**尚硅谷大数据技术之MySQL For Linux**

# 第1章MySQL简介

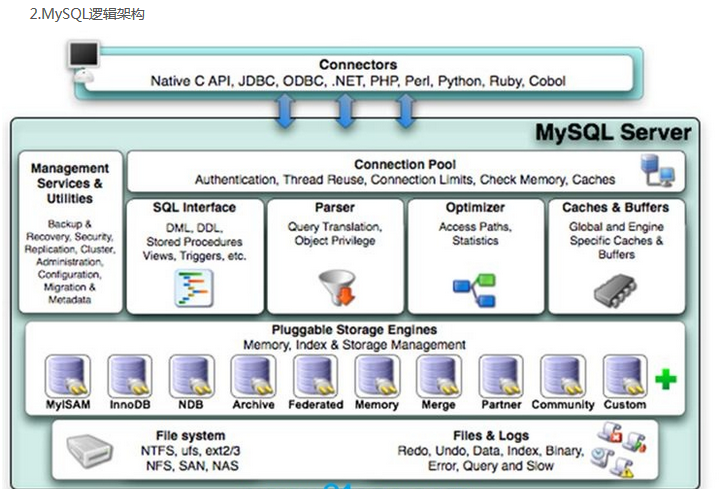
## 1.1 什么是Mysql

1. 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。
2. Mysql是开源的，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。
3. MySQL使用标准的SQL数据语言形式。
4. 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。
5. 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。

## 1.2 Mysql高手是怎样练成的

1. 数据库内部结构和原理
2. 数据库建模优化
3. 数据库索引建立
4. SQL语句优化
5. SQL编程(自定义函数、存储过程、触发器、定时任务)
6. mysql服务器的安装配置
7. 数据库的性能监控分析与系统优化
8. 各种参数常量设定
9. 主从复制
10. 分布式架构搭建、垂直切割和水平切割
11. 数据迁移
12. 容灾备份和恢复
13. shell或python等脚本语言开发
14. 对开源数据库进行二次开发

## 1.3 整体架构图



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

|  |  |
| --- | --- |
| Management Serveices & Utilities | 系统管理和控制工具 |
| 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 |
| 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 |
| 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，比如有where条件时，优化器来决定先投影还是先过滤。 |
| 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。

（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。

# 第2章MySQL安装使用

## 2.1在Linux上安装MySQL

### 2.1.1 MySQL官网

MySQL官网下载地址

<http://dev.mysql.com/downloads/mysql/>

MySQL搜狐镜像下载地址

http://mirrors.sohu.com/mysql/MySQL-5.7/

### 2.1.2 MySQL安装

（1）检查当前系统是否安装过Mysql, 需要说明的是CentOS7默认已安装mariadb(MySQL源代码的一个分支，MySQL的另外一个衍生版本) ，因此在CentOS6中要检查mysql，而CentOS7要检查mariadb

[root@hadoop102 ~]$ rpm -qa|grep mariadb

mariadb-libs-5.5.56-2.el7.x86\_64 //如果存在通过如下命令卸载

[root@hadoop102 ~]$ rpm -e --nodeps mariadb-libs //用此命令卸载mariadb

1. 将MySQL安装包拷贝到/opt/software目录下

[root@hadoop102 software]# ll

总用量 528384

-rw-r--r--. 1 root root 609556480 3月 21 15:41 mysql-5.7.28-1.el7.x86\_64.rpm-bundle.tar

1. 解压MySQL安装包

[root@hadoop102 software]# tar -xf mysql-5.7.28-1.el7.x86\_64.rpm-bundle.tar



1. 在安装目录下执行rpm安装

[root@hadoop102 software]$ rpm -ivh mysql-community-common-5.7.28-1.el7.x86\_64.rpm

[root@hadoop102 software]$ rpm -ivh mysql-community-libs-5.7.28-1.el7.x86\_64.rpm

[root@hadoop102 software]$rpm -ivh mysql-community-libs-compat-5.7.28-1.el7.x86\_64.rpm

[root@hadoop102 software]$ rpm -ivh mysql-community-client-5.7.28-1.el7.x86\_64.rpm

[root@hadoop102 software]$ rpm -ivh mysql-community-server-5.7.28-1.el7.x86\_64.rpm

注意:按照顺序依次执行

1. 删除/etc/my.cnf文件中datadir指向的目录下的所有内容:

查看datadir的值：

[root@hadoop102 software] cat /etc/my.cnf

[mysqld]

datadir=/var/lib/mysql

删除/var/lib/mysql目录下的所有内容:

[root@hadoop102 mysql]# pwd

/var/lib/mysql

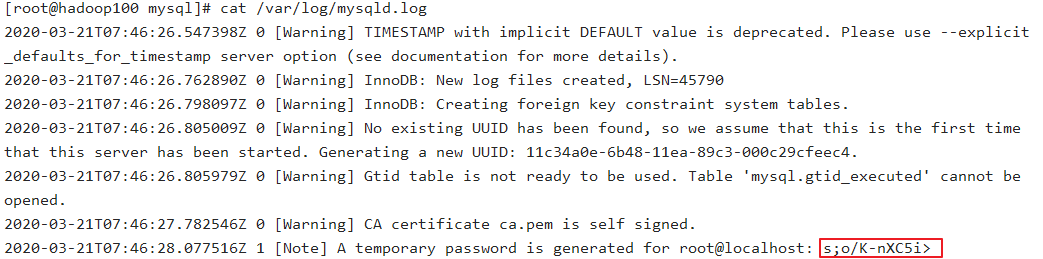
[root@hadoop102 mysql]# rm -rf \* //注意执行命令的位置

1. 初始化数据库

[root@hadoop102 opt]$ mysqld --initialize --user=mysql

1. 查看临时生成的root用户的密码

[root@hadoop102 opt]$ cat /var/log/mysqld.log



1. 启动MySQL服务

[root@hadoop102 opt]$ service mysqld start

1. 登录MySQL数据库

[root@hadoop102 opt]$ mysql -uroot -p

Enter password: 输入临时生成的密码

登录成功.

1. 必须先修改root用户的密码,否则会报错

mysql> set password = password("新密码");

### 2.1.3 Mysql的安装位置

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 路径 | 解释 | 备注 |
| --datadir | /var/lib/mysql/ | mysql数据库文件的存放路径 |  |
| --basedir | /usr/bin | 相关命令目录 | mysqladmin mysqldump等命令 |
| --plugin-dir | /usr/lib64/mysql/plugin | mysql插件存放路径 |  |
| --log-error | /var/log/mysqld.log | mysql错误日志路径 |  |
| --pid-file | /var/run/mysqld/mysqld.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 |

### 2.1.4 MySQL服务的自启动

（1）Mysql服务是开机自动启动

[root@hadoop102 ~]$ systemctl list-unit-files | grep mysqld.service

mysqld.service enabled

1. 如果要取消开机自启动，

[root@hadoop100 mysql]# systemctl disable mysqld.service

Removed symlink /etc/systemd/system/multi-user.target.wants/mysqld.service.

[root@hadoop100 mysql]# systemctl list-unit-files | grep mysqld.service

mysqld.service disabled

### 2.1.5 修改字符集

（1） 常用命令

|  |  |  |
| --- | --- | --- |
| SQL语句 | 描述 | 备注 |
| show databases | 列出所有数据库 |  |
| create database 库名 | 创建一个数据库 |  |
| create database 库名 character set utf8 | 创建数据库，顺便执行字符集为utf-8 |  |
| show create database 库名 | 查看数据库的字符集 |  |
| show variables like ‘%char%’ | 查询所有跟字符集相关的信息 |  |
| set [字符集属性]=utf8 | 设置相应的属性为utf8 | 只是临时修改，当前有效。服务重启后，失效。 |
| alter database 库名character set 'utf8' | 修改数据库的字符集 |  |
| alter table 表名convert to character set 'utf8' | 修改表的字符集 |  |

（2）案例

mysql>CREATE database mydb;

mysql>use mydb;

mysql>CREATE table mytbl(id int,name varchar(50));

mysql>insert into mytbl(id,name) values (1001,'jack');

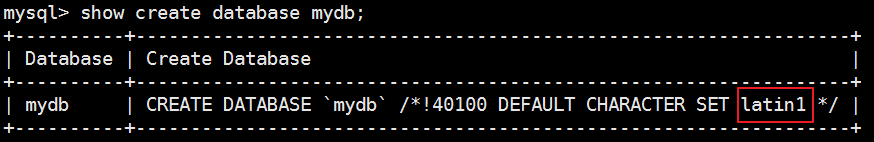
mysql>insert into mytbl(id,name) values (1002,'张三');

ERROR 1366 (HY000): Incorrect string value: '\xE5\xBC\xA0\xE4\xB8\x89' for column 'name' at row 1

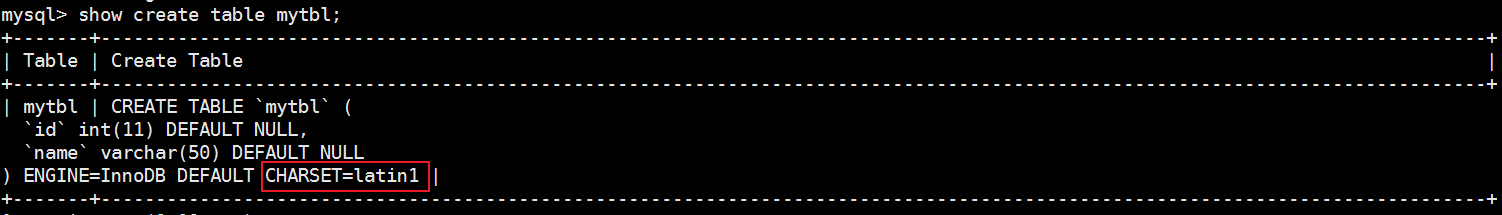
Error原因: 当前数据库的编码不支持中文。

（3）查看默认的编码字符集

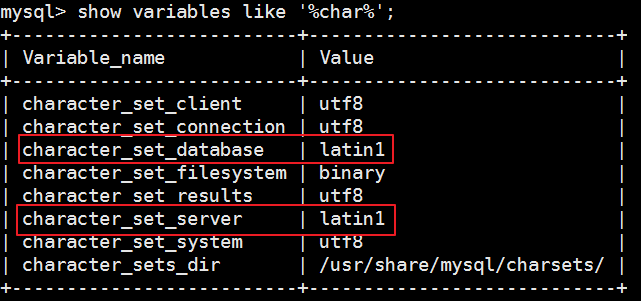
mysql>show create database mydb;



mysql>show create table mytbl;



mysql>show variables like '%char%'



（4）永久修改默认的编码字符集

[root@hadoop102 ~]$ vim /etc/my.cnf

#添加如下配置

[client]

default-character-set=utf8

[mysqld]

character\_set\_server=utf8

collation-server=utf8\_general\_ci

[mysql]

default-character-set=utf8

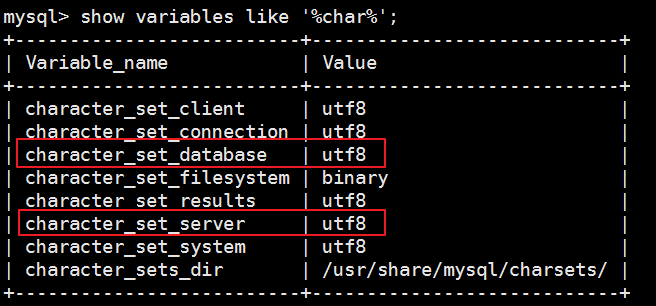
（5）重新启动MySQL服务

[root@hadoop102 ~]# service mysqld restart

停止 mysqld： [确定]

正在启动 mysqld： [确定]

再次查看：



注意：已经创建的数据库的设定不会发生变化，参数修改只对新建的数据库有效！

（6）修改已有库和表的编码

mysql> show create database mydb ; # 查看mydb库的 建库语句

mysql> show create table mytbl ; # 查看mytbl表的建表语句

mysql> alter database mydb character set 'utf8';

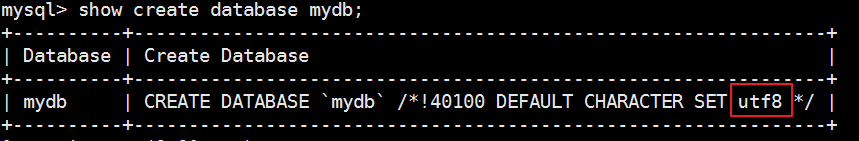
Query OK, 1 row affected (0.01 sec)

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

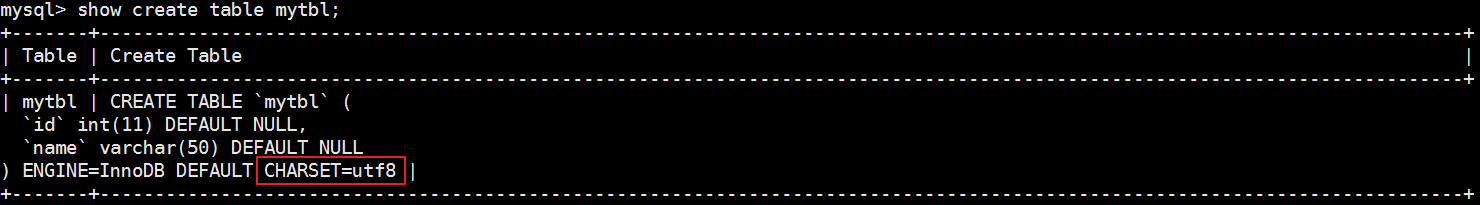
Query OK, 1 row affected (0.09 sec)

Records: 1 Duplicates: 0 Warnings: 0

mysql> show create database mydb;



mysql> show create table mytbl;



（7）再次插入中文数据

mysql> insert into mytbl (id, name) values ("1002","张三");

mysql> select \* from mytbl;

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

| id | name |`

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

| 1001 | zhangsan |

| 1002 | 张三 |

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

## 2.2 MySQL的一些杂项配置

### 2.2.1 sql\_mode

sql\_mode定义了对Mysql中sql语句语法的校验规则！

sql\_mode是个很容易被忽视的变量，如果设置为空值（非严格模式），在这种情况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。

（1） sql\_mode常用的值

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

（2） 查看当前的sql\_mode

mysql> select @@sql\_mode;

（3） 临时修改 sql\_mode

mysql> set @@sql\_mode='';

（4） 永久修改，需要在配置文件my.cnf中修改

[root@hadoop102 ~]$ vim /etc/my.cnf

#添加下列配置，然后重启mysql即可

[mysqld]

sql\_mode=''

（5） sql\_mode的影响案例：group by 查询语法错误！

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);

查询每个dept中年龄最大的人：

错误结果：

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

# 分组: group by以后， select后面只能跟 组标识 及 聚合函数(组函数)

正确结果：

SELECT id,name,ab.dept,ab.maxage 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;

## 2.3 MySQL的用户管理

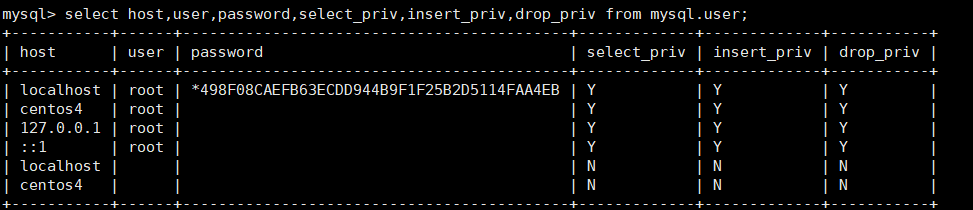
（1）MySQL的用户管理在 mysql库中的user表中

需要了解的列: Host,User, authentication\_string等， 可通过 desc user 查看user表结构

（1）相关命令

|  |  |  |
| --- | --- | --- |
| 命令 | 描述 | 备注 |
| create user zhang3 identified by '123123'; | 创建名称为zhang3的用户，密码设为123123； |  |
| select host,user,password,select\_priv,insert\_priv,drop\_priv from mysql.user; | 查看用户和权限的相关信息 |  |
| set password =password('123456') | 修改当前用户的密码 |  |
| update mysql.user set authentication\_string=password('123456') where user='li4'; | 修改其他用户的密码  注意:mysql 5.7 通过authentication\_string表示密码列 | 所有通过user表的修改，必须用flush privileges;命令才能生效 |
| update mysql.user set user='li4' where user='wang5'; | 修改用户名 | 所有通过user表的修改，必须用flush privileges;命令才能生效 |
| drop user li4 | 删除用户 | 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。 |

（2） 示例说明



**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连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 。

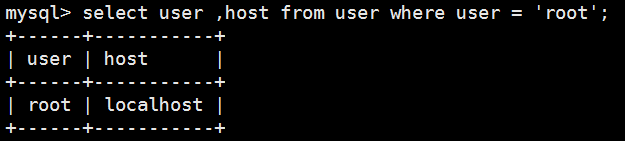
mysql 5.7 的密码保存到 authentication\_string 字段中不再使用password 字段。

**select\_priv , insert\_priv**等 :

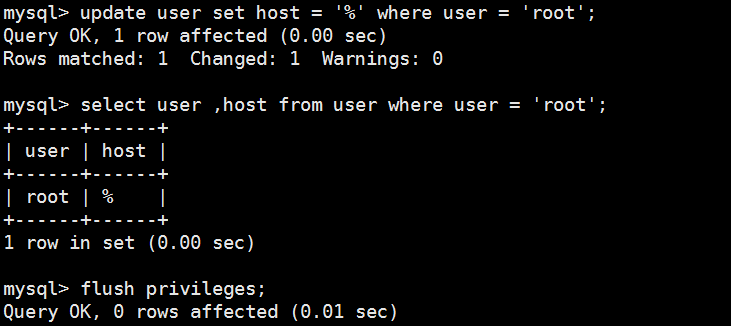
该用户所拥有的权限。

（3）通过远程工具访问MySQL

当前root用户对应的host值为localhost，意味着只允许本机连接



需要将host的值修改为%，表示允许所有远程通过 TCP方式的连接



通过远程工具测试:

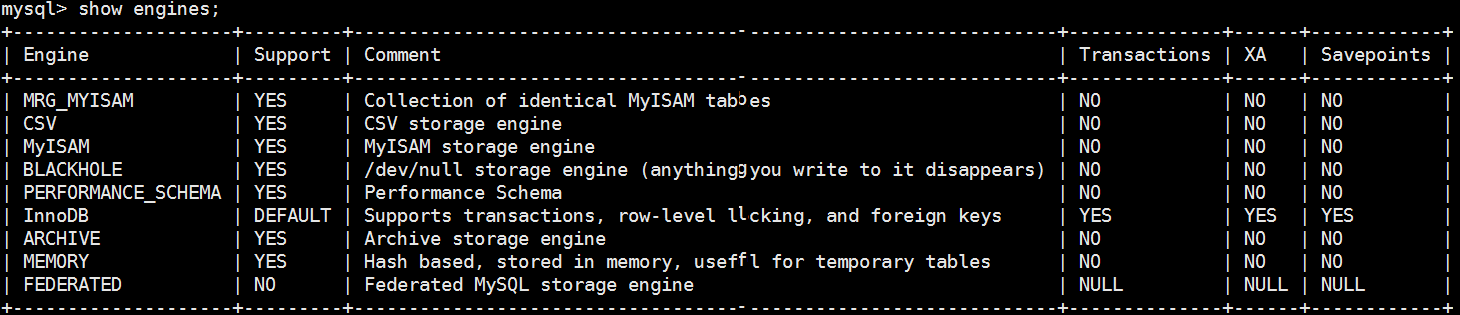


## 2.4 MySQL存储引擎

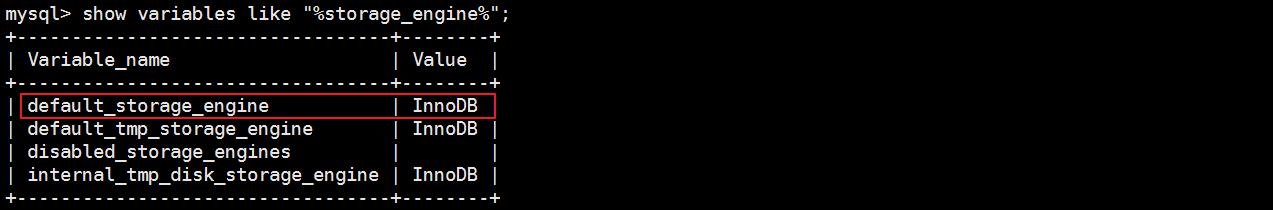
### 2.4.1 查看存储引擎

1. 查看支持的存储引擎

mysql> show engines;



1. 查看当前MySQL默认的存储引擎



### 2.4.2 各个存储引擎介绍

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。

1. 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。

1. 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%。

1. 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。

1. 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。

1. 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。

1. Federated引擎

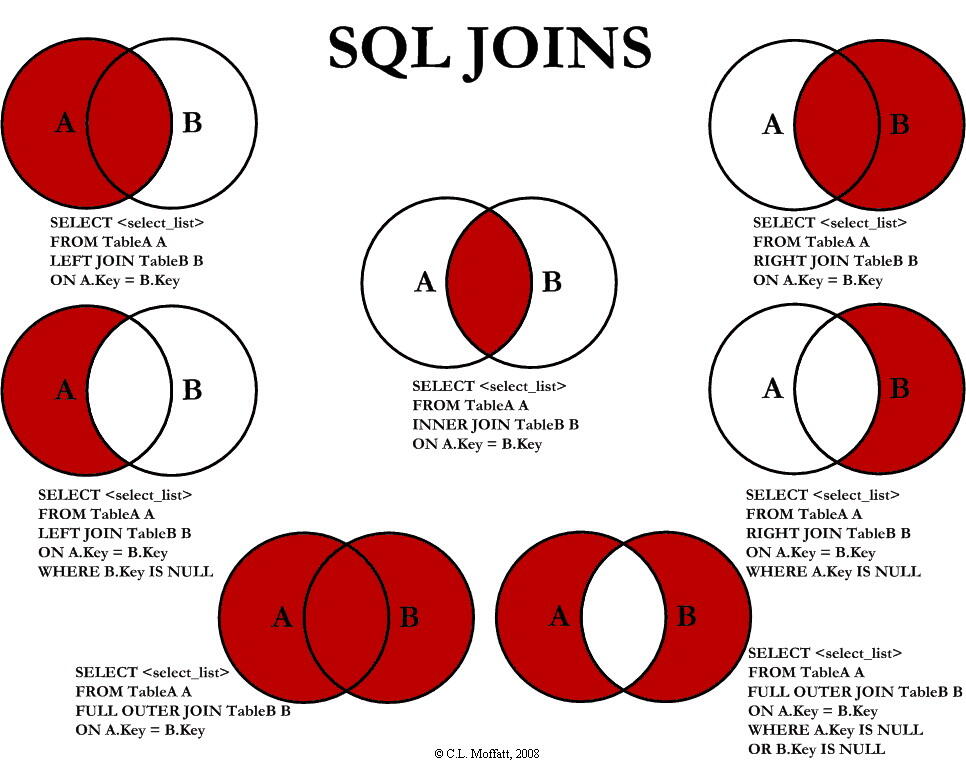
Federated引擎是访问其他MySQL服务器的一个代理，尽管该引擎看起来提供了一种很好的跨服务器的灵活性，但也经常带来问题，因此默认是禁用的。

### 2.4.3 MyISAM和InnoDB引擎

|  |  |  |
| --- | --- | --- |
| 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 | 读性能 | 并发写、事务、资源 |
| 默认安装 | Y | Y |
| 默认使用 | N | Y |
| 自带系统表使用 | Y | N |

# 第3章SQL预热

## 3.1 常见的Join查询图



## 3.2 Join示例

1. 建表语句

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. 所有有门派人员的信息（要求显示门派名称）

SELECT e.`name`,d.`deptName` FROM t\_emp e INNER JOIN t\_dept d ON e.`deptId`=d.`id`;

1. 列出所有人员及其门派信息

SELECT e.`name`,d.`deptName` FROM t\_emp e LEFT JOIN t\_dept d ON e.`deptId`=d.`id`;

1. 列出所有门派

SELECT \* FROM t\_dept;

1. 所有无门派人士

SELECT \* FROM t\_emp WHERE deptId IS NULL;

1. 所有无人门派

SELECT d.\* FROM t\_dept d LEFT JOIN t\_emp e ON d.`id`=e.`deptId` WHERE e.`deptId` IS NULL;

1. 所有人员和门派的对应关系

SELECT \* FROM t\_emp e LEFT JOIN t\_dept d ON e.`deptId`=d.`id`

UNION

SELECT \* FROM t\_emp e RIGHT JOIN t\_dept d ON e.`deptId`=d.`id`;

1. 所有没有入门派的人员和没人入的门派

SELECT \* FROM t\_emp e LEFT JOIN t\_dept d ON e.`deptId`=d.`id` WHERE e.deptId IS NULL

UNION

SELECT \* FROM t\_dept d LEFT JOIN t\_emp e ON d.`id`=e.`deptId` WHERE e.`deptId` IS NULL;

1. 添加CEO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;

1. 求各个门派对应的掌门人名称

SELECT d.deptName,e.name FROM t\_dept d LEFT JOIN t\_emp e ON d.ceo=e.id

1. 求所有当上掌门人的平均年龄

SELECT AVG(e.age) FROM t\_dept d LEFT JOIN t\_emp e ON d.ceo=e.id

1. 求所有人物对应的掌门名称

SELECT ed.name '人物',c.name '掌门'

FROM (SELECT e.name,d.ceo from t\_emp e LEFT JOIN t\_dept d on e.deptid=d.id) ed

LEFT JOIN t\_emp c on ed.ceo= c.id;

SELECT e.name '人物',tmp.name '掌门'

FROM t\_emp e LEFT JOIN (SELECT d.id did,e.name FROM t\_dept d LEFT JOIN t\_emp e ON d.ceo=e.id)tmp

ON e.deptId=tmp.did;

SELECT e1.name '人物',e2.name '掌门'

FROM t\_emp e1

LEFT JOIN t\_dept d on e1.deptid = d.id

LEFT JOIN t\_emp e2 on d.ceo = e2.id ;

SELECT e2.name '人物',

(SELECT e1.name FROM t\_emp e1 where e1.id= d.ceo) '掌门'

from t\_emp e2 LEFT JOIN t\_dept d on e2.deptid=d.id;

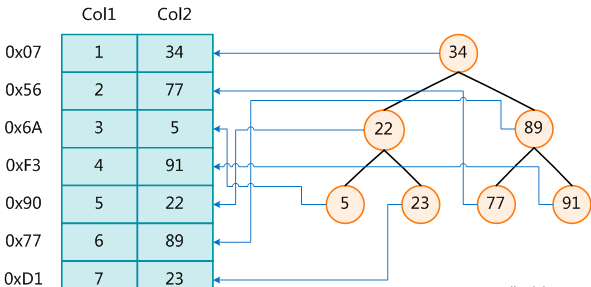
# 第4章 索引优化分析

## 4.1 索引简介

### 4.1.1 索引是什么

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。可以简单理解为排好序的快速查找数据结构。

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。

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。

### 4.1.2 索引的优劣势

1. 优点**：**

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

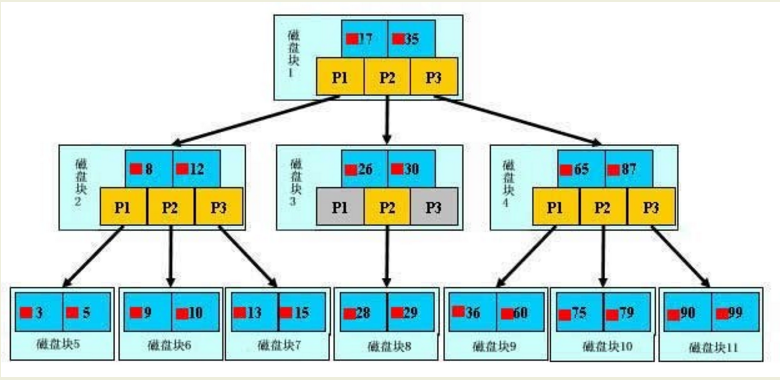
件每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。

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。

## 4.2 MySQL的索引结构

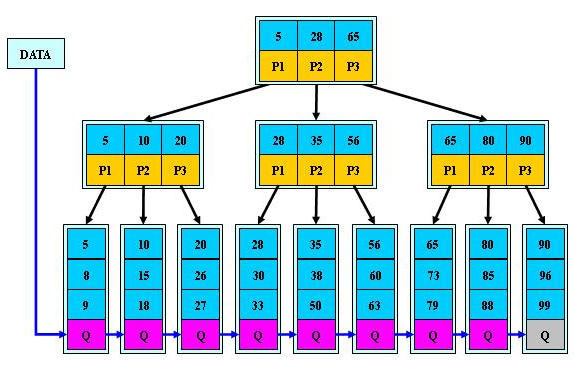
### 4.2.1 B tree索引



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。

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。

### 4.2.2 B+tree索引



B+树的非叶子结点仅仅存储着关键字信息和儿子的指针，B+树中的数据都存储在叶子结点上，也就是其所有叶子结点的数据组合起来就是完整的数据.因此每个磁盘块包含的关键字信息会更多。这样也就决定了加载一个磁盘块可以获取到更多的关键字，可以减少IO操作，

一次IO操作相当于成百上千次的内存比较.

在B+树上增加了顺序访问指针，也就是每个叶子节点增加一个指向相邻叶子节点的指针，这样一棵树成了数据库系统实现索引的首选数据结构

### 4.2.3 B Tree与B+Tree 的区别

1. B-树的关键字和记录是放在一起的， 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。

### 4.2.4 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

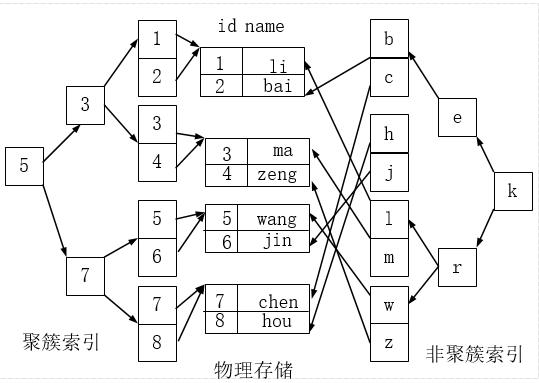
B+树的内部结点并没有指向关键字具体信息的指针。因此其内部结点相对B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多。相对来说IO读写次数也就降低了。

1. 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。

### 4.2.5 聚簇索引和非聚簇索引

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。

## 4.3. MySQL索引分类

### 4.3.1 单值索引

概念：即一个索引只包含单个列，一个表可以有多个单列索引

1. 随表一起创建：

CREATE TABLE customer (

id INT(10) AUTO\_INCREMENT ,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name)

);

1. 单独建单值索引：

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

1. 查看某个表的索引

show index from 表名

show keys from 表名

### 4.3.2 唯一索引

概念：索引列的值必须唯一，但允许有空值

1. 随表一起创建：

CREATE TABLE customer (

id INT(10) AUTO\_INCREMENT ,

customer\_no

VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name),

UNIQUE (customer\_no)

);

1. 单独建唯一索引：

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

### 4.3.3 主键索引

概念：设定为主键后数据库会自动建立索引，innodb为聚簇索引

1. 随表一起建索引

CREATE TABLE customer (

id INT(10) AUTO\_INCREMENT ,

customer\_no

VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id)

);

1. 单独建主键索引：

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

1. 删除建主键索引：

ALTER TABLE customer drop PRIMARY KEY ;

1. 需要注意的问题：

设置为自增的主键上的索引不能删除.

### 4.3.4 复合索引

概念：即一个索引包含多个列

1. 随表一起建索引：

CREATE TABLE customer (

id INT(10) AUTO\_INCREMENT ,

customer\_no VARCHAR(200),

customer\_name VARCHAR(200),

PRIMARY KEY(id),

KEY (customer\_name),

UNIQUE (customer\_name),

KEY (customer\_no,customer\_name)

);

1. 单独建索引：

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

### 4.3.5 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 | 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 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。 |

## 4.4 索引的创建时机

### 4.4.1 适合创建索引的情况

1. 主键自动建立唯一索引
2. 频繁作为查询条件的字段应该创建索引
3. 查询中与其它表关联的字段，外键关系建立索引
4. 单键/组合索引的选择问题， 组合索引性价比更高
5. 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
6. 查询中统计或者分组字段

总结: 查询中过滤、统计、分组、排序、关联所能用到的字段应该建立索引,

建索引优先考虑复合索引，其次考虑单值索引.

### 4.4.2 不适合创建索引的情况

1. 表记录太少
2. 经常增删改的表或者字段
3. Where条件里用不到的字段不创建索引
4. 过滤性不好的不适合建索引

# 第5章 Explain性能分析[了解]

## 5.1 概念

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。

官网地址：

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html

## 5.2 怎么用

Explain+SQL语句。

Explain执行后返回的信息：



## 5.2 Explain分析准备工作

### 5.2.1 Explain字段解释

id: select 查询的序列号`,表示查询中执行select子句或操作表的顺序。

select\_type: 主要用于区别普通查询、联合查询、子查询等的复杂查询。

table: 这个数据是基于哪张表的。

partitions: 查询数据匹配的分区

type: 是查询的访问类型，是较为重要的一个指标

possible\_keys: 显示可能应用在这张表中的索引，一个或多个。

key: 实际使用的索引。如果为NULL，则没有使用索引。

key\_len: 表示索引中使用的字节数

ref: 显示索引的哪一列被使用了

rows: 显示MySQL认为它执行查询时必须检查的行数，不精确。

filtered: 返回结果的行占需要读到的行(rows列的值)的百分比

Extra: 其他的额外重要的信息。

### 5.2.2 创建测试数据

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)));

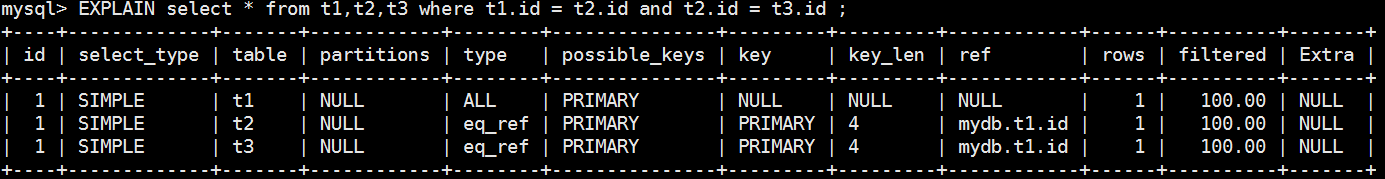
## 5.3 Explain字段分析

### 5.3.1 Explain 之id

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

1. id相同：执行顺序由上至下

mysql>EXPLAIN select \* from t1,t2,t3 where t1.id = t2.id and t2.id = t3.id ;



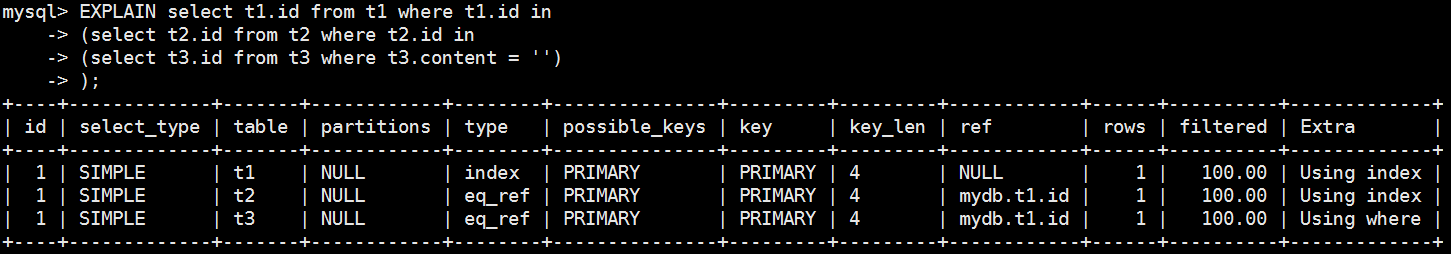
1. id不同：如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行

mysql> EXPLAIN select t1.id from t1 where t1.id in

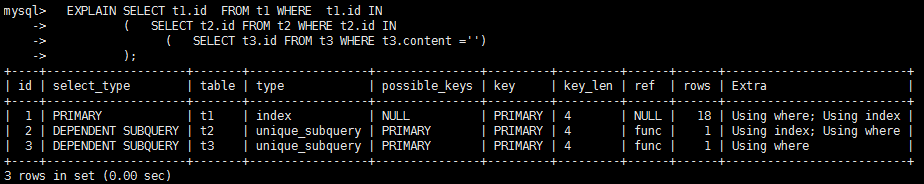
(select t2.id from t2 where t2.id in

(select t3.id from t3 where t3.content = '')

);



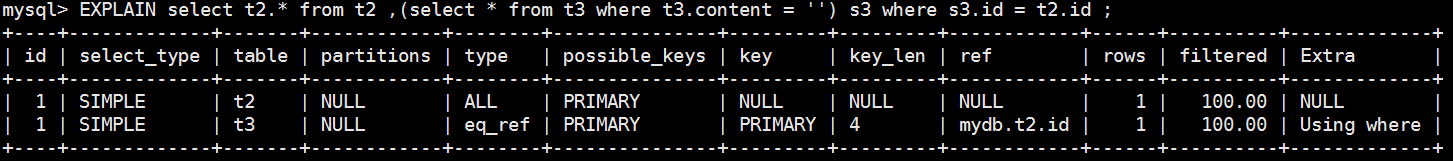
MySQL5.5结果:



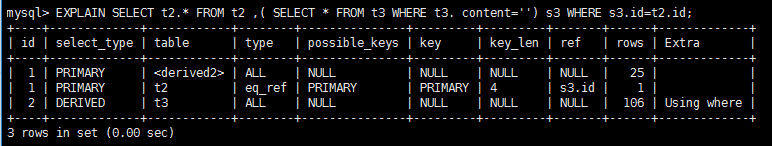
1. 有相同也有不同

mysql> EXPLAIN select t2.\* from t2 ,(select \* from t3 where t3.content = '') s3 where

s3.id = t2.id ;



MySQL5.5的结果:



1. 总结：

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 。

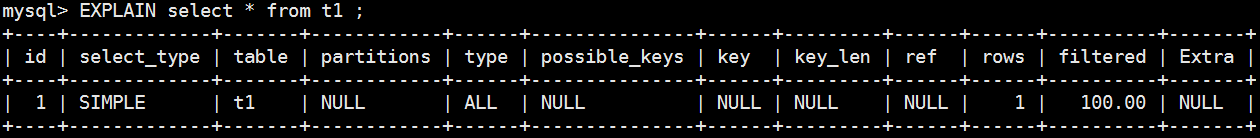
id号每个号码，表示一趟独立的查询。一个sql 的查询趟数越少越好。

### 5.3.2 Explain 之select\_type

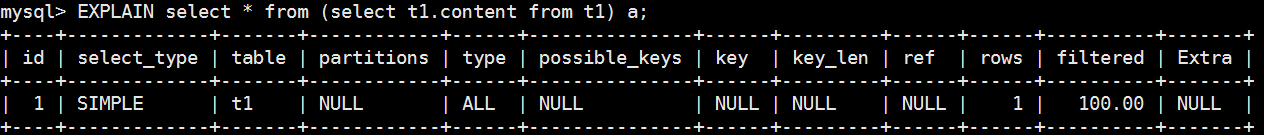
select\_type : 主要用于区别普通查询、联合查询、子查询等的复杂查询。

1. SIMPLE：代表单表查询

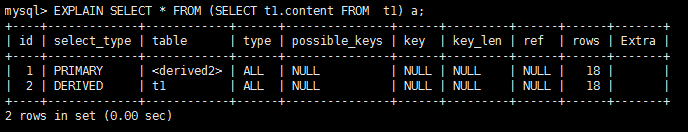
mysql>EXPLAIN select \* from t1 ;



1. 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。



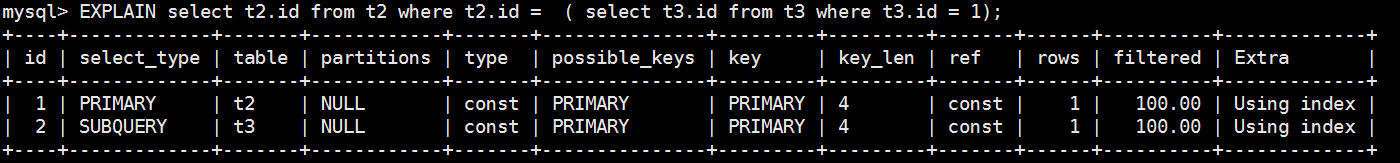
MySQL5.5结果:



1. DERIVED：在FROM查询中的子查询，结果存放在临时表中
2. SUBQUERY：在WHERE列表中包含了子查询。

mysql> EXPLAIN select t2.id from t2 where t2.id =

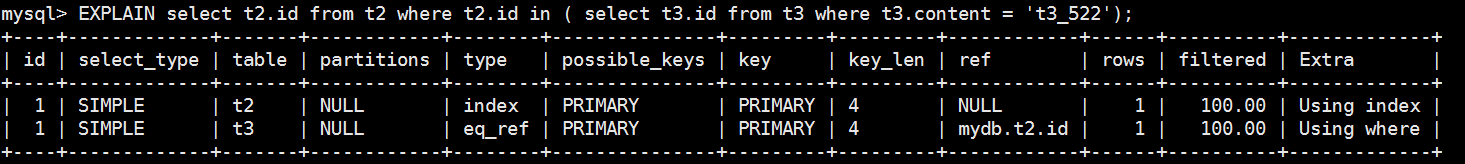
( select t3.id from t3 where t3.id = 1);



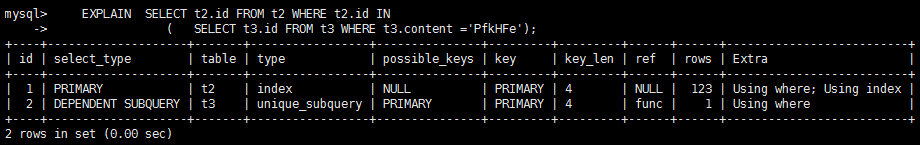
1. DEPENDENT SUBQUERY：在SELECT或WHERE列表中包含了子查询,子查询基于外层。

mysql> EXPLAIN select t2.id from t2 where t2.id in

(select t3.id from t3 where t3.content = 't3\_522');



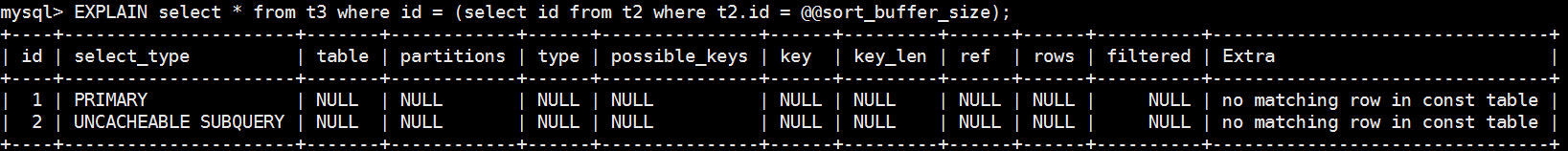
MySQL5.5的结果:

注意：SUBQUERY和DEPENDENT SUBQUERY都是where后面的条件，subquery是单个值，dependent subquery是一组值。

1. UNCACHEABLE SUBQUREY：当使用了@@来引用系统变量的时候，不会使用缓存。

mysql> EXPLAIN select \* from t3 where id =

(select id from t2 where t2.id = @@sort\_buffer\_size);



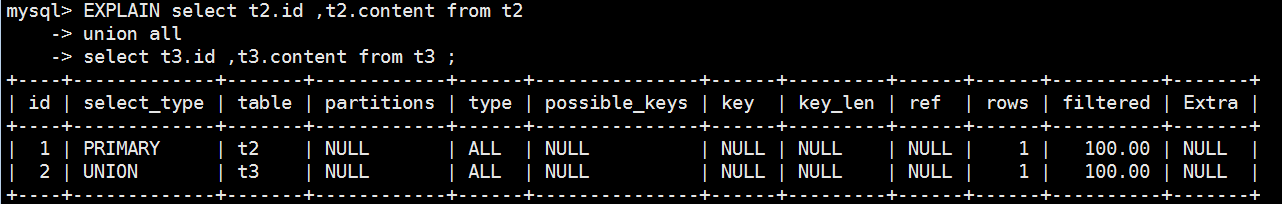
1. UNION：若第二个SELECT出现在UNION之后，则被标记为UNION；

若UNION包含在FROM子句的子查询中,外层SELECT将被标记为：DERIVED。

mysql> EXPLAIN select t2.id ,t2.content from t2

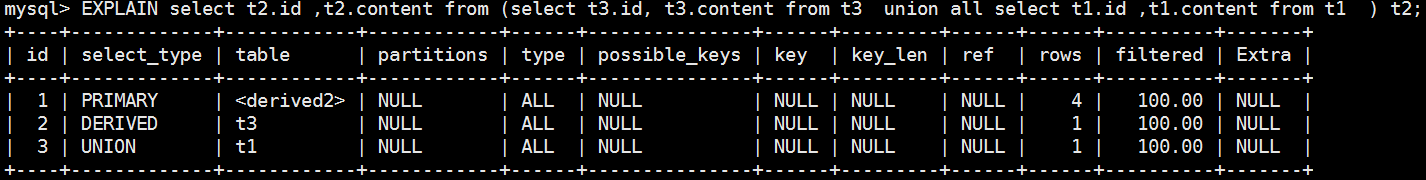
union all

select t3.id ,t3.content from t3 ;



mysql> EXPLAIN select t2.id ,t2.content from

(select t3.id, t3.content from t3 union all select t1.id ,t1.content from t1 ) t2;



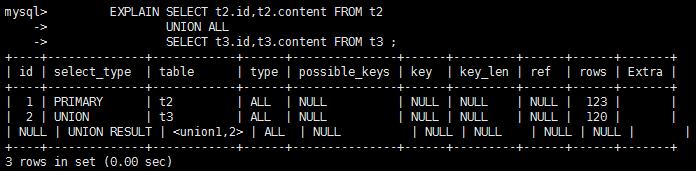
1. UNION RESULT：从UNION表获取结果的SELECT。

mysql> EXPLAIN select t2.id ,t2.content from t2

union all

select t3.id ,t3.content from t3 ;

MySQL5.5结果:



### 5.3.3 Explain 之type\*

type是查询的访问类型，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 ，

一般来说，得保证查询至少达到range级别，最好能达到ref。

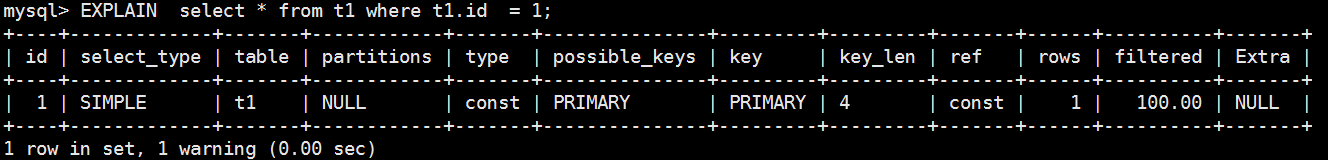
1. system

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。

1. 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。

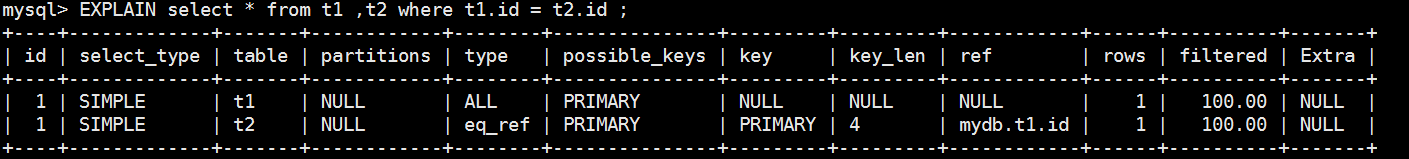
mysql> EXPLAIN select \* from t1 where t1.id = 1;



1. 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。

mysql> EXPLAIN select \* from t1 ,t2 where t1.id = t2.id ;

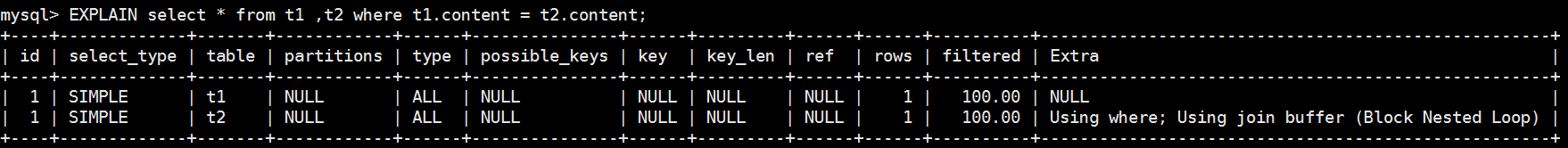


1. ref

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。

* 没用索引前：

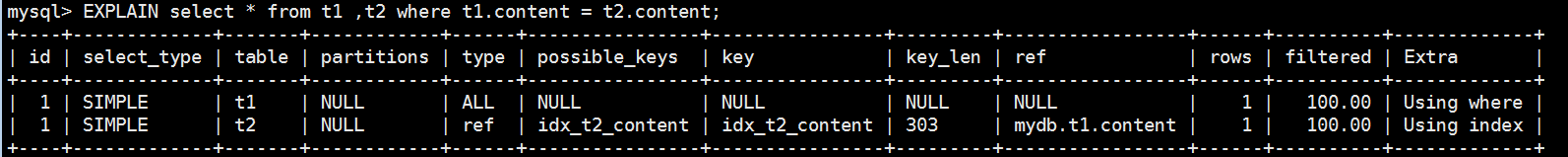
mysql> EXPLAIN select \* from t1 ,t2 where t1.content = t2.content;



* 建立索引后：

mysql> create index idx\_t2\_content on t2(content);

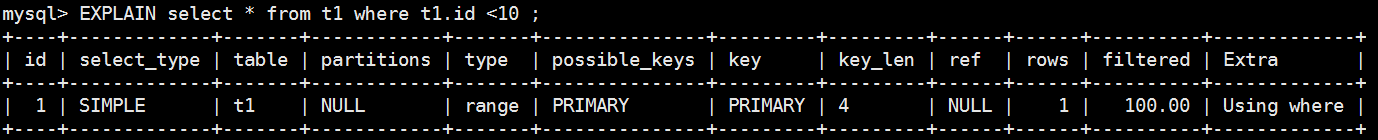
mysql> EXPLAIN select \* from t1 ,t2 where t1.content = t2.content;



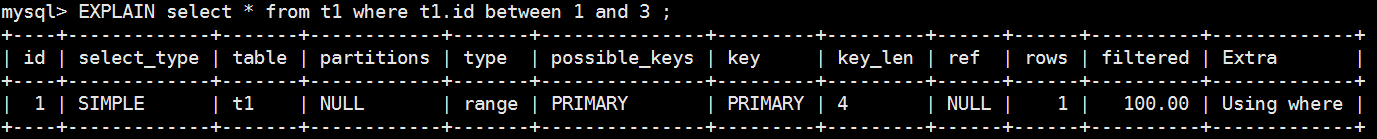
1. range

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。

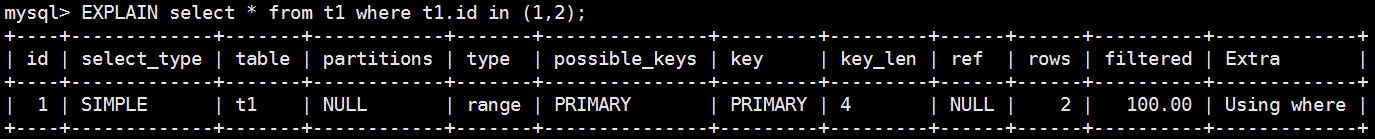
mysql> EXPLAIN select \* from t1 where t1.id >10 ;



mysql> EXPLAIN select \* from t1 where t1.id between 1 and 3 ;



mysql> EXPLAIN select \* from t1 where t1.id in (1,2);

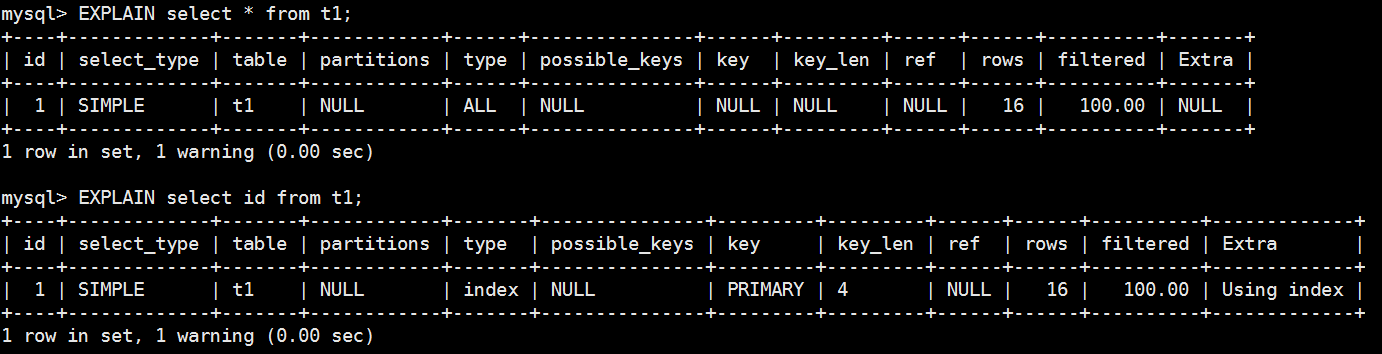


1. index

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。

mysql> EXPLAIN select \* from t1;

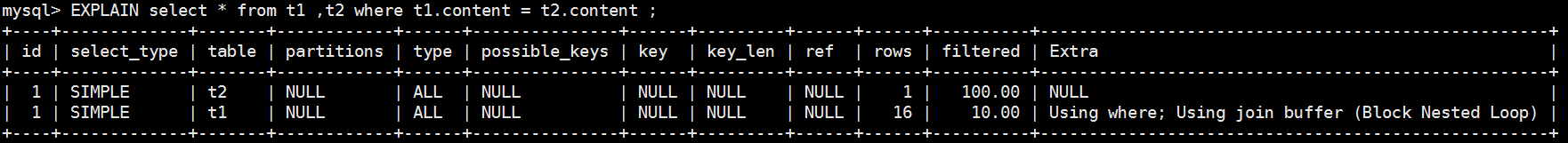
mysql> EXPLAIN select id from t1;



1. all

Full Table Scan，将遍历全表以找匹配的行。

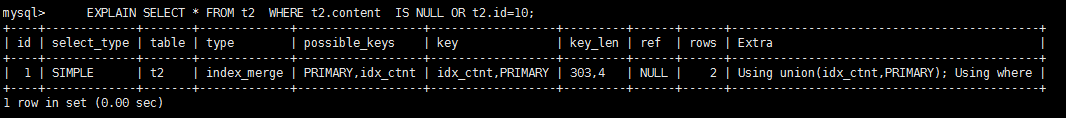
mysql> EXPLAIN select \* from t1 ,t2 where t1.content = t2.content ;



1. 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。

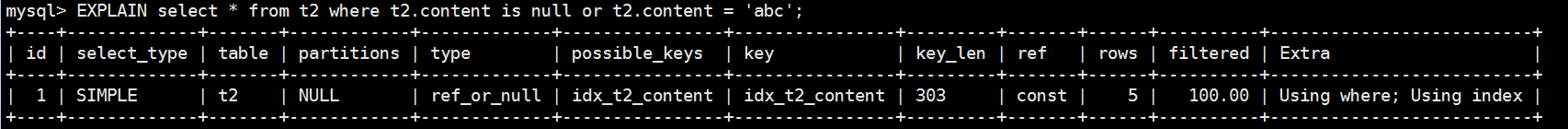
MySQL5.5的结果:



1. 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。

mysql> EXPLAIN select \* from t2 where t2.content is null or t2.content = 'abc';

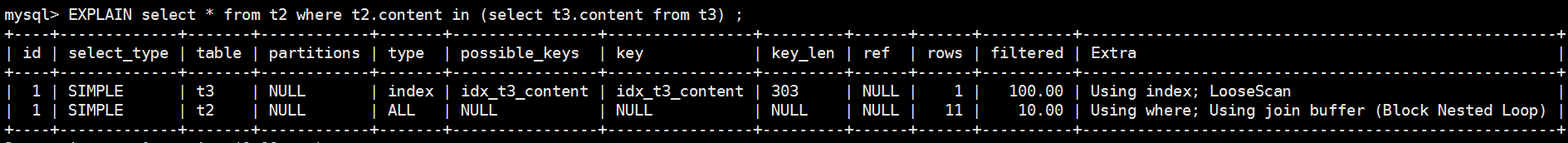


1. index\_subquery

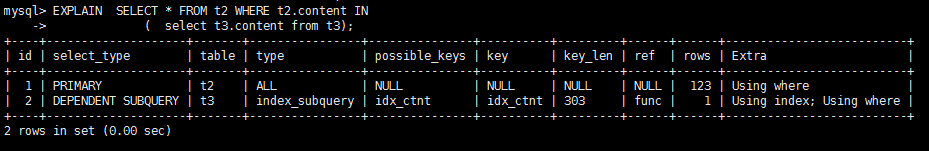
利用索引来关联子查询，不再全表扫描。

mysql> create index idx\_t3\_content on t3(content);

mysql> EXPLAIN select \* from t2 where t2.content in (select t3.content from t3) ;



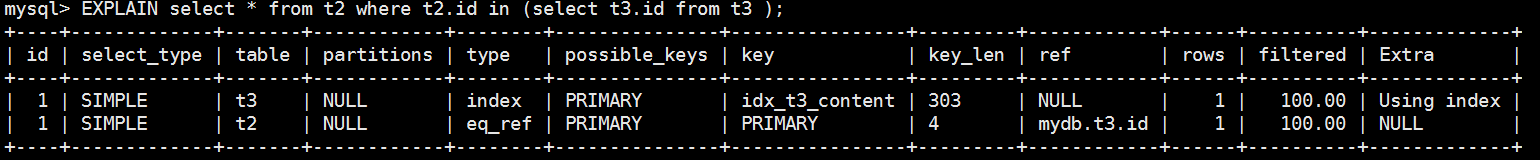
MySQL5.5的结果:



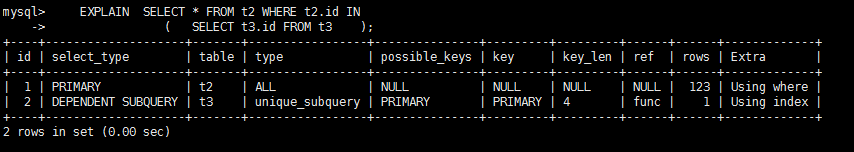
1. unique\_subquery

该联接类型类似于index\_subquery。 子查询中的唯一索引。

mysql> EXPLAIN select \* from t2 where t2.id in (select t3.id from t3 );



MySQL5.5的结果



说明：一般来说，得保证查询至少达到range级别，最好能达到ref。

### 5.3.4 Explain 之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。**

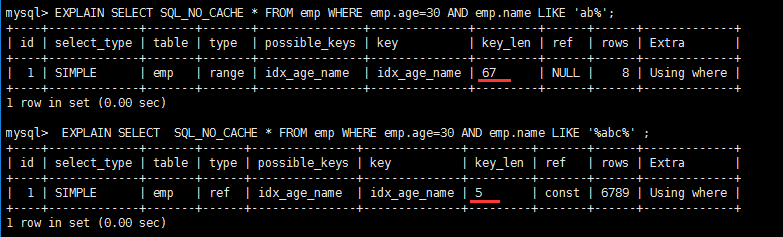
### 5.3.5 Explain 之 key\*

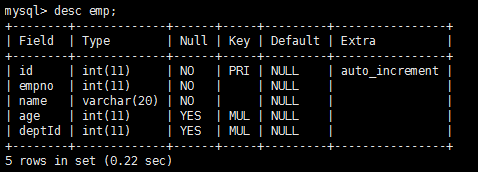
实际使用的索引。如果为NULL，则没有使用索引。

### Explain 之 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。 key\_len字段能够帮你检查是否充分的利用上了索引。ken\_len越长，说明索引使用的越充分。

示例如下





**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



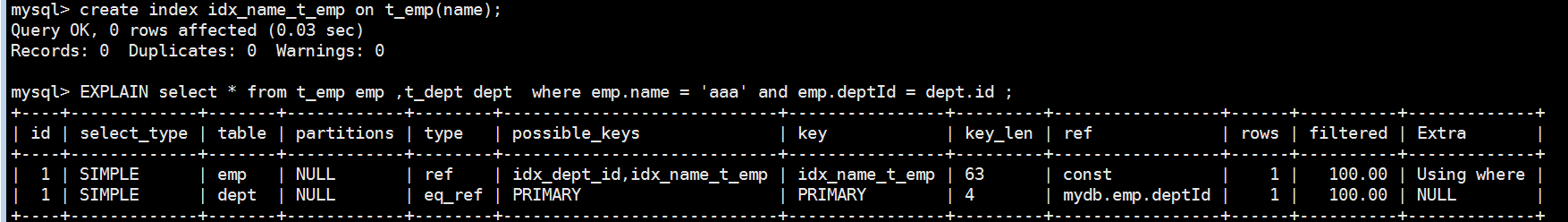
### 5.3.7 Explain 之 ref

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。

mysql> create index idx\_name\_t\_emp on t\_emp(name);

mysql> EXPLAIN select \* from t\_emp emp ,t\_dept dept where emp.name = 'aaa'

and emp.deptId = dept.id ;



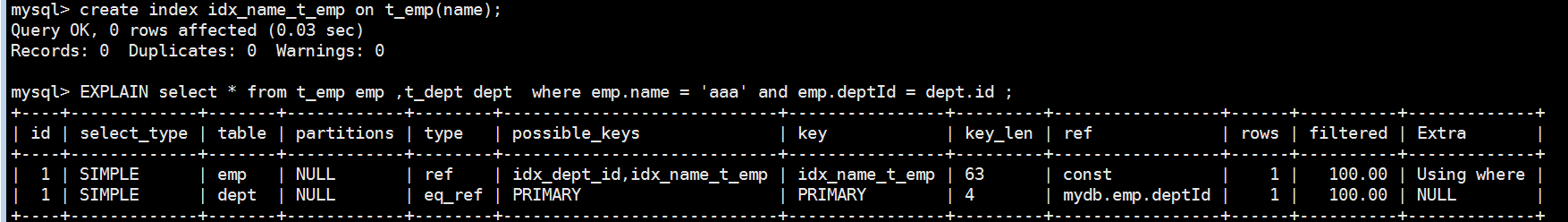
### 5.3.8 Explain 之 rows

rows列显示MySQL认为它执行查询时必须检查的行数。越少越好！

mysql> create index idx\_name\_t\_emp on t\_emp(name);

mysql> EXPLAIN select \* from t\_emp emp ,t\_dept dept where emp.name = 'aaa'

and emp.deptId = dept.id ;



### 5.3.9 Explain 之 Extra

1. Using filesort

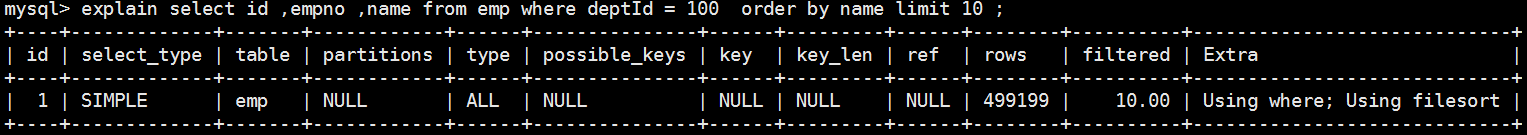
说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成的排序操作称为“文件排序”。

注意: 测试表数据不能太小。

* **优化前：**

mysql> explain select id ,empno ,name from t\_emp

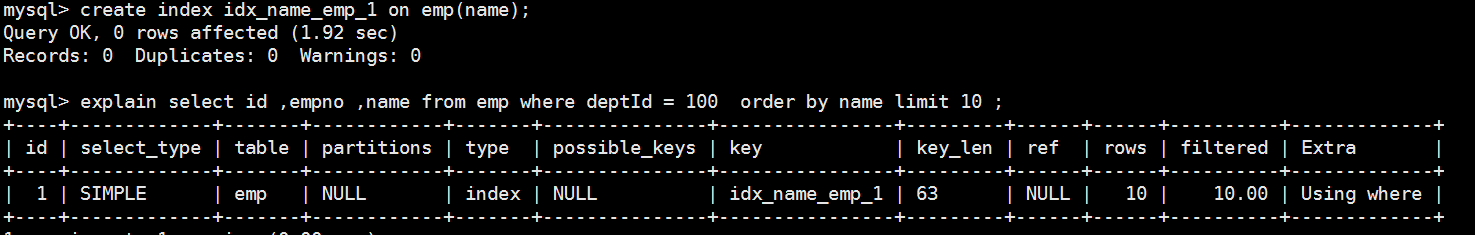
where deptid = 100 order by name limit 10 ;



* **优化后：**

mysql> create index idx\_name\_emp\_1 on emp(name);

mysql> explain select id ,empno ,name from emp where deptId = 100 order by name limit 10 ;

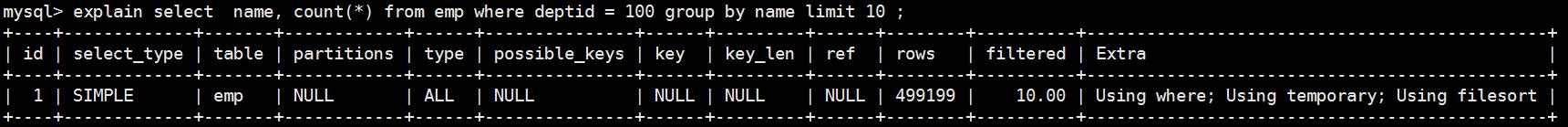


1. Using temporary

使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。

* **优化前：**

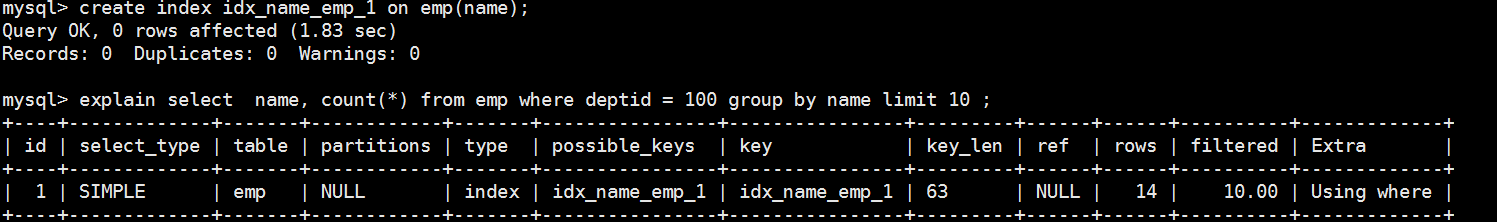
mysql> explain select name, count(\*) from emp where deptid = 100 group by name limit 10 ;



* **优化后：**

mysql> create index idx\_name\_emp\_1 on emp(name);

mysql> explain select name, count(\*) from emp where deptid = 100 group by name limit 10 ;

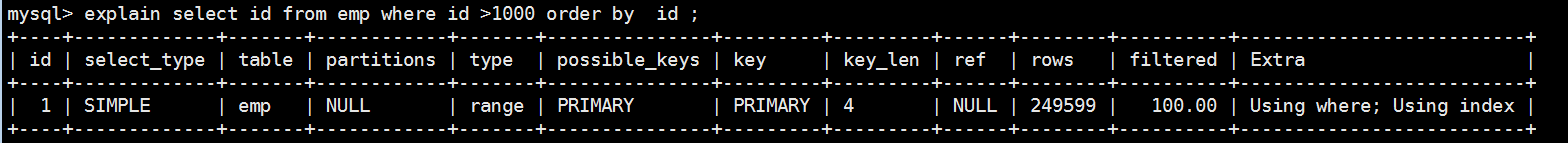


1. Using index

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！

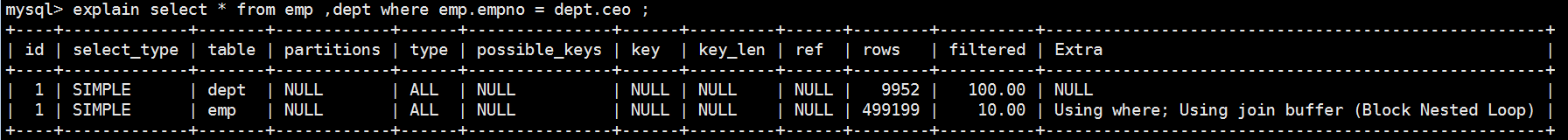
如果同时出现using where，表明索引被用来执行索引键值的查找;如果没有同时出现using where，表明索引只是用来读取数据而非利用索引执行查找。

mysql> explain select id from emp where id >1000 order by id ;



1. Using where：表明使用了where过滤。
2. Using join buffer：使用了连接缓存。

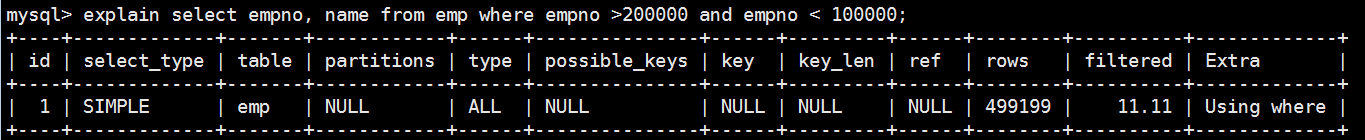
mysql> explain select \* from emp ,dept where emp.empno = dept.ceo ;



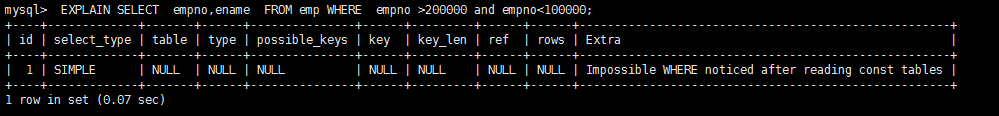
1. impossible where：

where子句的值总是false，不能用来获取任何元组。

mysql> explain select empno, name from emp where empno >200000 and empno < 100000;



MySQL5.5的结果:

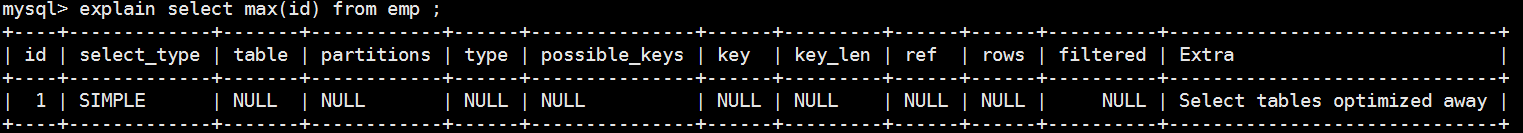


1. select tables optimized away

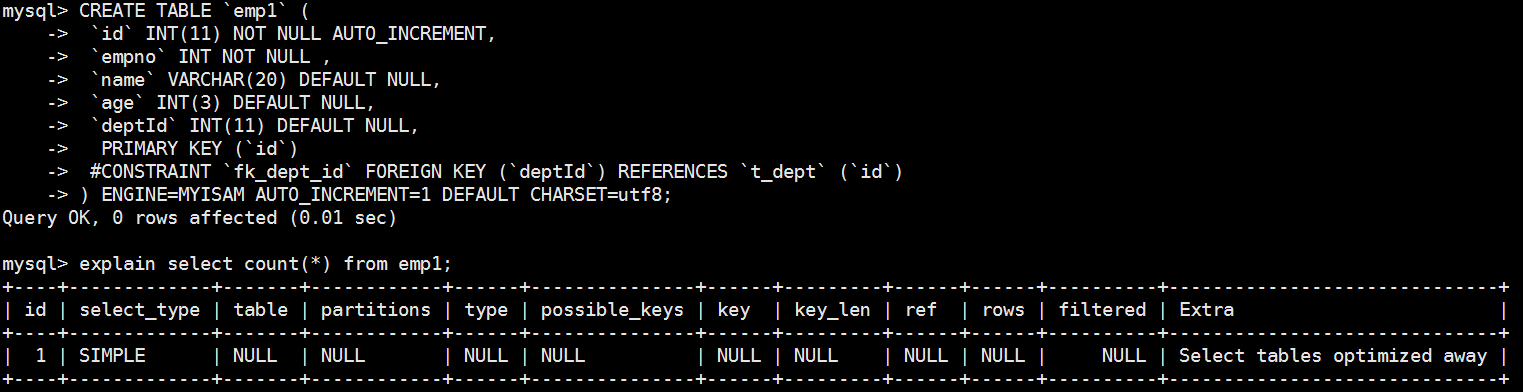
在没有GROUPBY子句的情况下，基于索引优化MIN/MAX操作或者对于MyISAM存储引擎优化COUNT(\*)操作，不必等到执行阶段再进行计算，查询Explian生成的阶段即完成优化。

* **在innodb中：**

mysql> explain select max(id) from emp ;



* **在Myisam中：**



# 第6章 批量数据脚本[了解]

## 6.1 准备工作

1. 建表语句

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;

1. 开启log\_bin\_trust\_function\_creators

它控制是否可以信任存储函数创建者

mysql> show variables like 'log\_bin\_trust\_function\_creators'; //查询

mysql> set global log\_bin\_trust\_function\_creators=1; //开启

设置永久生效方式：修改/etc/my.cnf 文件

[mysqld]

log\_bin\_trust\_function\_creators=1

## 6.2 编写随机函数

1. 生成随机字符串

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 $$

1. 如果要删除函数，则执行：drop function rand\_string
2. 生成随机数字

#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$$

1. 如果要删除函数：drop function rand\_num

## 6.3 创建存储过程

1. 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;

1. 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;

## 6.4 调用存储过程

1. 添加数据到部门表

#执行存储过程，往dept表添加1万条数据

DELIMITER ;

CALL insert\_dept(10000);

1. 添加数据到员工表

#执行存储过程，往emp表添加50万条数据

DELIMITER ;

CALL insert\_emp(100000,500000);

## 6.5 批量删除某个表上的所有索引

1. 删除索引的存储过程

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$$

1. 执行存储过程

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

## 6.6 批量数据脚本

同学们只需要在Mysql中执行如下脚本即可完成上述的准备工作.本章内容无须掌握，只是为后续的内容做准备。

[..\2.资料\批量数据脚本\batch.sql](../2.资料/批量数据脚本/batch.sql)

1. 首先将batch.sql 上传到Linux的某个目录下: 如 /opt/software/mysql/batch.sql ,然后在mysql中执行如下操作:

mysql> source /opt/software/mysql/batch.sql

# 单表使用索引常见的索引失效

## 7.1 全值匹配我最爱

1. 建立索引

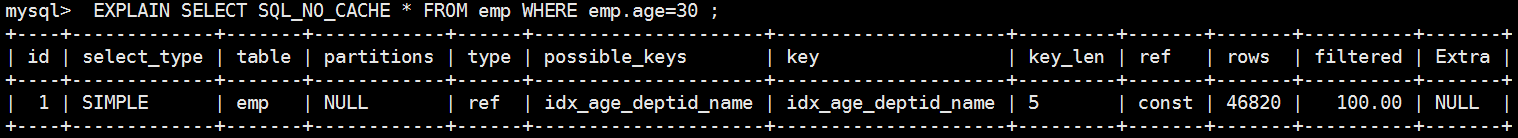
CREATE INDEX idx\_age\_deptid\_name ON emp(age,deptid,NAME);

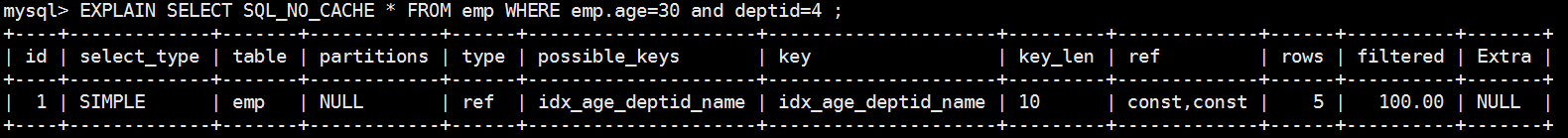
1. 查看下列sql的Explian

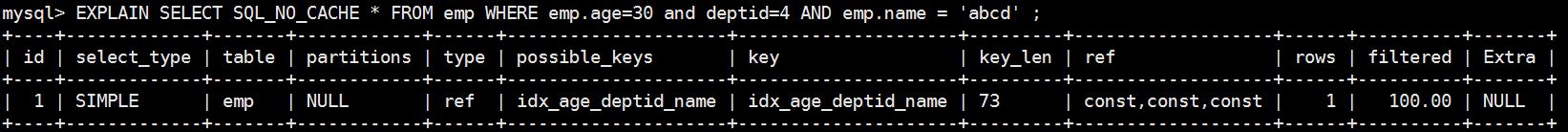
EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30

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

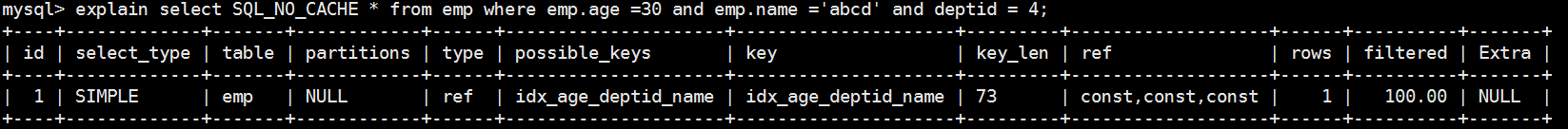
EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptid=4 AND emp.name = 'abcd'





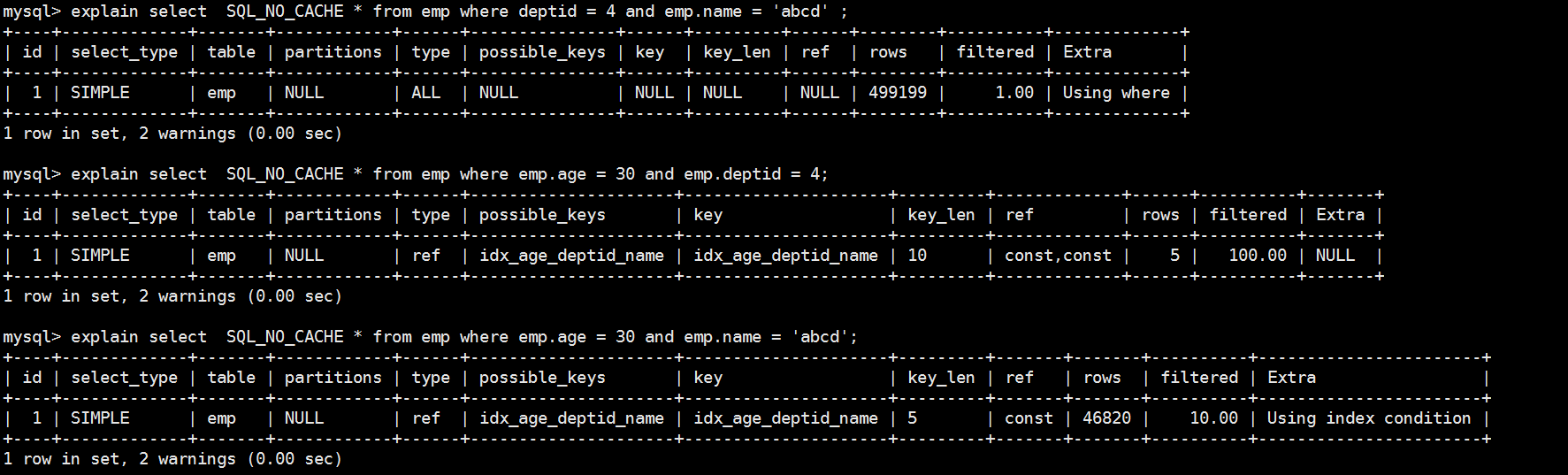
结论：全值匹配我最爱指的是，查询的字段按照顺序在索引中都可以匹配到！

1. 更换条件顺序查看索引能否被应用

结论：SQL中查询字段的顺序，跟使用索引中字段的顺序没有关系。

## 7.2 最佳左前缀法则

使用复合索引，需要遵循最佳左前缀法则，即如果索引了多列，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。



**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。**

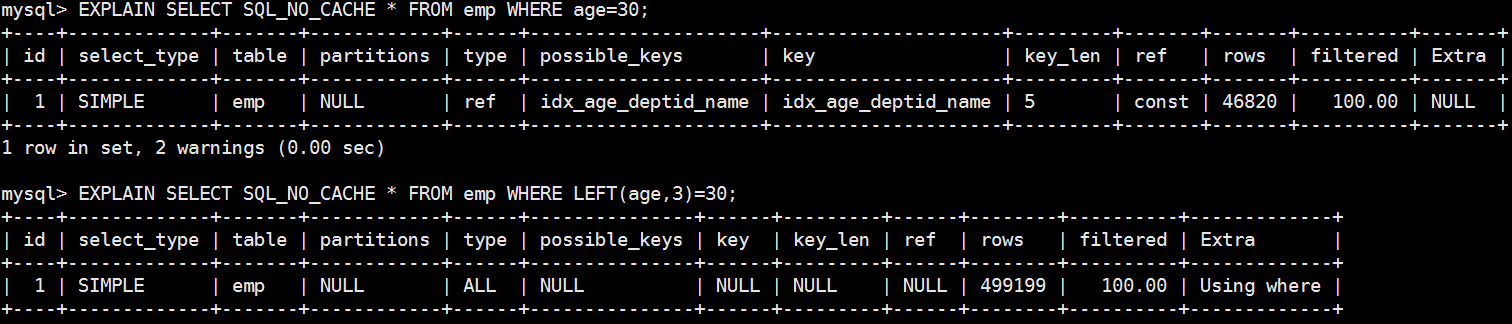
## 7.3 索引列上做计算

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。

1. 在查询列上使用了函数

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

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE LEFT(age,3)=30;



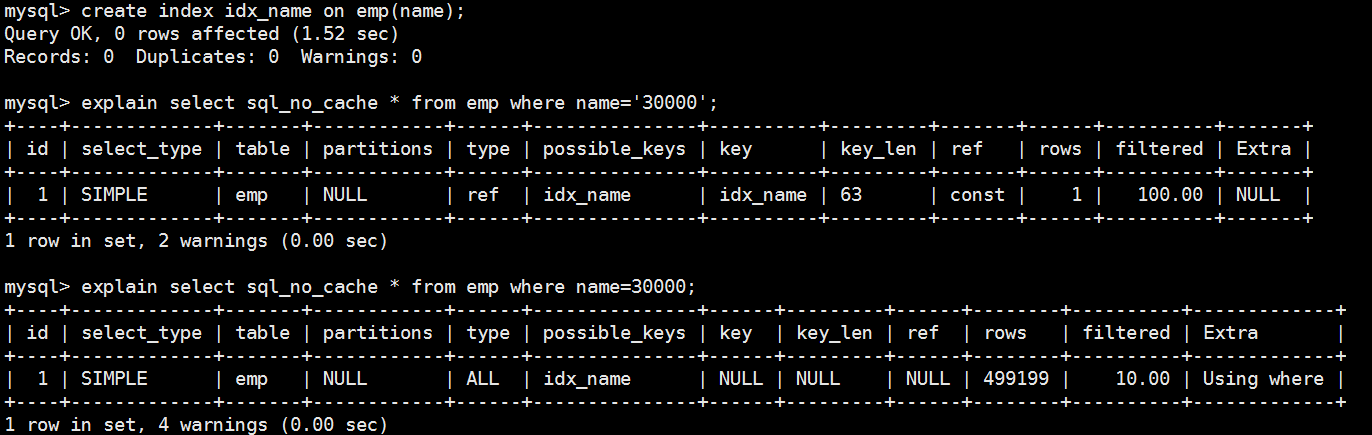
结论：等号左边无计算！

1. 在查询列上做了转换

create index idx\_name on emp(name);

explain select sql\_no\_cache \* from emp where name='30000';

explain select sql\_no\_cache \* from emp where name=30000;



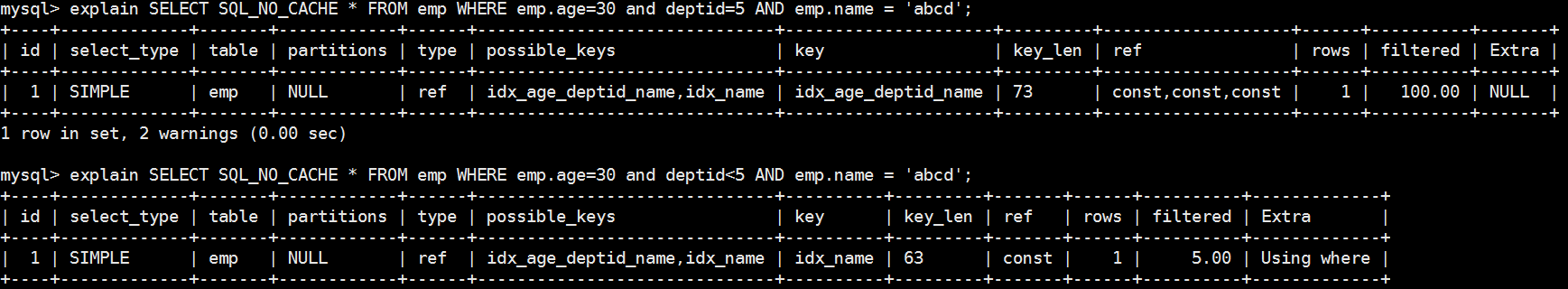
结论：等号右边无转换！

## 7.4 索引列上进行范围查询

1. 在索引列上使用范围查询

explain SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptid=5 AND emp.name = 'abcd';

explain SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptid<5 AND emp.name = 'abcd';



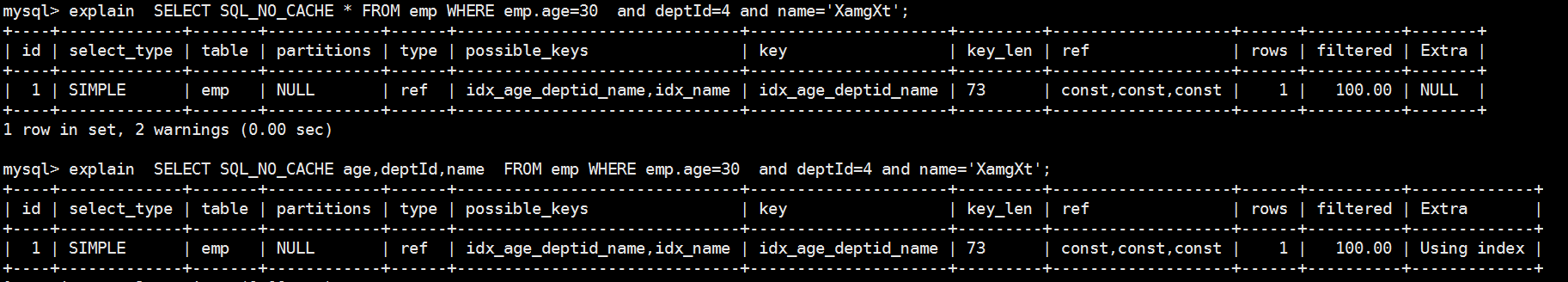
建议：将可能做范围查询的字段的索引顺序放在最后

## 7.5 使用覆盖索引

1. 查询列和索引列时，不要写select \*

explain SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.age=30 and deptId=4 and name='XamgXt';

explain SELECT SQL\_NO\_CACHE age,deptId,name FROM emp WHERE emp.age=30 and deptId=4 and name='XamgXt';

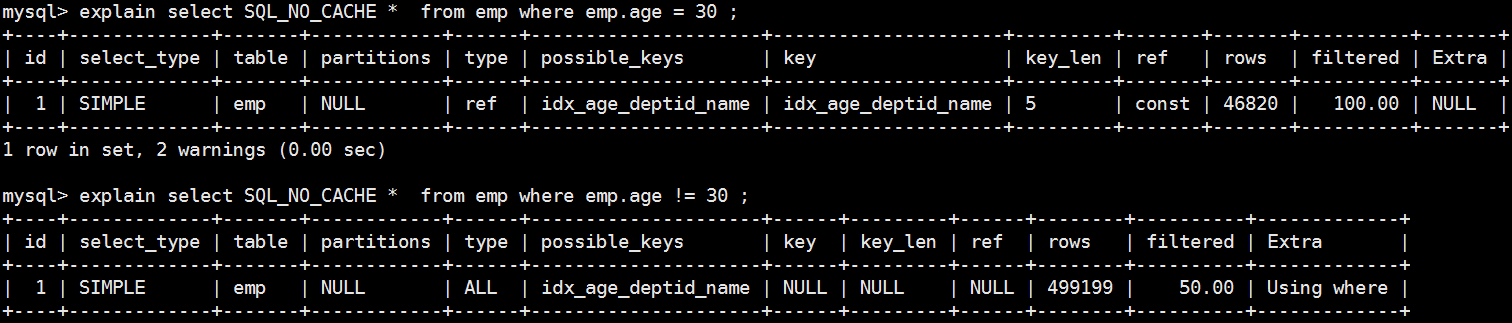


## 7.6 使用不等于(!= 或者<>)

1. 使用不等于(!= 或者<>)时，有时会无法使用索引会导致全表扫描。

mysql> explain select SQL\_NO\_CACHE \* from emp where emp.age = 30 ;

mysql> explain select SQL\_NO\_CACHE \* from emp where emp.age != 30 ;



## 7.7 is not null 和 is null

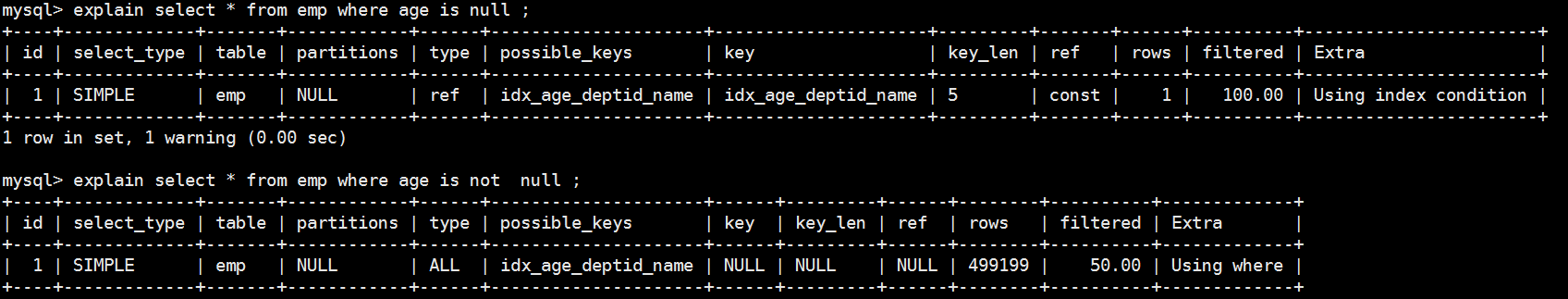
1. 查看索引字段是否允许为空

mysql> desc emp;

1. 当字段允许为Null的条件下

explain select \* from emp where age is null

explain select \* from emp where age is not null



结论：is not null用不到索引，is null可以用到索引。

## 7.8 like的前后模糊匹配

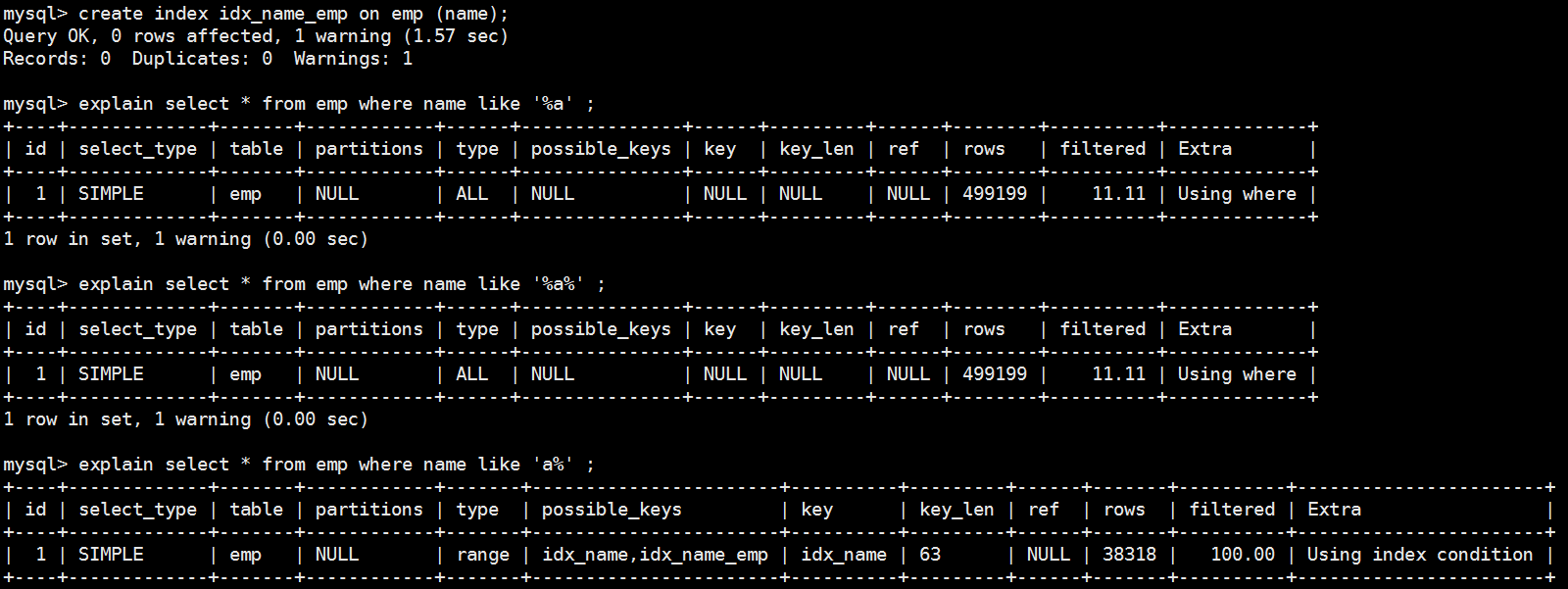
1. 前缀模糊查询会导致索引失效

mysql> create index idx\_name\_emp on emp (name);

mysql> explain select \* from emp where name like '%a' ;

mysql> explain select \* from emp where name like '%a%' ;

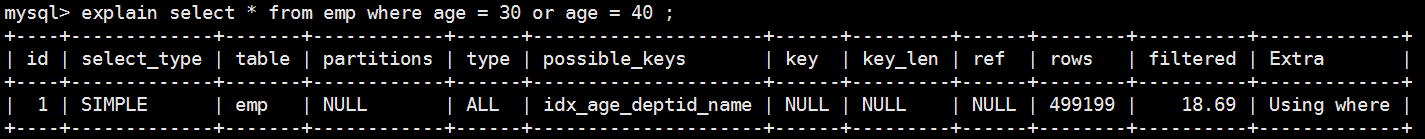
mysql> explain select \* from emp where name like '%a%' ;



## 7.9 使用or

1. OR查询会导致索引失效

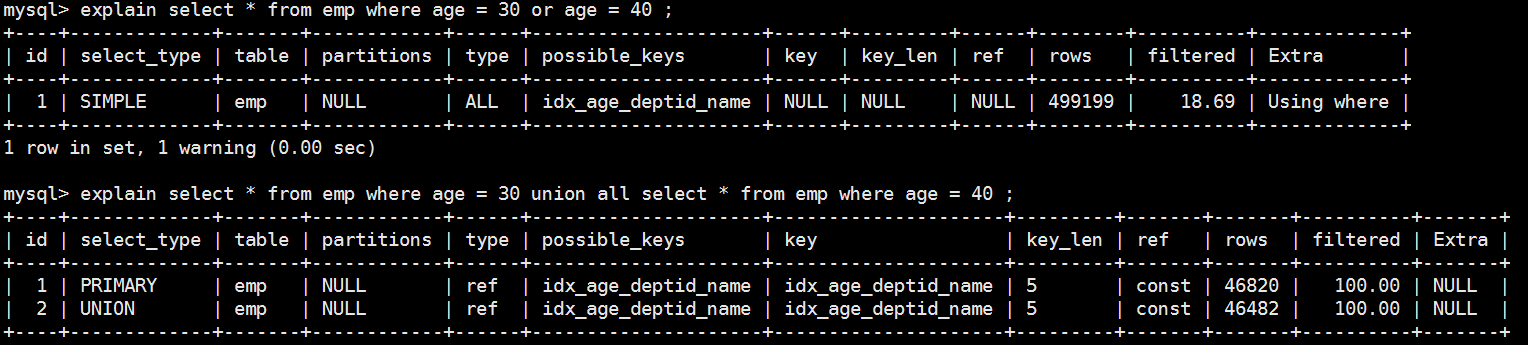
mysql> explain select \* from emp where age = 30 or age = 40 ;



1. 使用union all或者union来替代

mysql> explain select \* from emp where age = 30 union all select \* from emp where age =

40 ;



## 7.10 总结练习

1. 建立索引index(a,b,c)，判断下列索引引用情况。

|  |  |
| --- | --- |
| **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 |

## 7.11 口诀

全值匹配我最爱，最左前缀要遵守；

带头大哥不能死，中间兄弟不能断；

索引列上少计算，范围之后全失效；

LIKE百分写最右，覆盖索引不写\*；

不等空值还有OR，索引影响要注意；

# 第8 章 视图

## 8.1 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。

## 8.2 干什么

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

## 8.3 使用

1. 创建视图

CREATE OR REPLACE VIEW 视图名 AS SQL语句

1. 使用视图

select \* from 视图名