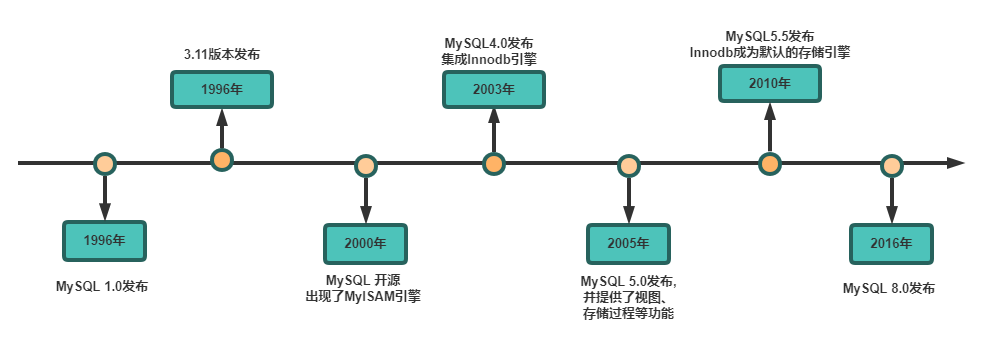
# 框架高级课程系列之MySQL

# MySQL\_简介

## MySQL简介

* MySQL是一个关系型数据库管理系统，由瑞典MySQL AB（创始人Michael Widenius）公司开发，2008被Sun收购（10亿美金），2009年Sun被Oracle收购。MariaDB
* MariaDB基于事务的Maria存储引擎，替换了MySQL的MyISAM存储引擎； 它使用了Percona的 XtraDB代替了InnoDB存储引擎。（虚拟机自带的）
* 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（GNU General Public License）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。
* 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++、C#、Python、Java、Perl、PHP、Eiffel、Ruby和Tcl等。

## MySQL发展史



## MySQL读音

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

参考: <https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>

## MySQL高手是怎样炼成的（开发工程师，DBA，运维）

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

# MySQL\_rpm安装

## MySQL\_Linux版的安装(mysql5.7)

### 下载地址

官网：[https://www.mysql.com](https://www.mysql.com/)

### 打开官网，点击DOWNLOADS

然后，点击MySQL Community(GPL) Downloads

MySQL分为社区版（Community Server）和企业版（Enterprise Server）

计算机生成了可选文字:
MySQL::MySQLDownloadsx+
口
CO
囱☆
丝巴‘
卜不习、
圈令
功e阴而，SmO5tpopularopensourcedatab仍e
q
ContactMySQL}Log.n}Reg.ster
MYSQL.COM
DOWNLOADS
OOCUMEN]-ATION
DEVELOPERZONE
f岁in益
龟
MySQLEnterpriseEdition
MySQLEnterpriseEditionincludesthemostcomprehensivesetofadvancedfeaturesandmanagementtoolsforMySQL.
MySQLDatabase
MySQLStorageEngineS(Inn0DB,MylSAM,etc·）
MySQLConnectorsUDBC,ODBc,.Net,etc·）
M卢QLReplication
MySQLPartitioning
MVSOLROUter尸
MysqLShell一一一一一一一一户
MVSOLWorkbench、、
2叙7飞c卜nicalsupport.
MVSOLEnteFDriseB日CkUDZ
MVSOLEnteFDriseMonit0F工
MySQLEnterprjseHAZ
MysqLEnterpriseTransparentDataEncryption任DE)
MysqLEnterPriseMaskingandDe一ident币cation
MySQLEnterPriseFirevvall
MVSOLEnteFDFISeEnCFVDtion
MySQLEnterpriseAuditl
r乒
LearnMore卜
CUStomerDOWn{oad卜
TFi日｝DOWn}oad卜
Cont日CtSaleS
USA:+1一866一221一0634
Canad日：+1一866一221一0634
Germany:+498914301280
France:+33157608357
ltaly:+390224959120
UK:+442075538447
oracleMySQLCloudService
oracleMySQLcI0udService15builtonMySQLEnterprISeEdltlonandPoweredbyoraclecloud,ProVidlnganenterprise一gradeMySQL
datab日SeSe四！Ce.
LeaFnMOFe"
japan:0120一065556
Chin日门0800一811一0823
Indl日：0008001005870
MOFeCOUntFleS,
ContactUsOnline"
MySQLCIUsterCGE
MySQLcluster15areal一tlmeopenSourCetransaCtlonaldatabasedeslgnedforfast,aIWayS一onaCCeSStodataunderhlghthroughpUt
CondltlonS.
·MySQLCluster
·MySQLclusterManager
·Plus,everythlnginMySQLEnterPriseEdition
Le日FnMOFe"
customerDOWnI0ad》（SelectPatcheS&UPdateSTab,ProductSearch)
TFia}DOWn}0日d》
厂．
MySQLCommunlty(GPL)DownloadS,
旬
日‘―下们月
‘一闷
SqL
NOSOL乙
丫！、叭协

### 点击MySQL Community Server

计算机生成了可选文字:
令C白‘dev.mysql.com/downloadsl
0MySQLCommunityDOWnloadS
·MySQLYumRep0Sitory
·MySQLAPTRepOSitory
·MySQLSUSERepository
.MySQLCIuster
.MySQLRouter
·MySQLShell
.MySQLWorkbench
·Connector/C(Iibmysqlclient)
。ConneCtor/C++
·Connector/J
。ConneCtOF/NET
·Connector/Node．」s
·Connector/ODBC
·ConnectorlPython
·MySQLNativeDriverforPHP
·MySQLInstallerforWindows
·MySQLforExcel
·MySQLforVisualStudio
·MySQLNotifier
·MySQLBenchmarkTOol
·Timezonedescriptiontables
。DOWnloadArChiVeS

### 在General Availability(GA) Releases中选择适合的版本

1. 如果安装Windows 系统下MySQL ，推荐下载MSI安装程序；点击Go to Download Page进行下载即可
2. Windows下的MySQL安装有两种安装程序

* mysql-installer-web-community-5.7.28.0.msi 下载程序大小：19020KB；安装时需要联网安装组件。
* mysql-installer-community-5.7.28.0.msi 下载程序大小：504828KB；安装时离线安装即可。推荐。

计算机生成了可选文字:
名称
大小
mysql一installer一community一5.7.28.0.msi
mysql一installer一web一community一5.7.28.0.msi
504,828KB
19,020KB
寻谬

1. 如果安装Linux系统下MySQL，官方给出多种安装方式

|  |  |
| --- | --- |
| 安装方式 | 特点 |
| rpm | 安装简单，灵活性差，无法灵活选择版本、升级 |
| rpm repository | 安装包极小，版本安装简单灵活，升级方便，需要联网安装 |
| 通用二进制包 | 安装比较复杂，灵活性高，平台通用性好 |
| 源码包 | 安装最复杂，时间长，参数设置灵活，性能好 |

* 目前最新版本为8.0.18，发布时间2019.10.14；8.0.0在2016.9.12日就发布了。
* 本课程中使用5.7版本。选择下图中⑤来到以前版本下载页面。

计算机生成了可选文字:
令O
白
0MySQLCommunityDOWnloadS
MysQLc。mmunityserve币5.0.15户
最新发布版：5．了之后版本名称改为8.0
OokingforprevI0uSGA
el一510nSP
ReC0mmendedDOWnload:
．口
硬那
L.．色口尸
州「
困．
MySQ
LInstaller
forWindowS
A11MySQLProducts.ForAllWindowsPlatforms.
InOnePackage.
Windows(x86,32&64一bit),MySQLInstallerM写1
O
OtherDOWnloddS:
wtndows系统下的安装程序：可执行程序扩展名MSI程序
W1ndows(x86,64·bit),ZIPArchive
8.0.18
272.3M
'm外ql一8.0.18一Winx64.zip)
MDS:3clfc0bc3368639的68几e弓b印afa23d}Signa'ule
W1ndows(x86,64·bit),ZIPArchive
DebugBinaries＆飞stSuite
(mysql·8.0.18一wlnx64一debug一test.zip)
8.0.18
402.6M
MDS:8d56a0f2418的6598496b4lldc29d3石9}Slgnature
Ow。Suggestt。。ty。日。5。t卜。M。5。。e〔ksum、。nd。n。p。、．gn。t。re、t。、er．句t。。．ntegr．砂。ftoe
paCkagesyoudownload.
.

### 下载MySQL Community Server 5.7.28

1. 这里不能直接选择CentOS7系统的版本，所以选择与之对应的Red Hat Enterprise Linux 7
2. 直接点Download下载RPM Bundle全量包。包括了所有下面的组件。不需要一个一个下载了。

计算机生成了可选文字:
令C
O
0MySQLCommunityDOWnloadS
《
GeneralAvailability(GA)Releases
妙
MySQLCommunityServer5.7.28
Sele改VeFSion:
Lookingfor
Version?
Selectoper“二］’·9System:
陀笋．笋资甲手任二于＝=．严．钾
匕KeaHatLnterP卫互旦卫匹些
口口口口口口口口口口口口口口口目
Se}e改05VersiDn:
/orac!eLinux
RedHatEnterprlseLlnux71oracleLlnux7(x86,64一bit),
InstallUsingYum:
MySQLYUmRepository
SupportedPlatforms:
RedHatEnterpriseLinux/oracleLinux
O,.
卜、／/
Fedora
DOWnlo日dNOW"
DOWnloadPaCka它es
一d二，}I全量包｛
(mysql一5.7.28一1.el7.x86es64.rpm一bundle.tar)
5.7.28
[581.3MI…I芯石忑刁‘
MDS:bg。105。6757b63。3占7aes。a7o扔flo7}Slgnature
RPMPackage,MySQLServer
(mysql{ommunity-,erver一5.7.28一1.el7.x86一4.rpmJ
5.7.28
199.0M
DO目们！O月d
MDS:0734b几c2306拓e7的2284ce74240Bfe
RPMPackage,ClientUtilities
(mysql{ommuni诊cllent一5.7.28一1.eI7.x86-64.rpm)
5.7.28
43.0M
MDS:sba58a52dae094bsfe55c18bZcld6f4c
RPMPackage,DevelopmentLibraries
(mysql{ommunity-devel一5·7.28一1.eI7·x86--64.rpm)
5.7.28
6.7M
MDS:oc石cds七02baDga5742f6lcb6fbfdc3aZ
RPMPackage,DevelopmentLibrarjes
(mysql(ommunity-embedded一devel一5.7.28一1.eI7.x86_64.rpm)
5.7.28
130.5M
MDS:ddclZ10b7lfD故dfsafb38doa7b706cf
二J二百‘．心t．才．1口。卜‘，山月二口口口叮．习．1't．」二诊二1‘『．勺．
面口卜月
】r曰口品甩．刀飞三自J勺

### 不需要输入账号登录

* 直接点：No thanks,just start my download.

计算机生成了可选文字:
令C白‘dev.mysql.com/downloads/file/?id=489764
0MySQLCommunityD0wnloadS
LoginNoworSignUpforafreeaccOUnt.
AnoracleWebAccountprovldesyouWiththefollowlngadvantages:
·FastacceSSt0M外QLSoftwaredownloadS
·DownloadtechnlcalWhltePaPerSandPresentationS
·POStmeSSageSintheMySQLDIScuSS10nForumS
·ReportandtrackbugsIntheMySQLbugSystem
L
ogln,,
.1.，冲
usingmyOr.cl.w.b.ccount
for.no阴仁leWeb．仁仁OUnt
MySQL·com15usingoracle550forauthenticatlon·IfyoualreadyhaVeanoracleweb
account,clicktheLoglnIink.othe阴ISe,youcanSlgnupforafreeaccountbycllckingthe
5IgnUPIinkandfollowingtheInstructionS·
Nothanks,juststartmydownload.

### 下载的tar包，用压缩工具打开

计算机生成了可选文字:
荃mysql一5.7.28一1.eI7.x86_64.rpm一bundle.tar

* 解压后rpm安装包

计算机生成了可选文字:
巨回竺沸亚宁邓酬亚吵酬讨
、O
名称
兀．月
mysql一community一client一5.7.28一1.eI7.x86_64.rpm
mysql一community一common一5.7.28一1.eI7·x86_64.rPm
mysql一community一devel一5.7.28一1.eI7·x86_64.rpm
mysql一community一embedded一5.7.28一1.el7.x86_64.rPm
mysql一community一embedded一compat一5.7.28一1.el7.x86_64,rpm
mysql一community一embedded一devel一5.7.28一1.el7.x86_64.rpm
而石裘濡而蕊雨万万万燕燕砚砰币而1
一吠｝衬｝
品mysql一community一libs一comPat一5.7.28一1.eI7.x86--64.rPm
mysql一community一server一5.7.28一1.eI7.x86_64.rpm
品mysql一community一test一5.7.28一1.eI7.x86_64.rpm

## CentOS7系统，检查mysql依赖环境

### 查看是否安装过MySQL

rpm -qa | grep -i mariadb CentOS7使用这个

-i忽略大小写

计算机生成了可选文字:
[root④zyopt]#rpm一qa}grepmariadb
mariadb一libs一5.5.56一2.e17.x8664

卸载MySQL; --nodeps 排除依赖，否则别的软件依赖mysql,无法删除

rpm -e --nodeps mariadb-libs

计算机生成了可选文字:
root@zyopt]#rpm一e一nodepsmariadb一libs

### 检查/tmp临时目权限；

chmod -R 777 /tmp

计算机生成了可选文字:
dr一XF一Xr一X.13FOOtFO0to
dFWXFWXFWt.35FOOtFO0t4096
dFWXF一XF一X.13rOOtFOOt155
drWXr一Xr一X.21rOOtFOOt4096
3116:57
3117:26
15:39
15:52
USF
VaF
月月月月
12
12
12
12
户匕产O

### 执行安装命令前，先执行查询命令检查依赖

rpm -qa|grep libaio

* 如果存在libaio包如下：

计算机生成了可选文字:
1r00t坦Zhangyumysql一5.7.28一1.e17.X86_64.rpm一bundle]#rpm一qa}greplibaio
tibaio一0·3·109一13·e17·x86一64口口口口口口口口口口口口口口口口

rpm -qa|grep net-tools

* 如果存在net-tools包如下：

计算机生成了可选文字:


* 如果不存在需要到centos安装盘里进行rpm安装。安装linux如果带图形化界面，这些都是安装好的。

## CentOS7系统，安装MySQL 5.7.28

### 将安装程序拷贝到/opt目录下

* 在mysql的安装文件目录下执行：（必须按照顺序执行）

rpm -ivh mysql-community-common-5.7.28-1.el7.x86\_64.rpm

rpm -ivh mysql-community-libs-5.7.28-1.el7.x86\_64.rpm

rpm -ivh mysql-community-client-5.7.28-1.el7.x86\_64.rpm

rpm -ivh mysql-community-server-5.7.28-1.el7.x86\_64.rpm

* rpm 是Redhat Package Manage缩写，通过RPM的管理，用户可以把源代码包装成以rpm为扩展名的文件形式，易于安装。
* -i, --install 安装软件包
* -v, --verbose 提供更多的详细信息输出
* -h, --hash 软件包安装的时候列出哈希标记 (和 -v 一起使用效果更好)，展示进度条

计算机生成了可选文字:
1root坦zhangyu门＃cd/opt/mysql一5.7.28一1.e17.X86_64.rpm一bundle
1r00t@zhangyumysql一5.7.28一1.e17.x86_64.rpm一bundle]#11
总用量595288
一rw一F一F－一1F00t
一rw一F一r－一1F00t
一rw一F一r－一1r00t
一rw一F一F－一1F00t
一rw一r一r－一1r00t
一rw一F一F－一1F00t
一rw一F一F－一1F00t
一rw一r一r－一1r00t
一rw一F一F－一1F00t
一r树一F一F一。1F00t
F00t45109364
F00t318768
r00t7637096
F00t49329100
r00t23354908
F00t136837816
r00t4374364
r00t1353312
F00t268694824
F00t133129992
16
10
16
16
10
16
10
10
16
18
月2723:56mysqt一co。。unity一ctient一5.7.28一1.e17.x86_64.rp。训
月2723:56oysql一co。。unity一co。。on一5.7.28一1.et7.x86_64.rp．、Z
月2723:56ovsql一coo.unitv一devel一5.7.28一1.e17.x86_64.rp.
月2723:56oysql一coo.unity一e.bedded一5.7.28一1.e17.x86_64.rp.
月2723:56ovsql一comounitv一eobedded一coopat一5.7.28一1.e17.x86_64.rp.
月2723:56oysqt一comounity一eobedded一devel一5.7.28一1.et7.x86_64.rp。
月2723:56oysql一community一libs一5.7.28一1.e17.x86_64.rp.,/
月2723:56ovsql一co。。unity一libs一coopat一5.7.28一1.e17.x86_64.rp.
月2723:57mysqt一coomunity一server一5.7.28一1.e17.x86_64.rp。、尸口
月2723:57oysql一co。。unity一test一5.7.28一1.et7.x86_64.rp.

### 安装四个安装包

计算机生成了可选文字:
.―卫―一―-―川―--
00t@zhangyumysql一5·7·28一1·e17·X86_64·rpm一bundle]#
告：mysql一community一common一5.7.28一1.e17.x86_64.rpm:
rpm一ivhmysql一communlty一common一5.7.28一1.e17.x86_64.rpm
头V3DSA/SHAISignature，密钥ID5072elf5:NOKEY
子警
准备中…幸幸幸券执洲月毋角洲翔毋幸本幸幸热胸翔烟毋幸幸幸孝券券幸幸幸【100%]
正在升级／安装…
1·m、，‘门1一广nmmlln布十、，一r八叨即nn一只7，只一10五业登企丑业业业女世业业双丑业世世世业五五五业世世业丑五五业丑丑业11nn头1
Lr00t@zhangyumysql一5.7.28一1.e17.X86_64.rpm一bundlel#rpm一Ivhmysql一community一11bs一5.7.28一1.el7.x86_64.rpm
警告：mysql一community一libs一5.7.28一1.e17.x86_64.rpm：头V3DSA/SHAISignature，密钥ID5072elf5:NOKEY
准备中…鹅洲月毋举们目毋幸们翔吩幸们目峭毋鹅脚翔【100%]
正在升级／安装…
1:mvsal一communitv一libs一5.7.28一1.e17券举们翔麟毋角州翔博幸角洲目I＃幸1106%l
1r00t@zhangyumysql一5·7·28一1·e17·x86_64·rpm一bundlel#rpm一ivhmysql一community一client一5.7.28一1.e17.x86_64.rpm
警告：mysql一community一client一5.7.28一1.e17.x86_64.rpm：头V3DSA/SHAISignature，密钥ID5072elf5:NOKEY
准备中…拍月用撇们材毋幸们娜毋豹黝博幸们【1叨％]
正在升级／安装…
1:mvsal一communitv一client一5.7028一1.e#####＃娜券钩哪峨＃##＃弃渊＃##########＃幸1100%l_
[root@zhangyumysql一5·7·28一1·e17·X86_64·rpm一bundlel#rpm一Ivhmysql一communlty一Server一5·7·28一1·e17·x86_64·rpm
警告：mysql一community一server一5.7.28一1.e17.x86_64.rpm：头V3DSA/SHAISignature，密钥ID5672elf5:NOKEY
准备中…拍材毋们翔准泪翔器们博券幸们黝峭【100%]
正在升级／安装…
1:"Iysql一community一server一5.7.28一1.e欺们料峭毋拱脚翔博＃粼脚翔毋＃角1100%l

### 查看mysql版本

* 执行 mysqladmin --version命令，类似java -version如果打出消息，即为成功

计算机生成了可选文字:
1root坦zhangyumysql一5.7.28一1.e17.
mysqladminVer8.42Distrib5.7.
64.rpm一bundle]#mysqladmin一version
forLinuxonx8664口口口口口口口口口口口

* 执行rpm -qa|grep -i mysql命令，查看是否安装成功，需要增加 -i 不用去区分大小写，否则搜索不到。

计算机生成了可选文字:
1r00t坦zhangyUmysql一5·7·28一1·e17·X86_64·rpm一bundle]#rpm一qa
.ysql一community一libs一5·7·28一1.e17.x86_64
'ysql一community一Server一5.7.28一1.e17.x86_64
.ysql一community一common一5·7·28一1.e17.x86_64
.ysql一community一cliont一5·7·28一1.017.x86_64
9rep一1mysql

### 服务的初始化

* 为了保证数据库目录为与文件的所有者为 mysql 登录用户，如果你是以 root 身份运行 mysql 服务，需要执行下面的命令初始化：
* mysqld --initialize --user=mysql
* 另外 --initialize 选项默认以“安全”模式来初始化，则会为 root 用户生成一个密码并将该密码标记为过期，登录后你需要设置一个新的密码
* 查看密码：cat /var/log/mysqld.log

计算机生成了可选文字:
[r00t@zhangyumysql一5.7.28一1.e17.x86_64.rpm一bundlel#cat/Var/109/mysqld.10g
2619一10一27T16:40:36.93688920[Warning]TIMESTAHPwithimplicitDEFAULTvalue15deprecated.Pleaseuse
lts_for--timestampserveroption(Seedocumentationformoredetails).
2619一10一27T16:40:37.49561120[Warning]InnoDB:New109filescreated,LSN=45796
2619一10一27T16:40:37.59578220[Warning]InnoDB:Creatingforeignkeyconstraintsystemtables.
2619一10一27T16:40:37.69072426[Warning]NoexistingUUIDhasbeenfound,50weassumethatthis15the
hisserverhasbeenstarted.GeneratinganewUUID:8144d4ff一fsds一11e9一b96f一006c29756eb0.
2019一10一27T16:40:37.69871920[Warning]Gtidtable15notreadytobeused.Table'mysql.gtid_executed'
2019一10一27T16:40:38.56540620[Warning]CAcertificateca.pem15selfsigned.
2619一10一27T16:40:38.77444521[Note]Atemporarypassword15generatedforroot@localhost:gspfsagsvk*A
一explicitdefau
fiFSttimethstt
cannotbeopened

* root@localhost: 后面就是初始化的密码

### 启动mysql，并查看状态(加不加.service后缀都可以)

* 启动：systemctl start mysqld.service
* 关闭：systemctl stop mysqld.service
* 重启：systemctl restart mysqld.service
* 查看状态：systemctl status mysqld.service

计算机生成了可选文字:
[rootozhangyumysql一5.7.28一1.e17.x86_64.rpm一bundle]#systemctlstartmysqld.service
[r00t@zhangyumysql一5·7·28一1·e17·x86_64·rpm一bundle]#SystemctlStatuSmysqld
.mysqld.service一HySQLServer
Loaded:loaded(/usr/lib/systemd/system/mysqld.service;enabled;vendorpreset:disabled)
Active:active(running)since一2619一10一2890:45:39CST;495ago
Docs:man:mysqld(8)
http://dev.mysql.com/doc/refman/en/using一systemd.html
process:7700Execstart=/usr/sbin/mysqld一daemonize一pid一file=/var/run/mysqld/mysqld.pid＄刚SQLD_OPTS(code=exited,
status=0/SUCCESS)
ProceSS:7665EXecstartPre=/usr/bin/mysqld_pre_Systemd(code=eXited,StatuS=0/SUCCESS)
HainPID:7703(mysqld)
CGroup:/system.slice/mysqld.service
L--77e3/usr/sbin/mysqld一daemonize一pid一file=/var/run/m丫sqld/mysqld.pid
2800:45:23zhangyu.atguigusystemdlll:
2800:45:39zhangyu.atguigusystemdll]:
StartingHySQLServer…
StartedHySQLServer.
月月
10
10

* 查看进程：ps -ef | grep -i mysql

计算机生成了可选文字:
[root@zhangyumysql一5.7.28一1.e17.X86_64.rpm一bundlel#ps一ef}grep一1mysql
.ysql77031000:45?00:00:02/usr/sbin／。ysqtd一daemonize一pid一file=/var/run／。ysqld／。ysqld.pid
root7883119115000:59pts/000:60:00可蔽而一uroot一pxxx
root86887994001:11pts/100:60:00grep一color=auto一1.ysql

### 查看mysql服务是否自启动（默认自启动）

systemctl list-unit-files|grep mysqld.service

计算机生成了可选文字:
[r00t坦zhangyumysql一5·7·28一1·e17·x86_64·rpm一bundle]#Systemctllist一unit一files
.ysqld.serviceenabled
9repmysqld.servic

* 如不是enabled可以运行如下命令设置自启动

systemctl enable mysqld.service

* 如果希望不进行自启动，运行如下命令设置

systemctl disable mysqld.service

计算机生成了可选文字:
[root@zhangyumysql一5.7.28一1.e17.x86_64.rpm一bundle]#systemctldisablemysqld.service
Removedsymlink/etc/systemd/system/multi一user.target.wants/mysqld.service.
[root@zhangyumysql一5.7.28一1.e17.X86_64.rpm一bundle]#Systemctlenablemysqld·Service
Createdsymlinkfrom/etc/systemd/system/multi一user.target.wants/mysqld.serviceto/usr/lib/systemd/system/mysqld.service.

### 首次登录

* 通过 mysql -uroot -p进行登录，在Enter password：录入初始化密码

计算机生成了可选文字:
[root@zhangyumysql一5.7.28一1.e17.X86_64.rpm一bundle]#mysql一ur00t一p
Enterpassword:
WelcometotheHySQLmonitor.Commandsendwith:or\g。
YourHySQLconnectionid152
Servervorsion:5.7.28
Copyright(c)2000,2019,oracleand/oritsaffiliates.Allrightsreserved.
OF日Cle15
affiliates
OWners。
aregisteredtrademarkoforacleCorporationand/orits
.othernamesmaybetrademarksoftheirrespective
Type'help;
mysql＞口
0F
\h'forhelp.Type'\c'toclearthecurrentinputstatement.

### 修改密码

* 因为初始化密码默认是过期的，所以查看数据库会报错
* 修改密码：ALTER USER 'root'@'localhost' IDENTIFIED BY 'new\_password'; 但新密码太简单会报错

计算机生成了可选文字:
mysql>
ERROR
ALTERUSER'root'@'localhost'IDENTIFIED
1819(HY980):Yourpassworddoesnotsatis
,123
the
currentpolicyrequirements
Yy
Bf

* 设置完密码就可以用新密码登录，正常使用数据库了

ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';

计算机生成了可选文字:
mysql>
ouery
81ter
USer
FOWS
'root'O'localhost'identifiedby
r00t
OK,0
affected(0.00sec)

* 使用quit退出，重新登录

计算机生成了可选文字:
[rootozhangyumysql一5.7.28一1.e17.
mysql:
We1C0me
[WarninglUsingapassword
X86
On
mysql一uroot一proot
interfaCe
\9.
CanbeinseCUre。
totheMySQLmonitor.Commands
rpm一bundle]#
Commandline
endwith;or
YourHySQLconnectionid153
SerVerVersion:
5.7.28HySQLCommunityServer(GPL)
Copyright(c)2000,2019,oracleand/oritsaffiliates.Allrightsreserved.
OF日Cle15
affiliates
OWners。
aregisteredtrademarkoforacleCorporationand/orits
.othernamesmaybetrademarksofthelrrespective
Type"help:
or"\h
forhelp.Type"\c'toclearthecurrentinputstatement。
mysql>showdatabases;
＋一－-－一一一一一一一一一一一一一一一一＋
Dstab8Se
＋一一一一－一一一一一一一一一一一一一一一＋
1information_Schema}
1mysql}
1performance_Schemal
1Sysl
＋一～~~~~~~～一～~~～一一～～一一＋
4rowsinset(0.00sec)
mysql>.

### 安装后目录结构

|  |  |  |  |
| --- | --- | --- | --- |
| 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/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 |
|  | /etc/systemd/system/multi-user.target.wants/mysqld.service | 服务启停相关脚本 |  |
|  | **/etc/my.cnf** | mysql配置文件 | cat /etc/my.cnf |

### 字符集问题

* 默认latin1字符编码，不支持中文

show variables like '%char%';

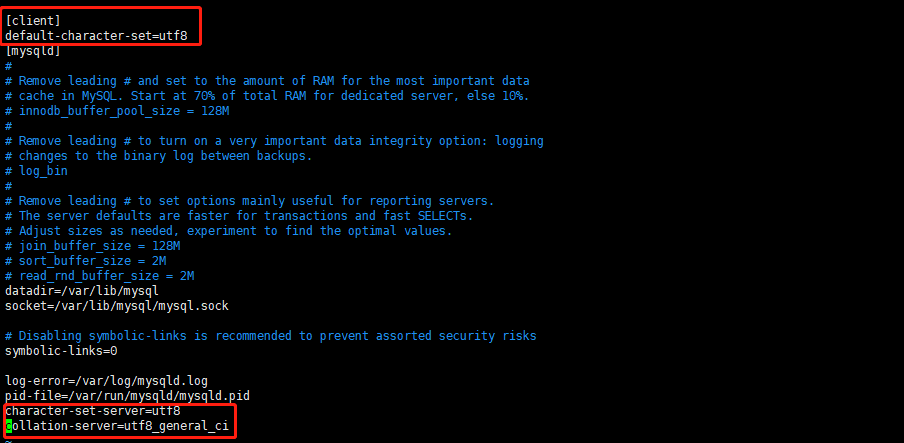
* 保存中文报错

计算机生成了可选文字:
mysql>showdatabases;
＋一一一一一一一一一一一一一一一一一一一一＋
Datab8Se
＋一一一一一一～一一一一一～一一一一一一一＋
＋一
4
informationschema
mysql
performance_Schema
SyS
rowsinset(0.01
一＋
SeC
mysql>
Query
createdatabasemydb;
O
rowaffected(0.00sec
mysql>
OK,1
USe
mydb
Databasechanged
O
mysql命令后面可以不用增加分号
mysql>
Query
createtablemytbl(id
OK,0rowsaffected(0
int,namevarchar(8))
.03sec)
mysql>insertinto
ERROR1366(HY006)
mytbl(id,name)values(1,‘张三’);
:Incorrectstringvalue:'\xES\xBC\XAO\xE4\xBS\x89'forcolumn
nallle'atr0WI

1. 修改字符集

* vim /etc/my.cnf

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



1. 重新启动mysql

systemctl restart mysqld

1. 已生成的库表字符集如何变更

* 以前创建的库，创建的表字符集还是latin1

计算机生成了可选文字:
mysql>showcreatetablemytbl;
一－一一＋
1Table1CreateTable
}
＋一－---－一＋一－----－一一－---
~～一～+
1mytbl
、id、
}cREATETABLE、mytbl、
int(11)DEFAULTNULL,
、name、varchar(8)DEFAULTNULL
)ENGINE=InnoDBDEFAULTCHARSET=latinll
+～一～~~~~＋一一一～~~~~~~~~~~~～一～~~~~~~--－一二，,～一～~~~
~~~~+
1FO树
inset(0.00sec

* 修改以前数据库的字符集

alter database mydb character set 'utf8';

* 修改以前数据表的字符集

alter table mytbl convert to character set 'utf8';

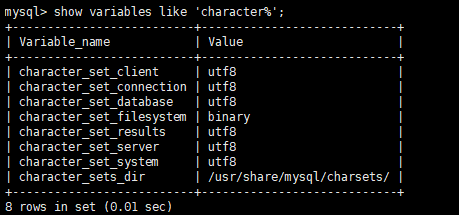
计算机生成了可选文字:
mysql>alterdatabasemydbcharacterset'utfs
QueryOK,1rowaffected(0.00sec)
mysql>altertablemytbl
ConVeFtto
(6.06sec
Warnings
Ch8FaCteFSet'Utfs
QueryOK,0
Records:6
r0WSaffeCted
Duplicates:0
0
mysql>insertintomytbl(id,name)values(1,”张三
QueryOK,1rowaffected(0.61sec)
mysql>5elect*frommytbl;
＋一一一一一一＋一一一一一一一一＋
1d
name
+----－一＋------－一＋
11张三
+~~~~~~＋二，,,,,，巴，,,~+
1rowinset(0.01sec
mysql>.

* 但是原有的数据如果是用非'utf8'编码的话，数据本身编码不会发生改变。
* 已有数据需要导出或删除，然后重新插入。

1. 查看字符集

show variables like 'character%';

show variables like '%char%';



# MySQL架构\_用户与权限管理【了解】

## MySQL的用户管理

### 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 \* from mysql.user \G; 以列的方式显示数据

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

查询特定字段

计算机生成了可选文字:
mysql>selecthost,user,authentication声tring,Select一rivfrom
USeF
hOSt
USer
1authentication一tring
卜～~~--~~~~~～一＋
1Selectjrivl
}localhost1rootl*E56All4692FEDDE073FgA五幻68月关班EB9703F3FI!Y
1localhost1mysql.sys!’一IslsNOTAvALIDPAsg"ORDTHATc月犯〔USEDHERE!N
}%!zhang3!*E56All4692FE0OE073F9AIDD68A0贬EB9703F3FI!N
}%1rootl*E56All4692FE0DE073F9AIDD68A0关EB9703F3FIIy
4ro树S
inset(0.01sec)

* 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 xxx 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。
* authentication\_string ： 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')

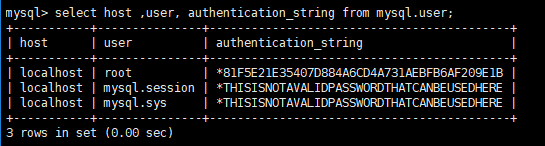
* 修改某个用户的密码:
* 必须使用root 用户: 权威

update mysql.user set authentication\_string=password('111111') where user='zhang3';

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

### 删除用户

drop user zhang3 ;  **推荐使用！**



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

Systemctl start|status|stop|enable|disable firewalld

Firewall -cmd –add-port=3306/tcp –permanent

* 重新加载

Firewall -cmd --reload

* 查看开放的端口号

Firewall -cmd –list-all

## 权限管理

### 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,update on atguigudb.\* to li4@localhost identified by ‘123’; # 必须有li4 用户才行！

* + #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 joe@'%' identified by '123';

如果没有joe这个用户 会自动创建这个用户

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

### 收回权限

* 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 \*.\* FROM joe@'%'; #收回全库全表的所有权限

REVOKE select,insert,update,delete ON \*.\* FROM joe@'%';

#收回mysql库下的所有表的插删改查权限,如果是test表的所有权限 test.\*

### 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 mysql.user ;

## 通过工具远程访问

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

### 先 ping 一下数据库服务器的ip地址确认网络畅通。

# ping 192.168.200.168 -t

### 关闭数据库服务的防火墙

systemctl start firewalld.service

systemctl status firewalld.service

systemctl stop firewalld.service

systemctl enable firewalld.service

systemctl disable firewalld.service

### 或者不关防火墙，而是，开发3306端口[了解]

查看开放的端口号

firewall-cmd --list-all

设置开放的端口号 生成环境！

firewall-cmd --add-service=http --permanent

firewall-cmd --add-port=80/tcp --permanent

firewall-cmd --add-port=3306/tcp --permanent

重启防火墙

firewall-cmd --reload

### 确认MySQL中已经有可以通过远程登录的账户

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

### 如果没有用户,先执行如下命令

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

### 测试连接



# MySQL架构\_杂项配置sql\_mode

## MySQL的sql\_mode合理设置

会话session:分为局部会话和全局变量。局部变量只对当前窗口起作用。全局变量对所有的窗口都起作用。

局部变量： show variables like ‘%变量名%’ set 变量=值

show session valuables like ‘%变量名%’

set session 变量=值

全局变量： show global variables like ‘%变量名%’

set global 变量 = 值

sql\_mode设置是否允许一些非法操作，比如允许一些非法数据的插入,数据查询等。

在生产环境必须将这个值设置为**严格模式**，所以开发、测试、生产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。

select @@GLOBAL.sql\_mode;全局的sql\_mode

select @@SESSION.sql\_mode;会话级别sql\_mode

计算机生成了可选文字:
mysql>SELECT阅GLOBAL.sql_mode;
＋一～~~~～一一一一～一～一～~～一一一一～～一一一一一一－·
一令
阅GLOBAL.Sql_mode
ONLYFULLGROUPBY,STRICTTRANSTABLES,NOZEROINDATE,NOZERODATE,ERRORFORDIVISIONBYZERO,NOAUTOCREATEUSER,NOENGINESUBSTITUTION
1rowinset(0.60sec)
mysql>SELECT阅SESSION.Sql_mode;
＋一～~~～一一一一一一一一一一一一一一一一一一一一一一一一一一－
阅SESSION.Sql_mode
＋一～~~~~~~~
~～一＋
}
一一～+
ONLYFULLGROUpBY,STRICTTRANSTABLES,NOZEROINDATE,NOZERODATE,ERRORFORDIVISIONBYZERO,NOAUTOCREATEUSER,NOENGINESUBSTITUTION
＋一－－一一一一－---－一一一一－--－一一一一－-－一
1rowinset(0.00sec)
一一＋

补充：

show variables like 'max\_connections'; 对应的修改： set xxx=1

show session variables like 'max\_connections'; 对应的修改： set xxx=1

show global variables like 'max\_connections'; 对应的修改： set global xxx=1

Session只对当前窗口生效,但是global对于所有的session都生效。

虽然设置了global变量、session变量，但是在mysql服务重启之后，数据库的配置又会重新初始化，一切按照my.ini的配置进行初始化，所以永久修改需要在: my.cnf|my.ini文件中修改。

## 示例：错误开发演示

|  |
| --- |
| CREATE TABLE mytbl2(id INT,NAME VARCHAR(16),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);  #查询每个部门年龄最大的人  正确的语句：  SELECT m.\* FROM mytbl2 m INNER JOIN(SELECT dept,MAX(age) maxage FROM mytbl2 GROUP BY dept) AS ab  ON ab.dept=m.dept AND m.age=ab.maxage  计算机生成了可选文字: 』曰a w队 mysql> ERROR iCh15 SELECTNAHE,dept,MAX(age) FROHmytb12 ofSELECT onC01UmnS GROUPBYdept不 1055(42000):Expression#1 notfunctionallydependent list15not inGROUPBY inGROUpBYclauseandcontainsnonaggregatedcolumn'mydb.mytb12.name clause; mysql>SELECTm.*FROHmytb12mINNERJOIN(SELECTdept,HAX(age) ge=ab.maxage; +~~~~~~+-~~~~~~+~~~~~~+~~~~~~+ }id1name1age}dept} +----－一＋一－---－一＋-－一一－一＋----－一＋ this m日Xage 15incompatiblewithsql_mode=only_full_group_by FROHmytb12GROUPBYdept)ASabUNab.dept=m.deptAND 114 t18n7 34 36 101 102 『乙一b ＋一一一－－一＋一一一一－－一＋一一一－－一＋－一一一一一＋ 2rowsinset(0.00sec)  注意：   * MySQL5.7默认已经设置了ONLY\_FULL\_GROUP\_BY。 |

## 设置sql\_mode

* 设置当前窗口中设置sql\_mode

SET GLOBAL sql\_mode = 'modes...';

SET SESSION sql\_mode = 'modes...';

* 在/etc/my.cnf中配置sql\_mode,永久生效。

|  |
| --- |
| [mysqld]  #set the SQL mode to strict  #sql-mode="modes..."  sql-mode = "**ONLY\_FULL\_GROUP\_BY**,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION" |

show variables like 'sql\_mode';

计算机生成了可选文字:
mysql>5howvariableslike'sql_mode';
VSrlablename
Va1Ue
＋一一～-－一一－一－-－一一一＋一一－-－一一一－-~~--~---~~~
---－一一干
l
..．·一一＋
sqlmode
ONLYFULLGROUPBY,STRTCTTRANSTABLES，付0ZERDINDAT仁，NOZ【RODAT仁，【RRORFORDIVISIONBYZ【RO,NOAUTOCRFATFUgER,NOENGIN【SUBS111UTION
+~～一～~-～一～~~~~~~+
~~～千
1r0w
set(6.00sec)
.1n

* 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，那么去掉该选项。
* STRICT\_ALL\_TABLES,STRICT\_TRANS\_TABLES

对于支持事务的表，这两种模式是一样的：如果发现某个值缺失或非法，MySQL将抛出错误，语句会停止运行并回滚。

对于不支持事务的表，这两种模式的效果：

1、如果在插入或修第一个数据行时就发现某个值非法或缺失，那该语句直接抛错，语句停止执行。这个和支持事务的数据表行为时一样的。

2、如果在插入或修改第n个（n>1)数据行时才发现错误，那就会出现下面的情况：

2.1 在STRICT\_ALL\_TABLES模式下，停止语句执行，存在部分更新的问题

2.2 在STRICT\_TRANS\_TABLES模式下，MySQL将继续执行该语句避免“部分更新问题”，对每个非法值将其转换为最接近的合法值。

* 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不存在，那么抛出错误。不设置此值时，用默认的存储引擎替代

案例1说明：

CREATE TABLE test(

id int(11) NOT NULL,

col1 int(11) DEFAULT NULL,

PRIMARY KEY (id)

);

insert into test values (1,'s'); #会发现当我们不设置该项的时候，字符串s也能插入进去

set sql\_mode='STRICT\_TRANS\_TABLES'; #开启严格模式,再次执行插入操作时，就会报错

案例2:

NO\_ZERO\_IN\_DATE: 在STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE模式下，只要日期的月和日中含有0值都报错，但是‘0000-00-00’除外:

NO\_ZERO\_DATE：1000-00-00 0000-01-00 0000-00-01（年月日中任何一个不为0）插入表，年月日全为0报错

演示:

set session sql\_mode='STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE';

create table t1(id int,age date);

insert into t1 values(1,'0000-00-00');

insert into t1 values(1,'1000-00-10');

set session sql\_mode='STRICT\_TRANS\_TABLES,**ERROR\_FOR\_DIVISION\_BY\_ZERO**';

set session sql\_mode='STRICT\_TRANS\_TABLES,**ERROR\_FOR\_DIVISION\_BY\_ZERO**';

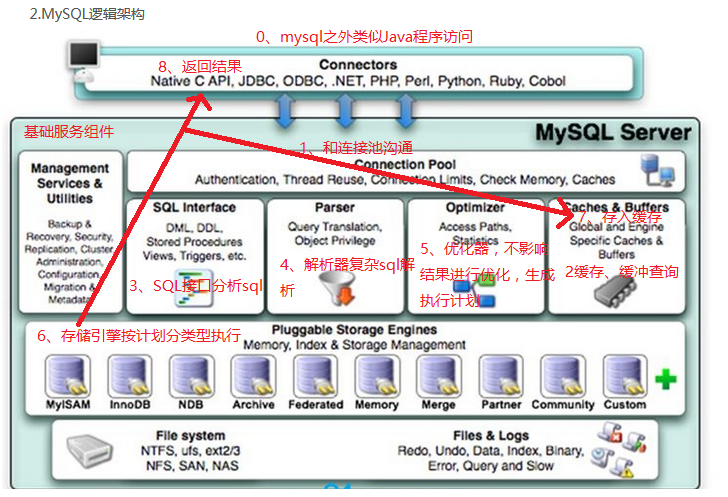
**insert into t1 values(1/0,'1990-12-12');**

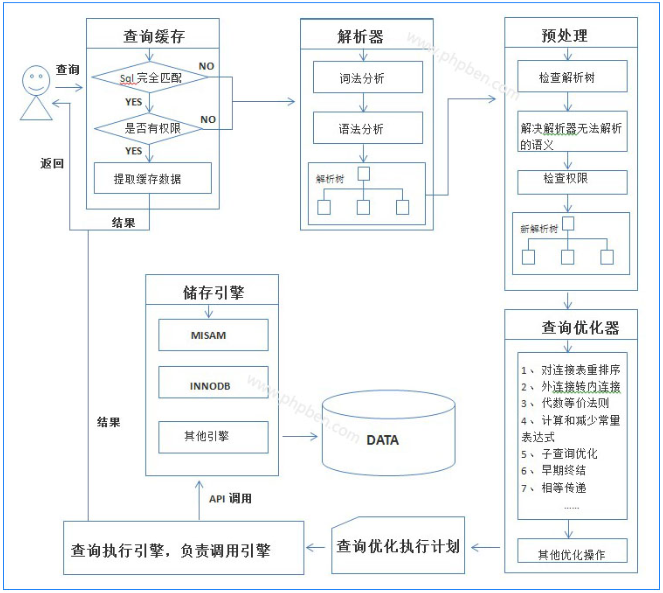
# MySQL架构\_逻辑架构

## 逻辑架构总体概览

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socket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

第二层架构主要完成大多数的核心服务功能，如SQL接口，并完成缓存的查询，SQL的分析和优化及部分内置函数的执行。

所有跨存储引擎的功能也在这一层实现，如过程、函数等。

在该层，服务器会解析查询并创建相应的内部解析树，并对其完成相应的优化：如确定查询表的顺序，是否利用索引等，最后生成相应的执行操作。

如果是select语句，服务器还会查询内部的缓存。如果缓存空间足够大，这样在解决大量读操作的环境中能够很好的提升系统的性能。

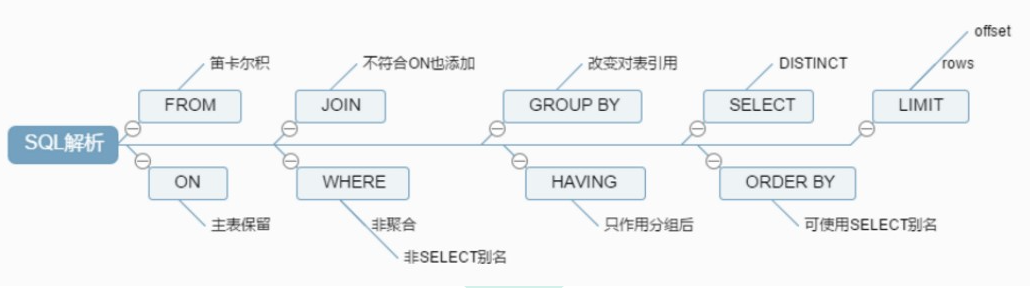
2.1 Management Serveices & Utilities： 系统管理和控制工具

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。

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。

## 各部分说明

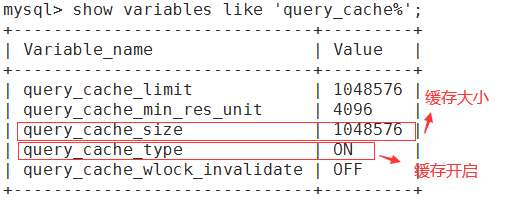
### 5.2.1连接器(Connectors)

连接器负责跟客户端建立连接，获取权限、维持和管理连接以及授权认证和安全等等。

* 用户名密码验证
* 查询权限信息,分配对一个的权限
* 可以使用show processlist查看当期存在的连接
* 如果太长时间没有动静，就好自动断开，通过wait\_timeout控制，默认8小时

### 5.2.2 缓存是否开启(cache)

show variables like 'query\_cache%';



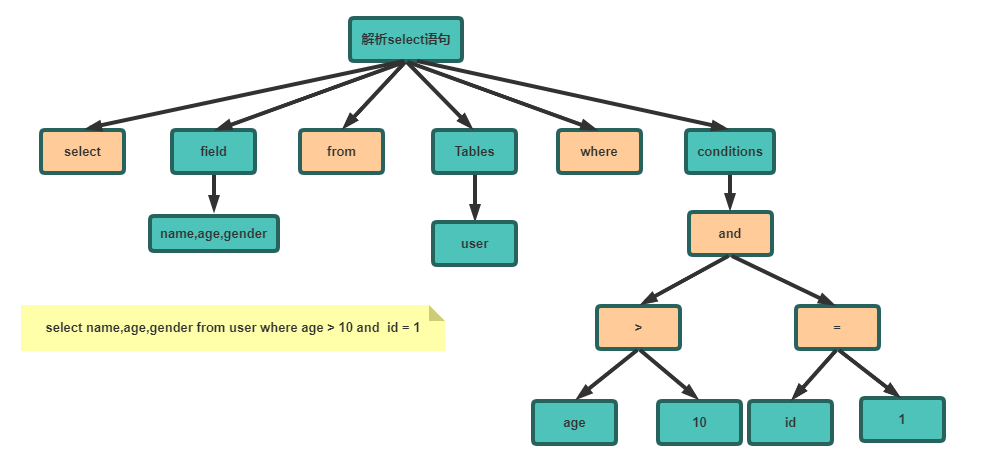
注意：sql语句要想使用缓存，要求：sql语句一模一样，一个空格也不允许多。

### 5.2.3 解析器（parser）

在解析器中，主要做词法解析和语法解析

词法解析：将传递过来的SQL解析成一个一个的词

语法解析：按着一定规则将SQL解析为一颗树桩结构，解析树



数据库中间件肯定也是有语法解析和词法解析功能的。

### 5.2.4预处理器(Preprocessor)

预处理也是在解析器部分完成,在解析器解析完毕SQL语句之后，还需要经过预处理器Preprocessor进行检查，比如表名是否存在: Select \* from table\_name(表名有可能是错的),是否有这个可执行权限(授权语句),最终得到一颗新的SQL解析树

### 5.2.5 优化器(optimizer)

当优化器拿到预处理器处理过的SQL解析树之后，SQL执行有可能有多种执行路径，优化器就是在不改变查询结果的前提下，对SQL语句进行优化，生成SQL的执行计划[Excution Plan],然后选择开销最小的方式执行SQL,达到较好的执行效果。

通俗的讲就是：就是当有多个表进行inner join连接的时候，优化器会决定谁作为驱动表，谁作为被驱动表。或者当一个表中出现多个索引的情况下，由优化器决定到底底层采用哪个索引被使用。

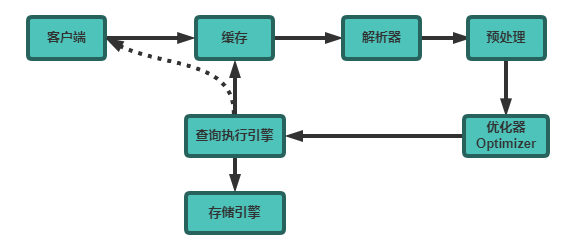
当然优化器不是万能的，如果我们写的SQL过于垃圾，优化器也救不了你.

注意：这里我们要明白，我们得到的这个执行计划，不一定是最优的执行计划,是因为它在生成执行计划的时候，也有可能覆盖不了所有的执行路径。那么我们怎么看mysql到底采用的是哪个执行计划呢？我们可以使用explain + SQL语句 查看最终采用了哪个执行计划。

### 5.2.6 查询执行引擎

注意：是**查询执行引擎**用优化器生成的执行计划，**去操作存储引擎,之所以我们的表随时可以换引擎**，就是因为**查询执行引擎**提供了一套统一的API，只要是各个版本的**存储引擎**都实现了这套API接口，无论底层存储引擎怎么变动，**查询执行引擎**调用的API接口是不变的，就能操作存储引擎。

最后查询执行引擎将存储引擎存储的数据查询出来放到缓存中一份，同时返回给客户端一份。整体流程就是这样，如下所示：



### 5.2.7存储引擎

Mysql的存储引擎都是插件式的存储引擎，都是可插拔的。存储引擎决定了表及数据的存储组织方式。在数据库里面，存储引擎使用的单位是表，也就是每张表都可以有不同的存储引擎,并且存储引擎也是可以修改的。

查看mysql存储数据库和表的本地目录:

show variables like 'datadir';

进入该目录，查看各个表的底层存储，如下所示:



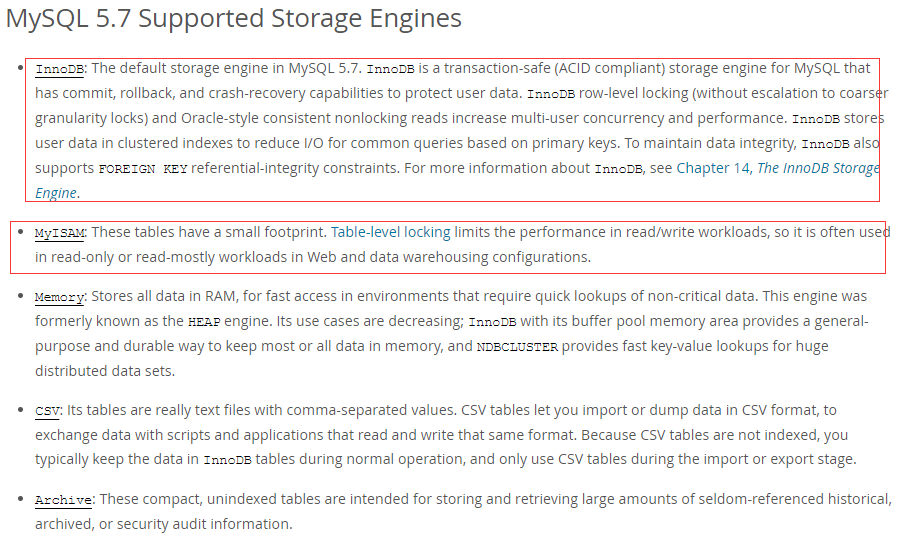
每个库都在一个目录里面，查看表在本地磁盘的的存储路径,可以看到innodb和myisam引擎存储的文件个数都不一样,innodb有两个文件存储，而myisam有三个文件存储。

show table status from test; #test是数据库名，查看一个库里面，每张表使用的数据库引擎

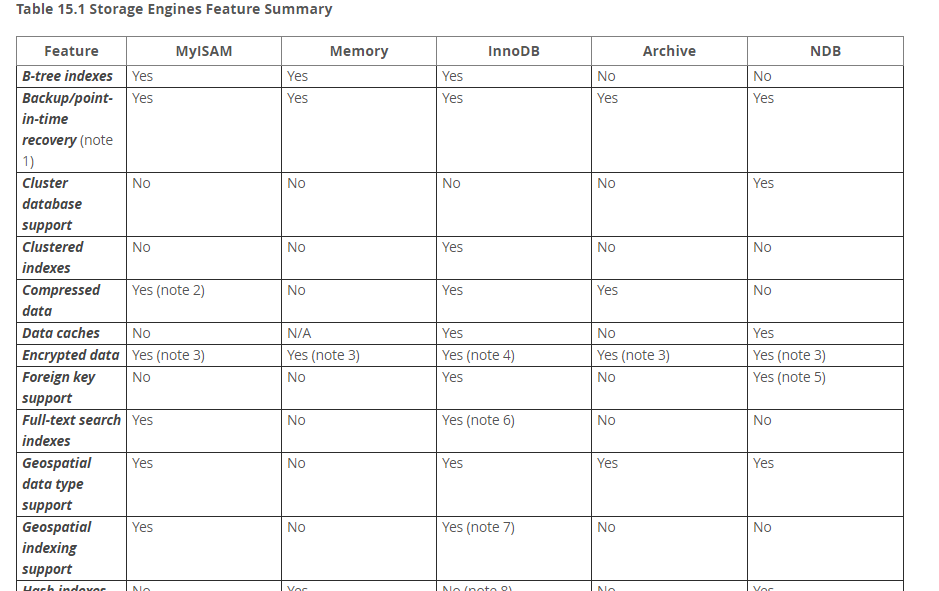
alter table dd engine = 'myisam'; #修改某个表的引擎

SHOW ENGINES： 查看当前mysql支持哪些数据库引擎，

参考: <https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html>



**看表分析：**



读多写少：建议myisam

其它：建议innodb

如果上述引擎还不能满足你，那咱就自己写一个存储引擎：请参考如下内部文档

<https://dev.mysql.com/doc/internals/en/custom-engine.html>

## 利用show profiles 查看sql具体消耗及是否命中缓存

了解查询语句底层执行的过程：查看是否开启计划。

1. 修改配置文件/etc/my.cnf，先开启查询缓存

新增一行：query\_cache\_type=1 //开启查询缓存

重启mysql：systemctl restart mysqld //重启生效

1. 再开启查询执行计划

show variables like '%profiling%'; //如果显示OFF说明关闭，ON说明开启

set profiling=1; //开启缓存日志查看功能

1. 执行语句两次：select \* from mydb.mytbl where id=1 ;
2. 显示最近执行的语句的SQL性能消耗

show profiles; //显示最近的几次查询

计算机生成了可选文字:
mysql>5howprofiles二
＋一－---－一一一一＋一－－一一一一－-
Quer先ID
DUFst10n
Query
1且勺乙
,J月呀
亡J(D
7
0.00024575
0。06633700
0.06325250
0。00041925
0。02169775
0.00672975
0。00012325
select阅profiling
SELECTDATABASE()
Sh0WdatabaseS
Sh0Wtables
showvariableslike'%profiling%
select*frommytblwhereid=1
select*frommytblwhereid=1
7FOWSinSet
1warning(6.00
SeC

1. 显示执行计划

show profile cpu,block io for query 6； //注意：6是上面的SQL编号

计算机生成了可选文字:
mysql>showprofilecpu,block10forquery6
StatUS
DUrstion
CPU_user1CPU_system
B10c悦ops_in1B10ck--ops_out
+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+~-~~~~~~~~+~~~~~~~~~~+~~~~~-~~~~~~+~~~~~~~~～一～~~~+~~~~-~~~~~~-~~~+
n)（日）
R)（日）
R)R)
（日）R)
（日）On
R)（日）
日）n)
（日）R)
R｝一日）
（日）(U
R)（日）
R)R}
starting
树aitingforquerycachelock
starting
checkingquerycacheforquery
checkingpermissions
openingtables
init
Systemlock
Waitingforquerycachelock
Systemlock
optimizlng
StatistiCS
pFeparing
executing
Sendingdata
end
queryend
closingtables
freeingitems
Waitingforquerycachelock
freeingitems
Waitingforquerycachelock
freeingitems
storingresultinquerycache
cleaningup
0。066026
0.000006
0。000004
0.006163
0。000021
0。000023
0.000033
6。000613
6.000605
0.000091
0。000623
0.000021
0。000016
0。000005
0.000099
0。000608
6.000010
0.000009
0。000074
0.000007
0.000043
0。000606
0.000605
0。000007
6。000616
6.600002
6.000000
6。000000
0.000000
0.000000
0。000000
0.000000
6。000000
6.000000
0.000000
6。000000
0.000000
0.000000
0.000000
0.000000
6.000000
6.000000
0.000000
0。000075
0.000006
0.000043
0。000006
6.000005
0.000006
6。000016
0。066619
0.006665
0。006664
0.002237
0。000016
0。000023
0.000033
0。006612
0.006665
0.000093
0。060022
0.060020
0。000016
0。000064
0.000100
0。006667
0.006669
0.000010
0。000000
0.000060
0.000060
0。006660
0.000060
0。000060
0。060066
0
0
0
3232
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
25r0WSInSet
1warnlng(0.00sec)

执行编号7时，比执行编号6时少了很多信息，从下面截图中可以看出查询语句直接从缓存中获取数据；

计算机生成了可选文字:
mysql>showprofilecpu,block10forquery7
－一一一一－-－一于一一－--－一一一－---－一＋
StatUS
DUrst10n
CpUUSeF
CPU_system
Block_ops_in1Block_ops_out
06666666
06666666
starting
树aitingforquerycachelock
starting
checkingquerycacheforquery
checkingprivilegesoncached
checkingpermissions
sendingcachedresulttoclien
cleaningup
0.000029
0.000010
0。000665
0.000011
0。000006
0。000013
0.000038
0。000012
0.000002
0.000001
0。000001
0.000001
0.000000
0.000001
0.000004
0.000001
0.000022
0.000009
6。000004
0.000010
0。006006
0。006012
0.000035
0。000010
8FO树5In
set,1warning(0.00sec)

注意：SQL必须是一致的，否则，不能命中缓存。

**如果对数据库表进行 insert ,update ,delete 这个时候，缓存会失效！**

如：select \* from mydb.mytbl where id=2 和 select \* from mydb.mytbl where id>1 and id<3 虽然查询结果一致，但并没有命中缓存。

问题：怎么命中缓存，或者命中缓存的条件是?

所以：要想命中缓存：就要保持SQL一模一样,一个空格也不要多,SQL关键字的大小写也不要改。

## 查询说明

* mysql的查询流程大致是：
* 首先，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树是否合法。
* 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**B+TREE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只少用到表中的一个索引**。

# 索引底层结构

## 6.1 什么是索引

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：索引是排好序的数据结构：便于快速查找到某个想要的信息。

索引的目的在于提高查询效率，可以类比字典，

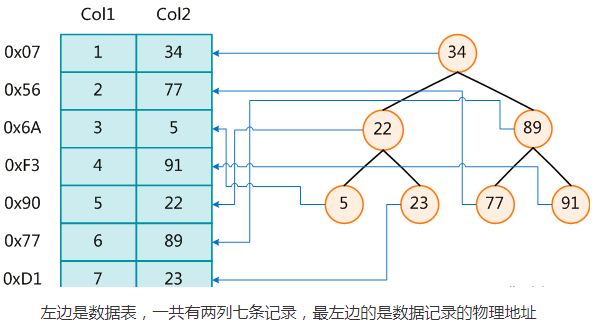
如果要查“mysql”这个单词，我们肯定需要定位到m字母，然后从下往下找到y字母，再找到剩下的sql。

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”。

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：



为了加快Col2的查找，可以维护一个右边所示的二叉查找树，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在磁盘上.

## 6.2索引的优劣势

优势：

1. 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。

2. 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。

劣势：

1. 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

2.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

## 6.3索引的一点思考

问题1: 表的索引越全越好,因为不管什么情况都能用到索引，对吗?

不对，因为使用索引会降低表的增删改操作速度

问题2: 为什么不要在性别字段上建索引?

就两个值

问题3: 为什么不建议使用身份证作为主键?

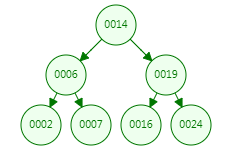
问题4: 模糊匹配like abc%, like %2673%, like %888都使用不到索引,对吗?

问题5: 不要使用select \*,写明具体查询字段,为什么?

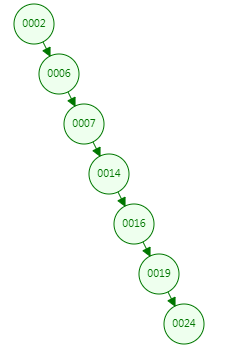
## 6.4 二叉树查找

特点: 左子树所有的节点都小于父节点，右子树所有的节点都大于父节点。投影到平面以后，就是一个有序的线性表。 动态演示,顺序插入： 14、6、2、7、19、16、24

得到:



这种折半查找速度还是挺快的。但是如果最坏的情况：如果我们插入的数据刚好是有序的，2、6、7、14、16、19、24



左子树和右子树相差太多，这时候二叉树就失去了平衡。查找速度也是全表扫描了，速度也就不快了。

那有没有可以平衡的树呢？有的，那就是平衡二叉树AVL Trees (Balanced binary search trees)

## 6.5 索引网站

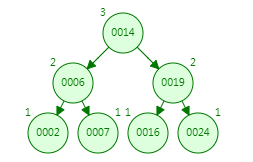
<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

二叉树(BST Binary Search Tree）)：

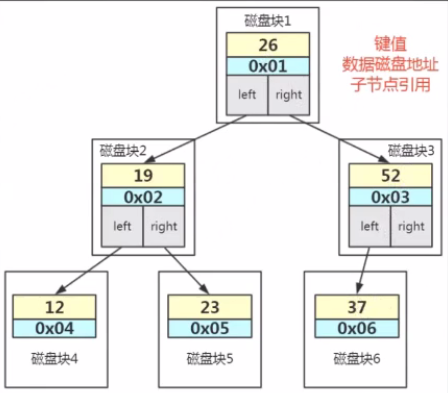
## 6.6平衡二叉树查找

平衡二叉树：规定左右子树深度差绝对值不会超过1，可以自动调节中间节点。达到两边平衡目的。

动态演示：插入数据：2、6、7、14、16、19、24



AVL数据存储：使用平衡二叉树保存索引至少应该包含三部分内容：

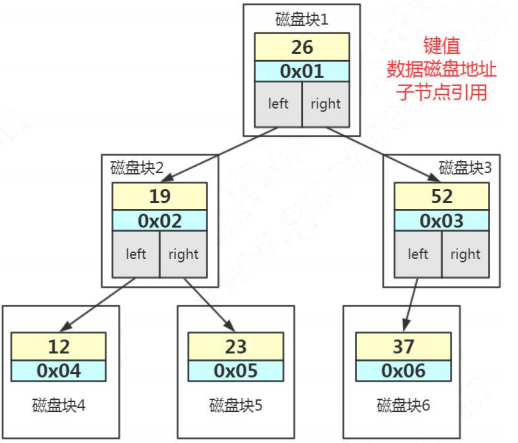


SHOW VARIABLES LIKE 'innodb\_page\_size';

如果我们采用这种方式的话，要知道innodb每页的数据是16kb,每页的数据都保存在一个磁盘块里面。

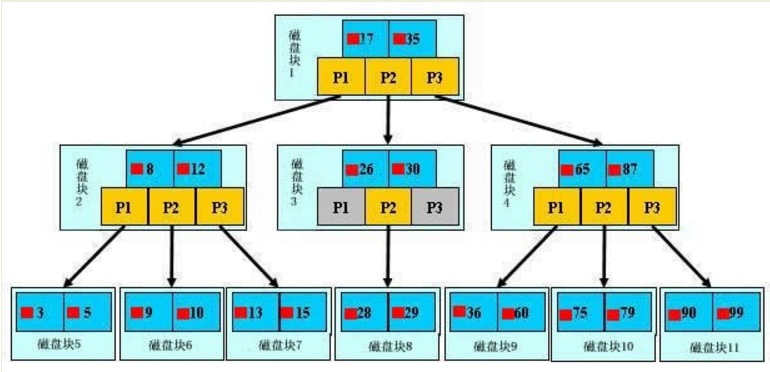


当我们用树的结构来存储索引的时候，由于索引也是存储在磁盘上，所以访问一个节点就要跟磁盘之间发生一次 IO。 InnoDB 操作磁盘的最小的单位是一页（或者叫一个磁盘块），大小是 16K(16384 字节)。 那么，一个树的节点就是 16K 的大小。 如果我们一个节点只存一个键值+数据+引用，例如整形的字段，可能只用了十几个 或者几十个字节，它远远达不到 16K 的容量，所以访问一个树节点，进行一次 IO 的时候， 浪费了大量的空间。 所以如果每个节点存储的数据太少，从索引中找到我们需要的数据，就要访问更多 的节点，意味着跟磁盘交互次数就会过多。 如果是机械硬盘时代，每次从磁盘读取数据需要 10ms 左右的寻址时间，交互次数 越多，消耗的时间就越多。



比如上面这张图，我们一张表里面有 6 条数据，当我们查询 id=37 的时候，要查询两个子节点，就需要跟磁盘交互 3 次，如果我们有几百万的数据呢？这个时间更加难以 估计。所以我们的解决方案是什么呢？ 第一个就是让每个节点存储更多的数据。 第二个，节点上的关键字的数量越多，我们的指针数也越多，也就是意味着可以有 更多的分叉（我们把它叫做“路数”）。 因为分叉数越多，树的深度就会减少（根节点是 0）。 这样，我们的树是不是从原来的高瘦高瘦的样子，变成了矮胖矮胖的样子？ 这个时候，我们的树就不再是二叉了，而是多叉，或者叫做多路。

## 6.7多路平衡查找树（B Tree）（分裂、合并）



注意：节点拥有的子树数量称为度

关键字数:N

度(Degree):N+1

这个就是我们的多路平衡查找树，叫做 B Tree（B 代表平衡）。 跟 AVL 树一样，B 树在枝节点和叶子节点存储键值、数据地址、节点引用。

它有一个特点：分叉数（路数）永远比关键字数多 1。比如我们画的这棵树，每个节 点存储两个关键字，那么就会有三个指针指向三个子节点。动态演示插入: 1、2、3、4、5、6

B Tree 的查找规则是什么样的呢？ 比如我们要在这张表里面查找 15。 因为 15 小于 17，走左边。 因为 15 大于 12，走右边。 在磁盘块 7 里面就找到了 15，只用了 3 次 IO。

和原来相比：

多路平衡二叉树每个节点存储的数据多了，路数减少了，IO次数也就减少了，显然提高了性能。

多路平衡二叉树是通过分裂和合并保持二叉树平衡的，而平衡二叉树是通过左旋和右旋保持平衡的。

## 6.8加强版的B+Tree

B Tree 的效率已经很高了，为什么 MySQL 还要对 B Tree 进行改良，最终使用了 B+Tree 呢？ 总体上来说，这个 B 树的改良版本解决的问题比 B Tree 更全面。 我们来看一下 InnoDB 里面的 B+树的存储结构:

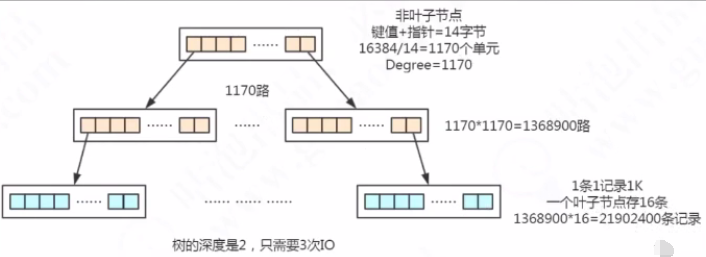


MySQL 中的 B+Tree 有几个特点：

1.它的关键字的数量是跟路数相等的；

2.B+Tree 的根节点和枝节点中都不会存储数据，只有叶子节点才存储数据。搜索 到关键字不会直接返回，会到最后一层的叶子节点。比如我们搜索 id=28，虽然在第一 层直接命中了，但是全部的数据在叶子节点上面，所以我还要继续往下搜索，一直到叶 子节点。

举个例子：假设一条记录是 1K，一个叶子节点（一页）可以存储 16 条记录。非叶 子节点可以存储多少个指针？ 假设索引字段是 bigint 类型，长度为 8 字节。指针大小在 InnoDB 源码中设置为 6 字节，这样一共 14 字节。非叶子节点（一页）可以存储 16384/14=1170 个这样的 单元（键值+指针），代表有 1170 个指针。 树 深 度 为 2 的 时 候 ， 有 1170^2 个 叶 子 节 点 ， 可 以 存 储 的 数 据 为 1170\*1170\*16=21902400。也就是说，一张 2000 万左右的表，查询 数据最多需要访问 3 次磁盘。 所以在 InnoDB 中 B+ 树深度一般为 1-3 层，它就能满足千万级的数据存储。



3、B+Tree 的每个叶子节点增加了一个指向相邻叶子节点的指针，它的最后一个数据会指向下一个叶子节点的第一个数据，形成了一个有序链表的结构。

4、它是根据左闭右开的区间 [ )来检索数据。

总结一下，InnoDB 中的 B+Tree 的特点：

1. 它是 B Tree 的变种,B Tree 能解决的问题，它都能解决。B Tree 解决的两大问题 是什么？（每个节点存储更多关键字；路数更多）
2. 扫库、扫表能力更强（如果我们要对表进行全表扫描，只需要遍历叶子节点就可以 了，不需要遍历整棵 B+Tree 拿到所有的数据,便于范围查询）
3. B+Tree 的磁盘读写能力相对于 B Tree 来说更强（根节点和枝节点不保存数据区， 所以一个节点可以保存更多的关键字，一次磁盘加载的关键字更多,3次IO能读取千万条数据）
4. 排序能力更强（因为叶子节点上有下一个数据区的指针，数据形成了链表）
5. 5)效率更加稳定（B+Tree 永远是在叶子节点拿到数据，所以 IO 次数是稳定的）

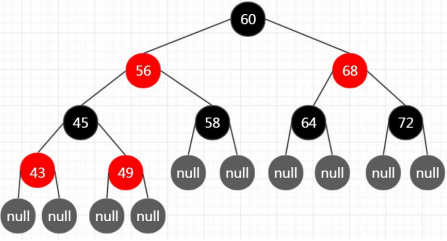
## 6.9为啥不用红黑树

红黑树也是 BST 树，但是不是严格平衡的。

必须满足 5 个约束：

1. 节点分为红色或者黑色。
2. 根节点必须是黑色的。
3. 叶子节点都是黑色的 NULL 节点。
4. 红色节点的两个子节点都是黑色（不允许两个相邻的红色节点）。
5. 从任意节点出发，到其每个叶子节点的路径中包含相同数量的黑色节点。

插入：60、56、68、45、64、58、72、43、49



基于以上规则，可以推导出： 从根节点到叶子节点的最长路径（红黑相间的路径）不大于最短路径（全部是黑色 节点）的 2 倍。

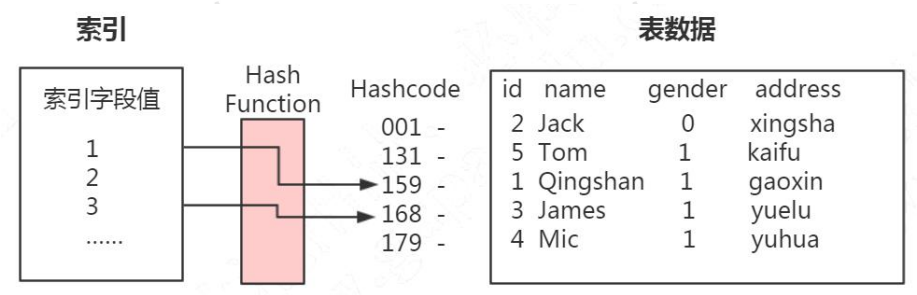
为什么不用红黑树？1、只有两路；2、不够平衡。

红黑树一般只放在内存里面用。例如 Java 的 TreeMap。

## 6.10索引方式：真的是用的 B+Tree 吗？

在 Navicat 的工具中，创建索引，索引方式有两种，Hash 和 B Tree。

HASH：以 KV 的形式检索数据，也就是说，它会根据索引字段生成哈希码和指针， 指针指向数据。



哈希索引有什么特点呢？

第一个，它的时间复杂度是 O(1)，查询速度比较快。因为哈希索引里面的数据不是 按顺序存储的，所以不能用于排序。

第二个，我们在查询数据的时候要根据键值计算哈希码，所以它只能支持等值查询 （= IN），不支持范围查询（> < >= <= between and）。

另外一个就是如果字段重复值很多的时候，会出现大量的哈希冲突（采用拉链法解 决）,效率会降低。

我们在 Navicat 工具里面选择索引方法是哈希，但是它创建的还是 B+Tree 索引，这个不是我们可以手动控制的。上次课我们说到bufferpool里面有一块区域是AdaptiveHashIndex自适应哈希索引，就是这个。这个开关默认是ON：

show variables like 'innodb\_adaptive\_hash\_index';

从存储引擎的运行信息中可以看到：

show engine innodb status\G;



# MySQL架构\_存储引擎简介

## 查看命令

* 查看mysql提供什么存储引擎：show engines;

计算机生成了可选文字:
mysql>5howengines;
＋一一－-－一－一－-－一一－一一－－一－
－一一一－-----－一一＋～一一一一一务－－一一一一－－一一－一＋
Engine
Support
Comment
TF8nsaCtionS}XA
Savepoints
1Inn0DB}OEFAULT1SupportStransactionS,row一level10cklng,andforeignkeys!YES!YES1YES}
1HRG＿畔ISAM1YES}C0llectionofidenticalHyISAHtables!NO1NO1NO!
1HEMORY}YES1Hashbased,storedinmemory,usefulfortemporarytables}No}NOIN01
1BLACKHoLE}YESI/dev/nullstorageengine(anythingyouwritetoitdisappearS)!No!NO1NO!
1HyISAH!YES1HyISAHStorageenglne!NO!NO1Nol
1CSV}YES1CSVStorageengine}NO}NOIN01
1ARCHIVE1YES1Archivestorageengine1NO1NO1NO!
1PERFORHANCE_SCHEMA}YES1PerformanceSchema!NO!NO1NO!
}FEDERATED}NO1FederatedHySQLStorageengin"!NULL!NULL1NULL!
＋一～一～~~~~～一一－--~~～一一～+~~～一～~-~~+～一一－--~~～一～~~~~~~~~-~~~～一一～~~～一～一～~~~~~~~----～一一一－--~~~~~~~-~~～一＋~~~～一一一－~~-~～一＋~~～一一一＋~~~～一一～-~~-~+
9rowsinset(6.00sec)

* 命令：show engines \G;

|  |
| --- |
| mysql> show engines \G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: InnoDB  Support: DEFAULT  Comment: Supports transactions, row-level locking, and foreign keys  Transactions: YES  XA: YES  Savepoints: YES  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: MRG\_MYISAM  Support: YES  Comment: Collection of identical MyISAM tables  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: MEMORY  Support: YES  Comment: Hash based, stored in memory, useful for temporary tables  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: BLACKHOLE  Support: YES  Comment: /dev/null storage engine (anything you write to it disappears)  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: MyISAM  Support: YES  Comment: MyISAM storage engine  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: CSV  Support: YES  Comment: CSV storage engine  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: ARCHIVE  Support: YES  Comment: Archive storage engine  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: PERFORMANCE\_SCHEMA  Support: YES  Comment: Performance Schema  Transactions: NO  XA: NO  Savepoints: NO  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Engine: FEDERATED  Support: NO  Comment: Federated MySQL storage engine  Transactions: NULL  XA: NULL  Savepoints: NULL |

* 查看默认的存储引擎：show variables like '%storage\_engine%';

计算机生成了可选文字:
mysql>Showvariableslike'%Storage_engine%'
VSrlablenallle
Va1Ue
＋一一一一－－一一－－一～～一一一～~~~~~～·
}default_storage_engine
1default_tmp_Storage_engine
1disabled_storage_engines
1internal_tmp_disk_storage_
一＋-－一－一一－一＋
Inn0DB
Inn0DB
englne
Inn0DB
4
rowsinset(0.01sec

## 各种引擎介绍

### InnoDB存储引擎

* **大于等于5.5之后，默认采用InnoDB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。可以确保事务的完整提交(Commit)和回滚(Rollback)。
* 除了增加和查询外，还需要更新，删除操作，那么，应优选选择InnoDB存储引擎。
* 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。

### 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。
* 5.5之前默认的存储引擎

### 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（档案）**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%。

### 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。

### 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。

### 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。

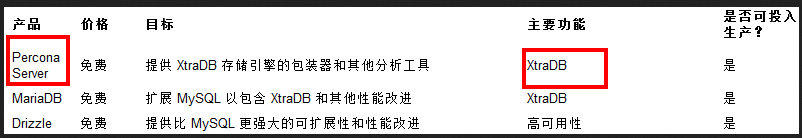
### Federated引擎

* 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。

## MyISAM和InnoDB 面试题！

|  |  |  |
| --- | --- | --- |
| **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 |
| 自带系统表使用 | Y | N |
| 关注点 | 性能：节省资源、消耗少、简单业务 | 事务：并发写、事务、更大资源 |
| 默认安装 | Y | Y |
| 默认使用 | N | Y |

## 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

# 索引优化分析\_预热\_慢&7-Join\_作业

## 性能下降SQL慢 执行时间长 等待时间长

* 数据过多——分库分表 mycat
* 索引失效，没有充分利用到索引——索引建立
* 关联查询太多join（设计缺陷或不得已的需求）——SQL优化
* 服务器调优及各个参数设置（缓冲、线程数等）——调整my.cnf

## 常见通用的Join查询

### SQL执行顺序

* 1. 手写

计算机生成了可选文字:
SELECTD工ST工NCT
<SeleCtlist>
FROM
<lef七七able><join七ype
Jo工N<righttab工e>ON<join
WHERE
COndi七ion>
<WhereCOndition>
GROUPBY
qroupby113七
HAV工NG
ORDER
L工M工T
havlnqcondi七ion>
BY
orderbycondi七ion>
<limi七ntlTnber>

* 1. 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：

计算机生成了可选文字:
1FROM<left七able>
2ON<joincondi七ion>
<join七yPe>JO工N<righ七七able
WHERE<WhereCOndi七ion>
GROUPBY<groupby115七＞
HAV工NG<havinqcondi七ion>
7SELECT
SD工ST工NCT<SeleC七115七＞
9ORDERBY<orderbycondi七ion>
10L工H工T<limi七nUmber>

* 1. 总结(鱼骨图)

计算机生成了可选文字:
O能et
宙卡尔积
不符合ON也添加
/
改变对表引用
DIST!NCT
FROM
JO!N
GROUP日Y」
SELECT
ON
V叮」ERE
O只OERBY
主表保留
非聚合
只作
下几使靛L〔C。
非SELEC铭时名
SQL"l

### Join图

计算机生成了可选文字:
SQLJOINS
州
SEI王CT<scl七ctlist>
FROMTablcAA
】王FTJOINTablcBB
ON凡Kcy二B．价y
飞
、、、匕．．口口尸，,．一尸
SEI卫CT<sclcctlist>
FROMTableAA
RIGHTJ0lNTab二eBB
ON凡Kcv二B.Kcv
SEUECT<select11st>
FROMTah!cAA
INNI乙RJOINTablcBB
ONA.Kcy二B．女y
A
SEI王CT<sclcc宜list>
FROMT决hlcAA
LEFr10INTahlcBB
oX^．旋y二B·枪y
WHEREB.Kcv15NUIJ。
SEI.ECT＜鱿kctlist>
FROMTabl亡AA
FRO、1TableAA
RIGHTJOINTahlcBB
ONA.Kcv
=B.Kcy
wHEREA-Kcv15NUIJ
SEUECT<selectlist>
SELECr<sclcctslist>
FRO、！TablcAA
FL!IJ.OUTERJOINTablcB
FUI'1.OUTERJOINTablcBI弓
oNA.Kcy二B·Kcy
OC工M。月沁t，太匡熔
ON人Kcy二B.Kcy
WHERE凡Key15NU】J·
ORB,K亡v15NUIJJ

### 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(11) 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(id,deptName,address) VALUES(1,'华山','华山');  INSERT INTO t\_dept(id,deptName,address) VALUES(2,'丐帮','洛阳');  INSERT INTO t\_dept(id,deptName,address) VALUES(3,'峨眉','峨眉山');  INSERT INTO t\_dept(id,deptName,address) VALUES(4,'武当','武当山');  INSERT INTO t\_dept(id,deptName,address) VALUES(5,'明教','光明顶');  INSERT INTO t\_dept(id,deptName,address) VALUES(6,'少林','少林寺');  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(1,'风清扬',90,1,100001);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(2,'岳不群',50,1,100002);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(3,'令狐冲',24,1,100003);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(4,'洪七公',70,2,100004);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(5,'乔峰',35,2,100005);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(6,'灭绝师太',70,3,100006);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(7,'周芷若',20,3,100007);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(8,'张三丰',100,4,100008);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(9,'张无忌',25,5,100009);  INSERT INTO t\_emp(id,NAME,age,deptId,empno) VALUES(10,'韦小宝',18,NULL,100010); |

### 7种JOIN

|  |  |
| --- | --- |
| 1 A、B两表共有(查询所有有部门的员工->员工和部门之间必须存在关联的数据)  SELECT \* FROM t\_emp a **INNER JOIN** t\_dept b ON a.deptid = b.id ;  计算机生成了可选文字: 口ldna止吧 卫仁二习几青提 已qe 功pno 100001 100002 100003 100004 100005 100006 100007 100008 100009 ep七Na功e 山山山布有胃省当教 2吞不群 3令狐冲 4洪七公 5乔峨 6灭绝师太 7周芷若 8张三丰 9张无忌 90 5O 24 ,O 35 ,O 2O 100 25 己ddre33 华山 华山 华山 洛刁 洛刁 峨省山 峨省山 武当山 允明顶 旦旦旦旦口回口一口  2 A、B两表共有+A的独有（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 ;  计算机生成了可选文字: 口」id n几In户 已ge dep七Id e功pno id dep七Na胭e 已ddre日3 U，。胃处 2吞不群 3令狐冲 4洪七公 5乔哇 6灭绝师太 尹周芷若 叔三丰 9张无忌 3军一 70 士 2O 100 25 口口口口 l已 100002 100003 100004 100005 100006 100007 100008 100009 口口口口口口口口 100010 1’华叮 1华山 1华山 2丐布 2丐有 3峨胃 3峨眉 4武当 5明教 华山 华山 习 习 省山 省山 当山 明顶 乎洛飞飞 、厂25+-－广广 六U月勺nU 仁〕『乙7 10韦小宝 口口 抓几L （讯几L){（抓几工） N仃LL 马学哥哥可可D'  3 A、B两表共有+B的独有（列出所有部门，并显示其部门的员工信息 ）B的全集  SELECT \* FROM t\_emp a **RIGHT JOIN** t\_dept b ON a.deptid = b.id ;  计算机生成了可选文字: n台拍e ep七Id e口pno 血p七Na角e 100002 100003 1"""”军一 100005 100006 100007 乍一 9 己ddre33 华正下一 华山 华山 洛习 洛习 峨省山 峨看山 武当山 光明顶 ．止．上2内乙，〕幼曰 n甘月勺n甘哎〕n甘n甘 匕〕『乙7，口7八乙 遨．匕曰 抓几L U凤清抒 2丢不群 3令狐冲 4洪七公 5乔峰 6灭绝厂太 ，周芷若 8张三丰 9张无忌 产谕蔽r= 100 十一一 25 而菌尸一而屁 10000 10000 口口口口口门口口 《机几L 1华山 1华山 2丐布 2丐帮 3峨省 3峨胃 4武当 5明教 创琴铆 口．口口口一旦口、口川口一回回口  4 A的独有 （查询没有加入任何部门的员工）  SELECT \* FROM t\_emp a **LEFT JOIN** t\_dept b ON a.deptid = b.id WHERE **b.id IS NULL** ;  计算机生成了可选文字: 已口 101案， 孕『pno dep七Nome N仃LL)（抓几L) 性ddre33 (NULL) 口口  5 B的独有（查询没有任何员工的部门）  SELECT \* FROM t\_emp a **RIGHT JOIN** t\_dept b ON a.deptid = b.id WHERE **a.id IS NULL** ;  计算机生成了可选文字: 已qe 1d der七Na正e 己ddre3日 NULL dep七Id 《抓几L e双甲nD 《机几L  6 AB全有（查询所有员工和所有部门）  #MySQL Full Join的实现 因为**MySQL不支持FULL JOIN**,下面是替代方法  #left join + union(去重)+ 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 ;  计算机生成了可选文字: m》’sql> ERROR se1ect'from 1064(42000): SerVerVerSIOn m》’sql> ERROR select今 for from 1064(42000): serVerVersionfor tbl_empafullJointbl_dept 、”ouha、产eanerrorinyourSQL therightsyntaxtousenear tbl_empafullouterjointbl Youha、reanerr0r1ny0urSQI. therightsyntaxtousenear bona.deptld=b.id: syntax:checkthemanualthatcorrespondstoyourMySQL fu11Jo1ntbl_deptbona.deptld=b.id'atlinel _deptbona.deptld=b.id: syntax:checkthemanualthatcorrespondstoyourMySQL fu11outerJointbl_deptbona.deptld=b.id'atline  计算机生成了可选文字: 口id n月丫n户 己qe 过ep七Id deptNa功e 口上．上22幼甘3乙．5 匀尸’、’月～ 2丢不群 3令狐；中 4洪七公 5乔峨 6灭绝卿太 7周芷若 8张三丰 9张无后 50 24 ,O 35 ,O 20 100 25 e功pnold ．甲而丽贫冲－ 100002 100003 10000习一一 100005 100006 100007 100008 100009 1华山 1华山 2丐有 2丐有 3峨省 3峨胃 4武当 5明教 10月阵勺、习又 花赢司而贡言 《ITUll）《抓几工 N习工L 抓几L 《1兀几L 台以目re33 罕正产一 华山 华山 洛习 洛阳 峨省山 峨圈山 武当山 尤导月顶 （上r-’二工） 不下马盯 口口口回回回回一口．口．口口     |  | | --- | | SELECT \* FROM t\_emp a LEFT JOIN t\_dept b ON a.deptid = b.id  **UNION ALL**  SELECT \* FROM t\_emp a RIGHT JOIN t\_dept b ON a.deptid = b.id ;  **1.UNION ALL 不会自动去重。**  **2.UNION在使用时，两张表的字段保证一致，如果不一致，请在slect后面列选字段，不要使用\***  计算机生成了可选文字: dep七Id e功pno id deD艺1万弓仍e 己ddre33 100001 100002 100003 100004 100005 100006 100007 100008 r华百 1华山 1华山 2丐帮 2丐布 3峨省 3峨胃 4武兰 叹日日公玲 尸华盯 华山 华山 洛刁 ｝洛习 峨圈山 峨省山 武当山 辛日目俪 月人1 ‘人2 idn己顶吧 亘亘亘啊潘骊一 2丢不群 3令狐冲 4洪七公 …5养哇 6灭绝，币太 ，夕芷着 8张三丰 90 5O 24 7O 35 7O 2O 100 旧厄粤局 月乙1〕 .1口口 10’韦小宝 le（抓兀L) 10口010 2岳不群 3令狐冲 少七公 5乔峰 6灭绝沛太 ，一周芷着 吩手丰 9张无忌 赢对临屁尸 50 24 7O 35 7O 2O 100 25 口口口口口口口 NULL) 100002 100003 100004 100005 100006 100007 100008 100009 口口口口口口口口口 （机几L) 《资刃LL》《NULL 一一气阅约r 1华山 1华山 2丐律 2丐布 3峨胃 3峨胃 4武当 5明教 N仃Ll. 华山 华山 洛刁 ｝洛习 峨雹山 峨胃山 武当山 光明攻 1月人 22+--3 幼〕月『 抓几L 巴口口｝口口工」旦一口 |   7 A的独有+B的独有（查询没有加入任何部门的员工，以及查询出部门下没有任何员工的部门）  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.id IS NULL ;  计算机生成了可选文字: a口e 当丝盈一二 （抓几L) 一片而右了－ deptNa.e 18 口口口口口口口口口口口 抓兀工， 丝，一 100010 抓几L N口LL》《机几L 一－下哆耳－ 己ddre33 （机几L) 多耳胃尸 口一口一口 |

### 扩展(掌门人)

1. 增加掌门人字段

|  |
| --- |
| 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 b.deptname,a.name FROM **t\_dept b** LEFT JOIN **t\_emp a** ON **b.ceo = a.id** ;

1. 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 ;

1. 求所有人，对应的掌门是谁(课堂练习，4种写法分析)

|  |
| --- |
| #临时表连接方式  #step1根据ceo 去查找每个部门的掌门是谁  {SELECT b.id AS deptId,a.name AS ceoname FROM t\_emp a **INNER JOIN** t\_dept b ON a.id = b.ceo }  #step2 与员工表整合 SELECT c.name,ab.ceoname FROM t\_emp c LEFT JOIN  ( SELECT b.id AS deptId,a.name AS ceoname FROM t\_emp a **INNER JOIN** t\_dept b ON a.id = b.ceo ) ab  ON c.deptId = ab.deptId ;    #临时表连接方式  #step1 根据部门Id 查询员工的信息  SELECT a.name,b.ceo FROM t\_emp a **LEFT JOIN** t\_dept b ON a.deptid = b.id  # step2 与员工表整合  SELECT ab.name,c.name AS ceoname FROM  ( SELECT a.name,b.ceo FROM t\_emp a **LEFT JOIN** t\_dept b ON a.deptid = b.id ) ab  LEFT JOIN t\_emp c ON ab.ceo = c.id ;    #三表左连接方式  SELECT a.name,c.name AS ceoname FROM t\_emp a  LEFT JOIN t\_dept b ON a.deptid = b.id  LEFT JOIN t\_emp c ON b.ceo = c.id ;    #子查询方式  SELECT a.name ,(SELECT c.name FROM t\_emp c WHERE c.id=b.ceo) AS ceoname  FROM t\_emp a LEFT JOIN t\_dept b  ON a.deptid = b.id ; |

## 作业：8个SQL

|  |
| --- |
| #1、列出自己的掌门比自己年龄小的人员  #2、列出所有年龄低于自己门派平均年龄的人员  #3、列出至少有2个年龄大于40岁的成员的门派  #4、至少有2位非掌门人成员的门派  #5、列出全部人员，并增加一列备注“是否为掌门”，如果是掌门人显示是，不是掌门人显示否  #6、列出全部门派，并增加一列备注“老鸟or菜鸟”，若门派的平均值年龄>50显示“老鸟”，否则显示“菜鸟”  #7、显示每个门派年龄最大的人  #8、显示每个门派年龄第二大的人 |

# 索引优化分析\_索引介绍

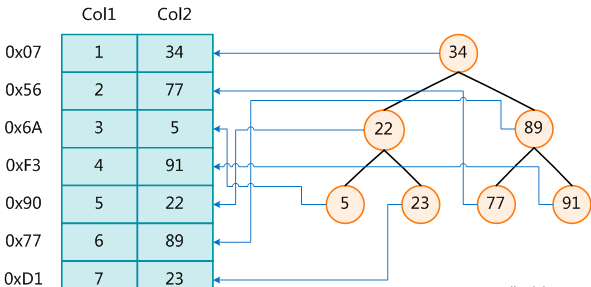
## 是什么

1. 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**。

索引的本质：索引是数据结构。你可以简单理解为“排好序的快速查找数据结构”。

1. 详解(重要)

* 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。

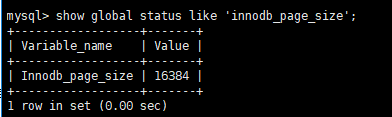
1. 结论

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**。

1. 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**以索引文件形式存储磁盘上**

磁盘块和磁盘页

* 数据库**索引是存储在磁盘上的**，如果数据很大，必然导致索引的大小也会很大，超过几个G（好比新华字典字数多必然导致目录厚）
* 当我们利用索引查询时候，是不可能将全部几个G的索引都加载进内存的，为了降低内存开销，InnoDB在把磁盘数据读入到内存时会以页（Page）为基本单位。（一次I/O操作）
* 每一个磁盘块在MySQL中是一个页，页大小是固定的，MySQL InnoDB的默认的页大小是16k，每个索引会分配在页上的数量是由索引字段的大小决定。当索引字段值的长度越长，每一页上的数量就会越少，因此在一定数据量的情况下，索引的深度会越深，影响索引的查找效率。
* SHOW GLOBAL STATUS LIKE 'Innodb\_page\_size'; -- **16KB**



## 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

## 劣势

* 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
* 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

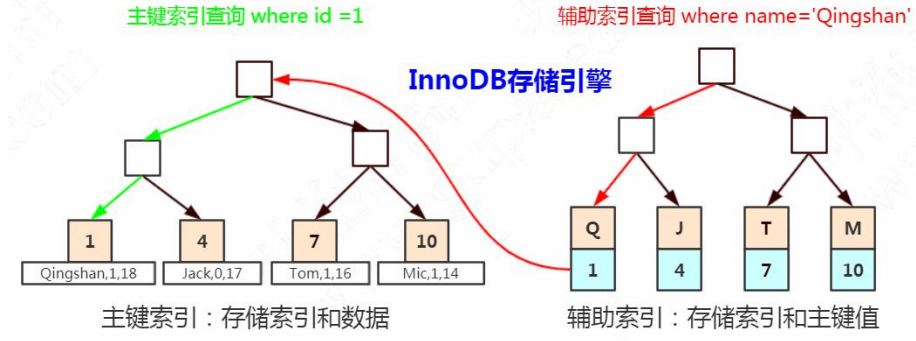
## MySQL索引

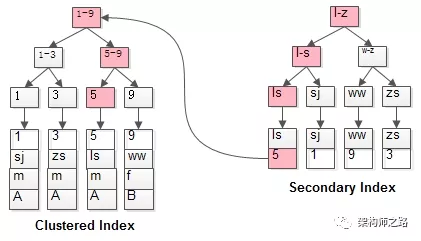
### 聚簇索引与非聚簇索引

* **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**。
* 就是索引键值的逻辑顺序跟表数据行的物理存储顺序是一致的。（比如字典的目录是按拼音排序的，内容也是按拼音排序的，按拼音排序的这种目录就叫聚集索引）。 在 InnoDB 里面，它组织数据的方式叫做叫做（聚集）索引组织表（clustered index organize table），所以主键索引是聚集索引，非主键都是非聚集索引。。
* 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。

计算机生成了可选文字:
ldfi8llle
琉
Wang
Jlfl
坟U一‘U
尸a一‘U
Chen
hou
【了工C己
聚簇索引
物理存储
到非聚簇索引

如果 InnoDB 里面主键是这样存储的，那主键之外的索引，比如我们在 name 字段 上面建的普通索引，又是怎么存储和检索数据的呢？





辅助索引存储的是辅助索引和主键值。

如果使用辅助索引查询，会根据主键值在主键索引中查询，最终取得数据。

比如我们用 name 索引查询 name= 'Qingshan'，它会在叶子节点找到主键值，也就是 id=1，然后再到主键索引的叶子节点拿到数据。一个问题，如果一张表没有主键怎么办？

1. 如果我们定义了主键(PRIMARY KEY)，那么 InnoDB 会选择主键作为聚集索引。
2. 如果没有显式定义主键，则 InnoDB 会选择第一个不包含有 NULL 值的唯一索引 作为主键索引。
3. 如果也没有这样的唯一索引，则 InnoDB 会选择内置 6 字节长的 ROWID 作为隐藏的聚集索引，它

会随着行记录的写入而主键递增。(\_rowid:隐含、自增的)

1. select \_rowid from 表名;

* 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。

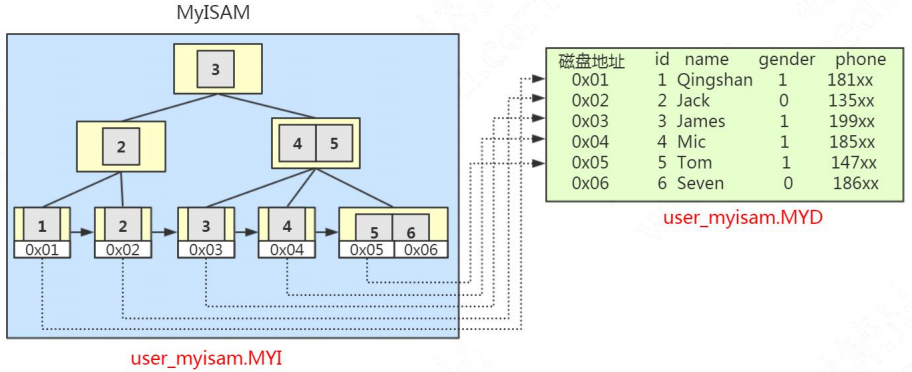
### 9.4.2 MyISAM

在 MyISAM 里面，另外有两个文件：

一个是.MYD 文件，D 代表 Data，是 MyISAM 的数据文件，存放数据记录,比如我们的 user\_myisam 表的所有的表数据。

一个是.MYI 文件，I 代表 Index，是 MyISAM 的索引文件，存放索引，比如我们在 id 字段上面创建了一个主键索引，那么主键索引就是在这个索引文件里面。

也就是说，在 MyISAM 里面，索引和数据是两个独立的文件。 那我们怎么根据索引找到数据呢？ MyISAM 的 B+Tree 里面，叶子节点存储的是数据文件对应的磁盘地址。所以从索引文件.MYI 中找到键值后，会到数据文件.MYD 中获取相应的数据记录。

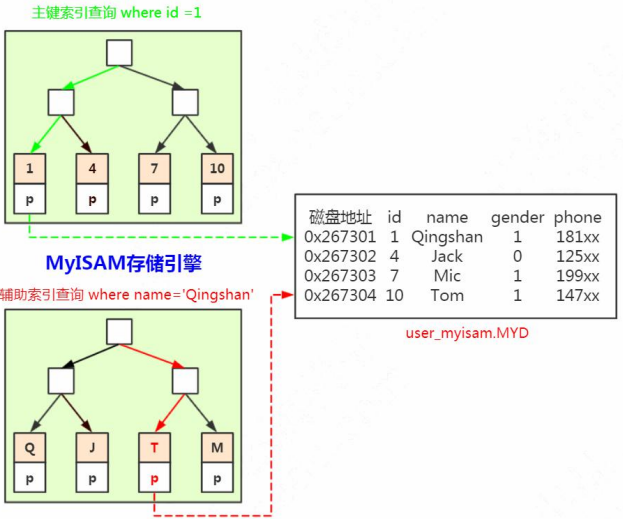


数据和索引是分开存放的。(**MyISAM引擎支持主键索引，但是没有聚簇索引(聚簇索引数据和索引在一起。只有Innodb引擎有聚簇索引**))

索引分为：主键索引和辅助索引(也称为:二级索引)

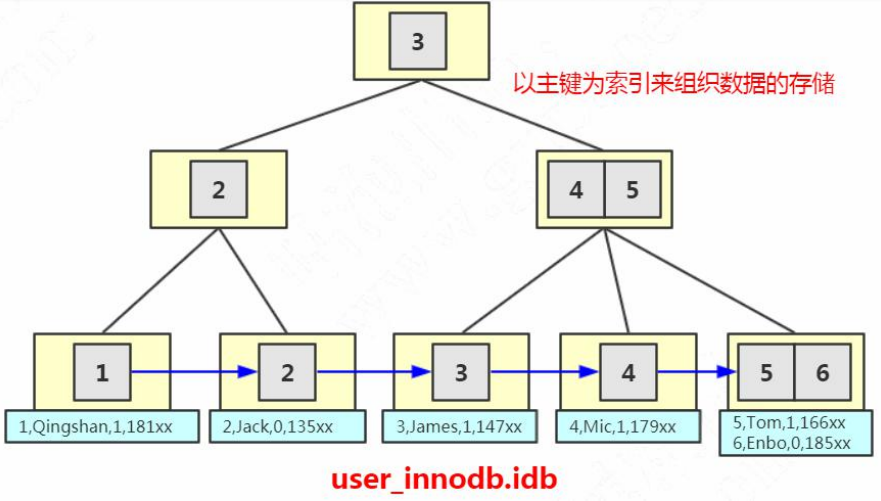
这里是主键索引，如果是辅助索引，有什么不一样呢？

在 MyISAM 里面，辅助索引也在这个.MYI 文件里面。辅助索引跟主键索引存储和检索数据的方式是没有任何区别的，一样是在索引文件里面找到磁盘地址，然后到数据文件里面获取数据。



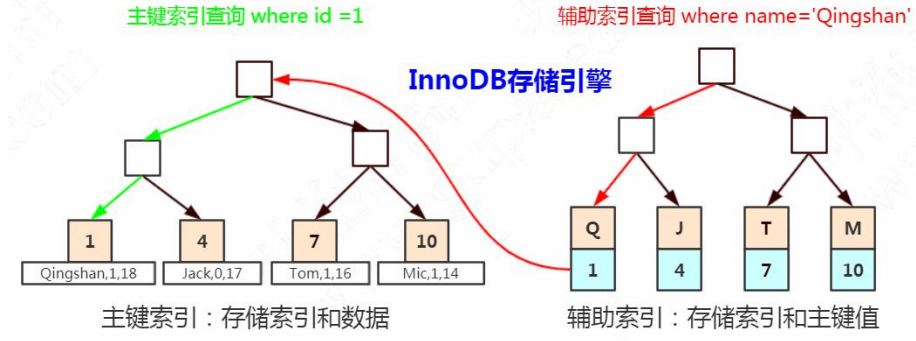
### 9.4.3 InnoDB

InnoDB 只有一个文件（.ibd 文件），那索引放在哪里呢？ 在 InnoDB 里面，它是以主键为索引来组织数据的存储的，所以索引文件和数据文件是同一个文件，都在.ibd 文件里面。 在 InnoDB 的主键索引的叶子节点上，它直接存储了我们的数据。



只有InnoDB引擎支持聚簇索引,对于聚簇索引而言，相邻索引指向的数据 是存在是在一起的。

**辅助索引**存储的是辅助索引和主键值。



这就是所谓的**回表查询【只有innodb引擎的辅助索引查询时会出现】。**

### 9.4.4小结

只有InnoDB引擎支持聚簇索引，聚簇索引索引键值的逻辑顺序跟表数据行的物理存储顺序是一致的。便于范围查询。

MyISAM引擎：只有主键索引，没有聚簇索引。

确切的说是：

1. 在Innodb引擎中，主键索引也即聚簇索引。
2. 在MyISAM引擎中，主键索引和辅助索引的搜索是一样的。
3. 在Innodb引擎中，主键索引(聚簇索引)和辅助索引(存储的是索引值和主键值)的搜索是不太一样。

## 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)  );  单独建主键索引：  ALTER TABLE customer2 add PRIMARY KEY customer2(id);  删除主键索引：  ALTER TABLE customer2 drop PRIMARY KEY ;  修改主键索引：  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\_customer\_no\_name ON customer(customer\_no,customer\_name);  删除索引：  DROP INDEX idx\_customer\_no\_name on customer ; |

### 基本语法

1. 创建

CREATE [UNIQUE ] INDEX indexName ON mytable(column name(length));

length ： 默认可以不用写

如果是CHAR，VARCHAR类型，length可以小于字段实际长度； 如果是BLOB和TEXT类型，必须指定length。

ALTER mytable ADD [UNIQUE ] INDEX [indexName] ON (columnname(length))

1. 删除

DROP INDEX [indexName] ON mytable;

1. 查看

SHOW INDEX FROM table\_name

1. 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。了解

## MySQL索引增删改查概括

### 9.6.1 创建的时候直接指定

CREATE TABLE customer (

id INT (10) UNSIGNED AUTO\_INCREMENT,

customer\_no VARCHAR (200),

customer\_name VARCHAR (200),

PRIMARY KEY (id), # 主键索引

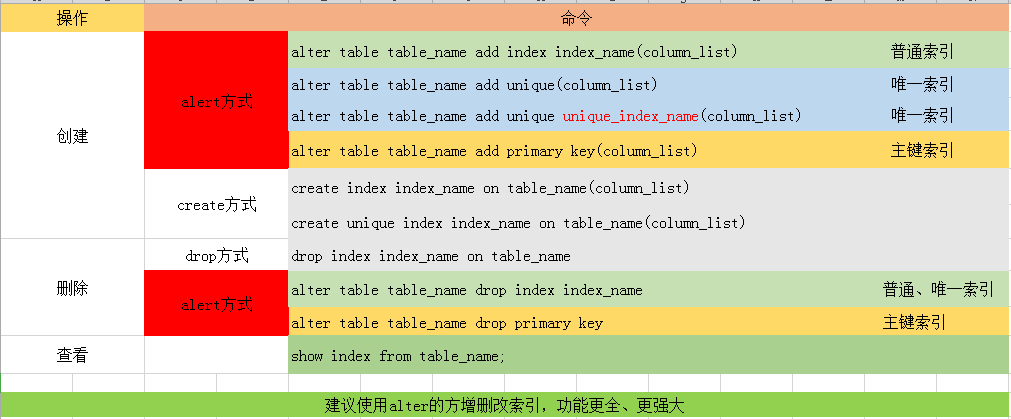
KEY (customer\_name), # 单值索引 ,可自己指定索引名

UNIQUE (customer\_name), # 唯一索引,可自己指定索引名

KEY (customer\_no, customer\_name) # 复合索引/组合索引,可自己指定索引名

);

### 9.6.2 后期再修改



需要注意以下三点：

1.唯一索引：通常针对手机号、用户名(非真实姓名)、身份证号、员工编号等字段创建

2.主键索引：建表时，表主键指定后，主键索引自动创建

3.一个表一旦建立主键之后后期几乎不会改动,如果要删除一个表的主键，必须保证该主键不能是自增的主键，否则删除不了，且提示:



## 哪些情况需要创建索引

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，who？(在高并发下倾向创建组合索引)
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

思考：排序和分组哪个更伤性能？

## 哪些情况不要创建索引

1. 表记录太少

300万数据时MySQL性能就开始下降了，这时就可以开始开始优化了

1. 经常增删改的表

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

1. where条件里用不到的字段不创建索引(索引也是占用空间的)

# 索引优化分析\_explain查看执行计划

## MySql Query Optimizer

* MySQL中有专门负责优化SELECT语句的优化器模块，主要功能：通过计算分析系统中收集到的统计信息，为客户端请求的Query提供他认为最优的执行计划（他认为最优的数据检索方式，但不见得是DBA认为是最优的，这部分最耗费时间）
* 当客户端向MySQL 请求一条Query，命令解析器模块完成请求分类，区别出是 SELECT 并转发给MySQL Query Optimizer时，MySQL Query Optimizer 首先会对整条Query进行优化，处理掉一些常量表达式的预算，直接换算成常量值。并对 Query 中的查询条件进行简化和转换，如去掉一些无用或显而易见的条件、结构调整等。然后分析 Query 中的 Hint 信息（如果有），看显示Hint信息是否可以完全确定该Query 的执行计划。如果没有 Hint 或Hint 信息还不足以完全确定执行计划，则会读取所涉及对象的统计信息，根据 Query 进行写相应的计算分析，然后再得出最后的执行计划。

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

计算机生成了可选文字:
令CO自dev.mysql.com/doc/refman/5.7/en/explain一eXtended.html
昏｝☆．尽，
犷个个J
q
MySQL5.7ReferenceManual，…，ExtendedEXPLAINoutputFormat
DOCUment日tionHome
·Pre伯Ce日ndLegdlNoticeS
》GeneFdl!nfo「mdtion
》InstallingandUpgradingMySQL
》丁日tOFial
,MySQLProgramS
》MySQLServerAdministration
》Security
》BdckupandRecovery
8.8.3ExtendedEXPLAINOUtpUtFormat
ForsEL已cTstatements,theExP认工Nstatementproducesextra(‘撼xtended")informationthat15notpartofExP认工NoutputbutcanbeviewedbyissuingasHo砚
闪几只N工NGsstatementfollowing弓xP认工N.The又essagevalueinsHowWARN工NGsoutputdisplayshowtheoptimizerqualifiestableandcolumnnamesinthe
翌些旦里Statement,Whatthe丝些旦里IOokSIikeaftertheapplicationofrewritingandOptimiZationrules,andpOSSiblyothernotesabouttheOptimiZation
pr0CeSS.
TheeXtended
1displayable
explainable
infoFm日tion
withasHowWA只N工NGsstatementfollowing三xP以工N15producedonlyforsE卫EcTstatements.sHo何
刁入RN工NGS
displays
anemptyresultforother
statements(D三LETE，工NsERT，吃P认cE,anduP队T乙）.
NOte
,optimizationoverview
》optimizingSQLStatementS
》optimizationandIndeXeS
》optimizingDatabaseStructure
》optimiZingforInnoDB殆bles
》optimiZingforMylSAM殆bles
·optimizingforMEMORY殆bles
,UnderstandingtheQueryExecution
P}dn
·optimiZingQuerieSWithEXPLAIN
.EXPLAIN0utputFormat
InolderMySQLreleases,extendedinformationwasproducedusingExP几入工NExTENDED.Thatsyntax15stillrecognizedforbackward
compatibilitybutextendedoutput15nowenabledbydefault,50theEx丁ENDE二·keyword15supe闭uousanddeprecated.Itsuseresultsina
Warning,anditWillberemovedfrom婴里三丛丝SyntaxinafutureMySQLrelease.
Here15anexampleOfextended醚里二丛丝output:
mysql>EXPLAIN
SELECTtla,tlaIN(SELECTtZaFROHtZ)FROMtl\G
良
***************************
1.ro树
***************************
123456789012
111
ExtendedEXPLAINoutput
FOFm日t
.obtainingExecutionPlan
Inf0Fmationf0FdN日med
ConneCtion
·EStim日tingQueryPerformance
ld:1
select_type:pRIMARY
table:tl
type:Index
poSSlble_keys:NULL
key:pRIMARY
key_len:4
ref:NULL
r0WS:4

### 能干嘛

* 表的读取顺序
* 数据读取操作的操作类型
* 哪些索引可以使用
* 哪些索引被实际使用
* 表之间的引用
* 每张表有多少行被优化器查询

### 怎么玩

* Explain + SQL语句

计算机生成了可选文字:
＋一一一～＋一一一一一－-－一一一一一＋~－一一一一一＋~～一－----～一～一＋--－一一一＋～一－一一一－－一－－一一一一＋~－一一一一＋-－一一一一－－一＋一一一－一一＋一一－-－一＋
+---－一一一＋
id1select_type}table}partitions
type1possible_keys1key
key--len1ref
FOWS
filtered1Extra
+～一一一＋一一一一－一一一一一～～一＋一一一一～一一＋
＋一一一一一一一＋

**EXPLAIN输出项（来源于mysql5.7文档）**

|  |  |  |
| --- | --- | --- |
| **Column** | **JSON Name** | **Meaning** |
| id | select\_id | The SELECT identifier |
| select\_type | None | The SELECT type |
| table | table\_name | The table for the output row |
| partitions | partitions | The matching partitions |
| type | access\_type | The join type |
| possible\_keys | possible\_keys | The possible indexes to choose |
| key | key | The index actually chosen |
| key\_len | key\_length | The length of the chosen key |
| ref | ref | The columns compared to the index |
| rows | rows | Estimate of rows to be examined |
| filtered | filtered | Percentage of rows filtered by table condition |
| Extra | None | Additional information |

翻译：

| **列** | **含义** |
| --- | --- |
| [id](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_id) | 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 |
| [select\_type](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_select_type) | 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 |
| [table](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_table) | 显示这一行的数据是关于哪张表的 |
| [partitions](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_partitions) | 匹配的分区,一般不用,直接忽略 |
| [type](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_type) | 联接类型 |
| [possible\_keys](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_possible_keys) | 可能用到的索引 |
| [**key**](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_key) | 实际选择的索引 |
| [key\_len](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_key_len) | 所选键的长度,值越大越好 |
| [ref](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_ref) | 显示索引的哪一列被使用了 |
| [rows](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_rows) | 估计要检查的行 |
| [filtered](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_filtered) | 按表条件过滤的行百分比 |
| [Extra](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_extra) | 附加信息,尤其要注意出现using filesort的情况,要尽量避免 |

### 建表语句

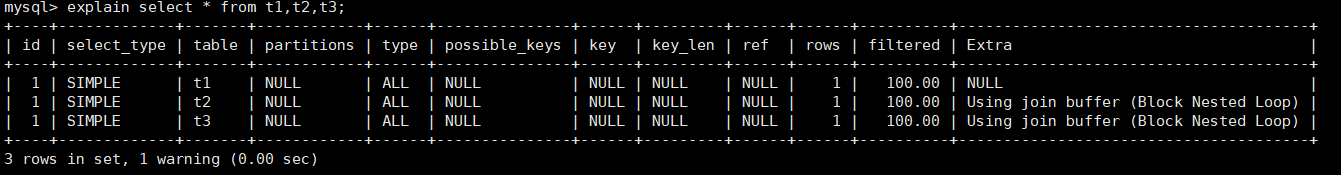
|  |
| --- |
| CREATE TABLE t1(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE t2(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE t3(id INT(10) AUTO\_INCREMENT,content VARCHAR(100) NULL , PRIMARY KEY (id));  CREATE TABLE 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))); |

# 索引优化分析\_explain\_各字段解释

## id★

1. 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
2. 三种情况
3. id相同，执行顺序由上至下

Explain select \* from t1,t2,t3;



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

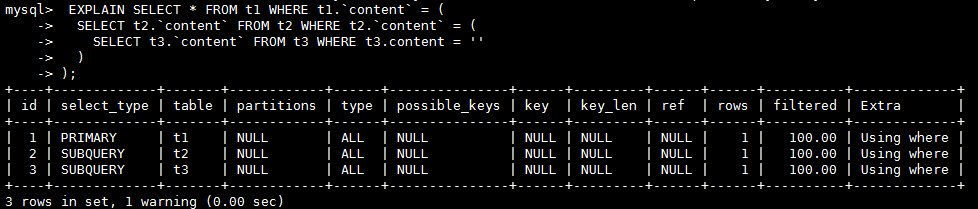
EXPLAIN SELECT \* FROM t1 WHERE t1.`content` = (

SELECT t2.`content` FROM t2 WHERE t2.`content` = (

SELECT t3.`content` FROM t3 WHERE t3.content = ''

)

)

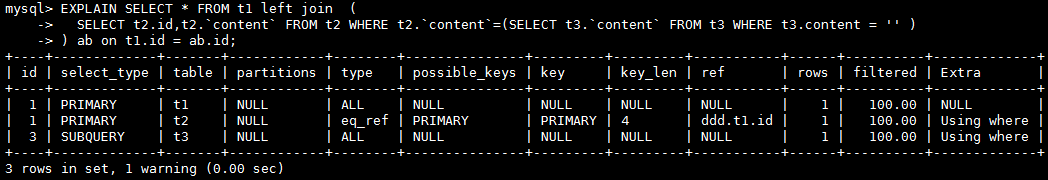


3）. id既有相同又有不同,同时存在: 先按着id从大到小，id相同(可认为是一组)再按着从上到下的顺序

EXPLAIN SELECT \* FROM t1 left join (

SELECT t2.id,t2.`content` FROM t2 WHERE t2.`content`=(SELECT t3.`content` FROM t3 WHERE t3.content = '' )

) ab on t1.id = ab.id;



* 关注点：1.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

1. 有哪些



1. 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
2. SIMPLE

* 简单的 select 查询,查询中不包含子查询或者UNION

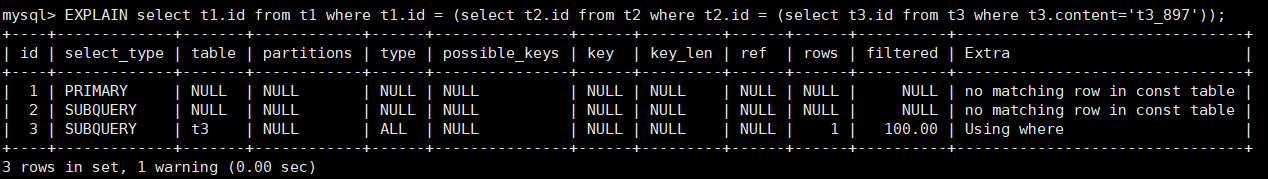
EXPLAIN SELECT \* FROM t1;

计算机生成了可选文字:
mysql>explainselect*fromtl
＋一～~
1id
一＋一～~~-~~~～一一～~+
一＋一～~~~
～一＋~~~~～一＋一～~~~~~~~~~~~~~+~~~~~~＋一～~~~~~～一＋-~~~～一＋一～~~~~+~~~~~~~~~~+~~~~~～一＋
select_type
table
partitlons
type
poSSible_keys
key
key--len
Fef
FOWS
filteFed
EXtFa
11SIHpLE
NULL
ALL
NULL
NULL
NULL
NULL
2
100.00
NULL
t1
1rowinset,1warning(6.00sec)

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

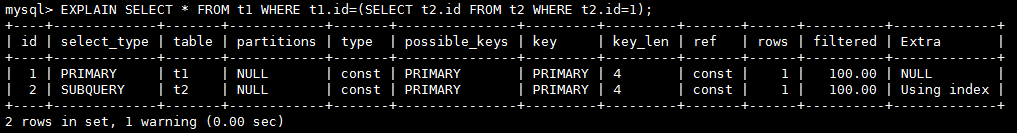
EXPLAIN select t1.id from t1 where t1.id = (select t2.id from t2 where t2.id = (select t3.id from t3 where t3.content='t3\_897'));



1. SUBQUERY

* 在SELECT或WHERE列表中包含了子查询

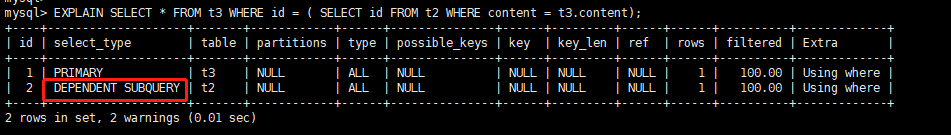
EXPLAIN SELECT \* FROM t1 WHERE t1.id=(SELECT t2.id FROM t2 WHERE t2.id=1);



1. DEPENDENT SUBQUERY

* 在SELECT或WHERE列表中包含了子查询,子查询基于(依赖于)外层查询

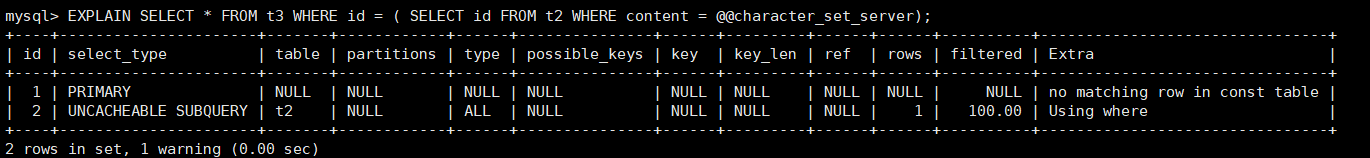
EXPLAIN SELECT \* FROM t3 WHERE id = ( SELECT id FROM t2 WHERE content = t3.content);



1. UNCACHEABLE SUBQUREY

* 表示这个subquery的查询要受到**外部系统变量**的影响

EXPLAIN SELECT \* FROM t3 WHERE id = ( SELECT id FROM t2 WHERE content = @@character\_set\_server);

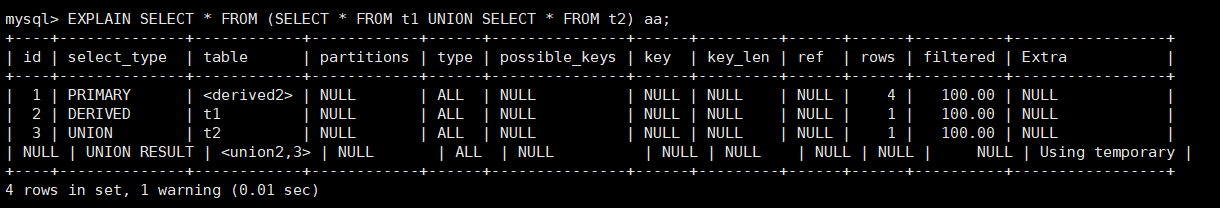


只要我们的SQL语句中出现SQL系统变量（@@sort\_buffer\_size:两个@后面有字符串:这就是系统变量），但凡是SQL中出现系统变量，MySQL就会认为这是不可用缓存的子查询,因为你用了变量，MySQL服务器就认为这是经常变动的，所以就不会去缓存中查询了。

1. UNION

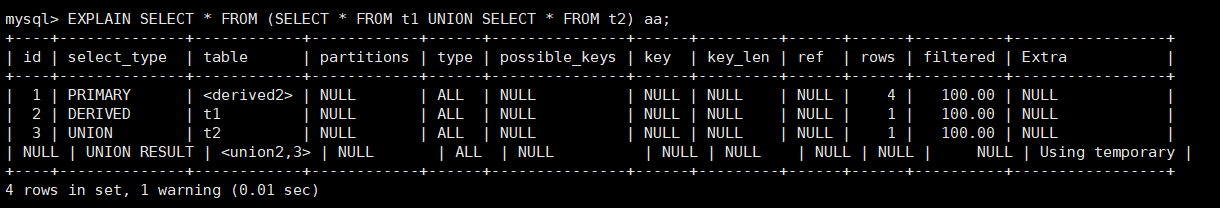
* 若第二个SELECT出现在UNION之后，则被标记为UNION； 若UNION包含在FROM子句的子查询中,外层SELECT将被标记为：DERIVED

EXPLAIN SELECT \* FROM (SELECT \* FROM t1 UNION SELECT \* FROM t2) aa;



1. UNION RESULT

* 从UNION表获取结果的SELECT



## table

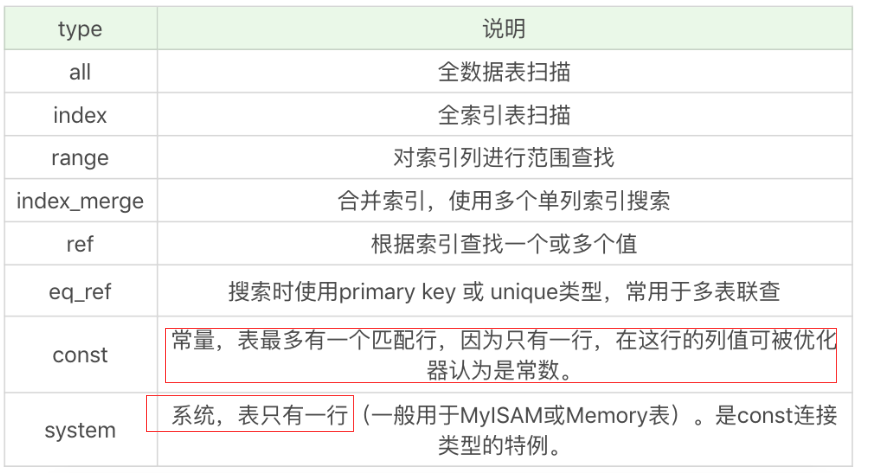
* 显示这一行的数据是关于哪张表的

## partitions

* 代表分区表中的命中情况，非分区表，该项为null
* <https://dev.mysql.com/doc/refman/5.7/en/alter-table-partition-operations.html>

## type★

计算机生成了可选文字:
自LL
indeX
r日nge
ref
eq_ref
const,s梦steo1HULL



1. 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**

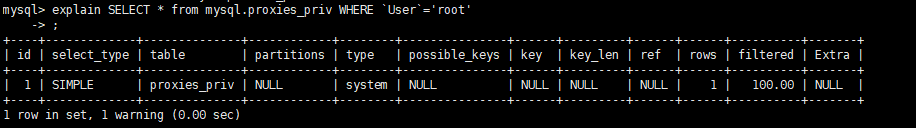
一般来说，得保证查询至少达到**range**级别，最好能达到**ref**。

1. 类型介绍

### system

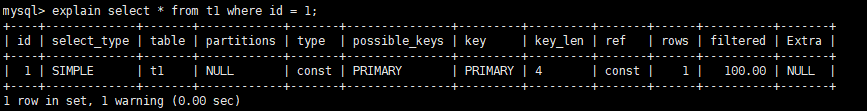
* 表仅有**一行记录**，必须是系统表，这是const类型的特例，查询起来非常迅速。

explain SELECT \* from mysql.proxies\_priv WHERE `User`='root';



### const

explain select \* from t1 where id = 1;



* 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

explain select \* from t1,t2 where t1.id = t2.id;

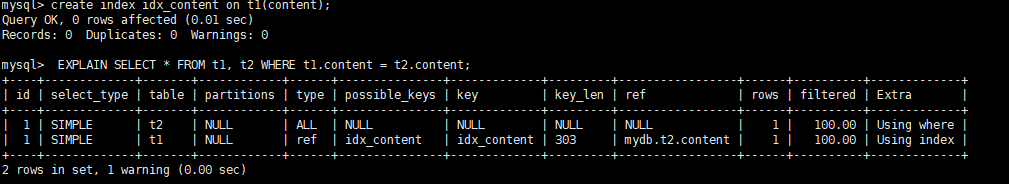
计算机生成了可选文字:
mysql>
＋一一～一＋一
}id!
explainselect*fromtl,tZwheretl.id=tZ.id;
一一＋一
一＋一
一＋一～一－~~～一＋~~~~-~～一一－~~~～一＋~~~～一～～一一＋一一一一一～~～一＋一
一＋一～一一一～+~~~~～一～~~~＋一一一－~～一＋
select_type
table
partitions
type
p0SSible_keys
key
key--len
ref
rOWS
filtered
EXtra
，上月工
，几勺乙
tt
SIHPLE
SIHPLE
NULL
NULL
1ALL}
1eq_refl
PRIHARY
pRIMARY
NULL
PRIHARY
NULL
4
NULL
mydb.tl.id
106。00
166。66
NULL
NULL
，几，几
＋一一一一＋－一一一一一－---－一一＋~
---－一＋一－一－----－一＋一－--－一一＋
2rowsinset,
1warning(0.00sec)

* 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。常见主键或唯一索引扫描
* 建议：小表作为驱动表[全表扫描]，大表作为被驱动表

### ref \*

create index idx\_content on t1(content);

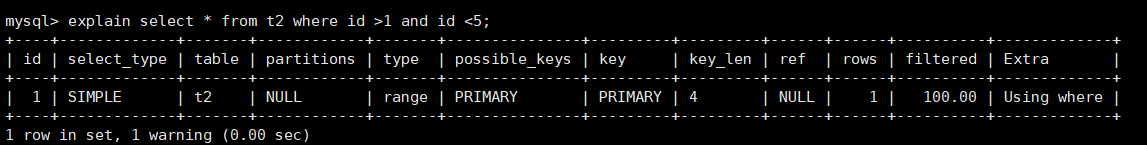
EXPLAIN SELECT \* FROM t1, t2 WHERE t1.content = t2.content;



* 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 \*

explain select \* from t2 where id >1 and id <5;



* 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 **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

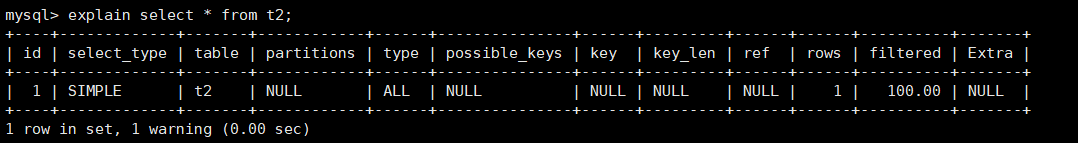
explain select id from t1;

计算机生成了可选文字:
mysql>
＋一一～一＋一
}id!
explainselectidfromtl;
一一＋~~～一～～一＋~~~
一＋一
一一＋一
一一＋~～一～~
一＋一
～一＋一～~~～一＋一
一＋一～~-～一～一～~＋一
一＋
selecttype
table
partitions
type
p0SSible_keys
key
ref
rOWS
filtered
EXtrs
1}SIHPLE
NULL
indeX
NULL
PRIHARY
key--len
4
NULL
1
106.00
Usingindex
t1
1rowinset,1warning(6.66sec)

* 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
* Full Index Scan，index与ALL区别为index类型**只遍历索引树**。
* 这通常比ALL快，因为索引文件通常比数据文件小。
* 也就是说虽然all和Index都是读全表，但index是从索引中读取的，而all是从硬盘中读的

### all

explain select \* from t2;



* Full Table Scan，将遍历全表以找到匹配的行

## 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

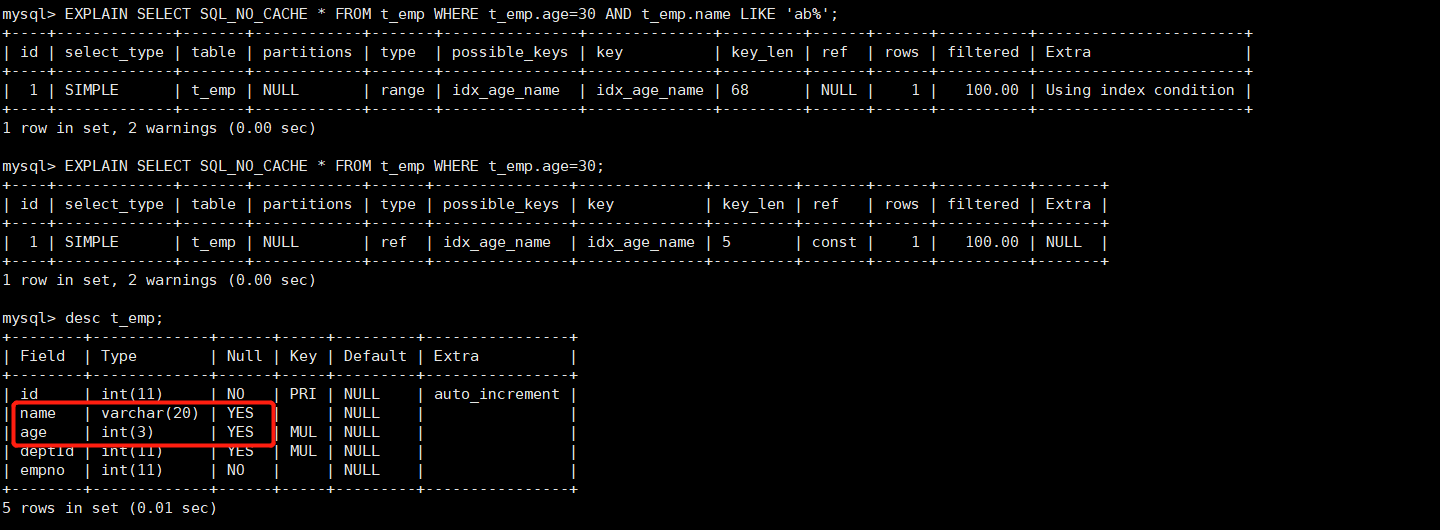
计算机生成了可选文字:
mysql>explainselect*fromtl
一一＋----－一＋-------－一＋一一一－－一＋－一一－－一＋-－一－一－--－一＋-----－一＋
select_type
table
partitions
type
p0SSible_keys
key
key--len
Fef
ro树S
flltered
EXtFS
id
＋一－一一＋-－一一一－--－一一一一＋--－一一一一＋-－一一－---－一一一＋一一－－一一＋一一－－一一－--－一一一一一＋--－一－一＋--－一－--－一＋一一一－一一＋一一一－－一＋一－一－一－--－一＋一一一－－一一＋
11SIHpL〔
NULL
ALL
NULL
NULL
NULL
NULL
1
100。00
NULL
t1
1rowinset,1
Warnlng
(6.66sec)
id
mysql>explainselect
fromtl:
-－一一＋一－-－一一一－－一＋一一一－一一－----－一＋
select_type
table
partitions
type
p0SSible_keys
key
key_len
4
ref
ro树S
filtered
EXtrs
id
--－一＋一－一－-－一－－一＋一－一－-－一－---－一＋
11SIHpLE
NULL
IndeX
NULL
PRIHARY
NULL
1
106.60
Usingindex
t1
1r0w
set,1
Warnlng
(6.61sec)
.1n

## key\_len ★

* key\_len表示索引使用的字节数，根据这个值可以判断索引的使用情况，特别是在组合索引的时候，判断该索引有多少部分被使用到非常重要。**值越大越好**。

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM t\_emp WHERE t\_emp.age=30 AND t\_emp.name LIKE 'ab%';

create index idx\_age\_name on t\_emp (age,name);



* 如何计算

第一组

key\_len=age的字节长度+name的字节长度=（4+1） + ( 20\*3+2+1) = 5+63 = 68

第二组

key\_len=age的字节长度=4+1=5

计算机生成了可选文字:
歹楼型
KEYLEN
备注
刚int
key_len=4+1
=5
允许NULL，加1一。yte
idintnotnUI!
key＿沦n二4
不允许NULL
userchar(30)utts
key＿犯n二
30*3+1
允许NOLL
uservarchar(30)
nU!!Utts
not
Utts
key＿犯n=
30*3+2
动态列类型，加2一。ytes
uservarchar(30)
key＿犯n=
30*3+2+1
动态列类型，加2一bytes，允许NuLL，再加1一byte
detailteXt(10)utts
key＿沦n=
30*3+2+1
TEXT歹l］截取部分，被视为动态歹l］类型，加2-
bytes，且允许NULL

**key\_len的长度计算公式：**

**varchr(10)变长字段且允许NULL    =  10 \* ( character set：utf8=3,gbk=2,latin1=1)+1(NULL)+2(变长字段)**

**varchr(10)变长字段且不允许NULL =  10 \* ( character set：utf8=3,gbk=2,latin1=1)+2(变长字段)**

**char(10)固定字段且允许NULL        =  10 \* ( character set：utf8=3,gbk=2,latin1=1)+1(NULL)**

**char(10)固定字段且不允许NULL    =  10 \* ( character set：utf8=3,gbk=2,latin1=1)**

**小结: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，

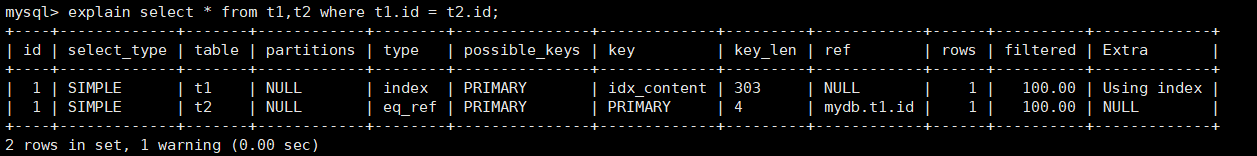
3 、varchar、text这种动态字符串要加2个字节

4、 允许为空的字段要加1个字节

## ref:意义不大

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。

explain select \* from t1,t2 where t1.id = t2.id;



## rows★:值越小越好

* rows列显示MySQL认为它执行查询时必须检查的行数。值越小越好,速度越快,注意这是个预估值，因为这是用explain关键字模拟执行的，不是真实执行的。

## filtered:意义不大，直接过

* 这个字段表示存储引擎返回的数据在mysql server层过滤后，剩下多少满足查询的记录数量的比例,不是具体记录数。最大值为100，这表示未过滤行。值从100减小表示过滤量增加。 rows显示检查的估计行数，rows× filtered显示将与下表连接的行数。例如，如果 rows为1000且 filtered为50.00（50％），则与下表连接的行数为1000×50％= 500。

## 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  \*:排序未用上索引

* 出现filesort的情况：**order by 没有用上索引**。

计算机生成了可选文字:
nlysql>resetqueryCache;
QueryOK,0rowsaffected(0.00sec
mysql>exPlainselectid，印pno,ename,]obfro。曰pwheredeptno=100orderbyename
+~~～一＋~~~~~--~--~~~+~---~~~+~--~~~+～·－～·～~~~~－一一～~+
1id!selectjype1table}type1possiblesekey,l
~~-~～一～~~~~~+~-~~~~-～一＋~~~~～一
key
!key--1en1ref
1加it10:
+~~~~-－一＋~~~---～·～·一一一～~~~~～····～~-~～一＋
!ro仍！Extral
11}51海工EI印plreflidx--deptnolidx--deptnolsIconst}591908}Using确ere:Usingfileso改l
+-－一＋----------－一＋----－一＋---－一＋------------－一＋---------－一＋------－一＋----－一＋-----－一＋一二二二二二二二二二二二二二二二二＋
1ro树inset(0.00sec
mysql>selectid，印pno,ename,jobfrom,p诵eredeptno=100orderbyenaoel加it10;
------－一＋------－一＋-----－一＋-------－一＋
id
日价pno
ename!]ob
+i
+---~~~~－一＋-~~~~-~~~+~~~~-－一＋~-~-~~-~~~+
11148142}1248143}aaaaaa1SALE洲AN}
}2575746}26757471aaaaaa1SALE驯AN}
}2569571!26695721aAAaAA1SALE驯AN!
13285315133853161aaaaaa1SALESNA刊｝
13556381}36563821aaaaab1SAI-ESNA刊｝
!3555069}36550701aAAaAB1SALESNA刊｝
13068712131687131AAaABg!SALESMA刊｝
}1819739}1919740}AAaABh!SALE驯AN}
11114073}12140741AaAAbH1SALE驯AN}
113011111140llU1AaAAbJ1SALE驯AN}
+－··……＋·····－－二＋····－－二＋···－-－·－二＋
10rowsinset(4.61sec】

* 优化后（给deptno和ename字段建立复合索引），去掉filesort

create index idx\_deptno\_ename on emp (deptno,ename);

计算机生成了可选文字:
。ysql>resetquerycache;
Query以，0ro呱affected(0.00sec)
mysql>exPlainselectid.,pno,enaoe.]obfro。曰p油eredeptno=100orderbyen溯el训it10;
',..!+!
·，..!+!
id1selectjype1table1type1possibleeskeys
key
1151陌tE
印p
ref1idxesdeptno.idxesdeptno夕name1idxesdeptno夕name
ref
COnst
ro树S
591908
＋··－·－·－··～－一＋
1Extral
＋·····～·～·～·～一＋
1using喃erel
＋·····～·～·～·～一＋
+!+l十
1ro树InSet(0．的SeC
mysql>selectid,,pno,ename.]obfrom印p油eredeptno=100orderbyenamel训it10;
＋·－·－·－·一＋·－·－··…＋
1idl曰pno!
＋·－·－·－·－·＋·－·……＋
·－·－-－一＋－···－·－·一＋
enamel]ob
···－·－一＋－···－·－·一＋
11148142!U481431aaaaaa!SALESN酬！
12569571126695721aAAaAA1SALESNA刊l
12575746126757471aaaaaa1SALE翎ANI
!3285315!33853161aaaaaa1SALESNA刊l
1355506913655070!aAAaAS1SALESN酬l
13556381136563821aaaaab1SALESNA刊l
130687U13168713!AAaA8g1SALESN酬l
111140731U140741AaAAbH1SALESNA刊l
11819739!19197401AAaA8h1SALE翎ANI
!13011111140llU!AaAAbJ1SALESNA刊l
+----~-－一十－-----－一十－--，甲－一十－------－一十
10ro咙inset(0.00sec)

* 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”
* 需要注意的是：由于出现了where查询条件，所以我们在创建索引的时候创建的是复合索引(where后面的字段和order by后面的字段一起创建复合索引，单独为order by后面的字段创建索引无法生效哦。)
* 面试题：什么时候需要创建复合索引？就是当出现where查询条件，并且需要排序的时候。||或者需要根据多个条件查询的时候。

### Using temporary \*:分组没用上索引

* 出现Using temporary情况：**分组没有用上索引**。产生临时表。注意：**分组操作是需要先排序后分组的**。所以，也会出现Using filesort。
* 优化前存在 using temporary 和 using filesort

计算机生成了可选文字:
mysql>resetquerycache;
QueryOK,0rowsaffected(0.00sec)
mysql>explainselectename,count(*)from印p汕eredeptno=100groupbyenamel确it19;
id1select一ype1table1type1possibleeskeys1key
1key--len}ref
FO树S
EXtF日
!+
1151呼tE
印plreflidX--deptno
1idX--deptno15
COnst
591908!Using喃ere:Usingt,porary;Usingfilesort
1rowinset(0.00sec)
。y,ql>,electen溯e,count(*)from印p汕eredeptno=100groupbyen抓el加it19
＋··……＋······～…＋
1ename1count(*)l
＋·－---－二＋····－··…＋
1aaaaaa141
}aAAaAB121
}AAaABg111
1AaAAbH121
1AaAAb」141
1aaaacm141
1aaaadm121
1aaaadn131
1AaAAYu111
1aaabdn121
+~~---－一＋-------～一＋
10ro树5inset(7.24sec)

* 优化后（给deptno和ename建立复合索引）去掉using temporary 和 using filesort，性能发生明显变化：
* 上面在出现using temporary的同时也出现了using filesort，这说明分组必排序。

create index idx\_deptno\_ename on emp (deptno,ename);

计算机生成了可选文字:
。ysql>resetquerycache;
Query以，0ro呱affected(0.00sec)
mysql>exPlainselectid.,pno,enaoe.]obfro。曰p油eredeptno=100orderbyen溯el训it10;
',..!+!
·，..!+!
id1selectjype1table1type1possibleeskeys
key
1151陌tE
印p
ref1idxesdeptno.idxesdeptno夕name1idxesdeptno夕name
ref
COnst
ro树S
591908
＋··－·－·－··～－一＋
1Extral
＋·····～·～·～·～一＋
1using喃erel
＋·····～·～·～·～一＋
+!+l十
1ro树InSet(0．的SeC
mysql>selectid,,pno,ename.]obfrom印p油eredeptno=100orderbyenamel训it10;
＋·－·－·－·一＋·－·－··…＋
1idl曰pno!
＋·－·－·－·－·＋·－·……＋
·－·－-－一＋－···－·－·一＋
enamel]ob
···－·－一＋－···－·－·一＋
11148142!U481431aaaaaa!SALESN酬！
12569571126695721aAAaAA1SALESNA刊l
12575746126757471aaaaaa1SALE翎ANI
!3285315!33853161aaaaaa1SALESNA刊l
1355506913655070!aAAaAS1SALESN酬l
13556381136563821aaaaab1SALESNA刊l
130687U13168713!AAaA8g1SALESN酬l
111140731U140741AaAAbH1SALESNA刊l
11819739!19197401AAaA8h1SALE翎ANI
!13011111140llU!AaAAbJ1SALESNA刊l
+----~-－一十－-----－一十－--，甲－一十－------－一十
10ro咙inset(0.00sec)

* 使用了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 \*:使用了覆盖索引

表示使用了**覆盖索引** [content是一个索引]

计算机生成了可选文字:
mysql>explainselect*fromtlwheretl.contentlike'ab%';
id1select_type
tsble
partitions1type
possible_keys
key
key--len
ref
r0WS
filteFed
EXtFS
11SIMPLE
t1
NULL
indeX
idXCtnt
idXCtnt
303
NULL
1
100。00
Usingwhere;Usingindex
1r0winSet
1warning(0.00
SeC

* 如果同时出现using where，表明索引被用来执行索引键值的查找; 通常出现在where 后面的字段没有用到索引的情况下。

计算机生成了可选文字:
mysql>explainselect*fromtl;
＋一～一一＋~~~～一～~~～一一一一＋~~~～一一～+~~~~~~~~~~~~+~~~~~～一＋一～~~~~~~~~~~~~～介－~~~~~~~~~+~～一～~~~～一＋一～~~~~+~~~~~~+~～一一～~~~~~+~~~~~~~~~~~~~+
id}select_type
table
partitions
type
p0Ssible_keys
key
key_len1ref
rows1filtered1Extra
1!SIHpLE
t1
NULL
indeX
NULL
idXCtnt
363
NULL
1
106。68
Usingindex
1FO树inSet
1warning(6.00
SeC

* 如果没有同时出现using where，表明索引用来读取数据而非执行查找动作。

### Using where|使用了where过滤[where过滤条件没有用到索引]

* 表明使用了where过滤！
* 一般情况筛选结果不止一条的时候会出现using where;

### using join buffer \*|关联查询字段没有用到索引

被驱动表的联表字段没有加索引。

如果有它则表明**关联字段没有**使用索引！

计算机生成了可选文字:
mysql>
E冷tAINSELECT*FROH印p,dept"{REdept.deptno＝曰p．、deptno、；
id}select一ype1table1type1possibleeskeys1key1key--1en1ref1ro"
EXtr己
51片tE
51协礼E
dept
日盯p
Al-L!M上L
八LLI限LL
M工LI限LL
限几LI限LL
M上L
M门！
10596
3077520
Using确ere;Usingjoinbuffer
2FO树5In
set(0.00sec)

* 使用了连接缓存

### impossible where|出现了不可能出现的where条件

计算机生成了可选文字:
z3
nlysql>EXPLAINSEL〔cT*rRoMstaffs
ANDNAME=
+l+I+
一一｝
+l+l
一一
l+l
一
记1select一ype
table
type
wH〔R〔N阴〔二’〕uly'
~~~~～一～~~~~~~~+~~~~
possible_keys!key
key--1en!
EXtra
＋一一一一＋
111
一一一一一一一＋一一一一一
NULL1NULL
NULL
一＋
SIMpLE
ref
NULL
ro例5
NULL
ImpossiblewH〔R〔
l+l
一
一L
一L
一U
一N
一
+l
一
一
一
一
一
一
一
一
一
一
一L
一L
一U
一N
一
l+l
一
l+l
一
1rowin
set(0.00sec)

* where 后面筛选条件有错误！

## 11.13拓展面试题：

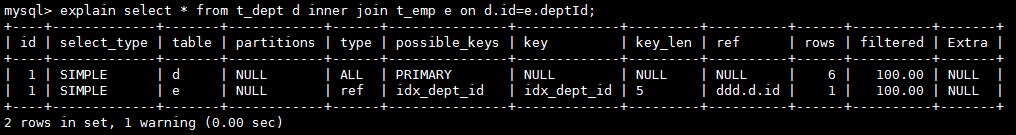
where和inner join有区别不？

where 会出现笛卡尔积

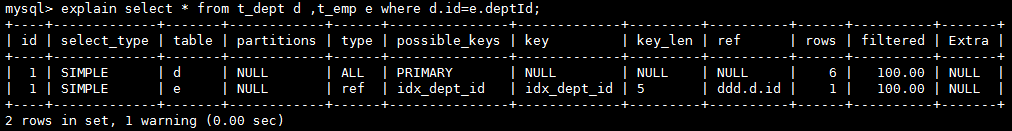
inner join 不会出现笛卡尔积，建议使用inner join连接

mysql5.7底层已经做了优化，mysql底层会将where查询转换成inner join,所以咱们看到效率是一样的。

explain select \* from t\_dept d inner join t\_emp e on d.id=e.deptId;



explain select \* from t\_dept d ,t\_emp e where d.id=e.deptId;



Inner join 会在on的条件上，从被驱动表中筛选出符合条件的条数。

Where不看条件，直接笛卡尔积。

但是5.5之后，where会被优化器优化成inner join

# 查询优化\_准备数据-保存50万条数据

## 员工表插50W条，部门表插10W条

### 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bin-log, 我们就必须为我们的function指定一个参数。  # 主从复制，主机会将写操作记录在bin-log日志中。从机读取bin-log日志，执行语句来同步数据。  # 如果使用函数来操作数据，会导致从机和主键操作时间不一致。所以，**默认情况下，mysql不开启创建函数设置**  #**查看mysql是否允许创建函数：**  show variables like 'log\_bin\_trust\_function\_creators';  #命令开启：允许创建函数设置：  set **global** log\_bin\_trust\_function\_creators=1;  # 不加global只是当前窗口有效。mysqld重启，上述参数又会消失。  #永久方法：   * 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 $$  CREATE FUNCTION **rand\_string**(**n** INT) RETURNS VARCHAR(255)  BEGIN  DECLARE chars\_str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  DECLARE return\_str VARCHAR(255) DEFAULT '';  DECLARE i INT DEFAULT 0;  WHILE i < **n** DO  SET return\_str =CONCAT(return\_str,SUBSTRING(chars\_str,FLOOR(1+**RAN**D()\*52),1));  SET i = i + 1;  END WHILE;  RETURN return\_str;  END $$    #假如要删除  DELIMITER ;  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  DELIMITER ;  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 $$  CREATE **PROCEDURE** insert\_emp( START INT , max\_num INT )  BEGIN  DECLARE i INT DEFAULT 0;  SET autocommit = 0; #设置手动提交事务  **REPEAT** #循环  **SET** i = i + 1; #赋值  INSERT INTO emp (empno, NAME ,age ,deptid ) VALUES ((START+i),rand\_string(6),rand\_num(30,50),rand\_num(1,10000));  **UNTIL** i = max\_num  **END REPEAT**;  **COMMIT**; #提交事务  END$$    #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10万条数据  CALL insert\_dept(100000); |

* Emp

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

## 批量删除某个表上的所有索引

### 建好的索引存在哪了？

SHOW INDEX FROM t\_emp ; 只能查看索引，但不能删除。

Drop index idx\_name on table\_name;

`information\_schema`.`STATISTICS` 存储索引的表（元数据库，统计表），我们可以对表数据进行删除操作。

### 知识点

* 1. 删除某一个索引

DROP INDEX idx\_xxx ON emp

* 1. 查出该表有哪些索引，索引名-->集合

SHOW INDEX FROM t\_emp

元数据：meta DATA 描述数据的数据

SELECT index\_name FROM **information\_schema.STATISTICS** WHERE table\_name='t\_emp' AND table\_schema='mydb'

AND index\_name <>'PRIMARY' AND seq\_in\_index = 1

* 1. 如何循环集合

CURSOR 游标

FETCH xxx INTO xxx

* 1. 如何让mysql执行一个字符串

PREPARE XXX 预编译语句

EXECUTE

* 1. 如何执行存储过程

CALL proc\_drop\_index ('mydb','t\_emp');

### 创建存储过程

删除表上所有索引

|  |
| --- |
| **DELIMITER** $$  CREATE **PROCEDURE** `proc\_drop\_index`(dbname VARCHAR(200),tablename VARCHAR(200))  BEGIN  DECLARE done INT DEFAULT 0;  DECLARE ct INT DEFAULT 0;  DECLARE \_index VARCHAR(200) DEFAULT '';  DECLARE \_cur CURSOR FOR **SELECT index\_name FROM information\_schema.STATISTICS WHERE table\_schema=dbname AND table\_name=tablename AND seq\_in\_index=1 AND index\_name <>'PRIMARY'** ;  #每个游标必须使用不同的declare continue handler for not found set done=1来控制游标的结束  **DECLARE CONTINUE HANDLER FOR NOT FOUND** set done=2 ;  #若没有数据返回,程序继续,并将变量done设为2  OPEN \_cur;  **FETCH \_cur INTO** \_index;  WHILE **\_index<>''** DO  SET @str = CONCAT("drop index " , \_index , " on " , tablename );  **PREPARE** sql\_str FROM @str ;  **EXECUTE** sql\_str;  **DEALLOCATE PREPARE** sql\_str;  SET \_index='';  FETCH \_cur INTO \_index;  END WHILE;  **CLOSE** \_cur;  END$$ |

### 执行存储过程

|  |
| --- |
| **CALL** proc\_drop\_index("mydb","emp");  **CALL** proc\_drop\_index("mydb","dept"); |

# 查询优化\_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

1. 系统中经常出现的sql语句如下：

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

1. 索引应该如何建立 ？

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

1. 建立索引前

计算机生成了可选文字:
mysql>
Empty
SELECT以荞‘N()--CACHE＊阳OH印p阳ERE印p.ag卜30ANDdeptid碑灿幻印p.name='abcd'
set(0.15sec

1. 建立索引后

计算机生成了可选文字:
mysql>
Empty
SELECT以l‘即esCAO{*FROH印p阳ERE翻p.ag卜30刊M〕deptid碑封幻曰p.name='abcd'
set(0.00sec

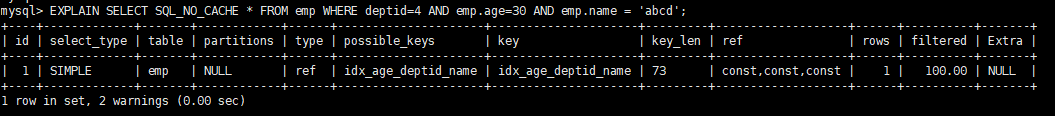
### 最佳左前缀法则

1. 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？不能

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



这说明mysql的优化器optimizer在起作用。

1. 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' ; # 部分被使用

计算机生成了可选文字:
mysql>EXPLAINSELECTSQL--NO_CACHE*FROHempWHEREemp.age=30
ANDemp.name='abcd"
＋一～一～＋一～
一＋~~~~~～一＋-~~~~~~~~~~~+~~~~~~＋一一～~~～一～~
～一＋一～~～一～一＋~~~~~~~~~～十～~~~~~~~～一一～~+
select_type
table
partitions
type
p0SSible_keys
key
key--len
ref
rOWS
filtered
EXtra
id
1}SIMPLE
emp
NULL
ref
id左age,id礼age_deptid,id左age_deptld_
name
id犯age15
Const
46624
10。06
Usingwhero
1rowinset,2warnings(0.00sec)

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.deptid=1 AND emp.name = 'abcd'; # 完全没有

计算机生成了可选文字:
mysql>EXPLAINSELECTSOL_NO_CACHE*FROHempWHEREemp.deptid=1ANDemp.name='abcd';
＋一～～一＋一
1id}
+~~~～一～一＋一～~~~
～一＋~~~~～一＋一～~~~~~~~~~~~~~+~~~~~~＋一～~~~~~～一＋~~~~～一＋一～~~~～一～+~~~~~~~~~~+～一～~~~~～一～~～一干
select_type
table
partitlons
type
poSSible_keys
key
key--len
Fef
FOWS
filteFed
EXtFa
11SIHpLE
omp
NULL
ALL
NULL
NULL
NULL
NULL
499086
1。06
Usingwhere
1rowinset,2warnings(0.00sec)

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。

如果where 条件后面索引都存在的情况下，mysql 的优化器会自动使用索引

### 计算、函数导致索引失效

1. 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';

1. 创建索引

CREATE INDEX idx\_name ON emp(NAME);

1. 第一种：索引优化生效

计算机生成了可选文字:
mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempWHERE
emp.nameLIKE"abc%”万
＋一～～一＋一
1idl
~~~~~～一～～一一～+~~～一一一一＋~~~~~
一＋~～一～～一＋~~~～一一＋~~~~~~~~~～可》一～~~~~~～一一一～~~~~~~~~~～一十
select_type
table
partitions
type
possible_keys
key
key--len
Fef
rows}filtered}Extra
11SIHpLE
0lflp
NULL
range
idXDame
idXnallle163
NULL
20
166。90
Usingindoxcondition
1rowinset,2warnings(0.00sec

计算机生成了可选文字:
mysql>SELECTSQL--NO_CACHE*FROHempWHERE
emp。nsme
LIKE'abC%'
＋一一一～~~~~+~～一～~~～一＋~~~~~～一一＋~~~~
id
empno1name
age
一一＋一～～一一～-
}deptld
(UCO一I.I）勺j叹J勺J叹JCO4118CO叹JOUC乙月呀月呀，了，一勺」
534334月呀34月呀4勺J月呀4勺J33月呀33
63772
473858
141845
136129
317458
350895
155530
406805
342215
182338
349935
401745
28242
399671
171686
16970
375130
99316
351848
479536
163772
573858
241845
230129
417458
450895
255530
506805
442215
282338
449935
501745
128242
499671
271686
110970
475130
199316
451848
579536
sbCIHO
AbCJNH
AbCJNN
aBCjnP
ABcJpw
AbCKQb
ABCKrC
日BCkre
日BCksh
aBCks]
abCkSJ
AbCKSk
AbCKTO
AbCKUp
ABCLUS
AbCLVU
ABCLVU
aBCIVV
aBClwy
AbCLXF
5447
9143
2497
3598
999
1746
1386
8068
375
7044
3037
4212
8325
3916
63
9034
4698
8697
1080
82
20FOWSinSet
1warning(6.00
SeC

1. 第二种：索引失效

计算机生成了可选文字:
mysql>EXPLAINSELECTSOL_NO_CACHE*FROHempWHERELEFT(emp.name,3)="abc';
＋一一一～+－一一一一一－-－一一一一＋一－－一一一一＋-－一一一－一－-－一一＋一一～~－一＋一一一－－一一一－一一－－一一＋----－一＋一一一－－一一一一＋~-－一一一＋一－-－一－－一＋~--－一一－-－一＋一－----－一－---~+
select_type
tsble
partitlons
type
possible_keys
key
key--len
Fef
FOWS
filtgFed
EXtFa
id
11SIHpL【
emp
NULL
ALL
NULL
1NU比｝NULL
一一＋--~～一一＋一一～~~
}
一－－一＋-
NULL
499686
166.00
Usingwher。
一
＋一－－一＋一一一一一－－一－一一－一＋--－一－
～一一－一＋·······……
1rowinset,2
WaFnlngS
一一＋-－一一一－---－一一＋－一－－一一＋一－
(0.00sec)

计算机生成了可选文字:
mysql>SELECTSQL--NO_CACHE*FROHempWHERELEFT(emp.name,3)='abc
＋一一～--~～一＋~-~~~~～一＋~~~～一一～一＋一～～一一一＋～一～~~~~~+
id
empno
name
age
deptld
月咔反J(l.I一气了勺J日一反J勺乙，了二J月呀no勺J，乙月呀noCOCOOUIJ
勺J415月呀勺J4月呀勺J月呀勺J月呀月呀4勺J34勺J勺J勺J勺」
10970
28242
63772
99316
130129
141845
155536
171686
182338
317458
342215
349935
350895
351848
375130
399671
401745
406805
473858
479536
110970
128242
163772
199316
230129
241845
255530
271686
282338
417458
442215
449935
450895
451848
475136
499671
501745
506805
573858
579536
AbCLVU
AbCKTO
abCIHO
SBCIVV
aBCjnp
AbCJNN
ABCKrC
ABCLUS
aBCksj
ABcjpw
aBCksh
日bCkSJ
AbCKQb
aBClwy
ABCLVU
AbCKUp
AbCKSk
日BCkFe
AbCJNM
AbCLXF
9034
8325
5447
8697
3598
2497
1386
63
7044
999
375
3037
1746
1086
4698
3916
4212
8068
9143
82
26ro树5InSet
1warning(0.21
SeC

### 范围条件右边的列索引失效

* 1. 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' ;

计算机生成了可选文字:
mysql>EXPLAINSELECTSQLNOCACHE*FROHempWHEREemp.age=30ANOemp.deptld>20ANDemp.name="abc'
＋一
~～一＋一
idl
~~~~~~~~~~~~+~～一～~-~+～一～~~~~~~~~
select_type}table}partitions
~~~~~~~~~+~~~~~~~～一＋一～~~~~+~~~~~~~+～一～-~～一～~~+~~~~~~~～一～~~~~~~～一一～~～一干
type
p0SSible_keys
key
key--len
Fef
FOWS
filteFed
EXtFS
11SIHpLE
emp
NULL
ranqe
idxagedeptidna"te
idxagedeptid
name110
NULL
4736B
16。60
Usingindoxcondition
1rowinset,2warnings(0.00sec

* 1. 那么索引 idx\_age\_deptid\_name这个索引还能正常使用么？
* 不能，范围右边的列不能使用。
* 如果这种sql 出现较多，应该建立：

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

* 将范围查询条件放置语句最后：

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

* 应用开发中范围查询，例如： 金额查询，日期查询往往都是范围查询。应将查询条件放置where语句最后。
  1. 效果

计算机生成了可选文字:
mysql>EXPLAINSELECTSQLNOCACHE*
FROHempWHERE
一一＋一－--－一一＋一－-
emp.age=30ANDemp.name='abc"ANDemp.deptld>20
~-－一一－---－一－-－一＋一一一一－-----------－一－一一＋~
select_type
table
partitions
type
p0SSible_keys
key
key--len
ref
ro树S
flltered
EXtra
id
l.f.
73
11SIHPLE
emp
NULL
range
idX--age,idX--age_deptid,idX--age_deptid_name,idX--age_
name_deptid
idX--age_name_deptid
l
·子，二
NULL
1
100。66
Usingindexcondition
，一～＋一一·········，·，·，二
，··一一＋一一一一一一＋～一一一一一＋一一一一一一一一一一＋一一·，·········，……
.1n
1FOW
set,2
W日FnlngS
(0.00sec)

### 不等于(!= 或者<>)索引失效

CREATE INDEX idx\_name ON emp(NAME);

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

计算机生成了可选文字:
mysql>EXPLAINSELECTSQL_NO_CACHE
＋一一一～+－一一一一一－－一一一一一＋-－一一一一一＋-－一－--
*FROHempWHEREemp.name<>'abc
select_type
tsble
partitions
type
p0SSible_keys
key
key--len
Fef
FOWS
filteFed
EXtFS
id
～一～一～~~＋一～~~~~+~~~~~~~~+～一～~~~~~~~+~~~~~~~-~~~~~+
11SIHpL〔
emp
NULL
ALL
NULL
NULL
NULL
NULL
499086
90.60
Usingwhere
·一－···········一＋
+--－一＋一一一一一－－一一一一一一＋-－一－-－一＋一一一一一－---－一一＋－一一一一一＋－一
－一＋-－一一一一＋-－一－---－一＋一－--～一＋··…
1rowinset,2
WaFnlngS
(0.01sec)
mysql>CREATEINDEXidx
name
QueryOK,0
ReCOFdS:6
r0WSaffeCted
Duplicates:0
ONemp(NAHE)
90sec)
Warnings:0
mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempWHEREemp.name<>'abc';
select_type
tsble
partitions
type
possible_keys
key
key--len
ref
ro树S
filtered
EXtFS
id
＋一－-
l1
SIHPLE
emp
NULL
ALL
idXname
NULL
NULL
NULL
499086
50.14
Usingwhere
1rowinset,2
W日FnlngS
(0.00sec)

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE emp.name != 'abc';

### is not null无法使用索引，is null可使用索引

下列哪个sql语句可以用到索引

CREATE INDEX idx\_age ON emp(age);

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE age **IS NULL**;

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE age **IS NOT NULL**;

计算机生成了可选文字:
mysql>CALLproc
QueryOK,0rows
_drop_index("mydb'',
affected(0.02sec
,'e叩’')
)
mysql>CREATEINDEXidX--ageONemp(age);
QuoryOK,0rowsaffocted(1.36soc)
Records:6Duplicates:0Warnings:0
－一＋一－---－一＋－一－－一一＋一一－---－一一一＋--
key_len
5
Fef
FOWS
flltered
EXtFS
mysql>EXpLAINSELECTSQL_NO_CACHE*FROHempWHEREage15NULL;
+--－一＋一一一一－-－一一一一－一＋－一一一一－一＋一一一一－-~－一一一一＋一－－一一一＋---－一一一一－－一一一一一＋-－··一
Select_type
table
partitions
type
p0Sslble_keys
key
id
11SIHPLE
emp
NULL
Fef
id左age
idX--age
Const
1
106。66
Usingindexcondition
1rowinset,2warnings(0.00sec)
mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempWHEREage15NOTNULL;
+--－一＋一一一一一－－一一一一－一＋－一一一一－一＋一－一一一－-－一一一一＋--－一一一＋一～-－一一一－--－一一一一＋-－一－一一＋------－一一＋-－一－一一＋～一－一－-－一非一－------－一＋一－---－一一－---~+
l+l
}
select_type
tsble
partitions
type
possible_keys
key
key--len
Fef
FOWS
filtered
EXtFS
id
-－一－-－一＋一－一－----－一＋一－----－一一一－-
11SIHPLE
emp
NULL
ALL
idX--age
NULL
NULL
NULL
499086
56。60
Usingwhere
+~~~~+~~~~~~～一～~~-~+-~~~~~~+~~~~~～·
1rowinset,2warnings(0.00sec)

### like以通配符%开头索引失效

1. 下列哪个sql语句可以用到索引

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE name LIKE 'ab%';

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE name LIKE '**%**ab%';

计算机生成了可选文字:
mysql>CALLproc
QueryOK,0rows
_drop_index("mydb"',
affected(0.02sec
"'emp',);
)
mysql>CREATEINDEXidx
name
QueryOK,0
ReCOFdS:6
r0WSaffeCted
Duplicates:0
ONemp(name);
81sec)
Warnings:0
mysql>EXPLAINSELECTSQL--NO_CACHE*FROHempWHEREnameLIKE'ab%';
＋一一一一＋一一一一一一一一一一一一一＋一一一一一一一＋一一一一一一一一一一一一＋一一一一一一一＋一一一一一一一一一一一一一一一＋一一一一一～～一～一＋～一一一一一一一一＋一一一一一一＋一一一一一一＋一一一一一一一一一一＋一一一一一一一～~~~~~~～一一一一一一一一＋
select_type
tsble
partitions
type
p0Ssible_keys
key
ke光len
ref
r0WS
filtered
EXtrs
id
63
11SIHPLE
emp
NULL
range
idXname
ldXnsme
NULL
767
166。66
Usingindexcondition
1rowinset,2
WaFnIngs
(0.03sec)
mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempWHERE
nameLIKE'%ab%";
＋一一一一＋一一一－一一一一一一一一一＋一一一一一一～＋一一一一一一一一一一一一＋一一一一一一＋一～～一一一一一一一一一一一一＋一一一一一一＋一一一一一一一一一＋一一一－一一＋～一一一一一一一＋一一一一－一一一一一＋一一一一一一一一一－一一～+
select_type
tsble
partitions
type
possiblekeys
key
key--len
ref
rOWS
filtered
EXtra
id
11SIHPLE
emp
NULL
ALL
NULL
NULL
NULL
NULL
499086
11。11
Usingwhere}
·、、～、～+
1r0w
set,2
WaFnIngs
(0.00sec)
.1n

### 类型转换导致索引失效

1. 下列哪个sql语句可以用到索引

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE name=123;

EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp WHERE name='123';

计算机生成了可选文字:
mysql>CALLproc
QueryOK,0rows
_drop_index("mydb"',''emp"'
affected(0.02sec
mysql>CREATE
INDEXidXnallle
QueryOK,6
Rocords:6
rowsaffected(1
0N
。80
emp
SeC
(name
)
Duplicates:0Warnings:0
mysql>EXPLAINSELECTSQL--NO_CACHE*
＋一～～一＋一一一一～~~～一～一～一＋~～一一一～~+~～一一一～~-~
FROHempWHEREname=123
一一＋~~~～一一＋一～~~～一一一～~-－一一一＋一～~~～一＋~~~~-~～一～+~~～一一一＋一～～一～一一～＋一～一一一～~-~~+-~~-~~~～一一一～一＋
l+l
}
id}Select_type!table
partitions
type1p0SSible_keys1key
key--len!ref
rOWS
filtered1Extra
------－一＋--------－一＋一－--－一一一一－--
11SIHPLE
emp
NULL
ALL
ldXnsme
NULL1NULL
NULL1499086
16。00
Usingwhere
1rowinset,4warnings(0.00sec
mysql>EXPLAINSELECTSQL--NO_CACHE*FROHempWHEREname="123';
＋一～~~＋一一～~~~~~～一一一～+~～一一一～一＋一～一～~~-~~～一一＋一一一一～一＋一～~~~~～一～~--～一一＋一～~~~~～一一一＋～一一一～~~～一＋一～一～~～一＋～一～~~~＋一一一－~～一一一一＋~~～一一一一＋
id}select_type!table!partitions1type1possible_keys1key
key--len1ref
rows1filtered!EXtra
~-－一－---－一－一一一＋一－-----－一～
11SIHPLE
emp
NULL
Fef
ldXnsme
ldXnsme
63
Const
1
100.66!NULL
1rowinset,2warnings(0.00sec

* name=123发生类型转换，索引失效。
* 设计实体类属性时，一定要与数据库字段类型相对应，否则会出现类型转换的情况。

## 小总结练习

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，尽量选择过滤性更好的索引（例如：手机号，邮件，身份证）
* 在选择组合索引的时候，过滤性最好的字段在索引字段顺序中，位置越靠前越好。
* 选择组合索引时，尽量包含where中更多字段的索引
* 组合索引出现范围查询时，尽量把这个字段放在索引次序的最后面
* 尽量避免造成索引失效的情况

# 查询优化\_关联查询优化

## 建表SQL

|  |
| --- |
| #分类  CREATE TABLE IF NOT EXISTS `class` (  `id` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,  `card` INT(10) UNSIGNED NOT NULL,  PRIMARY KEY (`id`)  );  #图书  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

### 采用左连接：left join

|  |
| --- |
| # 下面开始explain分析  EXPLAIN SELECT SQL\_NO\_CACHE \* FROM class LEFT JOIN book ON class.card = book.card;  #结论：type 有All  计算机生成了可选文字: mysql>EXPLAINSELECTSQL_NO_CACHE*FROHclaSSLEFTJOINb00kONclass.card=b00k.card; ---－一－一＋--－一一一＋--－一一一＋·…… select_type tsble partitions type p0SSible_keys key key--len Fef rOWS flltered EXtrs id ＋一一－ SIHpL〔 SIHPLE C18SS b00k NULL NULL ALL ALL NULL NULL NULL NULL NULL NULL NULL NULL 20 20 100 100 00 00 NULL Usingwh eFe: Usingjoinbuffer(BlockNestedLoop) ，上，上 ＋·…＋· 2FOWS ·…＋··～·…＋·～~~ 2warnings(0.00 二＋……＋二 ~＋……＋·····～…＋～··～二＋二 ~~+0000~000～·＋ 0十 Set, SeC ) .1n  # 添加索引优化  CREATE INDEX idx\_book\_card ON book(card);  #【右表被驱动表】，可以避免全表扫描  计算机生成了可选文字: mysql>EXPLAINSELECTSOL--NO_CACHE*FROHclassLEFTJOINbookONclass.card=book.card; ＋一～~ 1id 一＋一～ 一＋一～~~~～一＋一～~~~~ 一＋一～一－~~+~~~~~~~~~~~~~~~＋一～-~~~+～一～~~~~～一＋一～～一～~ 一＋一～~～一～+~~~～一～~~~~+~~~~~~~～一～~～一＋ select_type table partitions type possible_keys key key--len Fef FOWS filtered EXtFa SIHPLE SIHPLE ClaSS b00k NULL NULL ALL Fef NULL Y NULL Y NULL 4 NULL mydb.class.card 20 1 100。00 166。00 NULL Usingindex ，工IJL 2rowsinset,2warnings(0.00sec)  CREATE INDEX idx\_class\_card ON class(card);  #【左表驱动表】，无法避免全表扫描 |

### 采用内连接：inner join

换成inner join（MySQL自动选择驱动表）

先删除之前建立的索引！class book;

|  |
| --- |
| EXPLAIN SELECT SQL\_NO\_CACHE \* FROM class INNER JOIN book ON class.card=book.card;  计算机生成了可选文字: mysql>EXPLAINSELECTSOL--NO_CACHE*FROHclassINNERJOINbookONclass.card=book.card; ＋一～~ 1id 一＋一～ 一＋一～~~~～一＋一 一＋一～一－~~+～一一～~~~~~~~~~~~＋一～~~~~+～一～-~~~~ select_type table partitions type possible_keys key key--len Fef FOWS filteFed EXtFa Usingjoinbuffer(BlockNestedLoop) SIHPLE SIHPLE ClaSS b00k NULL NULL ALL ALL NULL NULL NULL NULL NULL NULL NULL NULL 20 26 160。00 10。66 NULL Usingwhere ，工IJL ~~~~~~~~～今 2rowsinset,2warnings(0.00sec)  # 添加索引优化  CREATE INDEX idx\_class\_card ON class(card);  计算机生成了可选文字: mysql>EXPLAINSELECTSQL_NO_CACHE*FROHclassINNERJOINbookONclass.card=book.card; ＋一～~ 1id 一＋一～~~~~～一－~~~~+~~～一一一一＋一 ~~-~-~~~+~~~~～一＋一～~ 一＋~~～一～ 一＋一～~~~~~~~+~~ ～一＋~~~~～一＋一 select_type table partitions type p0Ssible_keys key key--len Fef FOWS ~~~~~~~～一＋一 filteredl ~~~~~~~~~-－一＋ EXtral SIHPLE SIHPLE ClaSS b00k NULL NULL ALL Fef NULL Y NULL Y NULL 4 NULL mydb.class.card 20 1 100。00 100。00 NULL Usingindex ，上j上 小 2rowsinset,2warnings(0.00sec)  计算机生成了可选文字: mysql>EXPLAINSELECTSOL_NO_CACHE*FROHclaSSINNERJOINb00kONclaSS.card=b00k.card; ＋一一一～+－一一一一一－-－一一一一＋~-－一一一一＋-－一一一－一－-－一一＋一一－-－一一＋一一一－一－一一一－-－一一一＋一－-－一一＋一－-－一－一－一＋--－一一－-- ~＋一一一一－一＋一一一－一－-－一一＋一－---－一－---~~+ select_type tsble partitlons type possible_keys key key--len Fef FOWS fllteFed EXtF日 id ~~~+~~～一～~-~~~~～一～~~+~~~~~~+~~~~~~~~~~＋一～~~~~ l ～一～一～~~+ 160。00 160。00 Usingindex Usingindex 0月1 2 NULL mydb.book.card 月叶4 YX SIHpLE SIHPLE b0Ok ClaSS NULL NULL indeX Fef l丫 lX 月l,JL +~~~~+~~～一～~~～一～~~~+~~~~~~~+~~~~~~ ~~~+~～一一一～+~~～一～~~~~~+～一～~~~~~~-~~~+ 2rowsinset,2warnings (0.00sec) |

## 建议

1. 保证**被驱动表**的join字段被索引
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会自动将小结果集的表选为驱动表。选择相信mysql优化策略。
4. 能够直接多表关联的尽量直接关联，不用子查询

证明：

|  |
| --- |
| explain SELECT a.name ,(SELECT c.name FROM t\_emp c WHERE c.id=b.ceo) AS ceoname FROM t\_emp a LEFT JOIN t\_dept b ON a.deptid = b.id ; |
| explain SELECT a.name,c.name AS ceoname FROM t\_emp a LEFT JOIN t\_dept b ON a.deptid = b.id LEFT JOIN t\_emp c ON b.ceo = c.id ; |

# 查询优化\_子查询优化

* 两种：非掌门人员

|  |
| --- |
| 先创建一个索引  CREATE INDEX idx\_ceo ON t\_dept (ceo);  #①不推荐  Explain SELECT \* FROM t\_emp a WHERE a.id NOT IN  (SELECT b.ceo FROM t\_dept b WHERE b.ceo IS NOT NULL);  #②推荐  Explain SELECT a.\* FROM t\_emp a  LEFT JOIN t\_dept b  ON a.id = b.ceo  WHERE b.id IS NULL;  **结论：**  **NOT IN -->LEFT JOIN xxx ON xx WHERE xx IS NULL**  **尽量不要使用not in 或者 not exists** |

# 查询优化\_排序、分组优化

## 排序case

CALL proc\_drop\_index("mydb","emp");

CALL proc\_drop\_index("mydb","dept");

以下是否能使用到索引，能否去掉using filesort

### 总结：无过滤 不索引（where过滤或者limit过滤都行）

|  |
| --- |
| EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp **ORDER BY age,deptid**;  EXPLAIN SELECT SQL\_NO\_CACHE \* FROM emp **ORDER BY age,deptid LIMIT 10**;  计算机生成了可选文字: mysql> Query CALL 0 pFOC rOWS _drop_indeX(''mydb", affected(0.02sec ,'e叩”') ) mysql> Query CALL pFOC rOWS ""dept ) 0 _drop_index(''mydb'', affocted(0.00sec mysql>EXPLAINSELECT50仁NO_CACHE*FROHempORDERBYage,deptid; ＋一一一～+－一一一一－-~－一一一一＋一－－一一一一＋-－一一－---－一一一＋一－--－一＋一一一－－一一一－－一－－一一＋----－一＋一一一－－一一一一＋~-－一一一＋一－-－一－－一＋~-------－一＋一－一－---－一－----－一＋ }id ＋一～一 }1 select_type tsble partitions type possible_keys key key--len Fef l 一＋一 FOWS filteFed EXtFa 一＋一～一一一～-~~～一一～+~~~~～一一＋~~~～一－一－-～一一＋～一～~～一＋一一～~－一～～一～~~~～一＋~~~~～一＋~ + 一＋一～~～一一一～~~＋一一一一～~~~～一一一－~～一＋ SIHpLE emp NULL ALL NULL NULL NULL NULL 499086 160。06 Usingfilesort 1rowinset,2 Warnlngs (0.00sec) mysql>EXPLAINSELECTSOL_NO_CACHE*FROHempORDERBYage,deptidLIHIT10; ＋一一一～+－一一一一一－-－一一一一＋一－－一一一一＋-－一一一－一－-－一一＋一一～~－一＋一一一－－一一一一一－-－一一＋----－一＋一一一－－一一一一＋~-－一一一＋一－-－一－－一＋ ·＋一－--- ··－一＋ select_type tsble partitlons type possible_keys key key--len Fef FOWS filtgFed EXtFa id 11SIHpL【 emp ＋一一一一＋一一一一一一一一一一一一一＋一一一一一 } 一一－一＋- NULL1499686!166.001USingfilesortl ～一一－一＋-－一一一一－一＋---－一一－-－一＋－一一－--－一一一－--－一一＋ 1rowinset,2 WaFnlngS }NULL}ALL1NuLL1NULL}NULL 一一＋-－一一一－---－一一＋－一－－一一＋一－~～一一一－-~~-－一一＋--~～一一＋一一～~~ (0.00sec)  创建索引  CREATE INDEX idx\_age\_deptid\_name ON emp (age,deptid);  计算机生成了可选文字: mysql>CREATEINDEXidx--age oueryOK,0rowsaffected( Records:6Duplicates:0 _deptid_nameONemp(age,deptid,NAHE) 2.20sec) Warnings:0  增加limit过滤条件，使用上索引了。  计算机生成了可选文字: mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempORDERBYage,deptid; ＋一～～一＋一 1idl ~～一一～+~~～一一一一＋~~~~~~-~~~~~+~~~~～一＋一 一＋~-~～一一＋一～~~~~~~~+~~~~～一＋一一～~~~~~＋一一～-~~~~~~+~~~~~~~～一～～一～~－一卜 select_type table partitions type p0Ssible_keys key key--len Fef FOWS filteFed EXtFa 11SIHpLE 0lflp NULL ALL NULL NULL NULL NULL 499086 160。00 Usingfilesort 1rowinset,2warnings(0.00sec) mysql>EXPLAINSELECTSQL--NO_CACHE*FROMempORDERBYage,deptidLIHIT10; ＋一～～一＋一 }idl 一＋一～~～一～一＋~~~~~~-~~～一～+~～一一一一一＋一～ ～一＋一～～一～~~~~～一～~-~~~~~~~+~~~~~~~~~＋一～~~～一＋一～~~~~＋一～~~~~~~～一＋~～一～~～一十 select_type table partitions type p0SSible_keys key key--len ref rOWS filtered EXtrs +~~~~+~～一一～~－一～一一一～+~～一一一一～+~ 一～~~~～一一～~+~~～一～~~~~＋一一～~~~＋一一～~~~+~～一一～~~～一一＋~~~～一～~+ 73 11SIHPLE emp NULL indeX NULL idX--age_deptid_ name NULL 10 180。60 NULL 一＋－一－----－一一一一 ··…＋一一一一－一一－一＋一一－-－一＋一一－-－一＋ + } 一一＋ 1ro树inset,2warnings(0.00sec) |

### 总结：顺序错，必排序(文件filesort)

|  |
| --- |
| EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY deptid;  EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY deptid,NAME;  EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY deptid,empno;  //出现filesort  EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY NAME,deptid;  //出现filesort  EXPLAIN SELECT \* FROM emp WHERE deptid=45 ORDER BY age;  CREATE INDEX idx\_age\_deptid\_name ON emp (age,deptid,NAME)  计算机生成了可选文字: mysql>EXPLAINSELECT*FROHempWHEREage=45ORDERBYdeptid; ＋一一－一＋－一一－一一一一－一一一一＋- select_type table partitions type possible_keys key Fef FOWS filteFed EXtFa id 1!SIHPL〔 emp NULL ref idX--age_deptid_ name idX--age_deptid_ name key_len 5 Const 45828 100.00 Usingindexcondition 1r0win set,1warning(0.60sec)  计算机生成了可选文字: mysql>EXPLAINSELECT*FROHempWHEREage=45ORDERBYdeptid,NAHE; ·－一＋-－一一－--－一＋ 一＋ －一＋ select_type table partitions type p0SSible_keys key key--len ref r0WS filteFed EXtrs id ＋一一一一＋一一－－一－----－一一＋一一－－一一一＋一－-－一－一一－－一一＋一一一一－一＋一一－---－一一一一－---－一－－一一＋一一一一－-----~~~~~~---－一＋－一一一一－－一 1}SIHpLE emp NULL ref ldx--age_deptid_name idx--age_deptid_name 5 Const 45828 100。00 Usingindexcondition 1rowinset,1warning(0.60sec)  CREATE INDEX idx\_age\_deptid\_empno ON emp (age,deptid,empno);  计算机生成了可选文字: mysql>EXPLAINSELECT*FROMempWHEREage=45ORDERBYdeptid,empno; id1select_type tsble partitions type p0SSible_keys 氏。V Fef FOWS filteFed EXtFS key_len 5 11SIHPLF emp NULL ref idxagedeptid name idx--age_deptid_ name l 一＋- Const 45828 166.66 ～一－一＋· } 一＋· Usingindexcondition:Usingfilesort l· 1rowinset,1 WaFnIng (0.00sec) mysql>CREATEINDEXidX--age_deptid_ QueryOK,0rowsaffected(2.17sec empno ) ONemp(age,deptid,empno); Records:6Duplicates:0Warnings:0 mysql>EXPLAINSELECT*FROHempWHEREage=45ORDERBYdeptid,empno; select_type tsble partitions type possible_keys key Fef FO树S filteFed EXtFS id 1llSIMpL〔 ＋一一一一＋一一一一一－- 1rowInset, emp NULL ref idx--age_deptid_name,idX--age_deptid_empno idX--age_deptid_empno key--len 5 Const 45828 106.08 Usingindexcondition ～一－一一一一－－一－一一一－一＋一一一－－一一－－一一一－-－一一一一一～ 1 WaFnlng (0.00sec)  计算机生成了可选文字: mysql>EXPLAINSELECT*FROHempWHEREage=45ORDERBYNAHE,deptid; ＋一一一一＋~－一一一一－一－－一一一＋~-－一一一一＋~ ~＋一一一－--~＋一一一一－- select_type table partitions type possible_keys +~~~~+~~～一一～~~~~~~~+~~~~~~~+~~~~~~~~~~~~+~~~~～一＋~~~~~~~~~~ !key!key--len}ref1rows}filtered1EXtra ~~+~~~～一～~~－一一～~~~~～一～~~+-~~~～一～~~+~~～一－~~+~~~~~~~+~～一～~~~~~~+~~~~~~ ! ~～一－~~~~~~~~~~~~~-~~~～一～--~~~~～一～+ id 1}SIHPL〔 emp NULL ref idxagedeptidname,idxagedeptidempno idxagedeptidnam。 5 Const 4582B ＋一－－一＋-－一－--－一－－一一一＋-－一一－－一＋-－一一一－----－一＋一－--－一＋－一············…… －一＋－一 ·····一＋--－一－一－－一＋一一－-－一一＋一一一一－~ 1160.061 一＋一－······……牛· Usingindexcondition:Using filesort} 二尝 1rowinset, 1warning(6.60sec)  计算机生成了可选文字: mysql>EXPLAINSELECT*FROHempWHEREdeptid=45ORDERBYage; ＋一－一一＋一一一一一－一－ 一一一－一＋一一－一－-－一＋－一－---－一一一＋一－------~~~~~~~~~~~~~~~~~-－一一＋ select_type tsble partitions type possible_keys key key--len rof FOWS filtOFed EXtFa id 1}SIHPLE emp NULL ALL NULL NULL NULL NULL 499086 10。00 Usingwhere;Usingfilesort 1rowinset,1warning(6.00sec) |

### 总结：方向反 必排序

|  |
| --- |
| EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY deptid **DESC**, NAME **DESC** ;  EXPLAIN SELECT \* FROM emp WHERE age=45 ORDER BY deptid **ASC**, NAME **DESC** ;  计算机生成了可选文字: mysql>EXPLAIN ＋一～～一＋一～~~~~～一 *FROHempWHEREage=45ORDERBYdeptidDESC,NAHEDESC; ～一～一＋~~~~~~~~~~～一＋~~~~～一＋一 一＋一～一～~~~~~~~~~~～一～~~~~+~~~~~~~～一干一～~~~～一卜～~~~~～一＋一 id}select SELECT ~~～一～+~ typel table partitlons type1poSSible_keys key key--len Fef FOWS ～一～~~~~～一＋～一～~~~~~~~～一～+ filtered}Extral 1!SIHpLE omp NULL ref idxagedeptid name idxagedeptidompno idx--age_deptid_na"Ie15 COnst 45828 100.66!Usingwhor。 1FO树inSet 1warning(6.00 SeC  计算机生成了可选文字: mysql>EXPLAINSELECT*FROHempWHEREage=45ORDERBYdeptidASC,NAHEDESC二 ＋一－－一＋一一一－一－-- ~＋一一一－-----~~~~~~~~~-－一＋一一一一－－一一一＋一一－-－一 select_type tsble partitions type possible_keys key ke犯len ref FOWS filteFed EXtFS id 1!SIHPLE emp NULL Fef idX--age_deptid_name,idX--age_deptid_empno idx--age_deptid_name15 Const 45828 100。00 Usingindexcondition:Usingfilesort 1rowinset,1warning(6.00sec) |

* 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

CALL proc\_drop\_index("mydb","emp");

* 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

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

计算机生成了可选文字:
mysql>EXPLAINSELECTSOL--NO_CACHE*FROHempWHEREage=36ANDempno<101000ORDERBYNAHE
＋一～～一＋一
1id}
一＋一～~~~~~+~~~~~~
一＋一～~~~~+~~~~~~~~~~~~~~~＋一～~~~~+～一～~~~~～一＋一～～一～一＋~~
select_type
table
partitions
type
possible_keys
key
key--len
Fef
FOWS
filteFed
EXtFa
1}SIHPLE
0Illp
NULL
ALL
NULL
NULL
NULL
NULL
499086
3。33
Usingwher。；Usingfilesort
1rowinset,2warnings(6.01sec)

* 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

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);

计算机生成了可选文字:
mysql>CREATEINDEXidX--age
QueryOK,0rowsaffected(
Records:0Duplicates:0
_empno_nameONemp(age,empno,NAME)
1.87sec)
Warnings:0
mysql＞〔XPLAINSELECTSQL--NO_CACHE*FROHempWHEREage=36ANDempno<161000ORDERBYNAHE
＋一一一一＋－一一－---~-－一一一＋-－一一一－一＋-－一一－-----－一＋一－-~－一一＋一－一一一一－----－一一－一一一～一＋一一一－--－一一一一－~
id!select_type1table!partitions
type
possible_keys
key
key--len
ref
ro树S
filteFed
EXtFS
＋一－－一＋-
11SIHPLE
emp
NULL
range
idX--age_empno_name
idX--age_empno_name19
NULL
56
100。00
Usingindexcondition;Usingfilesort
1rowinset,2warnings(0.00sec)

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;

1. 为了去掉filesort我们可以把索引建成

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

计算机生成了可选文字:
mysql>CREATEINDEXidX--age
QueryOK,0rowsaffected(
Records:6Duplicates:0
nameON
2.09Sec
Warnings
emp(age,NAME)
0
mysql>EXPLAINSELECTSQL_NO_CACHE*FROHempWHEREage=30ANDempno<101000ORDERBYNAHE
id}Select_type!table
partitions
type1p0SSible_keys1key
key--len
ref
rOWS
filtered
EXtrs
11SIHPLE
emp
NULL
Fef
idxagename
idxagename!5
Const
47826
33.33
Usingindexcondition;Usingwhere
＋一一一～＋一一一一一一
1r0winSet
～一－一一一一一－-－一一一一＋~
2warnings(0.00sec)

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。与没有加任何索引性能提升了。

计算机生成了可选文字:
.ysql》CREATEINDEXidx_ag。
。呻（age,NA洲E);
QueryOK,6ro侧‘affected
Records:6Dupllcates:8
nd.eON
2.26sec)
树arnlngs:
.ysql》SELFCTSQ气NO_CACHE.FRON。呻姗EREag。＝36ANDe呻no'161686ORI)ERBYN月惬二
+~~~~~+~~~~～·～~+~~~~~~~~+~~~~~~+~~~~~~~~+
!e呻no
杏……
name
deptldl
.-..－…＋
86
488
372
11?
166986
160488
166372
10011?
白FdASI
AVZkCJ
。yrn护
RedpLm
l今1Ilf
.
16060
e．气j,J,J工J
g.
a.
.
I+1111
I+111
!+11!
606
工J,J气j
166395
106545
lee922
YudgxT
ZDVR目I
Zhl阅iA
4234
1275
5882
+~~~~~+~~~~~~~~+~~~~~~~~+~~~~~~+~~~～·
56ro树5inset.1warning(8.16sec)
口口暇口阅甲口口口口口口口口口．

1. 如果我们选择那个范围过滤，而放弃排序上的索引呢

DROP INDEX idx\_age\_name ON emp;

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

计算机生成了可选文字:
mysql>DROPINDEXidX--age_
QueryOK,0rowsaffected
Records:0Duplicates:0
nameONemp;
(0.02sec)
Warnings:0
mysql>CREATE
INDEXidxageempnoONemp(age，。mpno):
QueryOK,0
ReCOFdS:6
FO树S
affected(1.74sec
Duplicates:0Warnings:0
mysql>EXPLAINSELECTSQL_NO_CACHE*FROMempWHEREage=30ANDempno<101000ORDERBYNAHE
+-～一～＋一一一～一一一一一一一一～＋一一一一一～一＋一一一一～一一一一一一一＋～一～～一一一＋－一一一一一一一一一一一一一～＋一一一一一一一一一一一一一一～+～一一一一～一一一＋
id}select_type}table}partitions
type
possible_keys1key
key--len
Fef
FOWS
filteFed
EXtFa
l
J~~~~～一＋
1!SIHpLE
emp
NULL
range
idX--age_empno
id左age_empno19
NULL
56
166。00
Usingindexcondition;Usingfilesort
1rowinset,2warnings(0.00sec

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

计算机生成了可选文字:
.ysql》CREATEINDEXldX_age
OueryOK.
ROCOFdS:B
8ro树5affOCtod
1
e呻noONe叩（age,e呻no);
.56sec】
Duplicates:6树arnlngs:8
.ysql》SELECTSQL_NO_CACHE.FROHe呻
……＋。．...．……，．…＋二。窗，．…，,,,
城IERE
,，奋
ag。＝36ANDO呻no《161006ORDERBYNA洲E:
idle呻no1na触
age1deptld
毛一～～一～＋一·一～～一一～+～一～一一一·一＋~～一一·～＋一一～一～·－~+
6668066
333,J3工J,J
86
488
372
132
409
192
946
．肠
〕95
545
922
166886
166488
IB6372
IB6132
166489
166192
106940
aFdAsl!
AVZkcJ!
ayrnqPI
BBdpLol
Bi】xlal
Bta。。l
卜成Xlu!
5136
1813
7652
3467
1832
3273
9834
4234
1275
5882
686
勺1、j,J
186395
166545
188922
YudgxT!
ZDVR树11
zh初IAI
+....~+.．……＋.．……＋……＋.．……＋
56ro仍inset.1warning(8.61sec)_

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？

1. 原因

所有的排序都是在条件过滤之后才执行的，所以，如果条件过滤掉大部分数据的话，剩下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。

1. **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。

## 理论：双路排序和单路排序

当对sql进行order by排序的时候，需要尽可能的使用索引排序，如果无法使用索引排序的话，mysql就会使用文件排序。文件排序（filesort）分为两种:单路排序和双路排序。

### 双路排序（慢）:两次磁盘读取,占用内存空间少

Select id,age,name from stu order by name;

* 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（一般是主键列）和order by列，然后在 sort buffer 中对他们进行排序，排序完后需要再次去磁盘中取回其它需要的字段；
* 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（快）:一次磁盘读取,占用更多的内存空间

* 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内存空间， 因为它把每一行都保存在内存中了。

### 结论及引申出的问题

* 但是用单路有问题
* 在sort\_buffer中，单路比多路要多占用很多空间，因为单路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sort\_buffer容量大小，再排……从而多次I/O。
* 单路本来想省一次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。 禁止使用select \*

### 提高Order By的速度

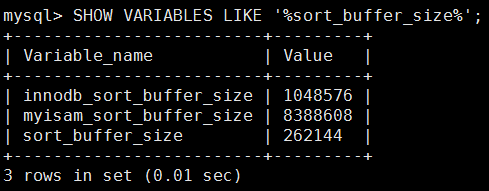
1. Order by时select \* 是一个大忌。只Query需要的字段， 这点非常重要。在这里的影响是：

* 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。
* 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（connection）的 1M-8M之间调整。 MySQL5.7，InnoDB存储引擎默认值是1048576字节，1MB。

SHOW VARIABLES LIKE '%sort\_buffer\_size%';



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。

SHOW VARIABLES LIKE '%max\_length\_for\_sort\_data%'; #默认1024字节

* 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。如果需要返回的列的总长度大于max\_length\_for\_sort\_data，使用双路算法，否则使用单路算法。1024-8192字节之间调整

## 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：

* group by 先排序再分组，遵照索引建的最佳左前缀法则
* 当无法使用索引列，增大max\_length\_for\_sort\_data和sort\_buffer\_size参数的设置
* group by没有过滤条件，也可以用上索引。Order By 必须有过滤条件才能使用上索引。(**这是group by与order by的区别**)

# 查询优化\_覆盖索引优化

## 什么是覆盖索引？

简单说就是，select 到 from 之间查询的列使用了索引！覆盖索引不用回表查询，通常出现于innodb引擎。

select \* from tName 不建议使用\*

数据库表结构：

create table user (

id int primary key,

name varchar(20),

sex varchar(5),

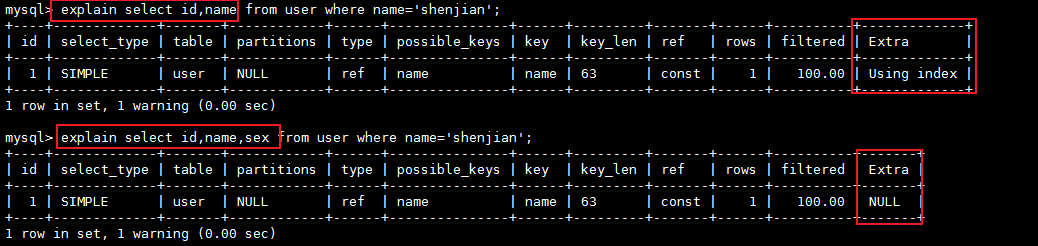
index(name)

)engine=innodb;

查看以下两条SQL的执行计划：

explain select id,name from user where name='shenjian';

explain select id,name,sex from user where name='shenjian';



**多查询了一个属性，为何检索过程完全不同？**

　什么是回表查询？

　什么是索引覆盖？

　如何实现索引覆盖？

哪些场景，可以利用索引覆盖来优化SQL？

问题1：这就是所谓的**回表查询**，先定位主键值，再定位行记录，它的性能较扫一遍索引树更低。

问题2:所谓覆盖索引是指：只需要在一棵索引树上就能获取SQL所需的所有列数据，无需回表。

explain的输出结果Extra字段为**Using index**时，能够触发索引覆盖。

**问题3：如何实现索引覆盖？**

常见的方法是：将被查询的字段，建立到联合索引里去。

对于前面的例子：能够命中name索引，索引叶子节点存储了主键id，通过name的索引树即可获取id和name，无需回表，符合索引覆盖，效率较高。

画外音，Extra：Using index。

drop table user;

create table user (

id int primary key,

name varchar(20),

sex varchar(5),

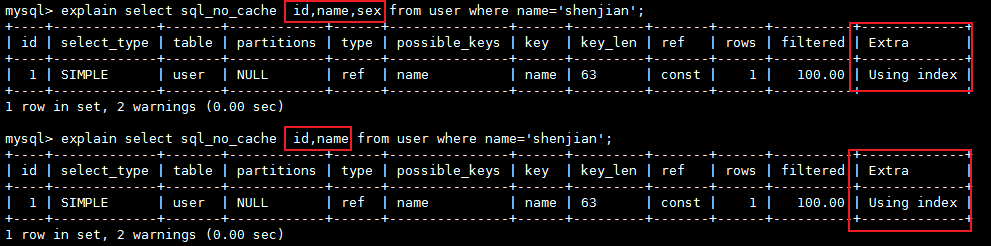
index(name, sex)

)engine=innodb;

查看以下两条SQL的执行计划：

explain select id,name from user where name='shenjian';

explain select id,name,sex from user where name='shenjian';



可以看到：都能够命中索引覆盖，无需回表。

问题四：**哪些场景可以利用索引覆盖来优化SQL？**

**场景1：全表count查询优化**

**重新创建表：默认只有id列有主键索引**

drop table user;

create table user (

id int primary key,

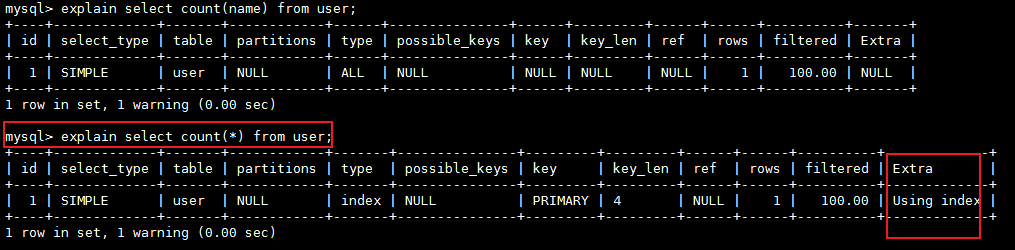
name varchar(20),

sex varchar(5)

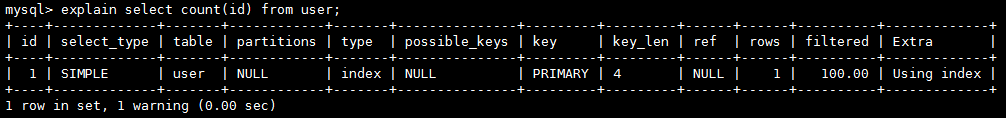
)engine=innodb;

explain select count(name) from user;

explain select count(\*) from user;



会发现mysql在使用select count(\*) from user;默认使用了索引覆盖。**作用等同于**:explain select count(id) from user; //id列是有主键索引的。



所以以后做count()统计时，尽量使用count(id)或者count(\*),不要使用没有建立索引的字段进行计算，如：count(name);count(name)不会使用到索引，除非我们为name字段建立了索引。

**场景2：列查询回表优化**

select id,name,sex ...where name='shenjian';

这个例子不再赘述，将单列索引(name)升级为联合索引(name, sex)，即可避免回表。

**场景3：分页查询**

select id,name,sex ...order by name limit 500,100;

将单列索引(name)升级为联合索引(name, sex)，也可以避免回表。

explain select sql\_no\_cache id,name,sex from user where name='shenjian' order by name limit 500,100;

例如：

drop table user;

create table user (

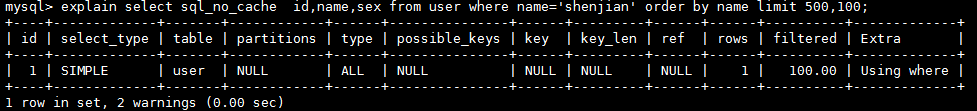
id int primary key,

name varchar(20),

sex varchar(5)

)engine=innodb;

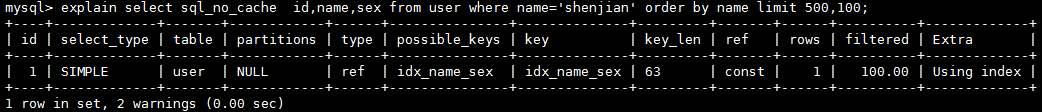
explain select sql\_no\_cache id,name,sex from user where name='shenjian' order by name limit 500,100;



会发现没有使用到覆盖索引,然后为分页查询的name,sex字段建立复合索引：

create index idx\_name\_sex on user(name,sex); 再次查询：

explain select sql\_no\_cache id,name,sex from user where name='shenjian' order by name limit 500,100;



会发现，使用到了覆盖索引。

## 总结

* 禁止使用select \*，一般情况下:索引树覆盖不了所有的字段
* 禁止查询与业务无关字段,容器占用较多的内存空间，影响排序性能,导致多路排序啥的。
* 尽量使用覆盖索引

# 查询截取分析\_慢查询日志

## 是什么

select \* from emp;

* 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中响应时间超过阀值的语句，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。
* 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。

## 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**。

### 查看及开启

1. 默认关闭

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，

计算机生成了可选文字:
mysql>SHOWVARIABLESLIKE'%slow--query--log%‘二
VaFiablensme
V81Ue
＋一一一－-－一－一－-~～一一一一－-－一＋一一一－-－一－--~---~~~~~-
l
－一＋
slow--query--10g
510吮query--log_file
OFF
/var/lib/mysql/zhangyu一slow.log
2FOWS
inset(0.62
SeC

1. 开启：set global slow\_query\_log=1; 只对窗口生效，重启服务失效

计算机生成了可选文字:
mysql>5et
oueryOK,0
globalslow--query--log=1;
rowsaffected(6.01sec)
mysql>SHOWVARIABLESLIKE'%slow--query--log%';
于？+
VaFiablename
1Value
＋········……
slow--query--10g
s10w--query--log_file
ONI
/var/lib/mysql/zhangyu一slow.logl
l+11
＋一一一一－－一一一一－一－－一－---－一＋一－--
2rowsinset(0.61sec)
-－一一一－--－一一一一－一于

1. 慢查询日志记录long\_query\_time时间

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

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

计算机生成了可选文字:
mysql>5howvariableslike"%10ng_query--time%';
＋一－-------------－一＋-----－一－-－一＋
VSr18blenallle
Va1Ue
＋一一一－-－一一－－一－－一一一一＋一一－－一一一－-－一＋
110ng_query--time110.0000001
＋一一一一一一一一～一一一一一一一～＋一一一一一一～一一一一十
1rowinset(0.00sec)

* + 全局变量设置，对所有客户端有效。但，必须是设置后进行登录的客户端。

SET GLOBAL long\_query\_time=0.1;

计算机生成了可选文字:
mysql>SETGLOBAL
QueryOK,0rows
10ng_query--time=0.1
affected(0.01sec)

* + 对当前会话连接立即生效，对其他客户端无效。

SET SESSION long\_query\_time=0.1; #session可省略

计算机生成了可选文字:
mysqL>5et
QueryOK,0
sessionlong_query--time=0.1;
rowsaffected(0.00sec
mysql>showvariableslike'%long_query--time%'
+---------------－一＋--------－一＋
Variablename
Va1Ue
＋一～~~---～一～一－一～一一一＋~~~-～一一～~~+
}10ng_query--time16·1006001
＋一－---－一一一－--－一一一一＋－一－－一一－-－一＋
1rowinset(0.00sec)

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。

1. 永久生效
   * 修改配置文件my.cnf（其它系统变量也是如此）
   * [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/localhost-slow.log

long\_query\_time=3

log\_output=FILE

* + 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，如果不设置，系统默认文件：[host\_name]-slow.log

### case

* 记录慢SQL并后续分析

SELECT \* FROM emp;

SELECT \* FROM emp WHERE deptid > 1;

* 查询当前系统中有多少条慢查询记录或者直接看慢查询日志

/var/lib/mysql/localhost-slow.log

SHOW GLOBAL STATUS LIKE '%Slow\_queries%';

计算机生成了可选文字:
mysql>SHOWGLOBALSTATUS
＋一～~~~～一一一～~~～一一＋一～~~~～一＋
}Variable_name}Value}
+~~～一～一一～～一一一～一～＋一一一一一～～礴
LIKE'%Slow--querieS%';
510吮queries
6
+~~~~~~~~~~~~~~~+~~~~~~~+
1rowinset(6.01sec)

## 日志分析工具mysqldumpslow

1. 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。
2. 查看mysqldumpslow的帮助信息
   1. mysqldumpslow --help

计算机生成了可选文字:
1r00t@Zhangyumysql]#
Usage:mysqldumpslow
mysqldumpslow一help
OPTS…11LOGS…
PaFSeandSUmmaFIZe
theHySQLslowquery109.optionsare
一Verb0Se
一debug
一help
Verb0Se
debug
writethistexttostandardoutput
一V
一d
一5ORDER
NUH
NUH
PATTERN
HOSTNAHE
Verb0Se
debug
whattosortby(al,at,ar,c,1,r,t),'at'15default
al:averagelocktime
ar:averagerowsSent
at:averagequerytime
C:COUnt
1:10Cktime
F:FOWS5ent
t:querytime
reversetheSortorder(largestlastinsteadoffirst)
justshowthetopnqueries
don'tabstractallnumberstoNandstringsto'S"
abstractnumberswithatleastndigitswithinnames
9rep:onlyconSlderStmtSthatincludethlsstrlng
hostnameofdbServerfor＊一slow.logfilename(canbewildcard)
default15'*",1.e.matchall
nameofserverinstance(ifusingmysql.serverstartupscript)
don'tSUbtFaCt10Cktimefromtotaltime
L-tlan
一｝一一
一g
一h
NAHE
·111
}}

* -a: 将数字抽象成N，字符串抽象成S
* -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；

|  |
| --- |
| 得到返回记录集最多的10个SQL  mysqldumpslow -s r -t 10 /var/lib/mysql/localhost-slow.log  得到访问次数最多的10个SQL  mysqldumpslow -s c -t 10 /var/lib/mysql/localhost-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/localhost-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/localhost-slow.log | more |

# View视图

## 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。

## 作用

* 封装复杂sql语句，提高复用性
* 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

## 适用场景

* 共用查询结果
* 报表

## 语法

* 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  删除  drop view view\_name; |

## 案例

|  |
| --- |
| CREATE VIEW v\_ceo AS  SELECT emp.name, ceo.name AS ceoname  FROM t\_emp emp  LEFT JOIN t\_dept dept ON emp.deptid = dept.id  LEFT JOIN t\_emp ceo ON dept.ceo = ceo.id; |
| INSERT INTO t\_emp (NAME,age,deptId,empno) VALUES ('岳灵珊',18,1,100011); |

总结：**mysql 视图会随着表中的数据变化而动态变化！**

# 主从复制

## 复制的基本原理

* slave会从master读取binlog来进行数据同步
* 三步骤+原理图

|  |
| --- |
| graphic   * MySQL复制三步骤：   1 master将写操作记录到二进制日志（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）；  从服务器I/O线程将主服务器的二进制日志读取过来记录到从服务器本地文件，然后SQL线程会读取relay-log日志的内容并应用到从服务器，从而使从服务器和主服务器的数据保持一致！  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，而且重启后从接入点开始复制。**   * 复制的最大问题（**延时**） |

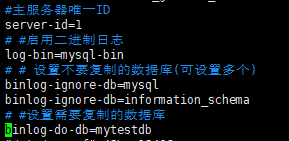
## 复制的基本原则

* 每个slave只有一个master
* 每个slave只能有一个唯一的服务器ID
* 每个master可以有多个salve

## 一主一从常见配置

### 主机配置文件

* + 建议mysql版本一致且后台以服务运行



logbin格式：

* binlog\_format=STATEMENT（默认）：数据操作的时间，同步时不一致
* binlog\_format=ROW：批量数据操作时，效率低
* binlog\_format=MIXED：是以上两种level的混合使用，有函数用ROW，没函数用STATEMENT，但是无法识别系统变量

在核心配置文件my.cnf 中配置

|  |
| --- |
| server-id=1  log-bin=mysql-bin  binlog-ignore-db=mysql  binlog-ignore-db=infomation\_schema  binlog-do-db=mytestdb  binlog\_format=STATEMENT |

先不要创建mytestdb数据库。主从搭建好了再创建。

### 从机配置文件

* 主从所有配置项都配置在[mysqld]节点下，且都是小写字母
* [必须]从服务器唯一ID：server-id=2
* [可选]启用中继日志：relay-log=mysql-relay

计算机生成了可选文字:
FOradVICeonh0w
http://dev.mysql.
tochangesettingspleasesee
com/doc/refman/5.7/en/server一configuration一defaults.html
晚曰＃
[mysqld]
#
#Removeleading#andsettotheamountofRAMforthemostimportant
#cacheinHySQL.Startat70%oftotalRAHfordedicatedserver,else
#innodb_buffer--pool_size=128H
#
#Removeleading#toturnonaveryirnportantdataintegrityoption:
#changestothebinary109betweenbackups.
#109_bin
#
#Removeleading#tosetoptionsmainlyusefulforreportingservers.
#TheSerVerdefaUltSarefasterfortransaCtionS日ndfastSELECTS.
#Adjustsizesasneeded,experimenttofindtheoptimalvalues
#join_buffer_SIZe=128H
#SOrtbUfferSIZe=ZH
#FeadFndbUffeFSIZe=ZM
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
data
10、。
logging
#Disablingsymbolic一links15recommendedtopreventassortedsecurityrlsks
symbolic一links=0
109一error=/var/109/mysqld.10g
pid一file=/var/run/mysqld/mysqld.pid
ChsraCtorSetSerVer=Utfs
query--cache_type=1
Server一id=2
relay一109=mysql一relay
#6eneratedbyCUN卜16RLWRITE

|  |
| --- |
| server-id=2  relay-log=mysql-relay |

### 主从机重启配置生效

systemctl restart mysqld

### 主从机都关闭防火墙

* systemctl stop firewalld

### 在主机上建立帐户并授权slave

GRANT REPLICATION SLAVE ON \*.\* TO 'slave'@'%' IDENTIFIED BY '123456';

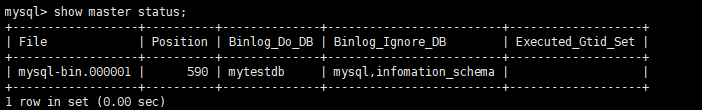
flush privileges;

# grant all privileges on \*.\* to root@'%' identified by 'root'; # 创建一个root 用户，并授权远程访问！

# flush privileges;

# 给root 用户授予远程访问权限

查询master的状态: show master status;



记下File和Position的值

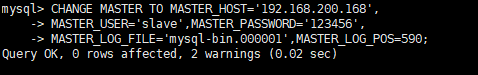
执行完此步骤后**不要再操作主服务器**MYSQL，防止主服务器状态值变化

### 在Linux从机上配置主机

CHANGE MASTER TO MASTER\_HOST='主机ip地址',

MASTER\_USER='X',MASTER\_PASSWORD='X',

MASTER\_LOG\_FILE='mysql-bin.具体数字',MASTER\_LOG\_POS=具体值;

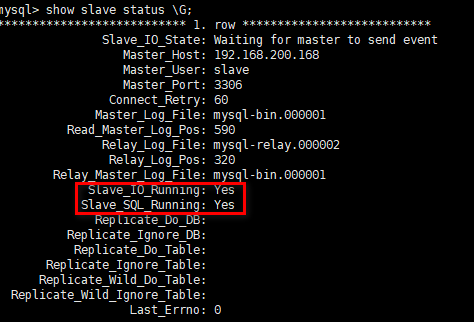


* 启动从服务器复制功能

start slave;

计算机生成了可选文字:
mysql>
Query
StaFt
OK,0
slave;
r0WSsffeCted
(6.00
SeC

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

**如果出现：**Slave\_IO\_Running：No

**则可能的解决办法是：**

1）停止stop slave; 再启动start slave;看是否能正常运行

2）两个服务器的防火墙是否关闭，是否互相能ping通

3）配置文件是否正确、是否重启了服务器

4）连接主机的语句是否正确

5) 可能是uuid 一致(master,slave uuid) vim /var/lib/mysql/auto.cnf 下更改uuid，重启服务（尤其是这个）

### 主机新建库、新建表、insert记录，从机复制

CREATE DATABASE mytestdb;

use mytestdb;

CREATE TABLE mytbl(id INT,NAME VARCHAR(16));

INSERT INTO mytbl VALUES(1, 'zhang3');

INSERT INTO mytbl VALUES(2, 'li4');

### 如果停止从服务复制功能，重新配置主从

在**从机**上执行。功能说明：停止I/O 线程和SQL线程的操作。

mysql> stop slave;

在**从机**上执行。功能说明：用于删除SLAVE数据库的relaylog日志文件，并重新启用新的relaylog文件。

mysql> reset slave;

在**主机**上执行。功能说明：删除所有的binglog日志文件，并将日志索引文件清空，重新开始所有新的日志文件。

用于第一次进行搭建主从库时，进行主库binlog初始化工作；

mysql> reset master;