执行计划中查询的序列号。表示查询中执行 select 子句或操作表的顺序,id 值越大优先级越高,越先被执行。id 相同,执行顺序由上至下。 |

|  |  |
| --- | --- |
| select\_type | 说明 |
| SIMPLE | 简单的 select 查询,不使用 union 及子查询 |
| PRIMARY | 最外层的 select 查询 |
| UNION | UNION 中的第二个或随后的 select 查询,不 依赖于外部查询的结果集 |
| DEPENDENT UNION | UNION 中的第二个或随后的 select 查询,依 赖于外部查询的结果集 |
| SUBQUERY | 子查询中的第一个 select 查询,不依赖于外 部查询的结果集 |
| DEPENDENT SUBQUERY | 子查询中的第一个 select 查询,依赖于外部 查询的结果集 |
| DERIVED | 用于 from 子句里有子查询的情况。 MySQL 会 递归执行这些子查询, 把结果放在临时表里。 |
| UNCACHEABLE SUBQUERY | 结果集不能被缓存的子查询,必须重新为外 层查询的每一行进行评估。 |
| UNCACHEABLE UNION | UNION 中的第二个或随后的 select 查询,属 于不可缓存的子查询 |

|  |  |
| --- | --- |
| 项 | 说明 |
| table | 输出行所引用的表 |

|  |  |
| --- | --- |
| type | 说明 |
| system | 表仅有一行(=系统表)。这是 const 连接类型的一个特例。 |
| const | const 用于用常数值比较 PRIMARY KEY 时。当 查询的表仅有一行时,使用 System。 |
| eq\_ref | const 用于用常数值比较 PRIMARY KEY 时。当 查询的表仅有一行时,使用 System。 |
| ref | 连接不能基于关键字选择单个行,可能查找 到多个符合条件的行。 叫做 ref 是因为索引要 跟某个参考值相比较。这个参考值或者是一 个常数,或者是来自一个表里的多表查询的 结果值。 |
| ref\_or\_null | 如同 ref, 但是 MySQL 必须在初次查找的结果 里找出 null 条目,然后进行二次查找。 |
| index\_merge | 说明索引合并优化被使用了。 |
| unique\_subquery | 在某些 IN 查询中使用此种类型,而不是常规的 ref:value IN (SELECT primary\_key FROM single\_table WHERE some\_expr) |
| index\_subquery | 在 某 些 IN 查 询 中 使 用 此 种 类 型 , 与 unique\_subquery 类似,但是查询的是非唯一 性索引: value IN (SELECT key\_column FROM single\_table WHERE some\_expr) |
| range | 只检索给定范围的行,使用一个索引来选择 行。key 列显示使用了哪个索引。当使用=、 <>、>、>=、<、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符,用常量比较关键字列时,可 以使用 range。 |
| index | 全表扫描,只是扫描表的时候按照索引次序 进行而不是行。主要优点就是避免了排序, 但是开销仍然非常大。 |
| all | 最坏的情况,全表扫描。 |

|  |  |
| --- | --- |
| 项 | 说明 |
| possible\_keys | 指出 MySQL 能在该表中使用哪些索引有助于 查询。如果为空,说明没有可用的索引。 |

|  |  |
| --- | --- |
| 项 | 说明 |
| key | MySQL 实际从 possible\_key 选择使用的索引。 如果为 NULL,则没有使用索引。很少的情况 下,MYSQL 会选择优化不足的索引。这种情 况下,可以在 SELECT 语句中使用 USE INDEX (indexname)来强制使用一个索引或者用 IGNORE INDEX(indexname)来强制 MYSQL 忽略索引 |

|  |  |
| --- | --- |
| 项 | 说明 |
| key\_len | 使用的索引的长度。在不损失精确性的情况 下,长度越短越好。 |

|  |  |
| --- | --- |
| 项 | 说明 |
| ref | 显示索引的哪一列被使用了 |

|  |  |
| --- | --- |
| 项 | 说明 |
| rows | MYSQL 认为必须检查的用来返回请求数据的行数 |

|  |  |
| --- | --- |
| 项 | 说明 |
| rows | MYSQL 认为必须检查的用来返回请求数据的行数 |

extra 中出现以下 2 项意味着 MYSQL 根本不能使用索引,效率会受到重大影响。应尽可能对此进行优化。

|  |  |
| --- | --- |
| extra 项 | 说明 |
| Using filesort | 表示 MySQL 会对结果使用一个外部索引排序,而不是从表里按索引次序读到相关内容。可能在内存或者磁盘上进行排序。MySQL 中无法利用索引完成的排序操作称为“文件排序” |
| Using temporary | 表示 MySQL 在对查询结果排序时使用临时表。常见于排序 order by 和分组查询 group by。 |

### 3.延迟关联

### ****4.optimize table****

如果您已经删除了表的一大部分，或者如果您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。被删除的记录被保持在链接清单中，后续的INSERT操作会重新使用旧的记录位置。您可以使用OPTIMIZE TABLE来重新利用未使用的空间，**并整理数据文件的碎片**。

**5.混合使用过程化和关系化逻辑的时候使用自定义变量**

## 6.范式

### 1.第一范式（1NF）

**1NF的定义为：符合1NF的关系中的每个属性都不可再分。**

如果每列(或者每个属性)都是不可再分的最小数据单元(也称为最小的原子单元),则满足第一范式.

### 2.第二范式

**第二范式:在第一范式的基础上更进一层,目标是确保表中的每列都和主键相关.**

     如果一个关系满足第一范式,并且除了主键以外的其它列,都依赖于该主键,则满足 第二范式.

### 3.第三范式

**第三范式:在第二范式的基础上更进一层,目标是确保每列都和主键列直接相关,而不是间接相关.**

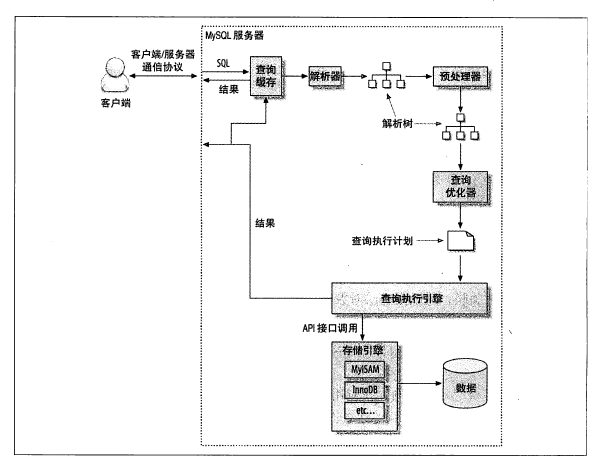
     如果一个关系满足第二范式,并且除了主键以外的其它列都不依赖于主键列,则满 足第三范式.

### 4.反范式

**反范式化指的是通过增加冗余或重复的数据来提高数据库的读性能。**

## **7.mysql查询优化**

### 1.查询执行路径



### 2.切分查询

将大查询切分为小查询。

### 3.分解关联查询

一条查询，分解为多条查询。

## 8.分区表

**分表：**分表是将一个大表按照一定的规则分解成多张具有独立存储空间的实体表，我们可以称为子表，每个表都对应三个文件，MYD数据文件，.MYI索引文件，.frm表结构文件。这些子表可以分布在同一块磁盘上，也可以在不同的机器上。

**分区：**分表将大表分解为若干个独立的实体表，而分区是将数据分段划分在多个位置存放，可以是同一块磁盘也可以在不同的机器。分区后，表面上还是一张表，但数据散列到多个位置了。app读写的时候操作的还是大表名字，db自动去组织分区的数据。

**分区的几种方式**

**Range：**

[?](https://www.2cto.com/database/201503/380348.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **create** **table** range(  　　id **int**(11),  　　money **int**(11) unsigned not null,  **date** datetime  　　)**partition by** range(year(**date**))(  　　partition p2007 **values** less than (2008),  　　partition p2008 **values** less than (2009),  　　partition p2009 **values** less than (2010)  　　partition p2010 **values** less than maxvalue  )；  对已有表分区：  alter table hat\_area partition by range(id) (  partition p0 values less than(1000),  partition p1 values less than(2000),  partition p2 values less than(3000),  partition p3 values less than maxvalue ); |
|  |  |

**List：**

[?](https://www.2cto.com/database/201503/380348.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **create** **table** list(  　　a **int**(11),  　　b **int**(11)  　　)(partition **by** list (b)  　　partition p0 **values** in (1,3,5,7,9),  　　partition p1 **values** in (2,4,6,8,0)  　); |

**Hash：**

[?](https://www.2cto.com/database/201503/380348.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | **create** **table** hash(  　　a **int**(11),  　　b datetime  　　)partition **by** hash (YEAR(b)  　　partitions 4; |

**Key：**

[?](https://www.2cto.com/database/201503/380348.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | **create** **table** t\_key(  　　a int(11),  　　b datetime)  　　partition by key (b)  　　partitions 4; |

**分区管理**

**新增分区**

[?](https://www.2cto.com/database/201503/380348.html)

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE sale\_data  ADD PARTITION (PARTITION p201010 VALUES LESS THAN (201011)); |

**删除分区  
 --当删除了一个分区，也同时删除了该分区中所有的数据。  
 ALTER TABLE sale\_data DROP PARTITION p201010;**

**1，分区可以分在多个磁盘，存储更大一点**

**2，根据查找条件，也就是where后面的条件，查找只查找相应的分区不用全部查找了**

**3，进行大数据搜索时可以进行并行处理。**

**4，跨多个磁盘来分散数据查询，来获得更大的查询吞吐量**

## **9.视图**

### **1.什么是视图**

视图是由查询结果形成的一张虚拟表。通俗的讲，视图就是一条SELECT语句执行后返回的结果集。所以我们在创建视图的时候，主要的工作就落在创建这条SQL查询语句上。

### **2.视图的优点**

**①简化查询语句**

比如：有一张商品表，我们经常要查每个栏目下商品的平均价格

这时候我们就可以创建一张视图：

  create view avgPrice as select cat\_id,avg(shop\_price) from goods gropy by cat\_id;

  创建完，以后我们要查每个栏目的平均价格时，只要这么写

  select \* from avgPrice;就可以了。

**②可以进行权限控制**

把表的权限封闭，但是开放相应的视图权限，视图里只开放部分数据列

 比如我们的goods商品表，我们不想让别人看到我们的销售价格，这时候我们就可以 把查看商品表的权限封闭，创建一张视图

  create view showGoods as select goods\_id,goods\_name from goods;

  不出现销售价格列就可以了。

**③大数据分表时可以用到**

比如表的行数据超过200万行时，速度就会变慢

  可以把一张表的数据拆成4张表来存放

可以用视图，把四张表形成一张视图

  create view news as select \* from news1 union select \* from news2 union ...

### **3.视图的创建语法**

create view 视图名 as select 语句;

**create view area\_view as select \* from hat\_area where id between 1 and 100**;

**select \* from area\_view where id = 60;//视图使用**

### **4.视图的修改**

 alter view 视图名 as select 语句;

### **5.更新视图**

**通过视图更新表数据。**通过视图更新时，都是转换到基本表来更新。更新视图时，只能 更新权限范围内的数据。超出了范围，就不能更新。

### **6.删除视图**

**drop view** IF EXISTS 视图名

### **7.查看视图**

desc area\_view;

show table status like 'area\_view';

### **8.视图与表的关系**

 视图是表的查询结果，自然表的数据变了，会影响视图的结果

①视图的增删改也会影响表；

  ②但视图并不总是能增删改的；

  视图的数据与表的数据一一对应时可以修改；

   对于视图的insert还应注意：视图必须包含表中没有默认值的列。

## 10.触发器

触发器(trigger)：监视某种情况，并触发某种操作。

触发器创建语法四要素：1.监视地点(table) 2.监视事件(insert/update/delete) 3.触发时间(after/before) 4.触发事件(insert/update/delete)

### 1.创建触发器

create trigger triggerName after/before insert/update/delete on 表名for each row;

#这句话在mysql是固定的

Begin sql语句; end;

### 2.删除触发器

drop trigger triggerName；

### 3.更改触发器

### 4.查看触发器

show  triggers

## 11.游标

游标简单来说就是查询出来的数据索引，通过对游标的操作（第一个位置、最后一个位置、上一个位置、下一个位置）可以遍历出数据。

### **1.定义游标**

Create procedure statisticstore()

Begin

--创建接收游标数据的变量

declare c int;

declare n varchar(20);

--创建总数变量

declare total int default 0;

--创建结束标志变量

declare done int default false;

--创建游标

declare cur cursor for select name,count from store where name = 'iphone';

--指定游标循环结束时的返回值

declare continue HANDLER for not found set done = true;

--设置初始值

set total = 0;

--打开游标

open cur;

--开始循环游标里的数据

read\_loop:loop

--根据游标当前指向的一条数据

fetch cur into n,c;

--判断游标的循环是否结束

if done then

leave read\_loop; --跳出游标循环

end if;

--获取一条数据时，将count值进行累加操作，这里可以做任意你想做的操作，

set total = total + c;

--结束游标循环

end loop;

--关闭游标

close cur;

--输出结果

select total;

end;

--调用存储过程

call statisticstore();

### **2.删除游标**

drop procedure if exists statisticstore;

## 12.绑定变量

绑定变量是为了减少解析的，比如你有个语句这样

select aaa,bbb from ccc where ddd=eee;

如果经常通过改变eee这个谓词赋值来查询，像如下

select aaa,bbb from ccc where ddd=fff;select aaa,bbb from ccc where ddd=ggg;select aaa,bbb from ccc where ddd=hhh;

每条语句都要被数据库解析一次，这样比较浪费资源，如果把eee换成“:1”这样的绑定变量形式，无论ddd后面是什么值，都不需要重复解析。

Java实现绑定变量的方法：

[java] view plaincopy

PreparedStatement pstmt;

for (id = 1; id < 10000; id ) {

if (null == pstmt) pstmt = con.prepareStatement("UPDATE employees SET salay = ? WHERE id = ?");

pstmt.setBigDecimal(1, 150.00);

pstmt.setInt(2, id);

pstmt.executeQuery();

}

不用绑定变量，则相当于反复解析、执行了1w个sql语句。使用绑定变量，解析sql语句只用了一次，之后的9999次复用第一次生成的执行计划。显然，后者效率会更高一些。

## 13.分布式xa事务

eXtended Architecture 在事务中意为分布式事务

XA由协调者(coordinator，一般为transaction manager)和参与者(participants,一般在各个资源上有各自的resource manager)共同完成。在MySQL中，XA事务有两种。

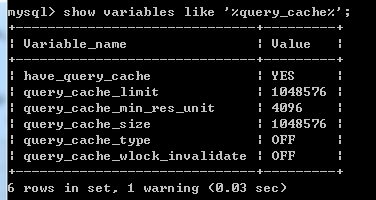
****内部XA事务****   
MySQL本身的插件式架构导致在其内部需要使用XA事务，此时MySQL即是协调者，也是参与者。例如，不同的存储引擎之间是完全独立的，因此当一个事务涉及两个不同的存储引擎时，就必须使用内部XA事务。

****外部XA事务****   
MySQL也可以仅作为一个外部XA的参与者。

****Two-phase commit****   
XA一般由两阶段完成，称为two-phase commit(2PC)。   
阶段一为准备阶段，即所有的参与者准备执行事务并锁住需要的资源。参与者ready时，向transaction manager汇报自己已经准备好。   
阶段二为提交阶段。当transaction manager确认所有参与者都ready后，向所有参与者发送commit命令。

## 14.查询缓存

show variables like '%query\_cache%';//查看数据库缓存配置情况



Set @@global.query\_cache\_size=100000;或 set global query\_cache\_size = 1000000;//设置缓存

show status like '%Qcache%';//缓存memory等信息

## 15.mysql配置

每60秒查看状态变量的增量变化：

mysqladmin -u root -p extended-status -ri60

**innodb\_buffer\_pool\_size**缓存用户表及索引数据的最主要缓存空间

select @@innodb\_buffer\_pool\_size;

set global innodb\_buffer\_pool\_size = 130000000;//默认 134217728

**innodb\_max\_dirty\_pages\_pct**

这个参数据控制脏页的比例如果是innodb\_plugin或是MySQL5.5以上的版本，建议这个参数可以设制到75%-90%都行。如果是大量写入，而且写入的数据不是太活跃，可以考虑把这个值设的低一点。 如果写入或是更新的数据也就是热数据就可以考虑把这个值设为：95%。

**Thread\_cache\_size**保持在缓存中的线程数。

**innodb flush\_log\_at\_trx\_commit**; 控制日志缓冲刷新频繁程度。

**Innodb\_thread\_concurrency;一次可以有多少线程进入内核。**

**Innodb\_log\_file\_size:**日志文件大小。

配置项：

Expire\_logs\_days:启用二进制日志，

## 16.mysql排序

MySQL内部实现排序主要有3种方式，常规排序，优化排序和优先队列排序，主要涉及3种排序算法：快速排序、归并排序和堆排序。

排序方式：

**两次传输排序two-pass（旧版本）**

读取行指针和需要排序的字段，对其进行排序，在根据排序结果读取所需要的行。

**单次传输排序single-pass（新版本）**

读取所需要的列，根据所需要的列进行排序，最后直接返回结果。

**Sort\_buffer\_size:排序缓冲区**

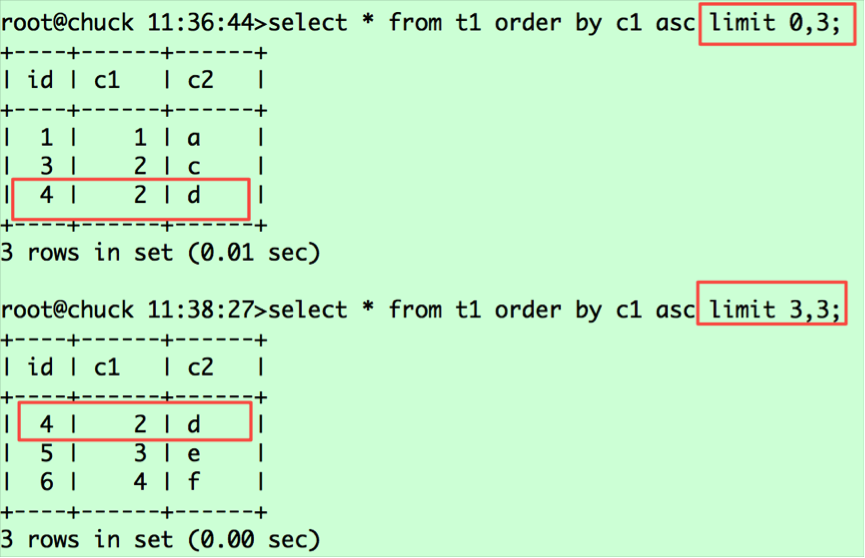
**Max\_length\_for\_sort\_data:**决定使用老式排序算法还是改进版排序算法是通过它来决定的。当所有**返回字段的最大长度**小于这个参数值时,MySQL 就会选择改进后的排序算法,反之,则选择老式的算法。所以,如果有充足的内存让MySQL 存放须要返回的非排序字段,就可以加大这个参数的值来让 MySQL 选择使用改进版的排序算法。

Max\_sort\_length:

1. 常规排序，双路排序  
   (1).从表t1中获取满足WHERE条件的记录  
   (2).对于每条记录，将记录的**主键+排序键(id,col2)取出放入sort buffer**  
   (3).如果**sort buffer可以存放**所有满足条件的(id,col2)对，则进行排序；**否则**sort buffer满后，进行排序并**写到临时文件中**。(排序算法采用的是快速排序算法)  
   (4).若排序中**产生了临时文件**，需要利用**归并排序**算法，保证临时文件中记录是有序的  
   (5).循环执行上述过程，直到所有满足条件的记录全部参与排序  
   (6).**扫描**排好序的(id,col2)队，即**sort buffer**，并**利用主键id去取SELECT需要返回的其他列**(col1,col2,col3)  
   (7).将获取的结果集返回给用户。  
         从上述流程来看，**是否使用文件排序主要看sort buffer是否能容下需要排序的(id,col2)的结果集，这个buffer的大小由sort\_buffer\_size参数控制**。此外一次排序**还需要两次IO**，**一次是取排序字段(id,col2)到sort buffer中，第二次是通过上面取出的主键id再来取其他所需要返回列**(col1,col2,col3)，由于**返回的结果集是按col2排序，因此id是乱序的**，通过乱序的id取(col1,col2,col3)时会产生大量的随机IO。对于第二次IO取MySQL本身会优化，即在取之前**先将主键id排序，并放入缓冲区**，这个缓存区大小由参数**read\_rnd\_buffer\_size**控制，然后有序去取记录，**将随机IO转为顺序IO**。

b.优化排序，单路排序，max\_length\_for\_sort\_data  
     常规排序方式除了排序本身，还需要额外两次IO。**优化排序方式相对于常规排序，减少了第二次IO**。**主要区别在于，一次性取出sql中出现的所有字段放入sort buffer中而不是只取排序需要的字段**(id,col2)。由于**sort buffer中包含了查询需要的所有字段**，因此**排序完成后可以直接返回，无需二次取数据**。这种方式的**代价**在于，同样大小的sort buffer，能存放的(col1,col2,col3)数目要小于(id,col2)，如果**sort buffer不够大，可能导致需要写临时文件，造成额外的IO**。当然MySQL提供了参数**max\_length\_for\_sort\_data**，只有当**排序sql里出现的所有字段小于max\_length\_for\_sort\_data时，才能利用优化排序方式，否则只能用常规排序方式。**  
c.优先队列排序  
     为了得到最终的排序结果，我们都需要将所有满足条件的记录进行排序才能返回。那么相对于优化排序方式，是否还有优化空间呢？**5.6版本针对Order by limit M，N语句，在空间层面做了优化，加入了一种新的排序方式--优先队列，这种方式采用堆排序实现**。堆排序算法特征正好可以解**limit M,N** 这类排序的问题，虽然仍然需要所有字段参与排序，但是**只需要M+N个元组的sort buffer空间即可**，对于M，N很小的场景，基本**不会因为sort buffer不够而导致需要临时文件进行归并排序的问题**。对于升序，采用大顶堆，最终堆中的元素组成了最小的N个元素，对于降序，采用小顶堆，最终堆中的元素组成了最大的N的元素。

假设每页3条记录，第一页limit 0,3和第二页limit 3,3查询结果如下：



我们可以看到 id为4的这条记录居然同时出现在两次查询中，这明显是不符合预期的，而且在5.5版本中没有这个问题。

使用**优先队列排序的目的**就是在不能使用索引有序性的时候，如果要**排序，并且使用了limit n**，那么只需要在排序的过程中，**保留n条记录即可，这样虽然不能解决所有记录都需要排序的开销，但是只需要 sort buffer 少量的内存就可以完成排序，上面已经说明。**

之所以**MySQL5.6**出现了第二页数据重复的问题，是因为**使用了优先队列排序，其使用了堆排序的排序方法**，而**堆排序是一个不稳定的排序方法，也就是相同的值（例子中的值2）可能排序出来的数据和读出来的数据顺序不一致，无法保证排序前后数据位置的一致，所以导致分页重复的现象**。

为了避免这个问题，有几种方法：

①：索引排序字段

利用索引的有序性，在字段添加上索引，就直接按照索引的有序性进行读取并分页，从而可以规避遇到的这个问题。

②：利用多列索引，对于单列相同无法排序的，利用其主键进行排序：

select \* from t1 order by c1,id asc limit 0,3;

select \* from t1 order by c1,id asc limit 3,3;

## 17.mysql双写缓冲

InnoDB使用了一种叫做doublewrite的特殊文件flush技术，在把pages写到date files之前，InnoDB先把它们写到一个叫doublewrite buffer的连续区域内，在写doublewrite buffer完成后，InnoDB才会把pages写到data file的适当的位置。如果在写page的过程中发生意外崩溃，InnoDB在稍后的恢复过程中在doublewrite buffer中找到完好的page副本用于恢复。

InnoDB的page size一般是16KB，其数据校验也是针对这16KB来计算的，将数据写入到磁盘是以page为单位进行操作的。操作系统写文件是以4KB作为单位的，那么每写一个InnoDB的page到磁盘上，操作系统需要写4个块。而计算机硬件和操作系统，在极端情况下（比如断电）往往并不能保证这一操作的原子性，16K的数据，写入4K时，发生了系统断电或系统崩溃，只有一部分写是成功的，这种情况下就是partial page write（部分页写入）问题。这时page数据出现不一样的情形，从而形成一个"断裂"的page，使数据产生混乱。这个时候InnoDB对这种块错误是无 能为力的.

有人会认为系统恢复后，MySQL可以根据redo log进行恢复，而MySQL在恢复的过程中是检查page的checksum，checksum就是pgae的最后事务号，发生partial page write问题时，page已经损坏，找不到该page中的事务号，就无法恢复。

**doublewrite buffer是InnoDB在tablespace上的128个页（2个区）大小是2MB。**为了解决 partial page write问题，当MySQL将脏数据flush到data file的时候, 先使用memcopy将脏数据复制到内存中的doublewrite buffer，之后通过doublewrite buffer再分2次，每次写入1MB到共享表空间，然后马上调用fsync函数，同步到磁盘上，避免缓冲带来的问题，在这个过程中，doublewrite是顺序写，开销并不大，在完成doublewrite写入后，再将double write buffer写入各表空间文件，这时是离散写入。

所以在正常的情况下, MySQL写数据page时，会写两遍到磁盘上，第一遍是写到doublewrite buffer，第二遍是从doublewrite buffer写到真正的数据文件中。如果发生了极端情况（断电），InnoDB再次启动后，发现了一个page数据已经损坏，那么此时就可以从doublewrite buffer中进行数据恢复了。

**doublewrite的缺点是什么？**

位于共享表空间上的doublewrite buffer实际上也是一个文件，写共享表空间会导致系统有更多的fsync操作, 而硬盘的fsync性能因素会降低MySQL的整体性能，但是并不会降低到原来的50%。这主要是因为：

doublewrite是在一个连续的存储空间, 所以硬盘在写数据的时候是顺序写，而不是随机写，这样性能更高。

将数据从doublewrite buffer写到真正的segment中的时候，系统会自动合并连接空间刷新的方式，每次可以刷新多个pages。

**是否一定需要doublewrite?**

在一些情况下可以关闭doublewrite以获取更高的性能。比如在slave上可以关闭，因为即使出现了partial page write问题，数据还是可以从中继日志中恢复。设置InnoDB\_doublewrite=0即可关闭doublewrite buffer。

## 18.mysql复制

## 19.mysql备份

### ****一、备份的目的****

做灾难恢复：对损坏的数据进行恢复和还原  
需求改变：因需求改变而需要把数据还原到改变以前  
测试：测试新功能是否可用

### ****二、备份需要考虑的问题****

可以容忍丢失多长时间的数据；  
恢复数据要在多长时间内完；   
恢复的时候是否需要持续提供服务；  
恢复的对象，是整个库，多个表，还是单个库，单个表。

### ****三、备份的类型****

#### 1、根据是否需要数据库离线

冷备（cold backup）：需要关mysql服务，读写请求均不允许状态下进行；  
温备（warm backup）： 服务在线，但仅支持读请求，不允许写请求；  
热备（hot backup）：备份的同时，业务不受影响。

**注：**

1、这种类型的备份，取决于业务的需求，而不是备份工具  
2、MyISAM不支持热备，InnoDB支持热备，但是需要专门的工具

#### **2、根据要备份的数据集合的范围** 完全备份：full backup，备份全部字符集。 增量备份: incremental backup 上次完全备份或增量备份以来改变了的数据，不能单独使用，要借助完全备份，备份的频率取决于数据的更新频率。 差异备份：differential backup 上次完全备份以来改变了的数据。 建议的恢复策略： 完全+增量+二进制日志 完全+差异+二进制日志

#### **3、根据备份数据或文件**

物理备份：直接备份数据文件

优点：

备份和恢复操作都比较简单，能够跨mysql的版本，  
恢复速度快，属于文件系统级别的

建议：

不要假设备份一定可用，要测试  
mysql>check tables；检测表是否可用  
**逻辑备份:** 备份表中的数据和代码

优点：

恢复简单、  
备份的结果为ASCII文件，可以编辑  
与存储引擎无关  
可以通过网络备份和恢复

缺点：

备份或恢复都需要mysql服务器进程参与  
备份结果占据更多的空间，  
浮点数可能会丢失精度  
还原之后，缩影需要重建

### ****四：备份的对象****

1、 数据；  
2、配置文件；  
3、代码：存储过程、存储函数、触发器  
4、os相关的配置文件  
5、复制相关的配置  
6、二进制日志

### ****五、备份和恢复的实现****

#### **1、利用select into outfile实现数据的备份与还原**

1.1把需要备份的数据备份出来

mysql> use hellodb;     //打开hellodb库   
mysql> select \* from students;  查看students的属性   
mysql> select \* from students where Age > 30 into outfile ‘/tmp/stud.txt' ;   //将年龄大于三十的同学的信息备份出来

**注意：**

备份的目录路径必须让当前运行mysql服务器的用户mysql具有访问权限

备份完成之后需要把备份的文件从tmp目录复制走，要不就失去备份的目的了

你会发现是个文本文件。所以不能直接导入数据库了。需要使用load data infile 恢复

回到mysql服务器端，删除年龄大于30的用户，模拟数据被破坏

mysql> delete from students where Age > 30;

mysql> load data infile '/tmp/stud.txt' into table students;

#### **2、利用mysqldump工具对数据进行备份和还原**

mysqldump 常用来做温备，所以我们首先需要对想备份的数据施加读锁，

2.1 施加读锁的方式：

1.直接在备份的时候添加选项

--lock-all-tables 是对要备份的数据库的所有表施加读锁

--lock-table 仅对单张表施加读锁，即使是备份整个数据库，它也是在我们备份某张表的时候才对该表施加读锁，因此适用于备份单张表

#### **2、在服务器端书写命令，**

mysql> flush tables with read lock; 施加锁，表示把位于内存上的表统统都同步到磁盘上去，然后施加读锁

mysql> flush tables with read lock;释放读锁

但这对于InnoDB存储引擎来讲，虽然你也能够请求道读锁，但是不代表它的所有数据都已经同步到磁盘上，因此当面对InnoDB的时候，我们要使用mysql> show engine innodb status; 看看InnoDB所有的数据都已经同步到磁盘上去了，才进行备份操作。

2.2备份的策略：

完全备份+增量备份+二进制日志

2.3 先给数据库做完全备份：

复制代码代码如下:

[root@www ~]# mysqldump -uroot --single-transaction --master-data=2 --databases hellodb > /backup/hellodb\_`date +%F`.sql

--single-transaction: 基于此选项能实现热备InnoDB表；因此，不需要同时使用--lock-all-tables；   
--master-data=2  记录备份那一时刻的二进制日志的位置，并且注释掉，1是不注释的   
--databases hellodb 指定备份的数据库

2.4回到mysql服务器端更新数据

复制代码代码如下:

mysql> create table tb1(id int); 创建表   
mysql> insert into tb1 values (1),(2),(3);  插入数据，这里只做演示，随便插入了几个数据

2.5先查看完全备份文件里边记录的位置：

复制代码代码如下:

[root@www backup]# cat hellodb\_2013-09-08.sql | less   
-- CHANGE MASTER TO MASTER\_LOG\_FILE='mysql-bin.000013', MASTER\_LOG\_POS=15684; 记录了二进制日志的位置

2.6 在回到服务器端：

复制代码代码如下:

mysql> show master status; 显示此时的二进制日志的位置   
 从备份文件里边记录的位置到我们此时的位置，即为增量的部分   
+------------------+----------+--------------+------------------+   
| File             | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |   
+------------------+----------+--------------+------------------+   
| mysql-bin.000004 |      15982 |              |                  |   
+------------------+----------+--------------+------------------+

2.7做增量备份

复制代码代码如下:

[root@www backup]# mysqlbinlog --start-position=15694 --stop-position=15982  
/mydata/data/mysql-bin.000013 > /backup/hellodb\_`date +$F\_%H`.sql

2.8再回到服务器

复制代码代码如下:

mysql> insert into tb1 values (4),(5); 在插入一些数值   
mysql> drop database hellodb;   删除hellodb库

2.9导出这次得二进制日志：

复制代码代码如下:

[root@www backup]# mysqlbinlog --start-position=15982 /mydata/data/mysql-bin.000013 查看删除操作时二进制日志的位置   
[root@www backup]# mysqlbinlog --start-position=15982 --stop-position=16176 /mydata/data/mysql-bin.000013 > /tmp/hellodb.sql  //导出二进制日志

2.10先让mysql离线

回到服务器端：

复制代码代码如下:

mysql> set sql\_log\_bin=0;  关闭二进制日志   
mysql> flush logs; 滚动下日志

2.11模拟数据库损坏

复制代码代码如下:

mysql> drop database hellodb;

2.12开始恢复数据：

复制代码代码如下:

[root@www ]# mysql < /backup/hellodb\_2013-09-08.sql  //导入完全备份文件   
[root@www ]# mysql < /backup/hellodb\_2013-09-08\_05.sql //导入增量备份文件   
[root@www ]# mysql< hellodb.sql //导入二进制文件

验证完成，显示结果为我们预想的那样

注：

1. 真正在生产环境中，我们应该导出的是整个mysql服务器中的数据，而不是单个库，因此应该使用--all-databases  
   2、在导出二进制日志的时候，可以直接复制文件即可，但是要注意的是，备份之前滚动下日志。

#### **3、利用lvm快照实现几乎热备的数据备份与恢复**

3.1策略：

完全备份+二进制日志；

3.2准备：

注：事务日志必须跟数据文件在同一个LV上；

3.3创建lvm Lvm的创建这里就不多说了，想了解话点击[http://www.jb51.net/LINUXjishu/105937.html](http://www.jb51.net/LINUXjishu/105937.html" \t "http://www.jb51.net/article/_blank)

3.4 修改mysql主配置文件存放目录内的文件的权限与属主属组，并初始化mysql

复制代码代码如下:

[root@www ~]# mkdir /mydata/data             //创建数据目录   
[root@www ~]# chown mysql:mysql /mydata/data  //改属组属主   
[root@www ~]#   
[root@www ~]# cd /usr/local/mysql/    //必须站在此目录下         
[root@www mysql]# scripts/mysql\_install\_db --user=mysql --datadir=/mydata/data  //初始化mysql

3.5修改配置文件：

复制代码代码如下:

vim /etc/my.cof   
datadir=/mydata/data   添加数据目录   
sync\_binlog = 1  开启此功能

3.6 启动服务

复制代码代码如下:

[root@www mysql]# service mysqld start   
mysql> set session sql\_log\_bin=0;  关闭二进制日志   
mysql> source /backup/all\_db\_2013-09-08.sql   读取备份文件

3.7回到mysql服务器：

复制代码代码如下:

mysql> FLUSH TABLES WITH READ LOCK; 请求读锁   
注：不要退出，另起一个终端：   
mysql> SHOW MASTER STATUS;          查看二进制文件的位置   
+------------------+----------+--------------+------------------+   
| File             | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |   
+------------------+----------+--------------+------------------+   
| mysql-bin.000004 |      107 |              |                  |   
+------------------+----------+--------------+------------------+   
1 row in set (0.00 sec)   
mysql> FLUSH LOGS;  建议滚动下日志。这样备份日志的时候就会很方便了

3.8导出二进制文件，创建个目录单独存放

复制代码代码如下:

[root@www ~]# mkdir /backup/limian   
[root@www ~]# mysql -e 'show master status;' > /backup/limian/binlog.txt   
[root@www ~]#

3.9为数据所在的卷创建快照：

复制代码代码如下:

[root@www ~]# lvcreate -L 100M -s -p r -n mysql\_snap /dev/myvg/mydata

回到服务器端，释放读锁

复制代码代码如下:

mysql> UNLOCK TABLES;   
[root@www ~]# mount /dev/myvg/mysql\_snap /mnt/data   
[root@www data]# cp \* /backup/limian/   
[root@www data]#lvremove /dev/myvg/mylv\_snap

3.10更新数据库的数据，并删除数据目录先的数据文件，模拟数据库损坏

复制代码代码如下:

mysql>  create table limiantb (id int,name CHAR(10));   
mysql> insert into limiantb values (1,'tom');   
[root@www data]# mysqlbinlog --start-position=187 mysql-bin.000003 > /backup/limian/binlog.sql   
[root@www backup]# cd /mydata/data/   
[root@www data]#  rm -rf \*   
[root@www ~]# cp -a /backup/limian/\* /mydata/data/   
[root@www data]# chown mysql:mysql \*

3.11测试

启动服务

复制代码代码如下:

[root@www data]# service mysqld start   
[root@www data]# mysql 登陆测试   
mysql> SHOW DATABASES;   
mysql> SET sql\_log\_bin=0  
mysql> source/backup/limian/binlog.sql; #二进制恢复   
mysql> SHOW TABLES;         #查看恢复结果   
mysql> SET sql\_log\_bin=1;   #开启二进制日志

注：此方式实现了接近于热备的方式备份数据文件，而且数据文件放在lvm中可以根据数据的大小灵活改变lvm的大小，备份的方式也很简单。

#### **4、基于Xtrabackup做备份恢复**

官方站点：www.percona.com

优势：

1、快速可靠的进行完全备份  
2、在备份的过程中不会影响到事务  
3、支持数据流、网络传输、压缩，所以它可以有效的节约磁盘资源和网络带宽。  
4、可以自动备份校验数据的可用性。

安装Xtrabackup

复制代码代码如下:

[root@www ~]# rpm -ivh percona-xtrabackup-2.1.4-656.rhel6.i686.rpm

其最新版的软件可从 http://www.percona.com/software/percona-xtrabackup/ 获得

注意：在备份数据库的时候，我们应该具有权限，但需要注意的是应该给备份数据库时的用户最小的权限，以保证安全性，

4.1前提：

应该确定采用的是单表一个表空间，否则不支持单表的备份与恢复。  
在配置文件里边的mysqld段加上

innodb\_file\_per\_table = 1

4.2备份策略  
完全备份+增量备份+二进制日志  
4.3准备个目录用于存放备份数据

复制代码代码如下:

[root@www ~]# makdir /innobackup

4.4做完全备份：

复制代码代码如下:

[root@www ~]# innobackupex --user=root --password=mypass /innobackup/

注：

1、只要在最后一行显示 innobackupex: completed OK!，就说明你的备份是正确的。  
2、另外要注意的是每次备份之后，会自动在数据目录下创建一个以当前时间点命名的目录用于存放备份的数据，那我们去看看都有什么

[root@www 2013-09-12\_11-03-04]# ls  
backup-my.cnf ibdata1 performance\_schema xtrabackup\_binary xtrabackup\_checkpoints  
hellodb mysql test xtrabackup\_binlog\_info xtrabackup\_logfile  
[root@www 2013-09-12\_11-03-04]#  
xtrabackup\_checkpoints ：备份类型、备份状态和LSN(日志序列号)范围信息；  
xtrabackup\_binlog\_info ：mysql服务器当前正在使用的二进制日志文件及至备份这一刻为止二进制日志事件的位置。  
xtrabackup\_logfile ：非文本文件，xtrabackup自己的日志文件  
xtrabackup\_binlog\_pos\_innodb ：二进制日志文件及用于InnoDB或XtraDB表的二进制日志文件的当前position。  
backup-my.cnf ：备份时数据文件中关于mysqld的配置

4.5回到mysql服务器端对数据进行更新操作

复制代码代码如下:

mysql> use hellodb;   
mysql> delete from students where StuID>=24;

4.6增量备份

复制代码代码如下:

innobackupex --user=root --password=mypass --incremental /innobackup/--incremental-basedir=/innobackup/2013-09-12\_11-03-04/   
--incremental  指定备份类型   
--incremental-basedir= 指定这次增量备份是基于哪一次备份的，这里是完全备份文件,这样可以把增量备份的数据合并到完全备份中去

4.7第二次增量

先去修改数据

复制代码代码如下:

mysql> insert into students (Name,Age,Gender,ClassID,TeacherID) values ('tom',33,'M',2,4);   
innobackupex --user=root --password=mypass --incremental /innobackup/ --incremental-basedir=/innobackup/2013-09-12\_11-37-01/   
 这里只须要把最后的目录改为第一次增量备份的数据目录即可

4.8最后一次对数据更改但是没做增量备份

复制代码代码如下:

mysql> delete from coc where id=14;

4.9把二进制日志文件备份出来，(因为最后一次修改，没做增量备份，要依赖二进制日志做时间点恢复)

复制代码代码如下:

[root@www data]# cp mysql-bin.000003 /tmp/

4.10模拟数据库崩溃

复制代码代码如下:

[root@www data]# service mysqld stop   
[root@www data]# rm -rf \*

恢复前准备

4.11对完全备份做数据同步

复制代码代码如下:

[root@www ~]# innobackupex --apply-log --redo-only /innobackup/2013-09-12\_11-03-04/

4.12对第一次增量做数据同步

复制代码代码如下:

innobackupex --apply-log --redo-only /innobackup/2013-09-12\_11-03-04/ --incremental-basedir=/innobackup/2013-09-12\_11-37-01/

4.13对第二次增量做数据同步

复制代码代码如下:

innobackupex --apply-log --redo-only /innobackup/2013-09-12\_11-03-04/ --incremental-basedir=/innobackup/2013-09-12\_11-45-53/   
--apply-log 的意义在于把备份时没commit的事务撤销，已经commit的但还在事务日志中的应用到数据库

注：

对于xtrabackup来讲，它是基于事务日志和数据文件备份的，备份的数据中可能会包含尚未提交的事务或已经提交但尚未同步至数据库文件中的事务，还应该对其做预处理，把已提交的事务同步到数据文件，未提交的事务要回滚。因此其备份的数据库，不能立即拿来恢复。

预处理的过程：

首先对完全备份文件只把已提交的事务同步至数据文件，要注意的是有增量的时候，不能对事务做数据回滚，不然你的增量备份就没有效果了。

然后把第一次的增量备份合并到完全备份文件内，

以此类推，把后几次的增量都合并到前一次合并之后的文件中，这样的话，我们只要拿着完全备份+二进制日志，就可以做时间点恢复。

4.14数据恢复

复制代码代码如下:

[root@www ~]# service mysqld stop   
[root@www data]# rm -rf \*  模拟数据库崩溃   
[root@www ~]# innobackupex --copy-back /innobackup/2013-09-12\_11-03-04/   
--copy-back数据库恢复，后面跟上备份目录的位置

4.15检测：

复制代码代码如下:

[root@www ~]# cd /mydata/data/   
[root@www data]# chown mysql:mysql \*   
[root@www data]#service mysqld start

检测结果数据正常。

## 20.数据共享

## 21.

## 22.

## 23.

## 24.

## 25.

## 26.

## 27.

## 28.

## 29.

## 30.