MySQL高级

1. mysql的架构介绍
   1. Mysql简介
      1. 概述

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司。

MySQL是一种关联数据库管理系统，将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL是开源的，所以你不需要支付额外的费用。

MySQL是可以定制的，采用了GPL协议，你可以修改源码来开发自己的Mysql系统。

MySQL支持大型的数据库。可以处理拥有上千万条记录的大型数据库。

MySQL使用标准的SQL数据语言形式。

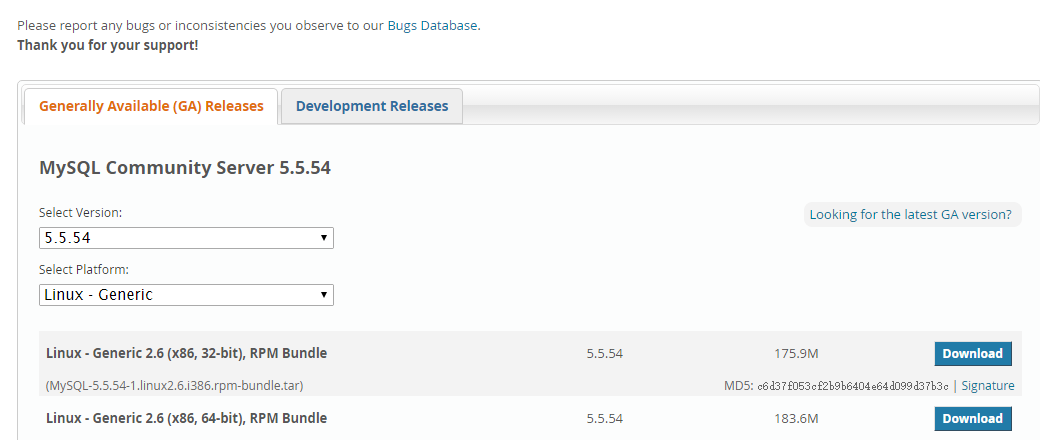
MySQL可以允许于多个系统上，并且支持多种语言。这些编程语言包括C、C++、Python、Java、Perl、PHP、Eiffel、Ruby和Tcl等。

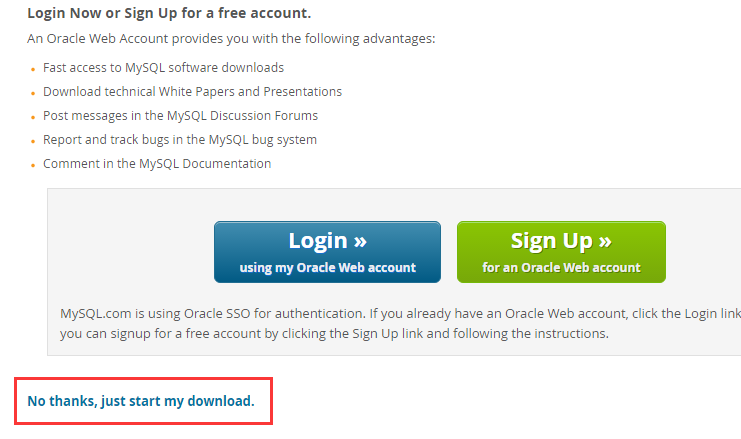
MySQL支持大型数据库，支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB，64位系统支持最大的表文件为8TB。

* 1. MysqlLinux版的安装
     1. mysql5.5

下载地址

官网下载地址：http://dev.mysql.com/downloads/mysql/





拷贝&解压缩

检查工作

检查当前系统是否安装过mysql，执行安装命令前，先执行查询命令

**1、CentOS6**

**rpm -qa|grep mysql**

如果存在mysql-libs的旧版本包如下：

graphic

请先执行卸载命令：**rpm -e --nodeps mysql-libs**

**1、CentOS7**

**rpm -qa|grep mariadb**

如果存在如下：

graphic

请先执行卸载命令：**rpm -e --nodeps mariadb-libs**

检查/tmp文件夹权限

由于mysql安装过程中，会通过mysql用户在/tmp目录下新建tmp\_db文件，所以请给/tmp较大的权限

执行 ：chmod -R 777 /tmp

安装

在mysql的安装文件目录下执行：

rpm -ivh MySQL-client-5.5.54-1.linux2.6.x86\_64.rpm

rpm -ivh MySQL-server-5.5.54-1.linux2.6.x86\_64.rpm

查看MySQL安装版本

或者可以执行 mysqladmin --version命令，类似java -version如果打出消息，即为成功。

graphic

通过vim 查看 mysql用户和mysql组

vim /etc/passwd

vim /etc/group

mysql服务的启+停

启动：

graphic

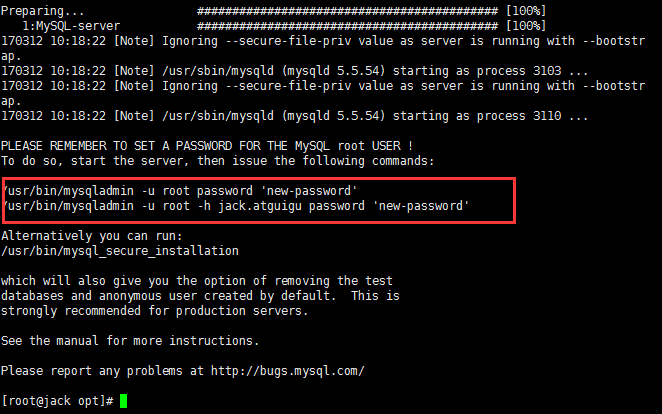
关闭：

graphic

首次登录

安装完成后会提示出如下的提示：

在mysql首次登录前要给 root 账号设置密码

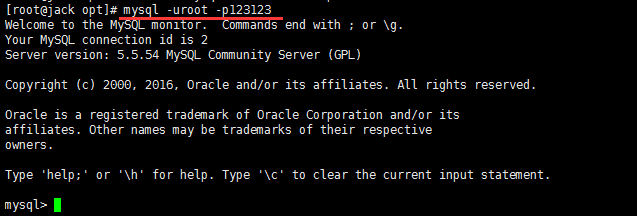


启动服务后，执行命令

/usr/bin/mysqladmin -u root password '123123'

graphic

然后通过 mysql -uroot -p123123进行登录

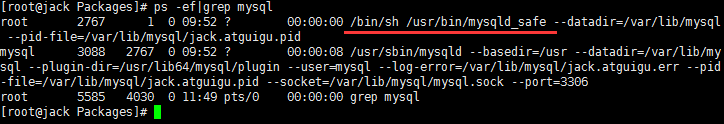


MySQL的安装位置

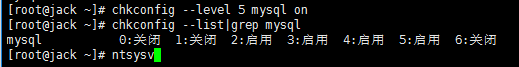
在linux下查看安装目录 ps -ef|grep mysql

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **路径** | **解释** | **备注** |
| --basedir | /usr/bin | 相关命令目录 | mysqladmin mysqldump等命令 |
| --datadir | /var/lib/mysql/ | mysql数据库文件的存放路径 |  |
| --plugin-dir | /usr/lib64/mysql/plugin | mysql插件存放路径 |  |
| --log-error | /var/lib/mysql/jack.atguigu.err | mysql错误日志路径 |  |
| --pid-file | /var/lib/mysql/jack.atguigu.pid | 进程pid文件 |  |
| --socket | /var/lib/mysql/mysql.sock | 本地连接时用的unix套接字文件 |  |
|  | /usr/share/mysql | 配置文件目录 | mysql脚本及配置文件 |
|  | /etc/init.d/mysql | 服务启停相关脚本 |  |

在linux下查看安装目录 ps -ef|grep mysql



自启动mysql服务

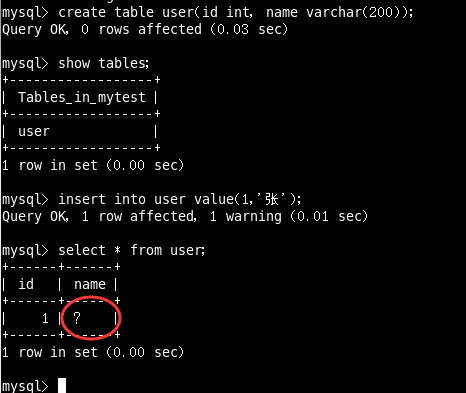


chkconfig --level 5 mysql

chkconfig --list | grep mysql

修改字符集问题

尝试插入输入：



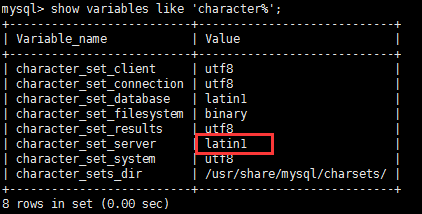
原因是字符集问题

**1 查看字符集**

show variables like 'character%';

show variables like '%char%';

看看出现的结果：



默认的服务器用了latin1，所以会乱码。

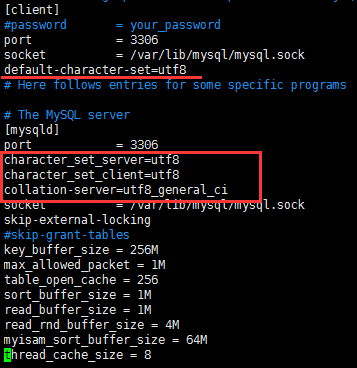
**2 修改my-huge.cnf**

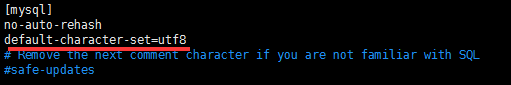
在/usr/share/mysql/ 中找到my-huge.cnf的配置文件，

拷贝其中的my-huge.cnf 到 /etc/ 并命名为my.cnf

然后修改my.cnf:

|  |
| --- |
| [client]  default-character-set=utf8  [mysqld]  character\_set\_server=utf8  character\_set\_client=utf8  collation-server=utf8\_general\_ci  [mysql]  default-character-set=utf8 |





**3、重新启动mysql**

查看原库的字符集：show create database mydb

但是原库的设定不会发生变化，参数修改之对新建的数据库生效

**4、已生成的库表字符集如何变更**

**修改数据库的字符集**

mysql> alter database mydb character set 'utf8';

**修改数据表的字符集**

mysql> alter table mytbl convert to character set 'utf8';

但是原有的数据如果是用非'utf8'编码的话，数据本身不会发生改变。

* 1. Mysql 的用户与权限管理
     1. MySQL的用户管理

创建用户

**创建用户**

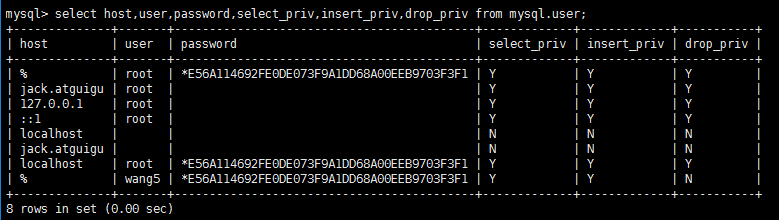
create user zhang3 identified by '123123';

表示创建名称为zhang3的用户，密码设为123123；

了解user表

**查看用户**

select host,user,password,select\_priv,insert\_priv,drop\_priv from mysql.user;



**host ：表示连接类型**

% 表示所有远程通过 TCP方式的连接

IP 地址 如 (192.168.1.2,127.0.0.1) 通过制定ip地址进行的TCP方式的连接

机器名 通过制定i网络中的机器名进行的TCP方式的连接

::1 IPv6的本地ip地址 等同于IPv4的 127.0.0.1

localhost 本地方式通过命令行方式的连接 ，比如mysql -u xxx -p 123xxx 方式的连接。

**User : 表示用户名**

同一用户通过不同方式链接的权限是不一样的。

**password ：密码**

所有密码串通过 password(明文字符串) 生成的密文字符串。加密算法为MYSQLSHA1 ，不可逆 。

mysql 5.7 的密码保存到 authentication\_string 字段中不再使用password 字段。

**select\_priv , insert\_priv等**

为该用户所拥有的权限。

设置密码

修改当前用户的密码:

set password =password('123456')

修改某个用户的密码:

update mysql.user set password=password('123456') where user='li4';

flush privileges; #所有通过user表的修改，必须用该命令才能生效。

修改用户

修改用户名：

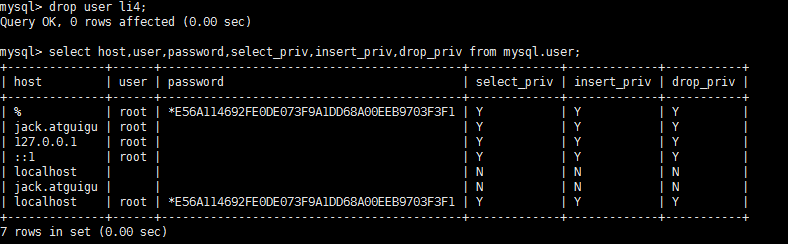
update mysql.user set user='li4' where user='wang5';

flush privileges; #所有通过user表的修改，必须用该命令才能生效。

graphic

删除用户

**drop user li4 ;**



不要通过delete from user u where user='li4' 进行删除，系统会有残留信息保留。

* + 1. 权限管理

授予权限

授权命令：

**grant 权限1,权限2,…权限n on 数据库名称.表名称 to 用户名@用户地址 identified by ‘连接口令’;**

**该权限如果发现没有该用户，则会直接新建一个用户。**

比如

**grant select,insert,delete,drop on atguigudb.\* to li4@localhost ;**

#给li4用户用本地命令行方式下，授予atguigudb这个库下的所有表的插删改查的权限。

**grant all privileges on \*.\* to root@'%' identified by '123';**

#授予通过网络方式登录的root用户 ，对所有库所有表的全部权限，密码设为123.

收回权限

查看当前用户权限

show grants;

**收回权限命令：**

**revoke 权限1,权限2,…权限n on 数据库名称.表名称 from 用户名@用户地址 ;**

REVOKE ALL PRIVILEGES ON mysql.\* FROM joe@localhost;

#收回全库全表的所有权限

REVOKE select,insert,update,delete ON mysql.\* FROM joe@localhost;

#收回mysql库下的所有表的插删改查权限

必须用户重新登录后才能生效

查看权限

查看当前用户权限

show grants;

查看某用户的全局权限

select \* from user ;

查看某用户的某个表的权限

select \* from tables\_priv;

* + 1. 通过工具远程访问

1、先 ping 一下数据库服务器的ip 地址确认网络畅通。

pint 192.168.0.121

2、关闭数据库服务的防火墙

service iptables stop

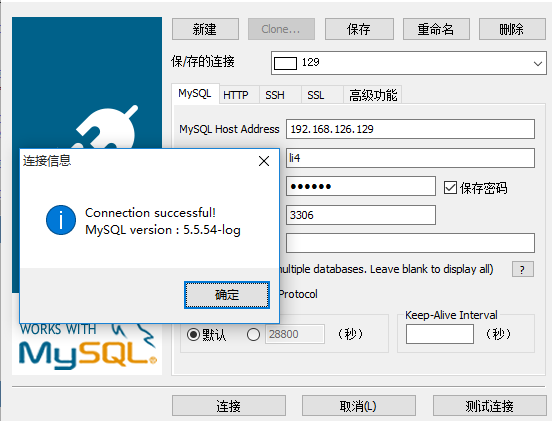
3、 确认Mysql中已经有可以通过远程登录的账户

select \* from mysql.user where user='li4' and host='%';

如果没有用户,先执行如下命令：

grant all privileges on \*.\* to li4@'%' identified by '123123';

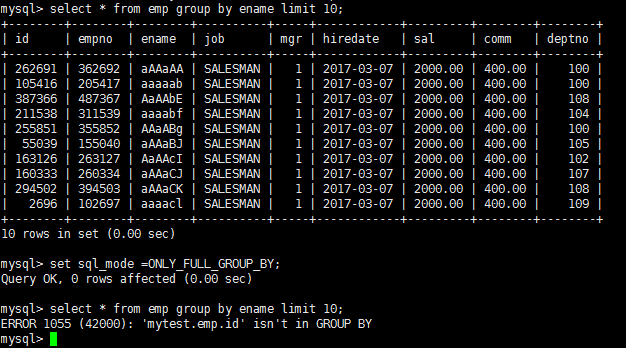
4、测试连接：



* 1. Mysql的一些杂项配置
     1. sql\_mode

**MySQL的sql\_mode合理设置**

sql\_mode是个很容易被忽视的变量，默认值是空值，在这种设置下是可以允许一些非法操作的，比如允许一些非法数据的插入。在生产环境必须将这个值设置为严格模式，所以开发、测试环境的数据库也必须要设置，这样在开发测试阶段就可以发现问题。



show variables like 'sql\_mode';

**sql\_mode常用值如下：**

set sql\_mode='ONLY\_FULL\_GROUP\_BY';

**ONLY\_FULL\_GROUP\_BY：**

对于GROUP BY聚合操作，如果在SELECT中的列，没有在GROUP BY中出现，那么这个SQL是不合法的，因为列不在GROUP BY从句中

**NO\_AUTO\_VALUE\_ON\_ZERO：**

该值影响自增长列的插入。默认设置下，插入0或NULL代表生成下一个自增长值。如果用户 希望插入的值为0，而该列又是自增长的，那么这个选项就有用了。

**STRICT\_TRANS\_TABLES：**

在该模式下，如果一个值不能插入到一个事务表中，则中断当前的操作，对非事务表不做限制

**NO\_ZERO\_IN\_DATE：**

在严格模式下，不允许日期和月份为零

**NO\_ZERO\_DATE：**

设置该值，mysql数据库不允许插入零日期，插入零日期会抛出错误而不是警告。

**ERROR\_FOR\_DIVISION\_BY\_ZERO：**

在INSERT或UPDATE过程中，如果数据被零除，则产生错误而非警告。如 果未给出该模式，那么数据被零除时MySQL返回NULL

**NO\_AUTO\_CREATE\_USER：**

禁止GRANT创建密码为空的用户

**NO\_ENGINE\_SUBSTITUTION：**

如果需要的存储引擎被禁用或未编译，那么抛出错误。不设置此值时，用默认的存储引擎替代，并抛出一个异常

**PIPES\_AS\_CONCAT：**

将"||"视为字符串的连接操作符而非或运算符，这和Oracle数据库是一样的，也和字符串的拼接函数Concat相类似

**ANSI\_QUOTES：**

启用ANSI\_QUOTES后，不能用双引号来引用字符串，因为它被解释为识别符

ORACLE：

设置等同：PIPES\_AS\_CONCAT, ANSI\_QUOTES, IGNORE\_SPACE, NO\_KEY\_OPTIONS, NO\_TABLE\_OPTIONS, NO\_FIELD\_OPTIONS, NO\_AUTO\_CREATE\_USER.

例子

|  |
| --- |
| CREATE TABLE mytbl2 (id INT,NAME VARCHAR(200),age INT,dept INT);  INSERT INTO mytbl2 VALUES(1,'zhang3',33,101);  INSERT INTO mytbl2 VALUES(2,'li4',34,101);  INSERT INTO mytbl2 VALUES(3,'wang5',34,102);  INSERT INTO mytbl2 VALUES(4,'zhao6',34,102);  INSERT INTO mytbl2 VALUES(5,'tian7',36,102); |

**#每个部门年龄最大的人**

|  |
| --- |
| SELECT NAME,dept,MAX(age) FROM mytbl2 GROUP BY dept; |

上面语句是错的

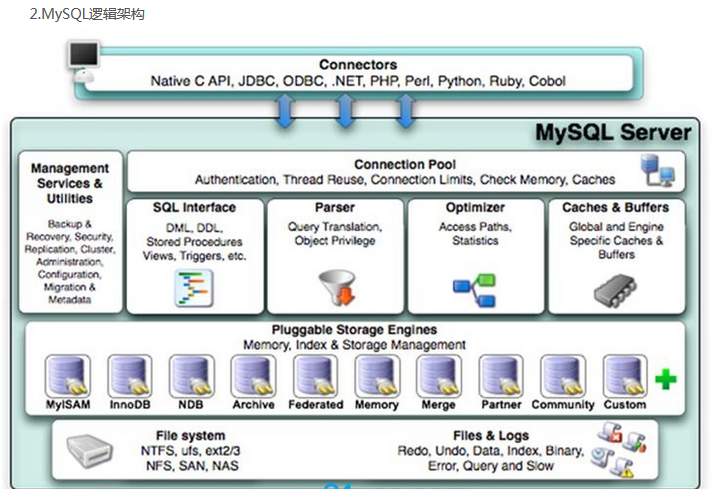
group by使用原则：select 后面只能放 函数 和group by后的字段

**1、每个机构最大年龄 2、的人**

|  |
| --- |
| SELECT \* FROM mytbl2 m INNER JOIN(  SELECT dept,MAX(age) maxage FROM mytbl2 GROUP BY dept  ) ab ON ab.dept=m.dept AND m.age=ab.maxage |

* 1. Mysql逻辑架构介绍
     1. 总体概览

和其它数据库相比，MySQL有点与众不同，它的架构可以在多种不同场景中应用并发挥良好作用。主要体现在存储引擎的架构上，插件式的存储引擎架构将查询处理和其它的系统任务以及数据的存储提取相分离。这种架构可以根据业务的需求和实际需要选择合适的存储引擎。



**1.连接层**

最上层是一些客户端和连接服务，包含本地sock通信和大多数基于客户端/服务端工具实现的类似于tcp/ip的通信。主要完成一些类似于连接处理、授权认证、及相关的安全方案。在该层上引入了线程池的概念，为通过认证安全接入的客户端提供线程。同样在该层上可以实现基于SSL的安全链接。服务器也会为安全接入的每个客户端验证它所具有的操作权限。

**2.服务层**

**2.1 Management Serveices & Utilities： 系统管理和控制工具**

**2.2 SQL Interface**: SQL接口

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

**2.3 Parser**: 解析器

SQL命令传递到解析器的时候会被解析器验证和解析。

**2.4 Optimizer**: 查询优化器。

SQL语句在查询之前会使用查询优化器对查询进行优化。

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

优化器来决定先投影还是先过滤。

**2.5 Cache和Buffer： 查询缓存。**

如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数据。

这个缓存机制是由一系列小缓存组成的。比如表缓存，记录缓存，key缓存，权限缓存等

**3.引擎层**

存储引擎层，存储引擎真正的负责了MySQL中数据的存储和提取，服务器通过API与存储引擎进行通信。不同的存储引擎具有的功能不同，这样我们可以根据自己的实际需要进行选取。后面介绍MyISAM和InnoDB

**4.存储层**

数据存储层，主要是将数据存储在运行于裸设备的文件系统之上，并完成与存储引擎的交互。

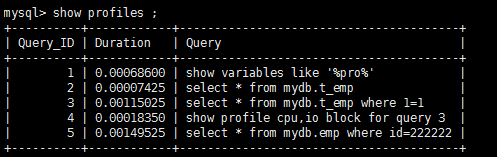
* + 1. 利用show profile 查看sql的执行周期

先开启 show variables like '%profiling%';

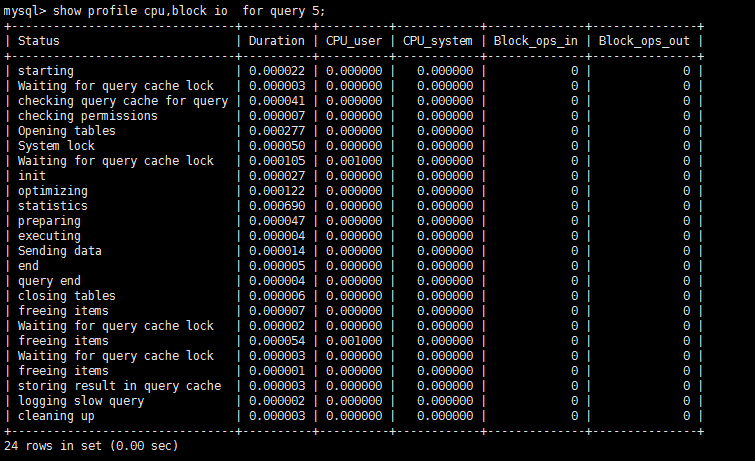
set profiling=1;

select \* from xxx ;

show profiles; #显示最近的几次查询



show profile cpu,block io for query Query\_ID #查看程序的执行步骤



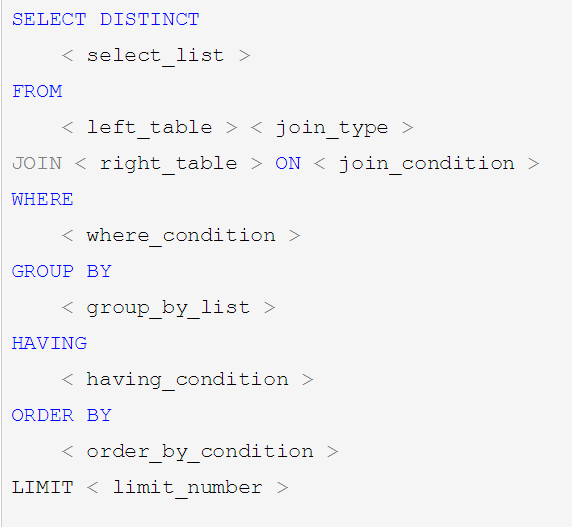
* + 1. 查询说明

查询流程图：

首先，mysql的查询流程大致是：

* mysql客户端通过协议与mysql服务器建连接，发送查询语句，先检查查询缓存，如果命中，直接返回结果，否则进行语句解析,也就是说，在解析查询之前，服务器会先访问查询缓存(query cache)——它存储SELECT语句以及相应的查询结果集。如果某个查询结果已经位于缓存中，服务器就不会再对查询进行解析、优化、以及执行。它仅仅将缓存中的结果返回给用户即可，这将大大提高系统的性能。
* 语法解析器和预处理：首先mysql通过关键字将SQL语句进行解析，并生成一颗对应的“解析树”。mysql解析器将使用mysql语法规则验证和解析查询；预处理器则根据一些mysql规则进一步检查解析数是否合法。
* 查询优化器当解析树被认为是合法的了，并且由优化器将其转化成执行计划。一条查询可以有很多种执行方式，最后都返回相同的结果。优化器的作用就是找到这其中最好的执行计划。。
* 然后，mysql默认使用的BTREE索引，并且一个大致方向是:无论怎么折腾sql，至少在目前来说，mysql最多只用到表中的一个索引。
  + 1. SQL执行顺序

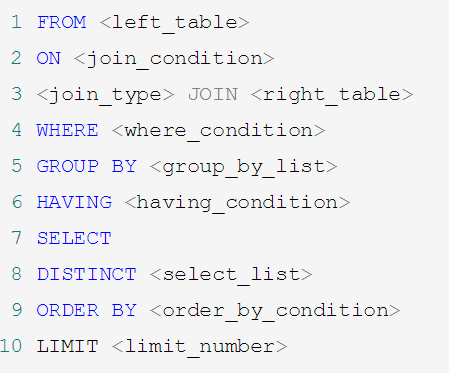
手写



机读

随着Mysql版本的更新换代，其优化器也在不断的升级，优化器会分析不同执行顺序产生的性能消耗不同而动态调整执行顺序。

下面是经常出现的查询顺序：

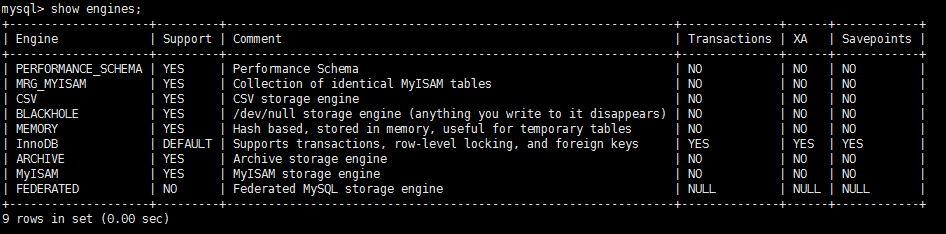


* 1. Mysql存储引擎
     1. 查看命令

1 如何用命令查看

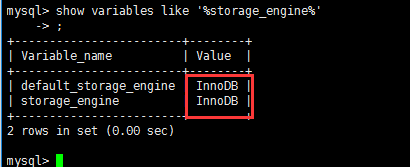
#看你的mysql现在已提供什么存储引擎:

mysql> show engines;



#看你的mysql当前默认的存储引擎:

mysql> show variables like '%storage\_engine%';



* + 1. 各个引擎简介

**1、InnoDB存储引擎**

InnoDB是MySQL的默认事务型引擎，它被设计用来处理大量的短期(short-lived)事务。除非有非常特别的原因需要使用其他的存储引擎，否则应该优先考虑InnoDB引擎。

**2、MyISAM存储引擎**

MyISAM提供了大量的特性，包括全文索引、压缩、空间函数(GIS)等，但MyISAM不支持事务和行级锁，有一个毫无疑问的缺陷就是崩溃后无法安全恢复。

**3、Archive引擎**

Archive档案存储引擎只支持INSERT和SELECT操作，在MySQL5.1之前不支持索引。

Archive表适合日志和数据采集类应用。

根据英文的测试结论来看，Archive表比MyISAM表要小大约75%，比支持事务处理的InnoDB表小大约83%。

**4、Blackhole引擎**

Blackhole引擎没有实现任何存储机制，它会丢弃所有插入的数据，不做任何保存。但服务器会记录Blackhole表的日志，所以可以用于复制数据到备库，或者简单地记录到日志。但这种应用方式会碰到很多问题，因此并不推荐。

**5、CSV引擎**

CSV引擎可以将普通的CSV文件作为MySQL的表来处理，但不支持索引。

CSV引擎可以作为一种数据交换的机制，非常有用。

CSV存储的数据直接可以在操作系统里，用文本编辑器，或者excel读取。

**6、Memory引擎**

如果需要快速地访问数据，并且这些数据不会被修改，重启以后丢失也没有关系，那么使用Memory表是非常有用。Memory表至少比MyISAM表要快一个数量级。

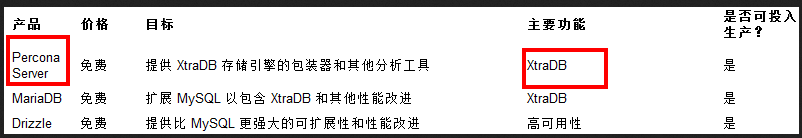
**7、Federated引擎**

Federated引擎是访问其他MySQL服务器的一个代理，尽管该引擎看起来提供了一种很好的跨服务器的灵活性，但也经常带来问题，因此默认是禁用的。

* + 1. MyISAM和InnoDB

|  |  |  |
| --- | --- | --- |
| **对比项** | **MyISAM** | **InnoDB** |
| 外键 | 不支持 | 支持 |
| 事务 | 不支持 | 支持 |
| 行表锁 | 表锁，即使操作一条记录也会锁住整个表，不适合高并发的操作 | 行锁,操作时只锁某一行，不对其它行有影响，  适合高并发的操作 |
| 缓存 | 只缓存索引，不缓存真实数据 | 不仅缓存索引还要缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响 |
| 关注点 | 节省资源、消耗少、简单业务 | 并发写、事务、更大资源 |
| 默认安装 | Y | Y |
| 默认使用 | N | Y |
| 自带系统表使用 | Y | N |

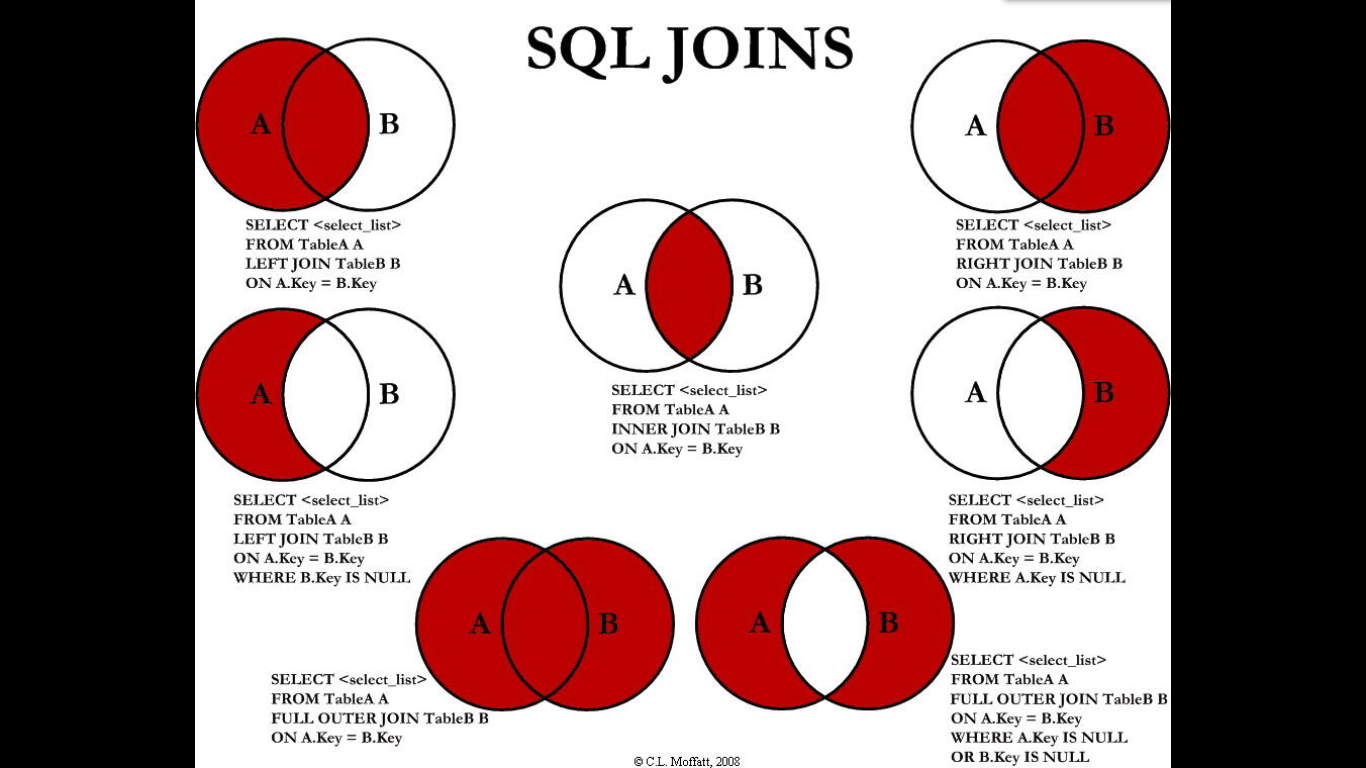
* + 1. 阿里巴巴、淘宝用哪个



* Percona 为 MySQL 数据库服务器进行了改进，在功能和性能上较 MySQL 有着很显著的提升。该版本提升了在高负载情况下的 InnoDB 的性能、为 DBA 提供一些非常有用的性能诊断工具；另外有更多的参数和命令来控制服务器行为。
* 该公司新建了一款存储引擎叫xtradb完全可以替代innodb,并且在性能和并发上做得更好,
* 阿里巴巴大部分mysql数据库其实使用的percona的原型加以修改。
* AliSql+AliRedis

1. 索引优化分析
   1. 性能下降SQL慢 执行时间长 等待时间长
      1. 数据过多

* 分库分表
  + 1. 关联了太多的表，太多join
* SQL优化
  + 1. 没有充分利用到索引
* 索引建立
  + 1. 服务器调优及各个参数设置
* 调整my.cnf
  1. SQL预热：常见通用的Join查询
     1. Join图



* + 1. 建表SQL

|  |
| --- |
| CREATE TABLE `t\_dept` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `deptName` VARCHAR(30) DEFAULT NULL,  `address` VARCHAR(40) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=INNODB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;    CREATE TABLE `t\_emp` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `name` VARCHAR(20) DEFAULT NULL,  `age` INT(3) DEFAULT NULL,  `deptId` INT(11) DEFAULT NULL,  empno INT NOT NULL,  PRIMARY KEY (`id`),  KEY `idx\_dept\_id` (`deptId`)  #CONSTRAINT `fk\_dept\_id` FOREIGN KEY (`deptId`) REFERENCES `t\_dept` (`id`)  ) ENGINE=INNODB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;    INSERT INTO t\_dept(deptName,address) VALUES('华山','华山');  INSERT INTO t\_dept(deptName,address) VALUES('丐帮','洛阳');  INSERT INTO t\_dept(deptName,address) VALUES('峨眉','峨眉山');  INSERT INTO t\_dept(deptName,address) VALUES('武当','武当山');  INSERT INTO t\_dept(deptName,address) VALUES('明教','光明顶');  INSERT INTO t\_dept(deptName,address) VALUES('少林','少林寺');    INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('风清扬',90,1,100001);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('岳不群',50,1,100002);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('令狐冲',24,1,100003);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('洪七公',70,2,100004);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('乔峰',35,2,100005);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('灭绝师太',70,3,100006);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('周芷若',20,3,100007);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('张三丰',100,4,100008);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('张无忌',25,5,100009);  INSERT INTO t\_emp(NAME,age,deptId,empno) VALUES('韦小宝',18,NULL,100010); |

* + 1. 7种JOIN

1 所有有门派的人员信息（A、B两表共有）

select \* from t\_emp a inner join t\_dept b on a.deptId = b.id;

2 列出所有用户，并显示其机构信息 （A的全集）

select \* from t\_emp a left join t\_dept b on a.deptId = b.id;

3 列出所有门派（B的全集）

select \* from t\_dept a left join t\_emp b on a.id = b.deptId;

4 所有不入门派的人员 （A的独有）

select \* from t\_emp a left join t\_dept b on a.deptId = b.id where b.id is null;

5 所有没人入的门派 （B的独有）

select \* from t\_dept b left join t\_emp a on a.deptId = b.id where a.deptId is null;

6 列出所有人员和机构的对照关系(AB全有)

#MySQL Full Join的实现 因为MySQL不支持FULL JOIN,下面是替代方法

#left join + union(可去除重复数据)+ right join

|  |
| --- |
| SELECT \* FROM t\_emp A LEFT JOIN t\_dept B ON A.deptId = B.id  UNION  SELECT \* FROM t\_emp A RIGHT JOIN t\_dept B ON A.deptId = B.id |

7 列出所有没入派的人员和没人入的门派（A的独有+B的独有）

|  |
| --- |
| SELECT \* FROM t\_emp A LEFT JOIN t\_dept B ON A.deptId = B.id WHERE B.`id` IS NULL  UNION  SELECT \* FROM t\_emp A RIGHT JOIN t\_dept B ON A.deptId = B.id WHERE A.`deptId` IS NULL; |

增加掌门字段

|  |
| --- |
| ALTER TABLE `t\_dept`  add CEO INT(11) ;    update t\_dept set CEO=2 where id=1;  update t\_dept set CEO=4 where id=2;  update t\_dept set CEO=6 where id=3;  update t\_dept set CEO=8 where id=4;  update t\_dept set CEO=9 where id=5; |

求各个门派对应的掌门人名称:

select \* from t\_emp a inner join t\_dept b on b.CEO=a.id;

求所有当上掌门人的平均年龄:

select avg(a.age) from t\_emp a inner join t\_dept b on a.id=b.CEO;

求所有人物对应的掌门名称:

* 1. 索引简介
     1. 是什么

MySQL官方对索引的定义为：索引（Index）是帮助MySQL高效获取数据的数据结构。 可以得到索引的本质：索引是数据结构。

索引的目的在于提高查询效率，可以类比字典，

如果要查“mysql”这个单词，我们肯定需要定位到m字母，然后从下往下找到y字母，再找到剩下的sql。

如果没有索引，那么你可能需要a----z，如果我想找到Java开头的单词呢？或者Oracle开头的单词呢？

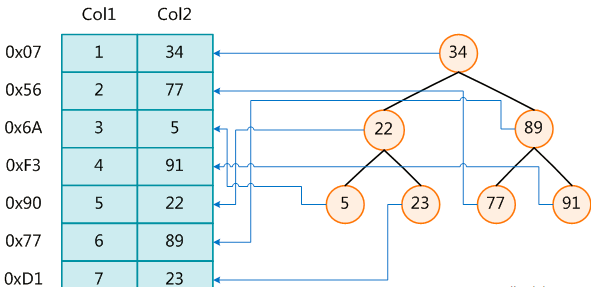
是不是觉得如果没有索引，这个事情根本无法完成？

你可以简单理解为“排好序的快速查找数据结构”。

详解(重要)

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，

这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。下图就是一种可能的索引方式示例：



左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址

为了加快Col2的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在一定的复杂度内获取到相应数据，从而快速的检索出符合条件的记录。

结论

数据本身之外，数据库还维护着一个满足特定查找算法的数据结构，这些数据结构以某种方式指向数据，这样就可以在这些数据结构的基础上实现高级查找算法，这种数据结构就是索引。

一般来说索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上

* + 1. 优势

类似大学图书馆建书目索引，提高数据检索的效率，降低数据库的IO成本

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

* + 1. 劣势

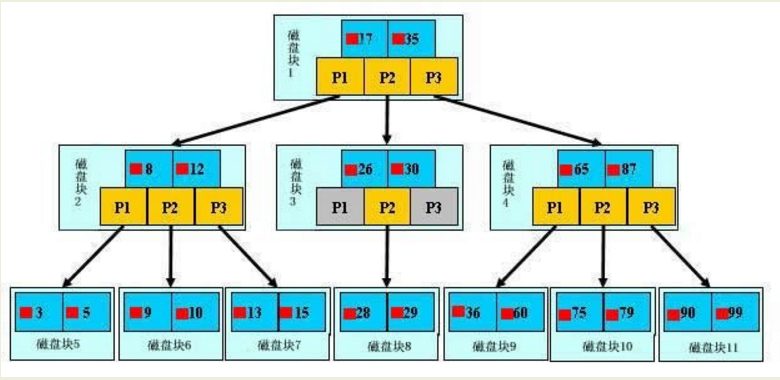
虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。 因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段， 都会调整因为更新所带来的键值变化后的索引信息

实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占用空间的

* + 1. mysql索引结构

BTree索引

原理图



**【初始化介绍】**

一颗b树，浅蓝色的块我们称之为一个磁盘块，可以看到每个磁盘块包含几个数据项（深蓝色所示）和指针（黄色所示），

如磁盘块1包含数据项17和35，包含指针P1、P2、P3，

P1表示小于17的磁盘块，P2表示在17和35之间的磁盘块，P3表示大于35的磁盘块。

***真实的数据存在于叶子节点***即3、5、9、10、13、15、28、29、36、60、75、79、90、99。

非叶子节点只不存储真实的数据，只存储指引搜索方向的数据项，如17、35并不真实存在于数据表中。

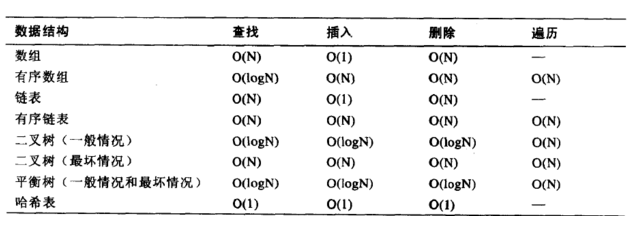
**【查找过程】**

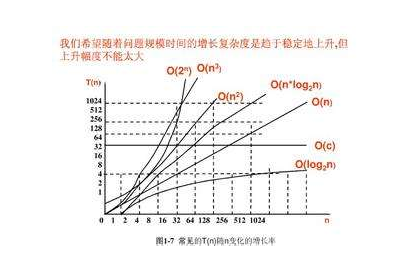
如果要查找数据项29，那么首先会把磁盘块1由磁盘加载到内存，此时发生一次IO，在内存中用二分查找确定29在17和35之间，锁定磁盘块1的P2指针，内存时间因为非常短（相比磁盘的IO）可以忽略不计，通过磁盘块1的P2指针的磁盘地址把磁盘块3由磁盘加载到内存，发生第二次IO，29在26和30之间，锁定磁盘块3的P2指针，通过指针加载磁盘块8到内存，发生第三次IO，同时内存中做二分查找找到29，结束查询，总计三次IO。

真实的情况是，3层的b+树可以表示上百万的数据，如果上百万的数据查找只需要三次IO，性能提高将是巨大的，如果没有索引，每个数据项都要发生一次IO，那么总共需要百万次的IO，显然成本非常非常高。

关于时间复杂度

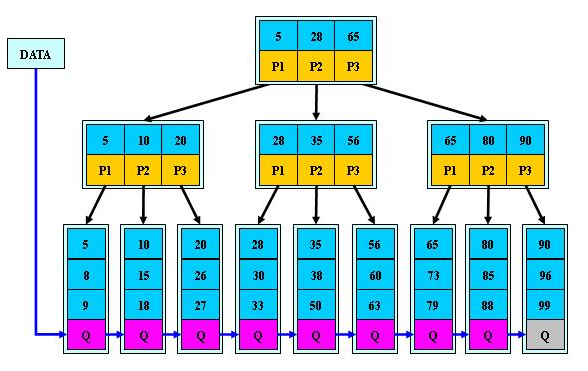
同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。算法分析的目的在于选择合适算法和改进算法。





B+Tree索引

原理图



B树和B+树的区别

B+Tree与B-Tree 的区别

　1）B-树的关键字和记录是放在一起的，叶子节点可以看作外部节点，不包含任何信息；B+树的非叶子节点中只有关键字和指向下一个节点的索引，记录只放在叶子节点中。

　 2）在B-树中，越靠近根节点的记录查找时间越快，只要找到关键字即可确定记录的存在；而B+树中每个记录的查找时间基本是一样的，都需要从根节点走到叶子节点，而且在叶子节点中还要再比较关键字。从这个角度看B-树的性能好像要比B+树好，而在实际应用中却是B+树的性能要好些。因为B+树的非叶子节点不存放实际的数据，这样每个节点可容纳的元素个数比B-树多，树高比B-树小，这样带来的好处是减少磁盘访问次数。尽管B+树找到一个记录所需的比较次数要比B-树多，但是一次磁盘访问的时间相当于成百上千次内存比较的时间，因此实际中B+树的性能可能还会好些，而且B+树的叶子节点使用指针连接在一起，方便顺序遍历（例如查看一个目录下的所有文件，一个表中的所有记录等），这也是很多数据库和文件系统使用B+树的缘故。

思考：为什么说B+树比B-树更适合实际应用中操作系统的文件索引和数据库索引？

1) B+树的磁盘读写代价更低

　　B+树的内部结点并没有指向关键字具体信息的指针。因此其内部结点相对B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多。相对来说IO读写次数也就降低了。

2) B+树的查询效率更加稳定

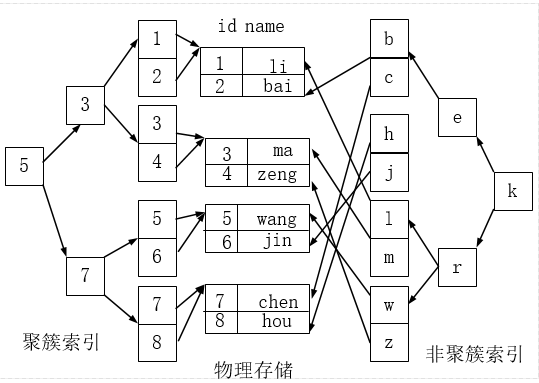
　　由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

聚簇索引与非聚簇索引

聚簇索引并不是一种单独的索引类型，而是一种数据存储方式。

术语‘聚簇’表示数据行和相邻的键值聚簇的存储在一起。

如下图，左侧的索引就是聚簇索引，因为数据行在磁盘的排列和索引排序保持一致。



**聚簇索引的好处：**

* 按照聚簇索引排列顺序，查询显示一定范围数据的时候，由于数据都是紧密相连，数据库不不用从多个数据块中提取数据，所以节省了大量的io操作。

**聚簇索引的限制：**

* 对于mysql数据库目前只有innodb数据引擎支持聚簇索引，而Myisam并不支持聚簇索引。
* 由于数据物理存储排序方式只能有一种，所以每个Mysql的表只能有一个聚簇索引。一般情况下就是该表的主键。
* **为了充分利用聚簇索引的聚簇的特性，所以innodb表的主键列尽量选用有序的顺序id，而不建议用无序的id，比如uuid这种。**
  + 1. mysql索引分类

单值索引

即一个索引只包含单个列，一个表可以有多个单列索引

语法

随表一起建索引：

CREATE TABLE customer (

id INT(10) UNSIGNED AUTO\_INCREMENT,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name)

);

单独建单值索引：

CREATE INDEX idx\_customer\_name ON customer(customer\_name);

删除索引：

DROP INDEX idx\_customer\_name on customer;

唯一索引

索引列的值必须唯一，但允许有空值

语法

随表一起建索引：

CREATE TABLE customer (

id INT(10) UNSIGNED AUTO\_INCREMENT,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name),

UNIQUE (customer\_no)

);

单独建唯一索引：

CREATE UNIQUE INDEX idx\_customer\_no ON customer(customer\_no);

删除索引：

DROP INDEX idx\_customer\_no on customer ;

主键索引

设定为主键后数据库会自动建立索引，innodb为聚簇索引

语法

随表一起建索引：

CREATE TABLE customer (

id INT(10) UNSIGNED AUTO\_INCREMENT,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id)

);

CREATE TABLE customer2 (

id INT(10) UNSIGNED,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id)

);

单独建主键索引：

ALTER TABLE customer add PRIMARY KEY customer(customer\_no);

删除建主键索引：

ALTER TABLE customer drop PRIMARY KEY ;

修改建主键索引：

必须先删除掉(drop)原索引，再新建(add)索引

复合索引

即一个索引包含多个列

语法

随表一起建索引：

CREATE TABLE customer (

id INT(10) UNSIGNED AUTO\_INCREMENT,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name),

UNIQUE (customer\_name),

KEY (customer\_no,customer\_name)

);

单独建索引：

CREATE INDEX idx\_no\_name ON customer(customer\_no,customer\_name);

删除索引：

DROP INDEX idx\_no\_name on customer ;

基本语法

创建

CREATE [UNIQUE ] INDEX [indexName] ON table\_name(column))

删除

DROP INDEX [indexName] ON mytable;

查看

SHOW INDEX FROM table\_name\G

使用ALTER命令

有四种方式来添加数据表的索引：

ALTER TABLE tbl\_name ADD PRIMARY KEY (column\_list): 该语句添加一个主键，这意味着索引值必须是唯一的，且不能为NULL。

ALTER TABLE tbl\_name ADD UNIQUE index\_name (column\_list): 这条语句创建索引的值必须是唯一的（除了NULL外，NULL可能会出现多次）。

ALTER TABLE tbl\_name ADD INDEX index\_name (column\_list): 添加普通索引，索引值可出现多次。

ALTER TABLE tbl\_name ADD FULLTEXT index\_name (column\_list):该语句指定了索引为 FULLTEXT ，用于全文索引。

* + 1. 哪些情况需要创建索引

主键自动建立唯一索引

频繁作为查询条件的字段应该创建索引

查询中与其它表关联的字段，外键关系建立索引

单键/组合索引的选择问题， 组合索引性价比更高

查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度

查询中统计或者分组字段

* + 1. 哪些情况不要创建索引

表记录太少

经常增删改的表或者字段

Why:提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。 因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件

Where条件里用不到的字段不创建索引

过滤性不好的不适合建索引

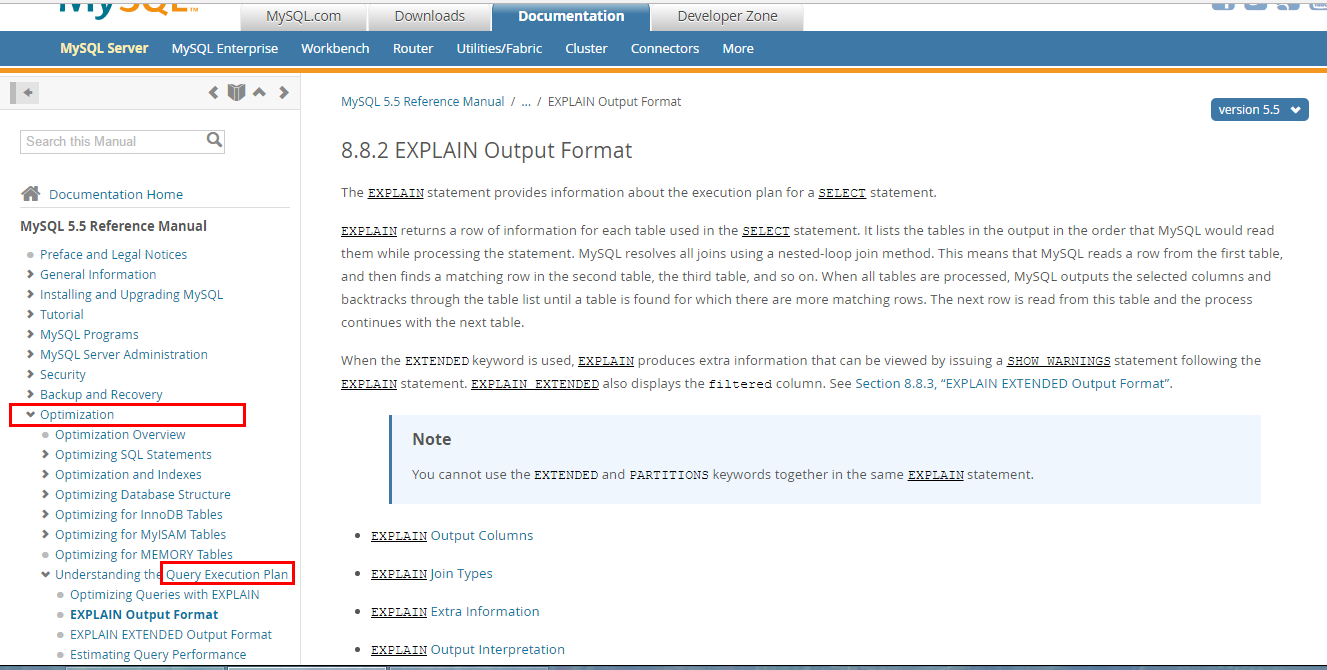
* 1. 性能分析
     1. Explain

是什么(查看执行计划)

使用EXPLAIN关键字可以模拟优化器执行SQL查询语句，从而知道MySQL是 如何处理你的SQL语句的。分析你的查询语句或是表结构的性能瓶颈

官网介绍

http://dev.mysql.com/doc/refman/5.5/en/explain-output.html



能干嘛

表的读取顺序

哪些索引可以使用

数据读取操作的操作类型

哪些索引被实际使用

表之间的引用

每张表有多少行被物理查询

怎么玩

Explain + SQL语句

执行计划包含的信息  


建表脚本

|  |
| --- |
| CREATE TABLE t1(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE t2(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE t3(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE t4(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  INSERT INTO t1(content) VALUES(CONCAT('t1\_',FLOOR(1+RAND()\*1000)));  INSERT INTO t2(content) VALUES(CONCAT('t2\_',FLOOR(1+RAND()\*1000)));  INSERT INTO t3(content) VALUES(CONCAT('t3\_',FLOOR(1+RAND()\*1000)));  INSERT INTO t4(content) VALUES(CONCAT('t4\_',FLOOR(1+RAND()\*1000))); |

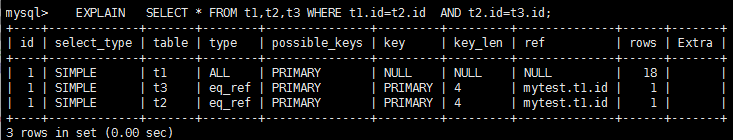
各字段解释

id

**select查询的序列号,包含一组数字，表示查询中执行select子句或操作表的顺序**

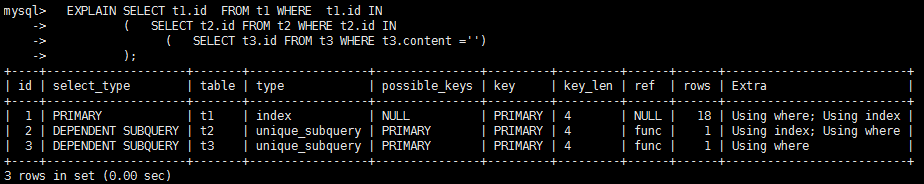
三种情况

id相同，执行顺序由上至下



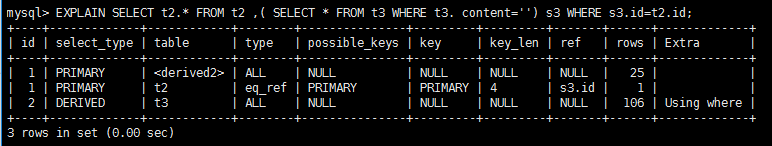
id相同，执行顺序由上至下

id不同，如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行



id不同，如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行

id相同不同，同时存在



id如果相同，可以认为是一组，从上往下顺序执行；

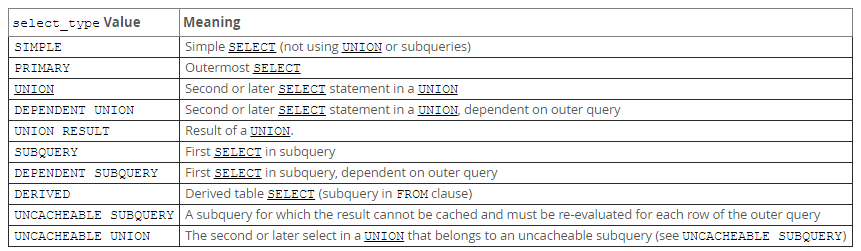
在所有组中，id值越大，优先级越高，越先执行

衍生 = DERIVED

关注点

id号每个号码，表示一趟独立的查询。一个sql 的查询趟数越少越好。

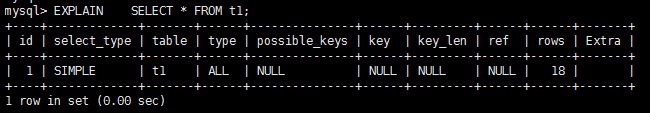
select\_type有哪些



查询的类型，主要是用于区别 普通查询、联合查询、子查询等的复杂查询

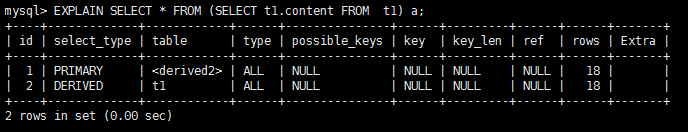
SIMPLE

简单的 select 查询,查询中不包含子查询或者UNION



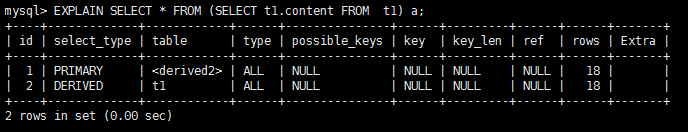
PRIMARY

查询中若包含任何复杂的子部分，最外层查询则被标记为Primary



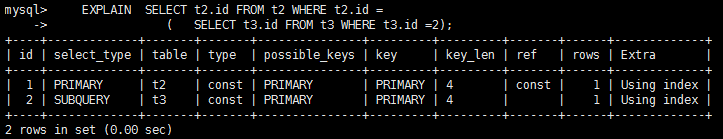
DERIVED

在FROM列表中包含的子查询被标记为DERIVED(衍生) MySQL会递归执行这些子查询, 把结果放在临时表里。



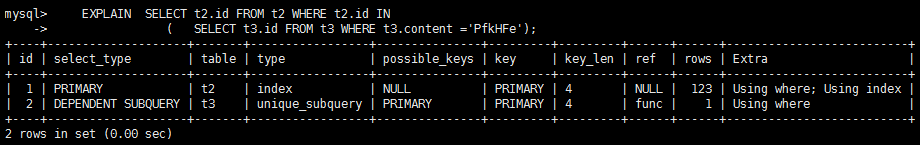
SUBQUERY

在SELECT或WHERE列表中包含了子查询

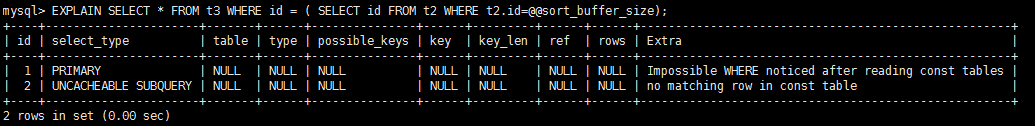


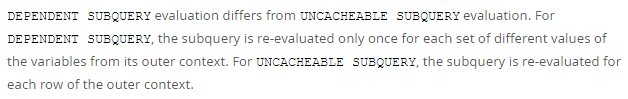
DEPENDENT SUBQUERY

在SELECT或WHERE列表中包含了子查询,子查询基于外层



UNCACHEABLE SUBQUREY



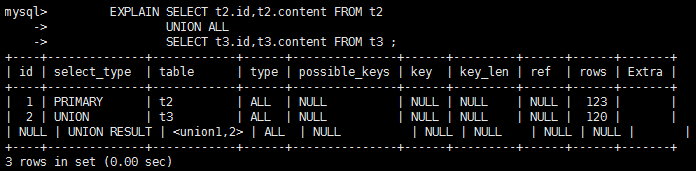


SHOW VARIABLES LIKE '%lower\_case\_table\_names%';

SELECT @@lower\_case\_table\_names FROM DUAL;

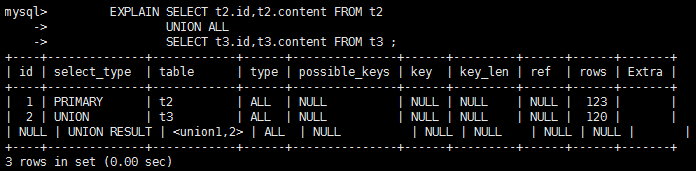
UNION

若第二个SELECT出现在UNION之后，则被标记为UNION； 若UNION包含在FROM子句的子查询中,外层SELECT将被标记为：DERIVED



UNION RESULT

从UNION表获取结果的SELECT



table

显示这一行的数据是关于哪张表的。

type



访问类型排列

type显示的是访问类型，是较为重要的一个指标，结果值从最好到最坏依次是：

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

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

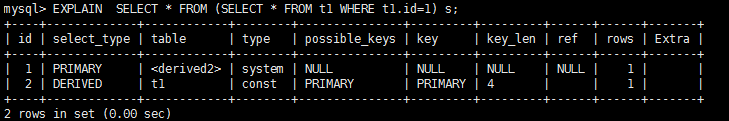
一般来说，得保证查询至少达到range级别，最好能达到ref。

显示查询使用了何种类型， 从最好到最差依次是： system>const>eq\_ref>ref>range>index>ALL

system

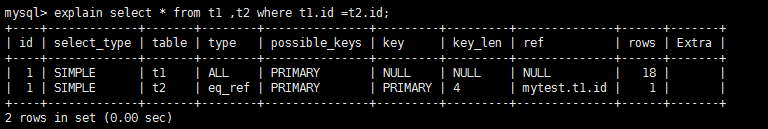
表只有一行记录（等于系统表），这是const类型的特列，平时不会出现，这个也可以忽略不计

const



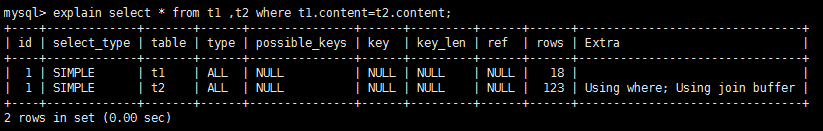
表示通过索引一次就找到了,const用于比较primary key或者unique索引。因为只匹配一行数据，所以很快 如将主键置于where列表中，MySQL就能将该查询转换为一个常量

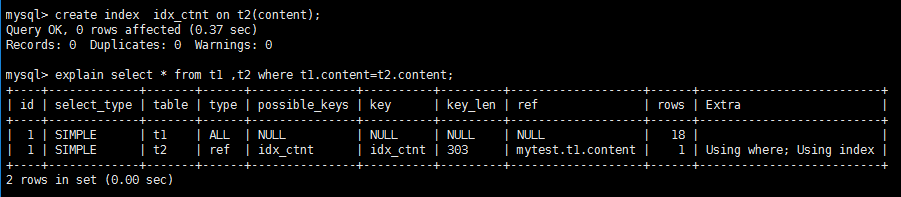
eq\_ref



唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描

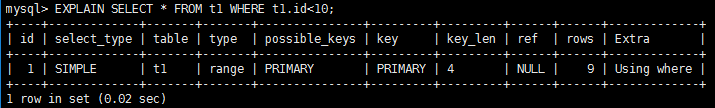
ref

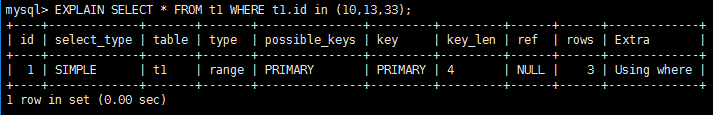




非唯一性索引扫描，返回匹配某个单独值的所有行. 本质上也是一种索引访问，它返回所有匹配某个单独值的行，然而， 它可能会找到多个符合条件的行，所以他应该属于查找和扫描的混合体

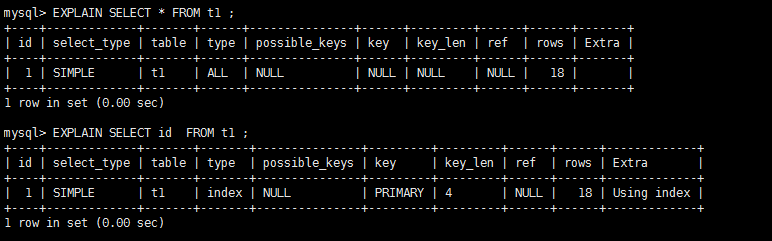
range





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 一般就是在你的where语句中出现了between、<、>、in等的查询 这种范围扫描索引扫描比全表扫描要好，因为它只需要开始于索引的某一点，而结束语另一点，不用扫描全部索引。

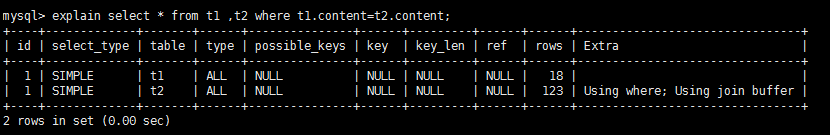
index



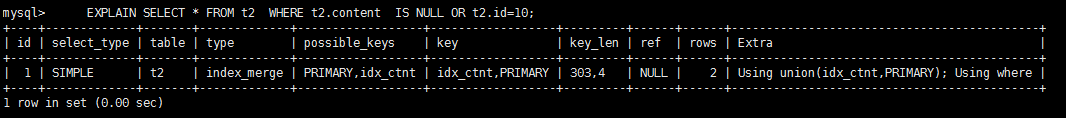
出现index是sql使用了索引但是没用通过索引进行过滤，一般是使用了覆盖索引或者是利用索引进行了排序分组

all

Full Table Scan，将遍历全表以找到匹配的行

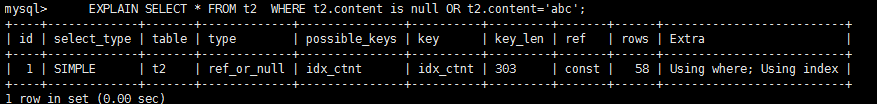


index\_merge



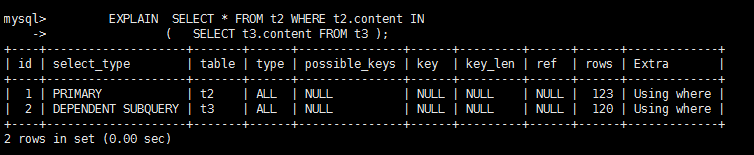
在查询过程中需要多个索引组合使用，通常出现在有 or 的关键字的sql中

ref\_or\_null

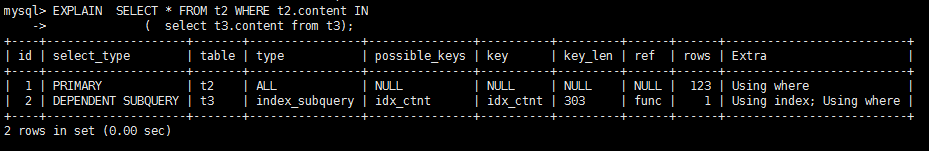


对于某个字段既需要关联条件，也需要null值得情况下。查询优化器会选择用ref\_or\_null连接查询。

index\_subquery

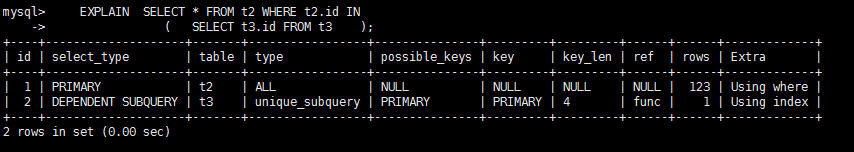


graphic



利用索引来关联子查询，不再全表扫描。

unique\_subquery



该联接类型类似于index\_subquery。 子查询中的唯一索引

备注：一般来说，得保证查询至少达到range级别，最好能达到ref。

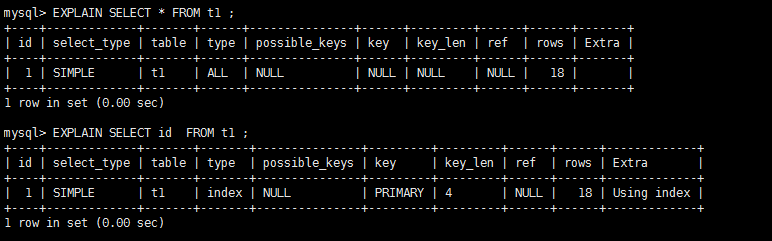
possible\_keys

显示可能应用在这张表中的索引，一个或多个。 查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询实际使用

key

实际使用的索引。如果为NULL，则没有使用索引

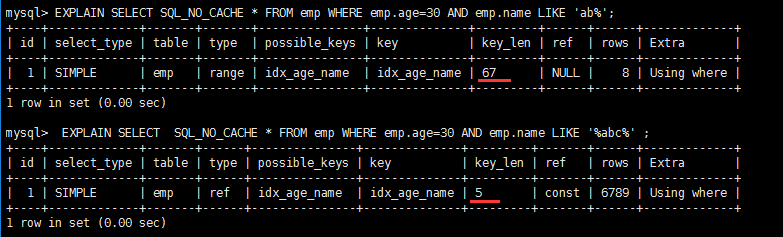
查询中若使用了覆盖索引，则该索引和查询的select字段重叠



key\_len

表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 AND emp.name LIKE 'ab%';



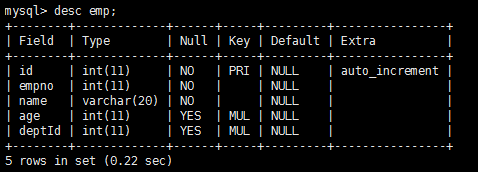
如何计算

1 、先看索引上字段的类型+长度比如 int=4 ; varchar(20) =20 ; char(20) =20

2 、如果是varchar或者char这种字符串字段，视字符集要乘不同的值，比如utf-8 要乘 3,GBK要乘2，

3 、varchar这种动态字符串要加2个字节

4、 允许为空的字段要加1个字节



第一组

key\_len=age的字节长度+name的字节长度=4+1 + ( 20\*3+2)=5+62=67

第二组

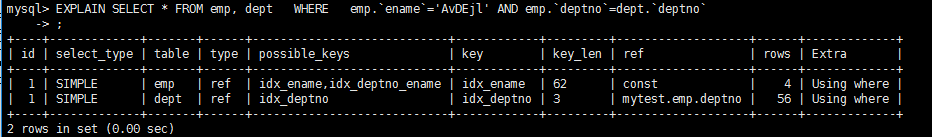
key\_len=age的字节长度=4+1=5



key\_len字段能够帮你检查是否充分的利用上了索引

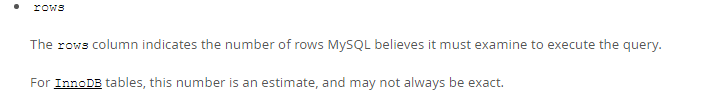
ref

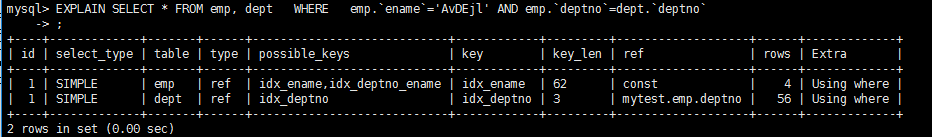
显示索引的哪一列被使用了，如果可能的话，是一个常数。哪些列或常量被用于查找索引列上的值



rows

rows列显示MySQL认为它执行查询时必须检查的行数。





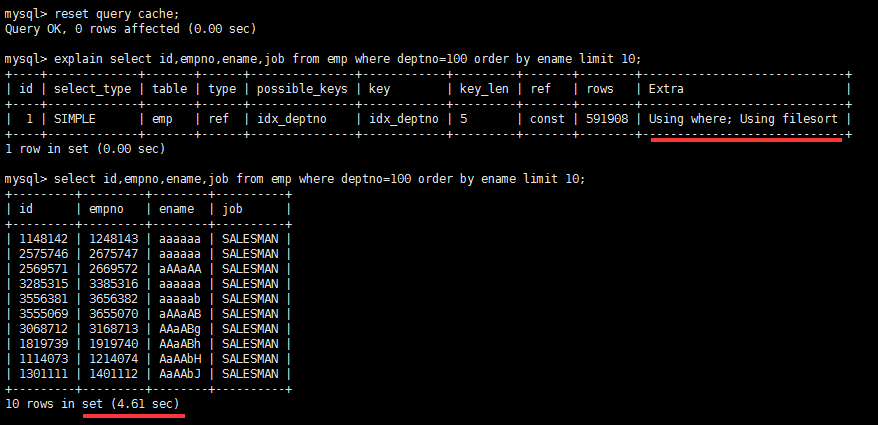
越少越好

Extra

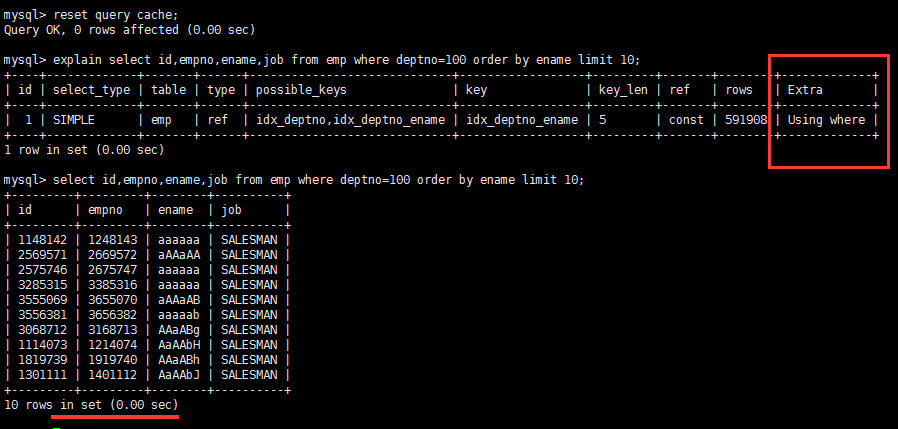
包含不适合在其他列中显示但十分重要的额外信息

Using filesort

出现filesort的情况：



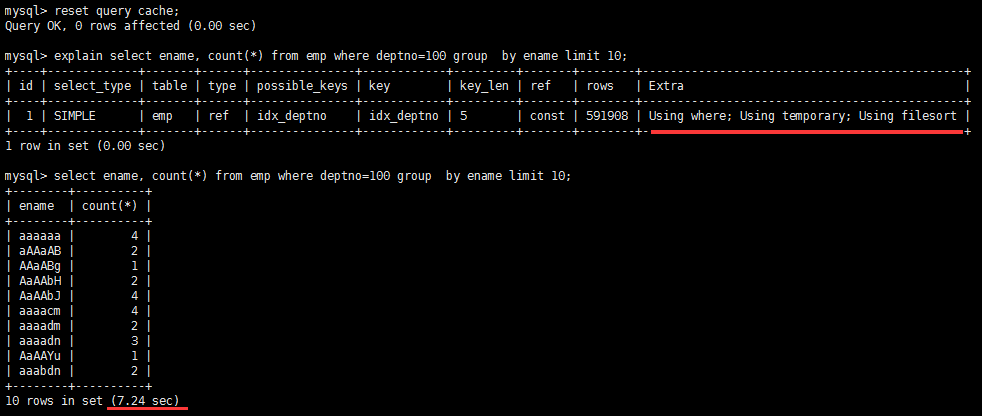
优化后，不再出现filesort的情况：



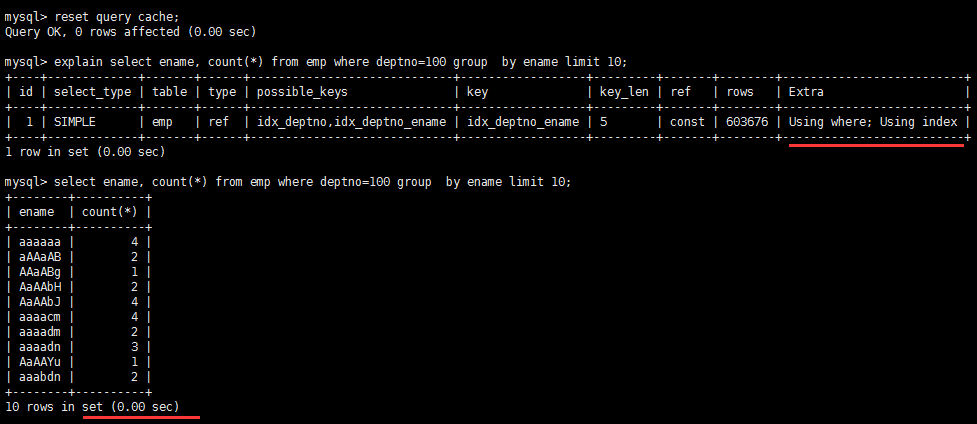
查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度

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进行读取。 MySQL中无法利用索引完成的排序操作称为“文件排序”

Using temporary

优化前存在 using temporary 和 using filesort 

优化前存在 using temporary 和 using filesort 不在，性能发生明显变化：



使了用临时表保存中间结果,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。

USING index

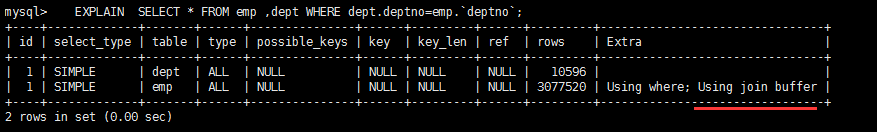
表示相应的select操作中使用了覆盖索引(Covering Index)，避免访问了表的数据行，效率不错！ 如果同时出现using where，表明索引被用来执行索引键值的查找; 如果没有同时出现using where，表明索引只是用来读取数据而非利用索引执行查找。

利用索引进行了排序或分组

Using where

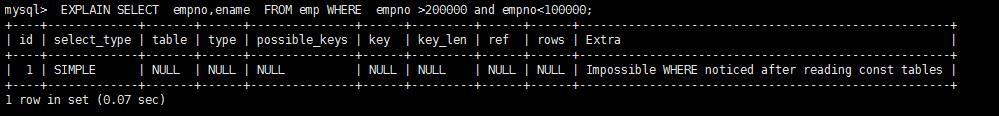
表明使用了where过滤

using join buffer



使用了连接缓存：

impossible where

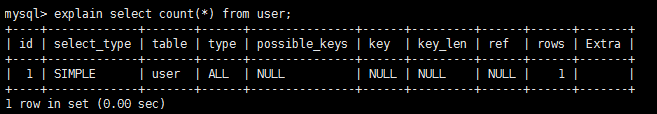


where子句的值总是false，不能用来获取任何元组

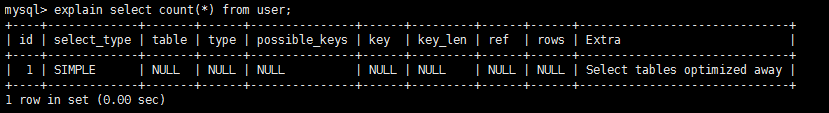
select tables optimized away

在没有GROUPBY子句的情况下，基于索引优化MIN/MAX操作或者 对于MyISAM存储引擎优化COUNT(\*)操作，不必等到执行阶段再进行计算， 查询执行计划生成的阶段即完成优化。

在innodb中：



在Myisam中：



* 1. 查询优化
     1. 批量数据脚本

往表里插入50W数据

建表

|  |
| --- |
| CREATE TABLE `dept` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `deptName` VARCHAR(30) DEFAULT NULL,  `address` VARCHAR(40) DEFAULT NULL,  ceo INT NULL ,  PRIMARY KEY (`id`)  ) ENGINE=INNODB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;    CREATE TABLE `emp` (  `id` INT(11) NOT NULL AUTO\_INCREMENT,  `empno` INT NOT NULL ,  `name` VARCHAR(20) DEFAULT NULL,  `age` INT(3) DEFAULT NULL,  `deptId` INT(11) DEFAULT NULL,  PRIMARY KEY (`id`)  #CONSTRAINT `fk\_dept\_id` FOREIGN KEY (`deptId`) REFERENCES `t\_dept` (`id`)  ) ENGINE=INNODB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8; |

设置参数log\_bin\_trust\_function\_creators

创建函数，假如报错：This function has none of DETERMINISTIC......

# 由于开启过慢查询日志，因为我们开启了 bin-log, 我们就必须为我们的function指定一个参数。

show variables like 'log\_bin\_trust\_function\_creators';

set global log\_bin\_trust\_function\_creators=1;

# 这样添加了参数以后，如果mysqld重启，上述参数又会消失，永久方法：

windows下my.ini[mysqld]加上log\_bin\_trust\_function\_creators=1

linux下 /etc/my.cnf下my.cnf[mysqld]加上log\_bin\_trust\_function\_creators=1

创建函数,保证每条数据都不同

随机产生字符串

|  |
| --- |
| DELIMITER $$  CREATE FUNCTION rand\_string(n INT) RETURNS VARCHAR(255)  BEGIN  DECLARE chars\_str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  DECLARE return\_str VARCHAR(255) DEFAULT '';  DECLARE i INT DEFAULT 0;  WHILE i < n DO  SET return\_str =CONCAT(return\_str,SUBSTRING(chars\_str,FLOOR(1+RAND()\*52),1));  SET i = i + 1;  END WHILE;  RETURN return\_str;  END $$  #假如要删除  #drop function rand\_string; |

随机产生部门编号

#用于随机产生多少到多少的编号

|  |
| --- |
| DELIMITER $$  CREATE FUNCTION rand\_num (from\_num INT ,to\_num INT) RETURNS INT(11)  BEGIN  DECLARE i INT DEFAULT 0;  SET i = FLOOR(from\_num +RAND()\*(to\_num -from\_num+1)) ;  RETURN i;  END$$  #假如要删除  #drop function rand\_num; |

创建存储过程

创建往emp表中插入数据的存储过程

|  |
| --- |
| DELIMITER $$  CREATE PROCEDURE insert\_emp( START INT , max\_num INT )  BEGIN  DECLARE i INT DEFAULT 0;  #set autocommit =0 把autocommit设置成0  SET autocommit = 0;  REPEAT  SET i = i + 1;  INSERT INTO emp (empno, NAME ,age ,deptid ) VALUES ((START+i) ,rand\_string(6) , rand\_num(30,50),rand\_num(1,10000));  UNTIL i = max\_num  END REPEAT;  COMMIT;  END$$  #删除  # DELIMITER ;  # drop PROCEDURE insert\_emp; |

创建往dept表中插入数据的存储过程

|  |
| --- |
| #执行存储过程，往dept表添加随机数据  DELIMITER $$  CREATE PROCEDURE `insert\_dept`( max\_num INT )  BEGIN  DECLARE i INT DEFAULT 0;  SET autocommit = 0;  REPEAT  SET i = i + 1;  INSERT INTO dept ( deptname,address,ceo ) VALUES (rand\_string(8),rand\_string(10),rand\_num(1,500000));  UNTIL i = max\_num  END REPEAT;  COMMIT;  END$$  #删除  # DELIMITER ;  # drop PROCEDURE insert\_dept; |

调用存储过程

dept

|  |
| --- |
| #执行存储过程，往dept表添加1万条数据  DELIMITER ;  CALL insert\_dept(10000); |

emp

|  |
| --- |
| #执行存储过程，往emp表添加50万条数据  DELIMITER ;  CALL insert\_emp(100000,500000); |

批量删除某个表上的所有索引

存储过程

|  |
| --- |
| DELIMITER $$  CREATE PROCEDURE `proc\_drop\_index`(dbname VARCHAR(200),tablename VARCHAR(200))  BEGIN  DECLARE done INT DEFAULT 0;  DECLARE ct INT DEFAULT 0;  DECLARE \_index VARCHAR(200) DEFAULT '';  DECLARE \_cur CURSOR FOR SELECT index\_name FROM information\_schema.STATISTICS WHERE table\_schema=dbname AND table\_name=tablename AND seq\_in\_index=1 AND index\_name <>'PRIMARY' ;  DECLARE CONTINUE HANDLER FOR NOT FOUND set done=2 ;  OPEN \_cur;  FETCH \_cur INTO \_index;  WHILE \_index<>'' DO  SET @str = CONCAT("drop index ",\_index," on ",tablename );  PREPARE sql\_str FROM @str ;  EXECUTE sql\_str;  DEALLOCATE PREPARE sql\_str;  SET \_index='';  FETCH \_cur INTO \_index;  END WHILE;  CLOSE \_cur;  END$$ |

执行存储过程

|  |
| --- |
| CALL proc\_drop\_index("dbname","tablename"); |

* + 1. 单表使用索引及常见索引失效

案例(索引失效)

全值匹配我最爱

系统中经常出现的sql语句如下：

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30;

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptid=4;

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptid=4 AND emp.name = 'abcd';

索引应该如何建立？

建立索引

CREATE INDEX idx\_age\_deptid\_name ON emp(age,deptid,NAME)

建立索引前

graphic

索引后

graphic

最佳左前缀法则

如果系统经常出现的sql如下：

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 AND emp.name = 'abcd'

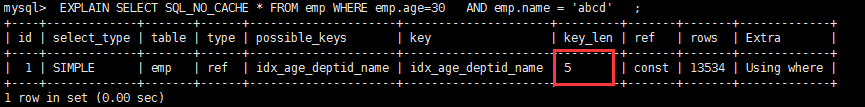
或者

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.deptid=1 AND emp.name = 'abcd'

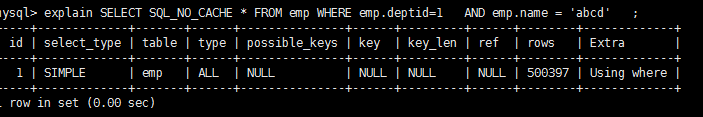
那原来的idx\_age\_deptid\_name 还能否正常使用？

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的列。

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 AND emp.name = 'abcd'



虽然可以正常使用，但是只有部分被使用到了。



完全没有使用上索引。

结论：过滤条件要使用索引必须按照索引建立时的顺序，依次满足，一旦跳过某个字段，索引后面的字段都无法被使用。

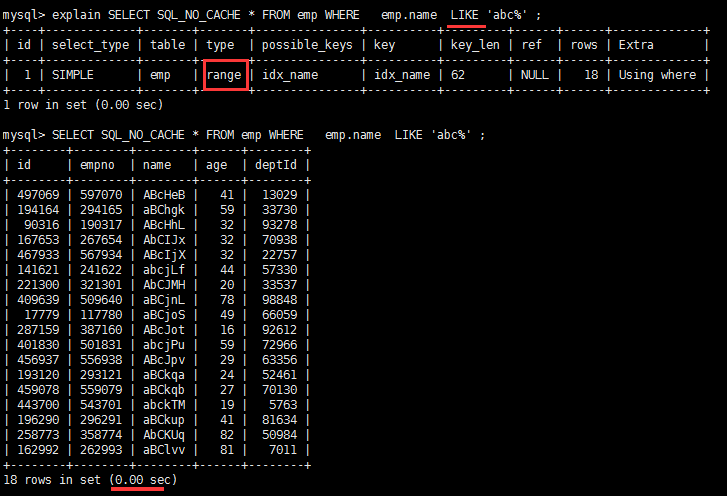
不在索引列上做任何操作（计算、函数、(自动or手动)类型转换），会导致索引失效而转向全表扫描

这两条sql哪种写法更好

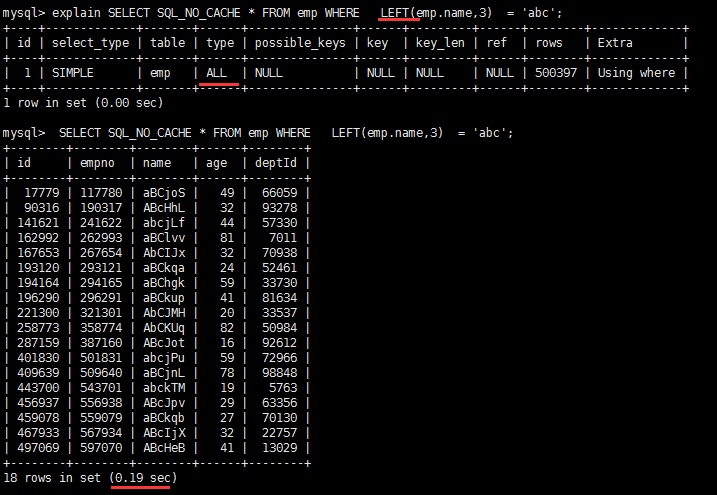
EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.name LIKE 'abc%'

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE LEFT(emp.name,3) = 'abc'

第一种



第二种

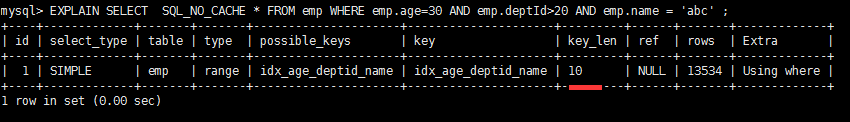


存储引擎不能使用索引中范围条件右边的列

如果系统经常出现的sql如下：

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 AND emp.deptId>20 AND emp.name = 'abc' ;

那么索引 idx\_age\_deptid\_name这个索引还能正常使用么？

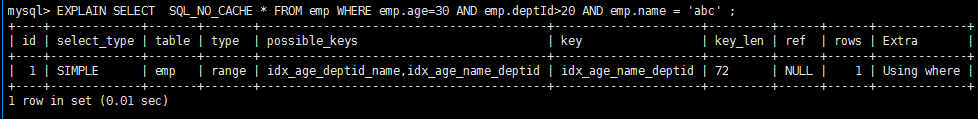


如果这种sql 出现较多

应该建立：

create index idx\_age\_name\_deptid on emp(age,name,deptid)

效果

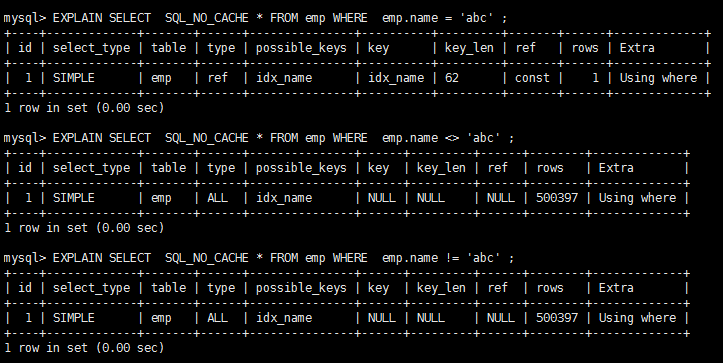


# drop index idx\_age\_name\_deptid on emp

mysql 在使用不等于(!= 或者<>)的时候无法使用索引会导致全表扫描

CREATE INDEX idx\_name ON emp(NAME)

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.name <> 'abc'



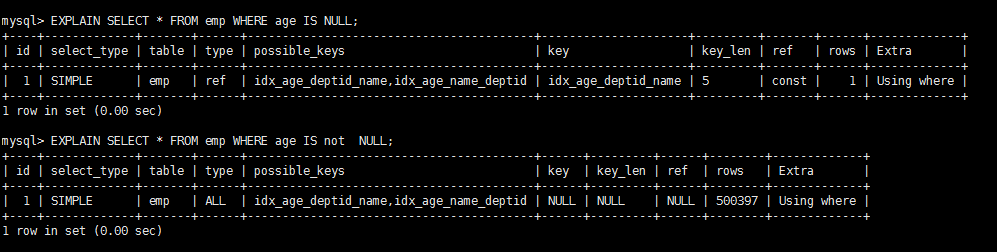
is not null 也无法使用索引,但是is null是可以使用索引的

UPDATE emp SET age =NULL WHERE id=123456;

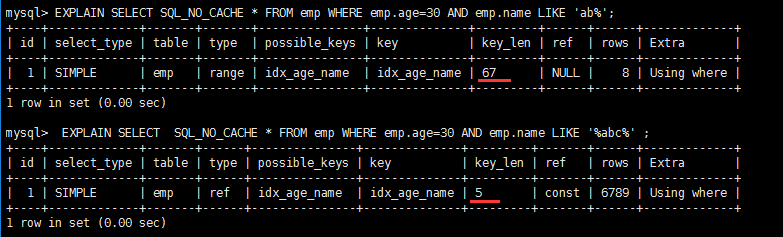
下列哪个sql语句可以用到索引

EXPLAIN SELECT \* FROM emp WHERE age IS NULL

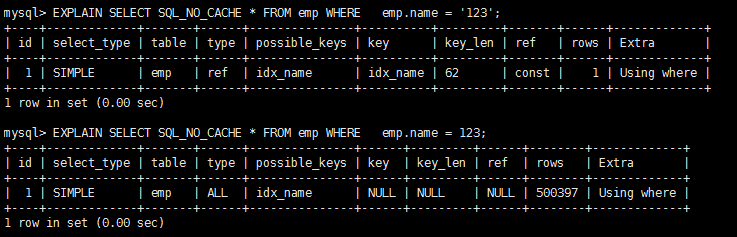
EXPLAIN SELECT \* FROM emp WHERE age IS NOT NULL



like以通配符开头('%abc...')mysql索引失效会变成全表扫描的操作



字符串不加单引号索引失效



小总结

假设index(a,b,c)

|  |  |
| --- | --- |
| **Where语句** | **索引是否被使用** |
| where a = 3 | Y,使用到a |
| where a = 3 and b = 5 | Y,使用到a，b |
| where a = 3 and b = 5 and c = 4 | Y,使用到a,b,c |
| where b = 3 或者 where b = 3 and c = 4 或者 where c = 4 | N |
| where a = 3 and c = 5 | 使用到a， 但是c不可以，b中间断了 |
| where a = 3 and b > 4 and c = 5 | 使用到a和b， c不能用在范围之后，b断了 |
| where a is null and b is not null | is null 支持索引 但是is not null 不支持,所以 a 可以使用索引,但是 b不可以使用 |
| where a <> 3 | 不能使用索引 |
| where abs(a) =3 | 不能使用 索引 |
| where a = 3 and b like 'kk%' and c = 4 | Y,使用到a,b,c |
| where a = 3 and b like '%kk' and c = 4 | Y,只用到a |
| where a = 3 and b like '%kk%' and c = 4 | Y,只用到a |
| where a = 3 and b like 'k%kk%' and c = 4 | Y,使用到a,b,c |

一般性建议

对于单键索引，尽量选择针对当前query过滤性更好的索引

在选择组合索引的时候，当前Query中过滤性最好的字段在索引字段顺序中，位置越靠前越好。

在选择组合索引的时候，尽量选择可以能够包含当前query中的where字句中更多字段的索引

在选择组合索引的时候，如果某个字段可能出现范围查询时，尽量把这个字段放在索引次序的最后面

书写sql语句时，尽量避免造成索引失效的情况

* + 1. 关联查询优化

建表SQL

|  |
| --- |
| CREATE TABLE IF NOT EXISTS `class` (  `id` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,  `card` INT(10) UNSIGNED NOT NULL,  PRIMARY KEY (`id`)  );  CREATE TABLE IF NOT EXISTS `book` (  `bookid` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,  `card` INT(10) UNSIGNED NOT NULL,  PRIMARY KEY (`bookid`)  );    INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));    INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));  INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20))); |

案例

# 下面开始explain分析

EXPLAIN SELECT \* FROM book LEFT JOIN class ON class.card = book.card;

#结论：type 有All

# 添加索引优化

ALTER TABLE `book` ADD INDEX Y ( `card`);

换成inner join

delete from class where id<5;

# 第2次explain

EXPLAIN SELECT \* FROM class LEFT JOIN book ON class.card = book.card;

#可以看到第二行的 type 变为了 ref,rows 也变成了优化比较明显。

#这是由左连接特性决定的。LEFT JOIN 条件用于确定如何从右表搜索行,左边一定都有,

#所以右边是我们的关键点,一定需要建立索引。

# 删除旧索引 + 新建 + 第3次explain

DROP INDEX Y ON book;

ALTER TABLE class ADD INDEX X (card);

EXPLAIN SELECT \* FROM class LEFT JOIN book ON class.card = book.card;

建议

1、保证被驱动表的join字段已经被索引

2、left join 时，选择小表作为驱动表，大表作为被驱动表。

3、inner join 时，mysql会自己帮你把小结果集的表选为驱动表。

4、子查询尽量不要放在被驱动表，有可能使用不到索引。

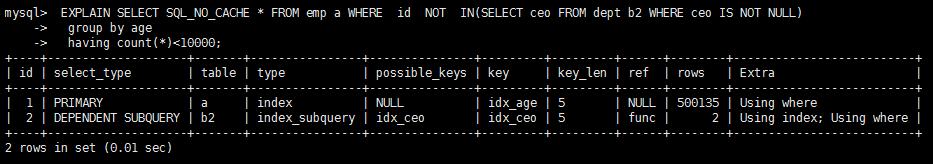
5、能够直接多表关联的尽量直接关联，不用子查询。

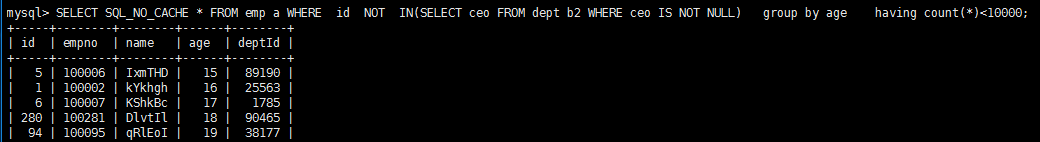
* + 1. 子查询优化

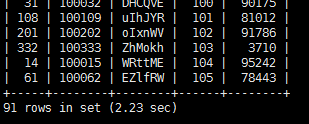
尽量不要使用not in 或者 not exists

取所有不为掌门人的员工，按年龄分组 ，每个年龄段多少人

|  |
| --- |
| SELECT SQL\_NO\_CACHE age,count(\*) FROM emp a WHERE id NOT IN(SELECT ceo FROM dept b2 WHERE ceo IS NOT NULL)  group by age  having count(\*)<10000 |

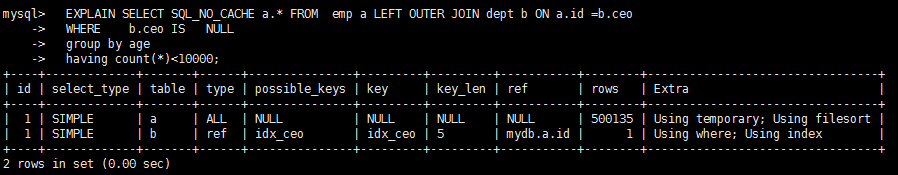


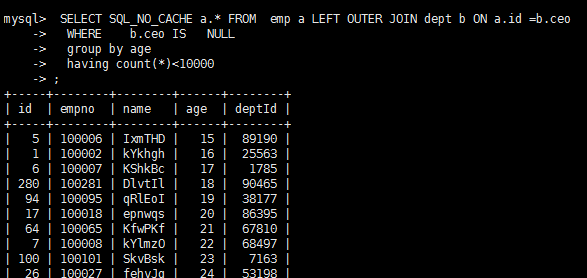


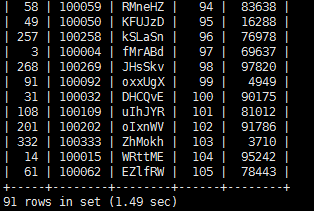


用left outer join on xxx is null 替代

|  |
| --- |
| EXPLAIN SELECT SQL\_NO\_CACHE age,count(\*) FROM emp a LEFT OUTER JOIN dept b ON a.id =b.ceo  WHERE b.ceo IS NULL  group by age  having count(\*)<10000 |







* + 1. 排序分组优化

case

create index idx\_age\_deptid\_name on emp (age,deptid,name)

以下 是否能使用到索引，能否去掉using filesort

1、explain select SQL\_NO\_CACHE \* from emp order by age,deptid;

2、 explain select SQL\_NO\_CACHE \* from emp order by age,deptid limit 10;

#无过滤 不索引

3、explain select \* from emp where age=45 order by deptid;

4、explain select \* from emp where age=45 order by deptid,name;

5、explain select \* from emp where age=45 order by deptid,empno;

6、explain select \* from emp where age=45 order by name,deptid;

7、 explain select \* from emp where deptid=45 order by age;

#顺序错，必排序

8、 explain select \* from emp where age=45 order by deptid desc, name desc ;

9、 explain select \* from emp where age=45 order by deptid asc, name desc ;

#方向反 必排序

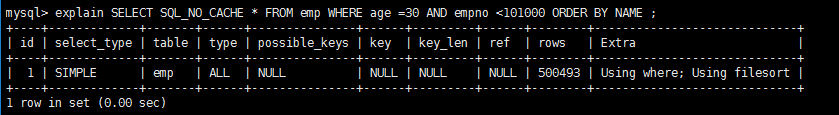
ORDER BY子句，尽量使用Index方式排序,避免使用FileSort方式排序

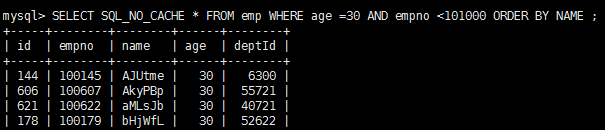
索引的选择

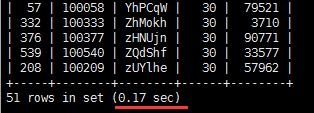
执行案例前先清除emp上的索引，只留主键索引的选择

#查询 年龄为30岁的，且员工编号小于101000的用户，按用户名称排序

SELECT SQL\_NO\_CACHE \* FROM emp WHERE age =30 AND empno <101000 ORDER BY NAME ;







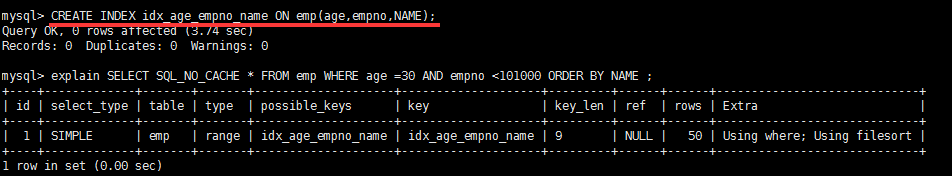
#结论：很显然,type 是 ALL,即最坏的情况。Extra 里还出现了 Using filesort,也是最坏的情况。优化是必须的。

**#开始优化：**

**思路： 尽量让where的过滤条件和排序使用上索引**

**但是一共两个字段(deptno,empno)上有过滤条件，一个字段(ename)有索引**

**1、我们建一个三个字段的组合索引可否？**



CREATE INDEX idx\_age\_empno\_name ON emp(age,empno,NAME);

我们发现using filesort 依然存在，所以name 并没有用到索引。

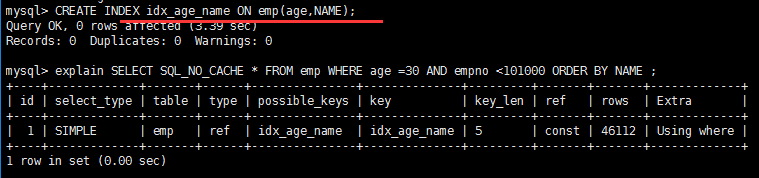
原因是因为empno是一个范围过滤，所以索引后面的字段不会再使用索引了。

所以我们建一个3值索引是没有意义的

那么我们先删掉这个索引，DROP INDEX idx\_age\_empno\_name ON emp

为了去掉filesort我们可以把索引建成

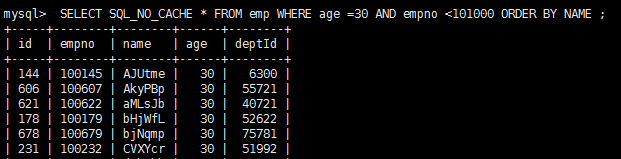
CREATE INDEX idx\_age\_name ON emp(age,NAME);

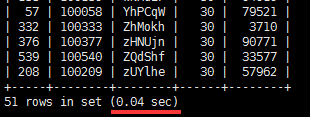


也就是说empno 和name这个两个字段我只能二选其一。

这样我们优化掉了 using filesort。

执行一下sql





速度果然提高了4倍。

但是

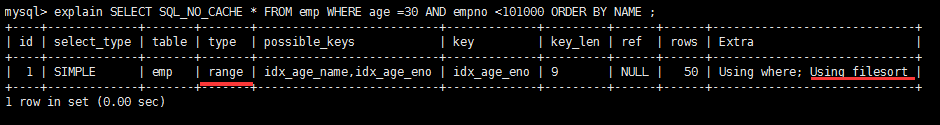
如果我们选择那个范围过滤，而放弃排序上的索引呢

建立

DROP INDEX idx\_age\_name ON emp

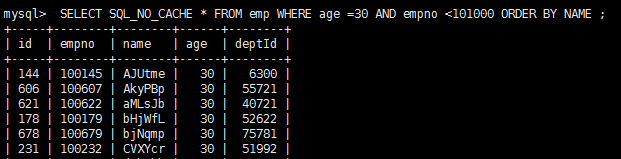
create index idx\_age\_eno on emp(age,empno);

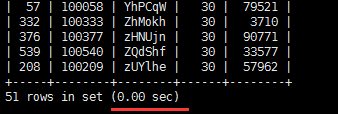
graphic



果然出现了filesort，而且type还是range光看字面其实并不美好。

我们来执行以下sql





结果竟然有 filesort的 sql 运行速度，超过了已经优化掉 filesort的 sql ，而且快了好多倍。何故？

原因是所有的排序都是在条件过滤之后才执行的，所以如果条件过滤了大部分数据的话，几百几千条数据进行排序其实并不是很消耗性能，即使索引优化了排序但实际提升性能很有限。 相对的 empno<101000 这个条件如果没有用到索引的话，要对几万条的数据进行扫描，这是非常消耗性能的，所以索引放在这个字段上性价比最高，是最优选择。

**结论**： **当范围条件和group by 或者 order by 的字段出现二选一时 ，优先观察条件字段的过滤数量，如果过滤的数据足够多，而需要排序的数据并不多时，优先把索引放在范围字段上。反之，亦然。**

如果不在索引列上，filesort有两种算法： mysql就要启动双路排序和单路排序

双路排序

MySQL 4.1之前是使用双路排序,字面意思就是两次扫描磁盘，最终得到数据， 读取行指针和orderby列，对他们进行排序，然后扫描已经排序好的列表，按照列表中的值重新从列表中读取对应的数据输出

从磁盘取排序字段，在buffer进行排序，再从磁盘取其他字段。

取一批数据，要对磁盘进行了两次扫描，众所周知，I\O是很耗时的，所以在mysql4.1之后，出现了第二种改进的算法，就是单路排序。

单路排序

从磁盘读取查询需要的所有列，按照order by列在buffer对它们进行排序，然后扫描排序后的列表进行输出， 它的效率更快一些，避免了第二次读取数据。并且把随机IO变成了顺序IO,但是它会使用更多的空间， 因为它把每一行都保存在内存中了。

结论及引申出的问题

由于单路是后出的，总体而言好过双路

但是用单路有问题

在sort\_buffer中，方法B比方法A要多占用很多空间，因为方法B是把所有字段都取出, 所以有可能取出的数据的总大小超出了sort\_buffer的容量，导致每次只能取sort\_buffer容量大小的数据，进行排序（创建tmp文件，多路合并），排完再取取sort\_buffer容量大小，再排……从而多次I/O。

本来想省一次I/O操作，反而导致了大量的I/O操作，反而得不偿失。

优化策略

增大sort\_buffer\_size参数的设置

增大max\_length\_for\_sort\_data参数的设置

减少select 后面的查询的字段。

Why

提高Order By的速度

1. Order by时select \* 是一个大忌只Query需要的字段， 这点非常重要。在这里的影响是：

1.1 当Query的字段大小总和小于max\_length\_for\_sort\_data 而且排序字段不是 TEXT|BLOB 类型时，会用改进后的算法——单路排序， 否则用老算法——多路排序。

1.2 两种算法的数据都有可能超出sort\_buffer的容量，超出之后，会创建tmp文件进行合并排序，导致多次I/O，但是用单路排序算法的风险会更大一些,所以要提高sort\_buffer\_size。

2. 尝试提高 sort\_buffer\_size

不管用哪种算法，提高这个参数都会提高效率，当然，要根据系统的能力去提高，因为这个参数是针对每个进程的 **1M-8M之间调整**

3. 尝试提高 max\_length\_for\_sort\_data

提高这个参数， 会增加用改进算法的概率。但是如果设的太高，数据总容量超出sort\_buffer\_size的概率就增大，明显症状是高的磁盘I/O活动和低的处理器使用率. **1024-8192之间调整**

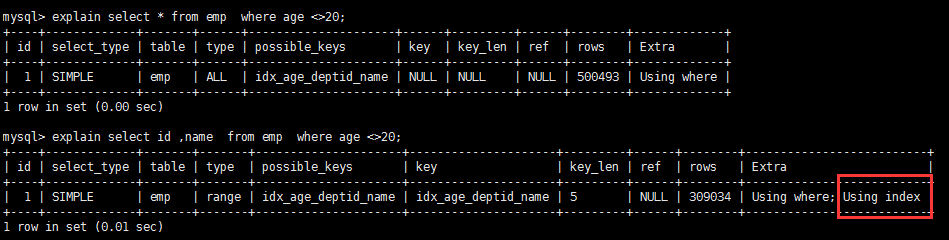
GROUP BY关键字优化

group by 使用索引的原则几乎跟order by一致 ，唯一区别是groupby 即使没有过滤条件用到索引，也可以直接使用索引。

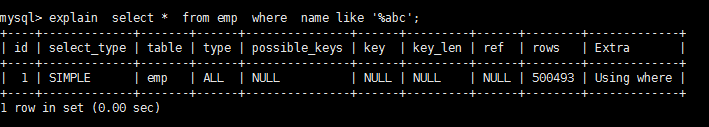
* + 1. 最后使用索引的手段：覆盖索引

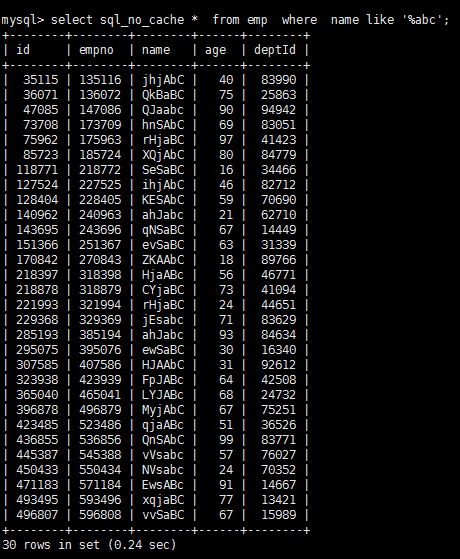
什么是覆盖索引？

简单说就是，select 到 from 之间查询的列 <=使用的索引列+主键

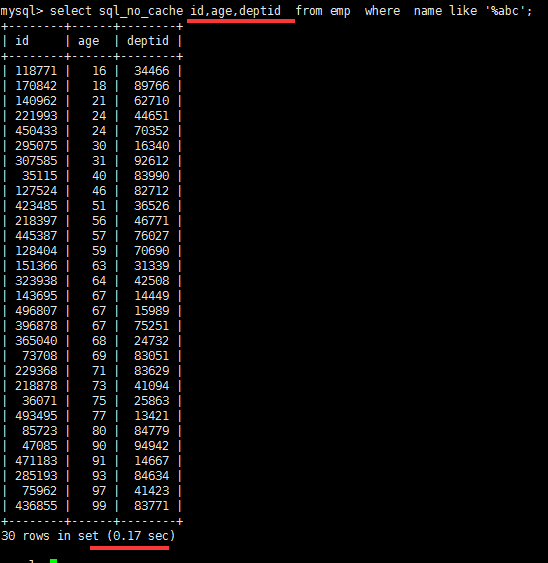


explain select \* from emp where name like '%abc';





使用覆盖索引后



1. 查询截取分析
   1. 慢查询日志
      1. 是什么

* MySQL的慢查询日志是MySQL提供的一种日志记录，它用来记录在MySQL中响应时间超过阈值的语句，具体指运行时间超过long\_query\_time值的SQL，则会被记录到慢查询日志中。
* 具体指运行时间超过long\_query\_time值的SQL，则会被记录到慢查询日志中。long\_query\_time的默认值为10，意思是运行10秒以上的语句。
* 由他来查看哪些SQL超出了我们的最大忍耐时间值，比如一条sql执行超过5秒钟，我们就算慢SQL，希望能收集超过5秒的sql，结合之前explain进行全面分析。
  + 1. 怎么玩

说明

默认情况下，MySQL数据库没有开启慢查询日志，需要我们手动来设置这个参数。

当然，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件

查看是否开启及如何开启

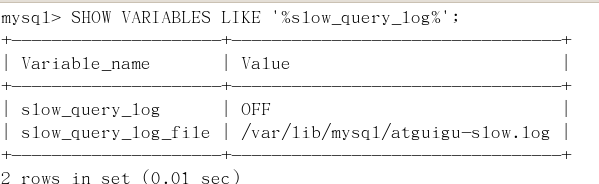
默认

SHOW VARIABLES LIKE '%slow\_query\_log%';

默认情况下slow\_query\_log的值为OFF，表示慢查询日志是禁用的，

可以通过设置slow\_query\_log的值来开启

SHOW VARIABLES LIKE '%slow\_query\_log%';

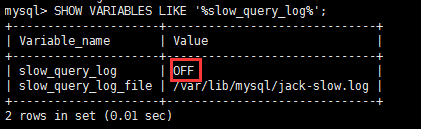


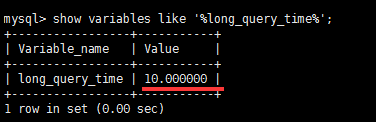
开启

set global slow\_query\_log=1;

使用set global slow\_query\_log=1;开启了慢查询日志只对当前数据库生效，

如果MySQL重启后则会失效。

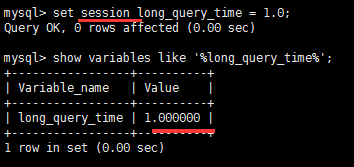




全局变量设置，对当前连接不影响

graphic

对当前连接立刻生效



如果要永久生效，就必须修改配置文件my.cnf（其它系统变量也是如此）

修改my.cnf文件，[mysqld]下增加或修改参数

slow\_query\_log 和slow\_query\_log\_file后，然后重启MySQL服务器。也即将如下两行配置进my.cnf文件

slow\_query\_log =1

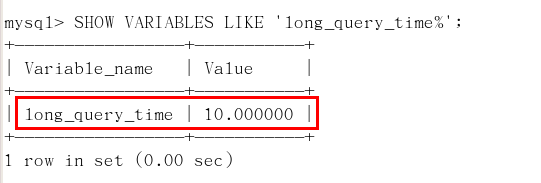
slow\_query\_log\_file=/var/lib/mysql/atguigu-slow.log

关于慢查询的参数slow\_query\_log\_file ，它指定慢查询日志文件的存放路径，系统默认会给一个缺省的文件host\_name-slow.log（如果没有指定参数slow\_query\_log\_file的话）

那么开启了慢查询日志后，什么样的SQL才会记录到慢查询日志里面呢？

这个是由参数long\_query\_time控制，默认情况下long\_query\_time的值为10秒，

命令：SHOW VARIABLES LIKE 'long\_query\_time%';



可以使用命令修改，也可以在my.cnf参数里面修改。

假如运行时间正好等于long\_query\_time的情况，并不会被记录下来。也就是说，

在mysql源码里是判断大于long\_query\_time，而非大于等于。

Case

查看当前多少秒算慢

SHOW VARIABLES LIKE 'long\_query\_time%';

设置慢的阙值时间

使用命令

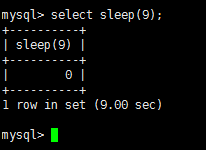
set long\_query\_time=1

修改为阙值到1秒钟的就是慢sql

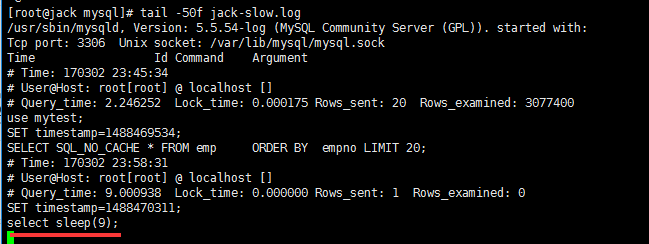
set long\_query\_time=1

记录慢SQL并后续分析

实验一条慢sql

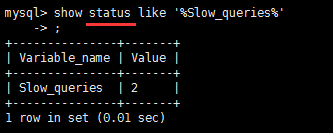


跟踪日志信息



查询当前系统中有多少条慢查询记录

show global status like '%Slow\_queries%';



配置版

my.cnf

【mysqld】下配置：

slow\_query\_log=1;

slow\_query\_log\_file=/var/lib/mysql/atguigu-slow.log

long\_query\_time=3;

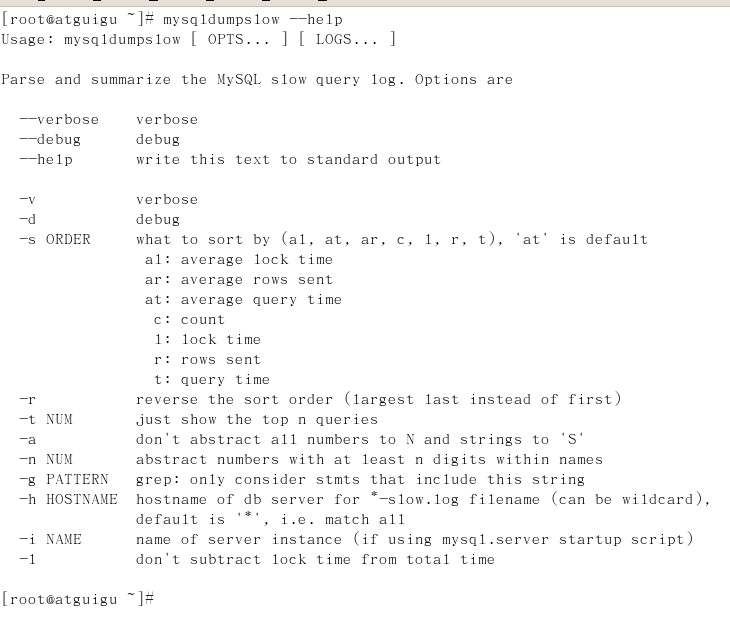
log\_output=FILE

* + 1. 日志分析工具mysqldumpslow

在生产环境中，如果要手工分析日志，查找、分析SQL，显然是个体力活，MySQL提供了日志分析工具mysqldumpslow。

查看mysqldumpslow的帮助信息

mysqldumpslow --help



-s: 是表示按照何种方式排序；

c: 访问次数

l: 锁定时间

r: 返回记录

t: 查询时间

al:平均锁定时间

ar:平均返回记录数

at:平均查询时间

-t:即为返回前面多少条的数据；

-g:后边搭配一个正则匹配模式，大小写不敏感的；

工作常用参考

得到返回记录集最多的10个SQL

mysqldumpslow -s r -t 10 /var/lib/mysql/atguigu-slow.log

得到访问次数最多的10个SQL

mysqldumpslow -s c -t 10 /var/lib/mysql/atguigu-slow.log

得到按照时间排序的前10条里面含有左连接的查询语句

mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/atguigu-slow.log

另外建议在使用这些命令时结合 | 和more 使用 ，否则有可能出现爆屏情况

mysqldumpslow -s r -t 10 /var/lib/mysql/atguigu-slow.log | more

* 1. SHOW PROCESSLIST
     1. 能干什么：查询所有用户正在干什么

如果出现不顺眼的

kill [id]

1. 工具和技巧拾遗
   1. 视图 VIEW
      1. 是什么

将一段查询sql封装为一个虚拟的表。

这个虚拟表只保存了sql逻辑，不会保存任何查询结果。

* + 1. 作用

1、封装复杂sql语句，提高复用性

2、逻辑放在数据库上面，更新不需要发布程序，面对频繁的需求变更更灵活

* + 1. 适用场景

很多地方可以共用的一组查询结果

报表

* + 1. 语法

CREATE VIEW view\_name

AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

使用：

select \* from view\_name

创建

CREATE VIEW view\_name

AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

使用

查询

select \* from view\_name

更新

CREATE OR REPLACE VIEW view\_name

AS

SELECT column\_name(s)

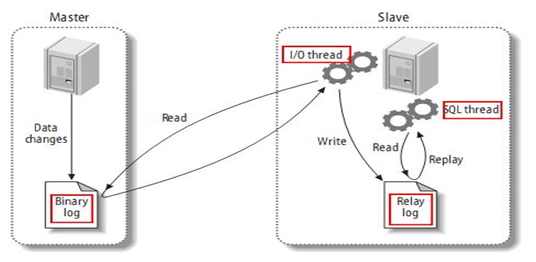
FROM table\_name

WHERE condition

* + 1. 注意事项

mysql的视图中不允许有from 后面的子查询，但oracle可以

1. 主从复制
   1. 复制的基本原理
      1. slave会从master读取binlog来进行数据同步
      2. 三步骤+原理图



MySQL复制过程分成三步：

1 master将改变记录到二进制日志（binary log）。这些记录过程叫做二进制日志事件，binary log events；

2 slave将master的binary log events拷贝到它的中继日志（relay log）；

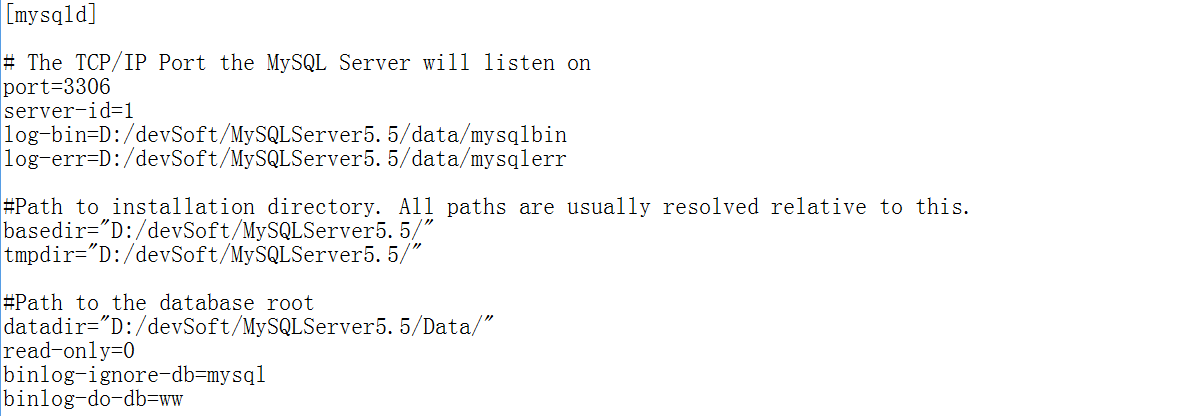
3 slave重做中继日志中的事件，将改变应用到自己的数据库中。 MySQL复制是异步的且串行化的

master将改变记录到二进制日志（binary log）。这些记录过程叫做二进制日志事件，binary log events

slave将master的binary log events拷贝到它的中继日志（relay log）

slave重做中继日志中的事件，将改变应用到自己的数据库中。 MySQL复制是异步的且串行化的

* 1. 复制的基本原则
     1. 每个slave只有一个master
     2. 每个slave只能有一个唯一的服务器ID
     3. 每个master可以有多个salve
  2. 复制的最大问题
     1. 延时
  3. 一主一从常见配置
     1. mysql版本一致且后台以服务运行
     2. 主从都配置在[mysqld]结点下，都是小写
     3. 主机修改my.ini配置文件



主服务器唯一ID

server-id=1

启用二进制日志

log-bin=自己本地的路径/data/mysqlbin

log-bin=D:/devSoft/MySQLServer5.5/data/mysqlbin

设置不要复制的数据库

binlog-ignore-db=mysql

设置需要复制的数据库

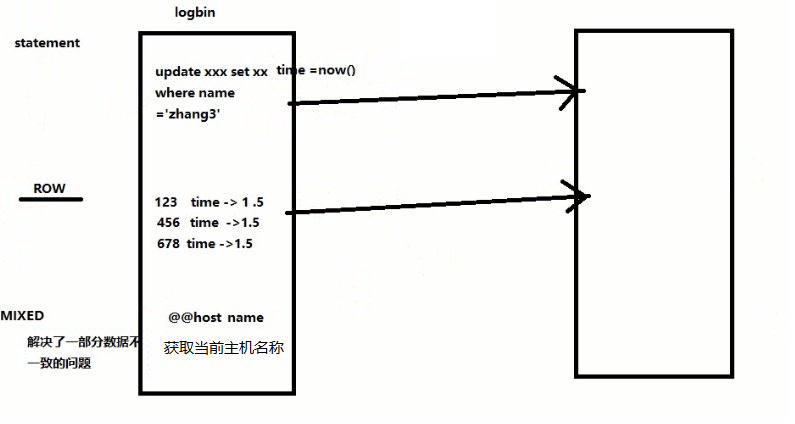
binlog-do-db=需要复制的主数据库名字（设置一个之前没有的数据库）

设置logbin格式

binlog\_format=STATEMENT（默认）

binlog\_format=STATEMENT（默认）

binlog\_format=ROW



* + 1. mysql主从复制起始时，从机不继承主机数据
    2. 从机配置文件修改my.cnf的[mysqld]栏位下

server-id = 2

注意my.cnf 中有server-id = 1

relay-log=mysql-relay

* + 1. 因修改过配置文件，请主机+从机都重启后台mysql服务
    2. 主机从机都关闭防火墙

windows手动关闭

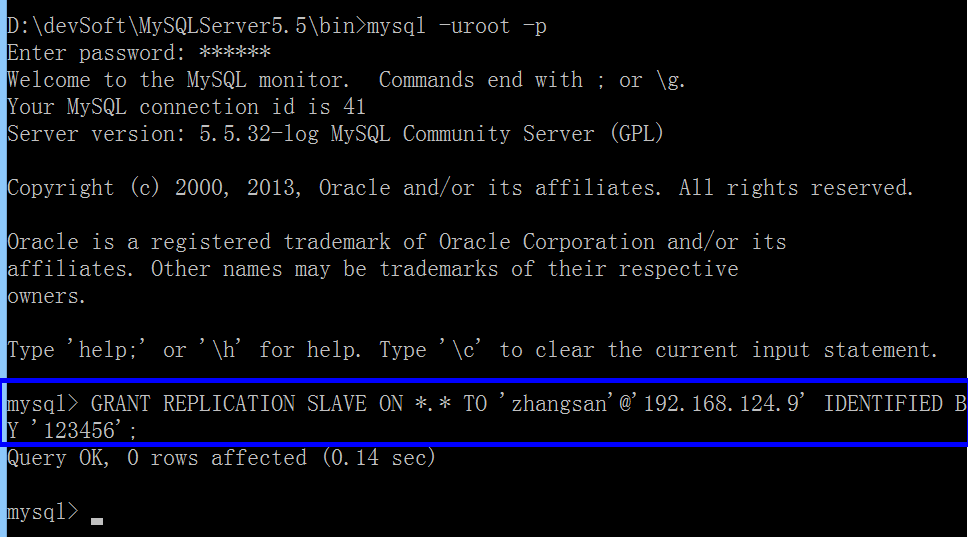
安全工具关上：腾讯管家

360不影响

关闭虚拟机linux防火墙 service iptables stop

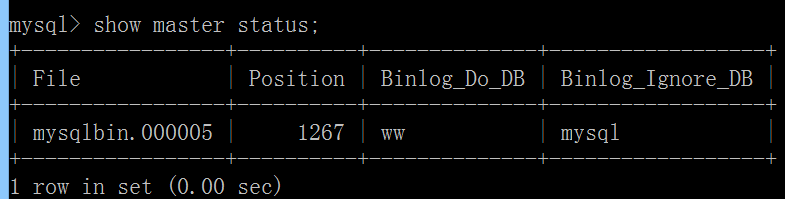
* + 1. 在Windows主机上建立帐户并授权slave

GRANT REPLICATION SLAVE ON \*.\* TO 'slave'@'从机器数据库IP' IDENTIFIED BY '123456';



查询master的状态

show master status;



记录下File和Position的值

执行完此步骤后不要再操作主服务器MYSQL，防止主服务器状态值变化

* + 1. 在Linux从机上配置需要复制的主机

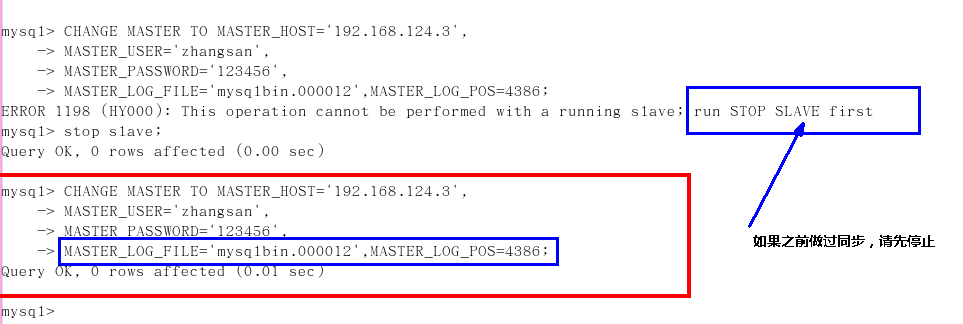
CHANGE MASTER TO MASTER\_HOST='主机IP',MASTER\_USER='zhangsan',MASTER\_PASSWORD='123456',MASTER\_LOG\_FILE='File名字',MASTER\_LOG\_POS=Position数字;

CHANGE MASTER TO MASTER\_HOST='192.168.154.1',

MASTER\_USER='zhangsan',

MASTER\_PASSWORD='123456',

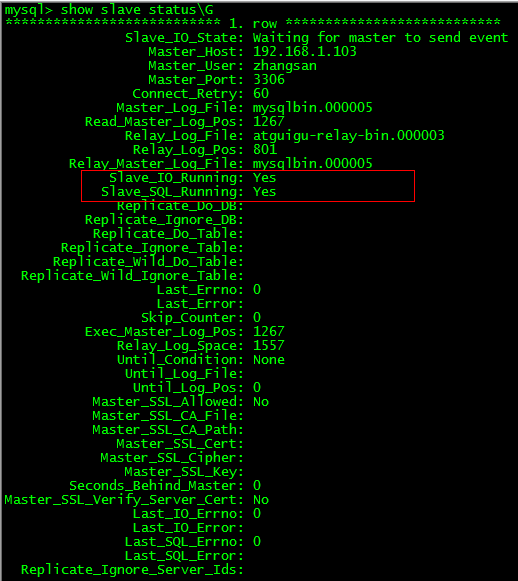
MASTER\_LOG\_FILE='mysqlbin.具体数字',MASTER\_LOG\_POS=具体值;



启动从服务器复制功能

start slave;

show slave status\G



下面两个参数都是Yes，则说明主从配置成功！

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

* + 1. 主机新建库、新建表、insert记录，从机复制
    2. 如何停止从服务复制功能

stop slave;