# 知识

1.选择B+树原因

hash索引

## 存储引擎类型

InnoDB：B+ 、自适应hash 、.frm结构文件、.ibd 数据文件+索引

MyISAM：B+

MEMORY: Hash

## 调优

1.SQL语句及索引的优化

2. 数据库表结构的优化

3.系统配置的优化

4.硬件的优化

### 定位问题：

1.如何辨别是sql语句慢

开启mysql的慢查询日志，超过阈值的语句会记录。

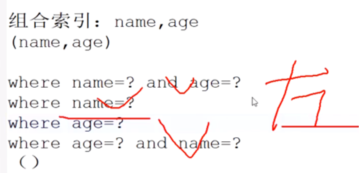
### 知识点：

1. sql优化最基本的原则，就是让数据库尽早、高效率的过滤数据，避免无效的运算，具体的手段比较多，需要根据不同的数据库来确定实现方案。
2. 优先where和order by字段建立索引。
3. 不给数据库留null值，where不使用 null 判断会扫描全表，!=、<> 会放弃索引
4. or两个字段假如一个没有索引会扫全表, or/in : 假如查询组合索引列以外的字段不会走索引？
5. 慎用in 连续的数值用between and 否则全扫·
6. like '%abc%'
7. 避免使用select \* 会解析更多的字段，消耗io交互资源
8. 量大的时候，允许部分数据冗余数据，避免join查询
9. 出现范围查询后续失效

------------------------------------------------------------------------------------

1. 回表：指用二级索引先找到id然后在通过id在主索引找到数据的过程
2. 索引覆盖：

当用二级索引找到id，而字段只需要返回id时，避免了再去主索引找其他数据（回表）。

explain的输出结果Extra字段为Using index时，能够触发索引覆盖，针对索引覆盖优化时将被查询的字段，建立到组合索引中

1. 最左匹配：组合索引 A列+B列的索引 先从A匹配（右图）
2. 索引下推（5.6ver after）

select \*from table where name=？ and age=？

mysql服务跟磁盘交互

没有索引下推：1、先从存储引擎中拉取数据（根据name筛选的数据）再mysql server 根据age进行数据的筛选

有索引下推：会再拉取数据的时候直接根据name，age来获取数据，不需要server做任何的数据筛选

磁盘消耗增加，但是减少io交互

为什么刚开始的时候没有使用索引下推？

select tl.name，t2.name from tl join t2 on tl.id=t2.id

1.先做表连接，然后查询需要的字段

2.先把需要的所有字段都拿出来，然后在做关联谓词下推√

索引下推唯一的缺点是需要在磁盘上多做数据筛选，原来筛选在内存中，现放到磁盘匹配，这样做看起来成本比较高，但是别忘了，数据的排序的，所有的数据是聚集存放所以性能不会有影响，而且整体的io量会大大减少，反而会提升性能I

1. MRR （mult\_range read 范围选取）

二级索引匹配后的数据id，

回表后对id进行排序，

再根据范围在主索引进行查找，性能提高√。

1. FIC(fast index create)

就是DML操作上了个锁，读数据不影响

插入和删除数据需要修改索引时

原来：1）先创建临时表，将数据导入到临时表2）把原始表删除 3）修改临时表的名字

IFC后：

给当前表加锁（Share），不会有创建临时文件的资源消耗，

DML操作等待，读取不影响

### 索引：

帮助mysql高效获取数据的数据结构和算法，文件形式存磁盘上（.ibd）

组合索引：字段拼接做索引（name+age+add），顺序不对会优化

全文索引：大文件查询，定义词库记录词条出现的频率和位置，归纳成为索引

设计原则：

1）是经常匹配的字段

2）低区分度和数据量小不用

3）频繁分组和排序，选用组合索引

### 索引方法:

B+ :

Hash: 键值唯一优势，无法范围查询，hash碰撞效率低

### Explain

<https://blog.csdn.net/wuseyukui/article/details/71512793?utm_medium=distribute.pc_relevant.none-task-blog-title-3&spm=1001.2101.3001.4242>

### 分库分表

垂直拆分：不同表拆到不同库

水平拆分：一个表拆成多个表

### 案例

1. w与InnoDB区别
2. InnoDB支持事务、外键，myisam不支持
3. InnoDB是聚集索引（存id）+辅助索引（字段索引，联合索引等）需要查询两次，查出主键id再根据主键id查询叶子节点，就是数据文件

MyISAM是非聚集索引， 叶子节点数据文件的地址指针

1. InnoDB count(\*) 扫全表，MyISAM 属性记录。
2. InnoDB5.7以前不支持全文索引
3. InnoDB支持表、行（默认）锁，行锁在索引上

MyISAM支持表锁，

# 安装

<https://www.runoob.com/w3cnote/windows10-mysql-installer.html>

## mysql\_Info

2020年2月29日03:59:16

port:

3306/33060/3310

root/123456

ghaya/123456

# 其他

[**mysql优化方面的面试题**](https://www.cnblogs.com/lc2817/p/12795919.html)

## 1.MySQL数据库作发布系统的存储，一天五万条以上的增量，预计运维三年，怎么优化？

a. 设计良好的数据库结构，允许部分数据冗余，尽量避免join查询，提高效率。  
b. 选择合适的表字段数据类型和存储引擎，适当的添加索引。  
c. mysql库主从读写分离。  
d. 找规律分表，减少单表中的数据量提高查询速度。  
e.添加缓存机制，比如memcached，apc等。  
f. 不经常改动的页面，生成静态页面。  
g. 书写高效率的SQL。比如 SELECT \* FROM TABEL 改为 SELECT field\_1, field\_2, field\_3 FROM TABLE.

## 2.实践中如何优化MySQL

最好是按照以下顺序优化：

1.SQL语句及索引的优化

2. 数据库表结构的优化

3.系统配置的优化

4.硬件的优化

## 3.优化数据库的方法

1. 选取最适用的字段属性，尽可能减少定义字段宽度，尽量把字段设置NOTNULL，例如’省份’、’性别’最好适用ENUM
2. 使用连接(JOIN)来代替子查询
3. 适用联合(UNION)来代替手动创建的临时表
4. 事务处理
5. 锁定表、优化事务处理
6. 适用外键，优化锁定表
7. 建立索引
8. 优化查询语句

## 4.如何通俗地理解三个范式？



第一范式：两个字段内容不可拼一个里面，一个字段存独立的数据。字段原子性。

第二范式：尽量细分多张表存储数据。表数据唯一性。

第三范式：冗余字段不能有。

范式化设计优缺点:

优点：数据少冗余，更新快，体积小。

缺点：对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引优化

反范式化：

优点：可以减少表得关联，可以更好得进行索引优化

缺点：数据冗余以及数据异常，数据得修改需要更多的成本

## 5.说说对SQL语句优化有哪些方法？（选择几条）

（1）Where子句中：where表之间的连接必须写在其他Where条件之前，那些可以过滤掉最大数量记录的条件必须写在Where子句的末尾.HAVING最后。

（2）用EXISTS替代IN、用NOT EXISTS替代NOT IN。

（3） 避免在索引列上使用计算

（4）避免在索引列上使用IS NULL和IS NOT NULL

（5）对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

（6）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

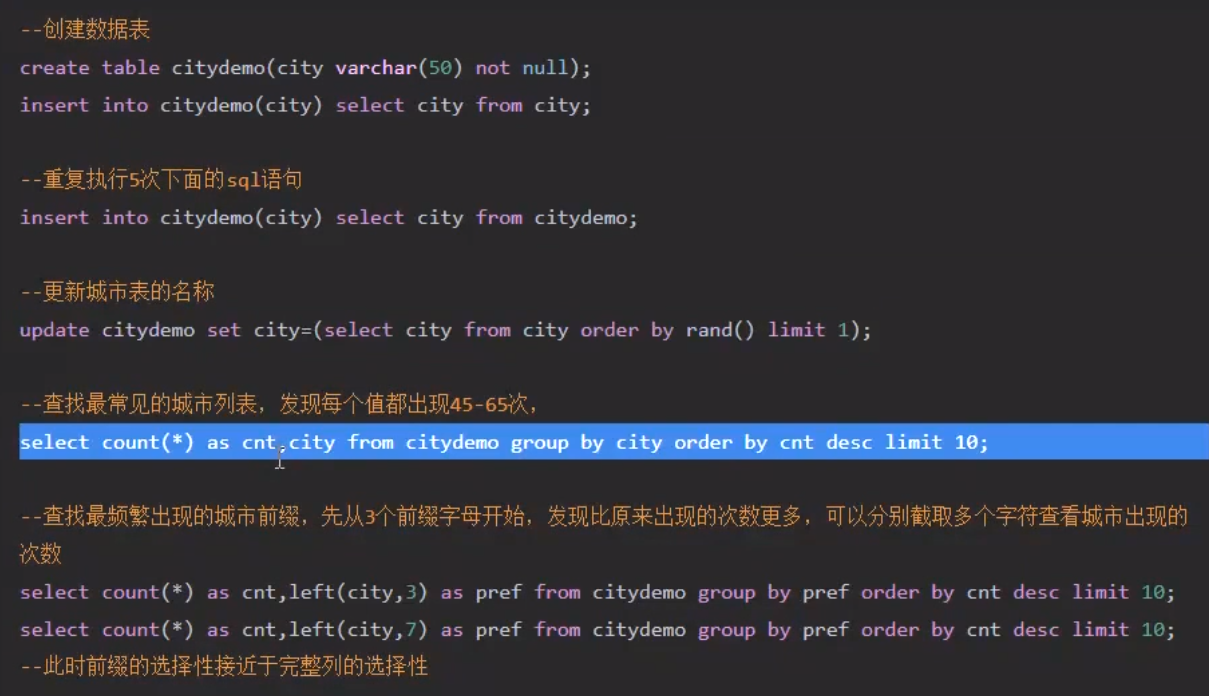
（7）应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描

## 6. [SQL优化思路大全](https://www.cnblogs.com/wcwen1990/p/7204739.html)

<https://www.cnblogs.com/wcwen1990/p/7204739.html>



## 7.案例



# 存储过程使用

delimiter $$ --声明结束符 可以用任何符号组合大概

CREATE PROCEDURE fun\_name

# 调优（未完善）

## 性能监控

<https://www.mysql.com/>

查看查询时间

show profiles;

查看详细步骤时间

show profile;

查看所有消耗

show profile all for query 1;

开启监控

set profiling=1;

显示连接数

show processlist;

## schema与数据类型优化

### 数据类型优化，通常在设计时就改好

更小的数据类型通常更好（快）

例子：储存ip地址

select inet\_aton('192.168.100.1');

select inet\_ntoa(3232261121);

#### 尽量避免null

如果查词中包含可为NULL的列.对mysql

来说很难优化，因为可为null的列使得索引、索引统计和值比较都团加复杂，坦白来说，适常情况下null的列改为not null带来的性能提升比较小，所有没有必要将所有的表的schema进行修改，但是应该尽量避免设计成可为null的列

#### 实际细则

枚举代替字符串类型

create table enum\_test(e enum('fish','apple','dog') not null);

insert into enum\_test(e) values('fish'),('dog'),('apple');

select \* from enum\_test;

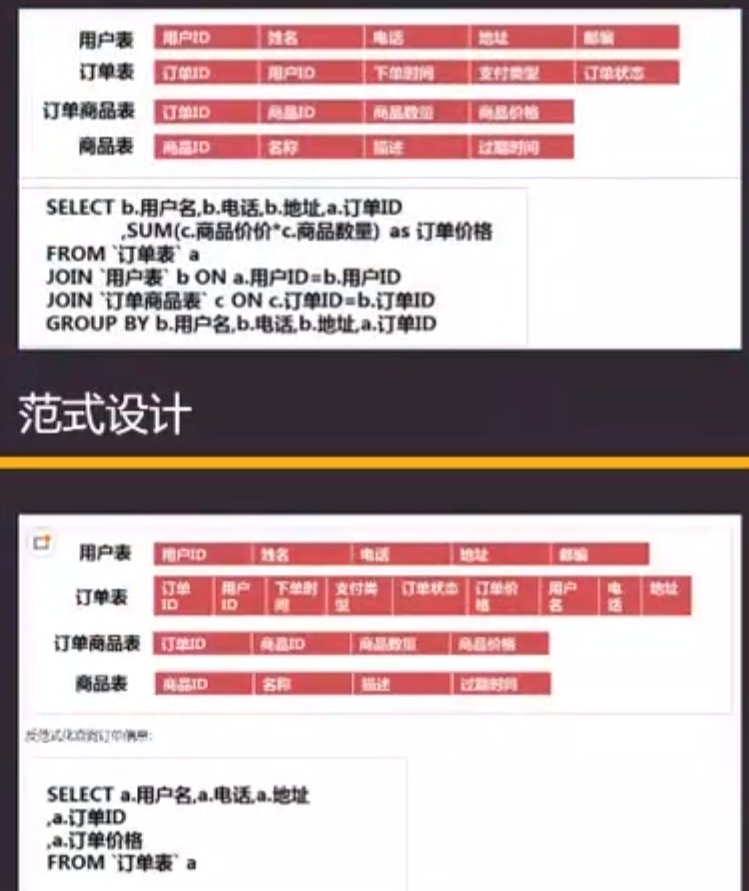
# 枚举：存整型的值，对应字符串

select e+1 from enum\_test;

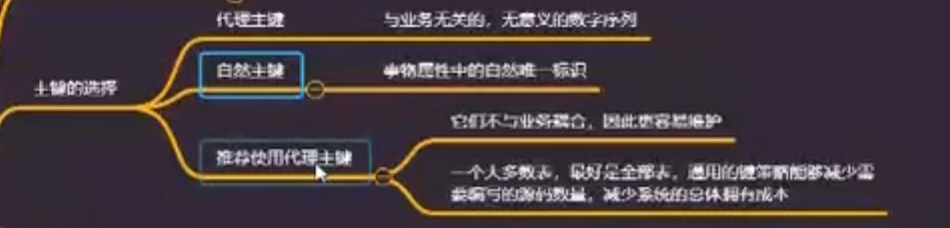
### 范式与反范式选择

#### 适当的冗余

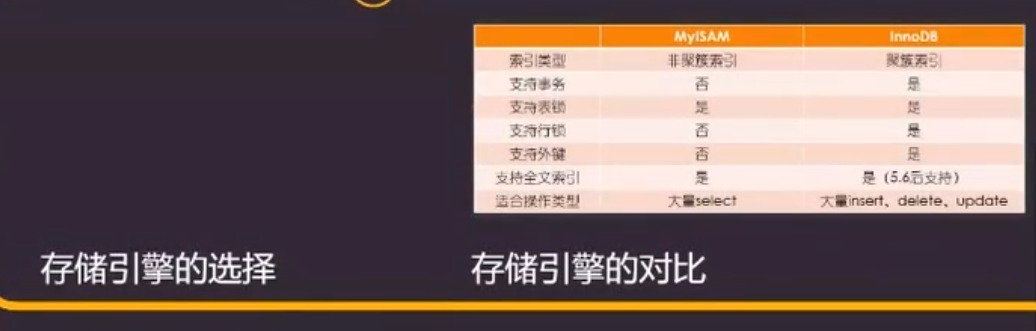
不一定要使用标准范式，适当的冗余字段可以减少表关联，查询更快速



### 主键选择



### 存储引擎的选择



## 执行计划

explain select \* from tb\_category;



## 通过索引进行优化



## 查询优化



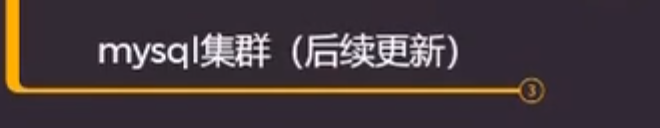
## 分区表



## 服务器参数设置



## mysql集群



# 问题记录

## Navicat Premium 12连接MySQL出现Authentication plugin 'caching\_sha2\_password' cannot be loaded

1. 管理员权限运行命令提示符，登陆MySQL

#登入mysql

   mysql -u root -p

   password

2. 修改账户密码加密规则并更新用户密码

   ALTER USER 'root'@'localhost' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;   #修改加密规则

   ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'password';   #更新一下用户的密码

3. 刷新权限并重置密码

   FLUSH PRIVILEGES;   #刷新权限

再重置下密码：ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';