# 数据库设计

## 什么是数据库范式

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。

## 数据库三大范式

第一范式：1NF是对属性的原子性约束，要求属性(列)具有原子性，不可再分解；(只要是关系型数据库都满足1NF)

第二范式：2NF是对记录的惟一性约束，表中的记录是唯一的, 就满足2NF, 通常我们设计一个主键来实现，主键不能包含业务逻辑。

第三范式：3NF是对字段冗余性的约束，它要求字段没有冗余。 没有冗余的数据库设计可以做到。

但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是： 在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

# 索引

## 什么是索引

索引用来快速地寻找那些具有特定值的记录，所有MySQL索引都以B-树的形式保存。如果没有索引，执行查询时MySQL必须从第一个记录开始扫描整个表的所有记录，直至找到符合要求的记录。表里面的记录数量越多，这个操作的代价就越高。如果作为搜索条件的列上已经创建了索引，MySQL无需扫描任何记录即可迅速得到目标记录所在的位置。如果表有1000个记录，通过索引查找记录至少要比顺序扫描记录快100倍。

## 索引的分类

### 主键索引

主键是一种唯一性索引，但它必须指定为“PRIMARY KEY”。如果你曾经用过AUTO\_INCREMENT类型的列，你可能已经熟悉主键之类的概念了。主键一般在创建表的时候指定，例如“CREATE TABLE tablename ( [...], PRIMARY KEY (列的列表) ); ”。但是，我们也可以通过修改表的方式加入主键，例如“ALTER TABLE tablename ADD PRIMARY KEY (列的列表); ”。每个表只能有一个主键。

#### 创建主键索引

主键是一种唯一性索引，但它必须指定为“PRIMARY KEY”。如果你曾经用过AUTO\_INCREMENT类型的列，你可能已经熟悉主键之类的概念了。主键一般在创建表的时候指定，例如“CREATE TABLE tablename ( [...], PRIMARY KEY (列的列表) ); ”。但是，我们也可以通过修改表的方式加入主键，例如“ALTER TABLE tablename ADD PRIMARY KEY (列的列表); ”。每个表只能有一个主键。

当一张表，把某个列设为主键的时候，则该列就是主键索引

create table aaa

(id int unsigned primary key auto\_increment ,

name varchar(32) not null default '');

这是id 列就是主键索引.

create table bbb (id int , name varchar(32) not null default '');

如果你创建表时，没有指定主键索引，也可以在创建表后，在添加, 指令:

实例:

alter table 表名 add primary key (列名);

删除主键索引

alter table articles drop primary key;

### 查询索引

desc 表名; 不能显示索引名称

show index from 表名

show keys from 表名

### 全文索引

#### 创建表结构

|  |
| --- |
| CREATE TABLE articles (  id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,  title VARCHAR(200),  body TEXT,  FULLTEXT (title,body)  )engine=myisam charset utf8;  INSERT INTO articles (title,body) VALUES  ('MySQL Tutorial','DBMS stands for DataBase ...'),  ('How To Use MySQL Well','After you went through a ...'),  ('Optimizing MySQL','In this tutorial we will show ...'),  ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),  ('MySQL vs. YourSQL','In the following database comparison ...'),  ('MySQL Security','When configured properly, MySQL ...'); |

错误用法:

select \* from articles where body like '%mysql%'; 错误用法 索引不会生效

正确用法:

select \* from articles where match(title,body) against ( 'database')

说明:

在mysql中fulltext 索引只针对 myisam生效

mysql自己提供的fulltext针对英文生效->sphinx (coreseek) 技术处理中文

使用方法是 match(字段名..) against(‘关键字’)

全文索引：停止词, 因为在一个文本中，创建索引是一个无穷大的数，因此，对一些常用词和字符，就不会创建，这些词，称为停止词.比如（a，b，mysql，the）

mysql> select match(title,body) against ('database') from articles;（输出的是每行和database的匹配度）

### 唯一索引

这种索引和前面的“普通索引”基本相同，但有一个区别：索引列的所有值都只能出现一次，即必须唯一。唯一性索引可以用以下几种方式创建：

创建索引，例如CREATE UNIQUE INDEX <索引的名字> ON tablename (列的列表)；

修改表，例如ALTER TABLE tablename ADD UNIQUE [索引的名字] (列的列表)；

创建表的时候指定索引，例如CREATE TABLE tablename ( [...], UNIQUE [索引的名字] (列的列表) )；

#### 创建表结构

create table ddd(id int primary key auto\_increment , name varchar(32) unique);

#### 注意

unique字段可以为NULL,并可以有多NULL, 但是如果是具体内容，则不能重复，

但是不能存有重复的空字符串’’

### 普通索引

 普通索引（由关键字KEY或INDEX定义的索引）的唯一任务是加快对数据的访问速度。因此，应该只为那些最经常出现在查询条件（WHEREcolumn=）或排序条件（ORDERBYcolumn）中的数据列创建索引。只要有可能，就应该选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据列）来创建索引。

create table ccc(

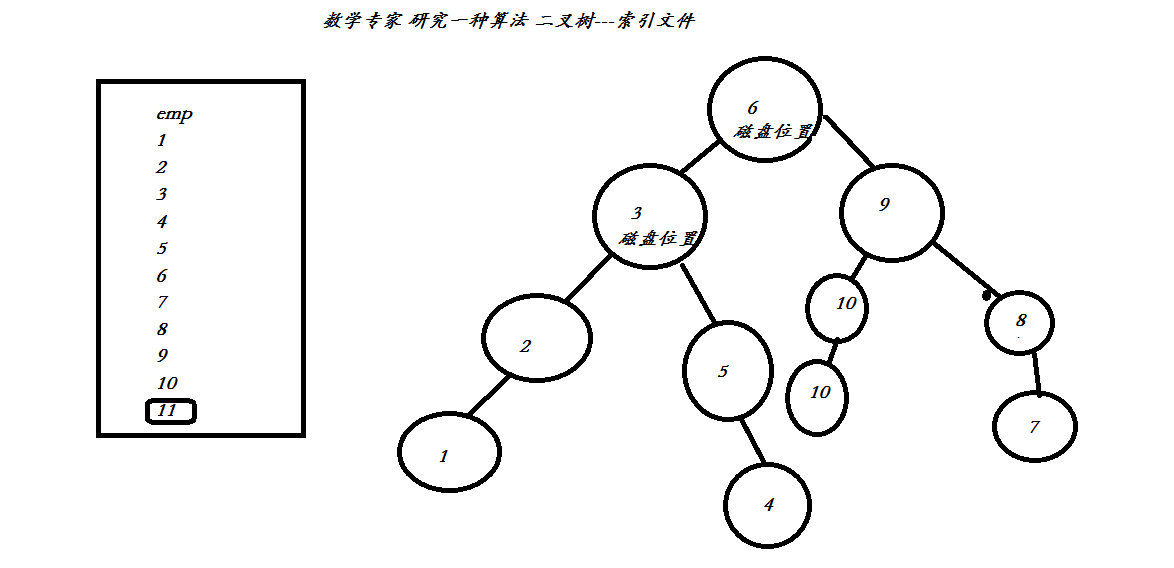
id int unsigned,

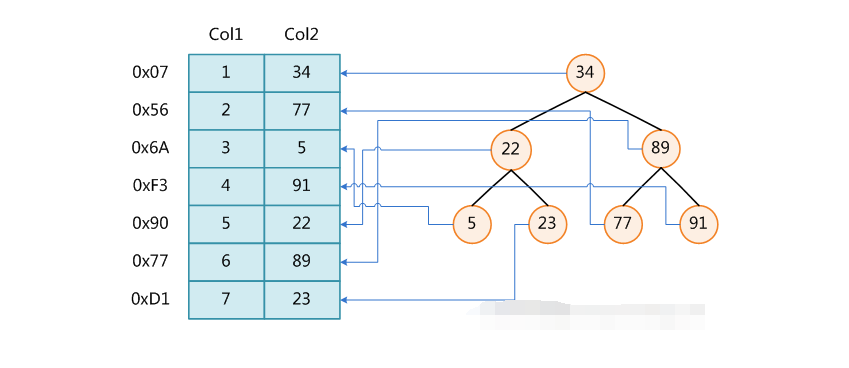
name varchar(32)

)

create index 索引名 on 表 (列1,列名2);

## 索引的实现原理





**数据库索引**，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。**索引的实现通常使用 B 树及其变种 B+ 树**。

## MySQL explain执行计划解读

引言：

实际项目开发中，由于我们不知道实际查询的时候数据库里发生了什么事情，数据库软件是怎样扫描表、怎样使用索引的，因此，我们能感知到的就只有

sql语句运行的时间，在数据规模不大时，查询是瞬间的，因此，在写sql语句的时候就很少考虑到性能的问题。但是当数据规模增大，如千万、亿的时候，我们运

行同样的sql语句时却发现迟迟没有结果，这个时候才知道数据规模已经限制了我们查询的速度。所以，查询优化和索引也就显得很重要了。

**问题：**

当我们在查询前能否预先估计查询究竟要涉及多少行、使用哪些索引、运行时间呢？答案是能的，mysql提供了相应的功能和语法来实现该功能。

分析：

MySql提供了EXPLAIN语法用来进行查询分析，在SQL语句前加一个"EXPLAIN"即可。比如我们要分析如下SQL语句：

explain select \* from table where table.id = 1 

运行上面的sql语句后你会看到，下面的表头信息：

table | type | possible\_keys | key | key\_len | ref | rows | Extra

**EXPLAIN列的解释**

***table***   
显示这一行的数据是关于哪张表的

***type***   
这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq\_reg、ref、range、indexhe和ALL

**说明：不同连接类型的解释（按照效率高低的顺序排序）**

system：表只有一行：system表。这是const连接类型的特殊情况。

const ：表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行，这个值实际就是常数，因为MYSQL先读这个值然后把它当做常数来对待。

eq\_ref：在连接中，MYSQL在查询时，从前面的表中，对每一个记录的联合都从表中读取一个记录，它在查询使用了索引为主键或惟一键的全部时使用。

ref：这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如，利用最左边前缀）时发生。对于之前的表的每一个行联合，全部记录都将从表中读出。这个类型严重依赖于根据索引匹配的记录多少—越少越好。

range：这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西时发生的情况。

index：这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好，因为索引一般小于表数据）。

ALL：这个连接类型对于前面的每一个记录联合进行完全扫描，这一般比较糟糕，应该尽量避免。

***possible\_keys***   
显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的语句

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

***key\_len***   
使用的索引的长度。在不损失精确性的情况下，长度越短越好

***ref***   
显示索引的哪一列被使用了，如果可能的话，是一个常数

***rows***   
MYSQL认为必须检查的用来返回请求数据的行数

***Extra***   
关于MYSQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，意思MYSQL根本不能使用索引，结果是检索会很慢

**说明：extra列返回的描述的意义**

Distinct ：一旦mysql找到了与行相联合匹配的行，就不再搜索了。

Not exists ：mysql优化了LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行，就不再搜索了。

Range checked for each Record（index map:#） ：没有找到理想的索引，因此对从前面表中来的每一个行组合，mysql检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一。

Using filesort ：看到这个的时候，查询就需要优化了。mysql需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行。

Using index ：列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候。

Using temporary ：看到这个的时候，查询需要优化了。这里，mysql需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上。

Where used ：使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型ALL或index，这就会发生，或者是查询有问题。

因此，弄明白了explain语法返回的每一项结果，我们就能知道查询大致的运行时间了，如果查询里没有用到索引、或者需要扫描的行过多，那么可以感到明显的延迟。因此需要改变查询方式或者新建索引。mysql中的explain语法可以帮助我们改写查询，优化表的结构和索引的设置，从而最大地提高查询效率。当然，在大规模数据量时，索引的建立和维护的代价也是很高的，往往需要较长的时间和较大的空间，如果在不同的列组合上建立索引，空间的开销会更大。因此索引最好设置在需要经常查询的字段中。

## 索引的代价

占用磁盘空间

对DML(update、delete、insert)语句的效率影响

增删改会对索引影响，因为索引要重新整理。

|  |  |
| --- | --- |
| 存储引擎 | 允许的索引类型 |
| myisam | btree |
| innodb | btree |
| memory/yeap | Hash,btree |

### 那些列上适合添加索引

查询作为查询条件字段应该创建索引

唯一性太差的字段不适合单独创建索引,即使频繁

Select \* from emp where sex=’男’

频繁更新字段，也不要定义索引。

不会出现在where语句的字段不要创建索引

总结:满处一下条件的字段，才应该创建索引

1. 肯定在where条件经常使用
2. 该字段的内容不是唯一的几个值
3. 字段内容不是频繁变化

# SQL优化技巧

1. 使用group by 分组查询是，默认分组后，还会排序，可能会降低速度，

在group by 后面增加 order by null 就可以防止排序.

explain select \* from emp group by deptno order by null;

1. 有些情况下，可以使用连接来替代子查询。因为使用join，MySQL不需要在内存中创建临时表。

select \* from dept, emp where dept.deptno=emp.deptno; [简单处理方式]

select \* from dept left join emp on dept.deptno=emp.deptno; [左外连接，更ok!]

1. 对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引

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，如：

select id from t where num is null

最好不要给数据库留 NULL，尽可能的使用 NOT NULL 填充数据库.

备注、描述、评论之类的可以设置为 NULL，其他的，最好不要使用 NULL。

不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（NULL 也包含在内），都是占用 100 个字符的空间的，如果是 varchar 这样的变长字段， null 不占用空间。

可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：

select id from t where num = 0

# MySQL数据引擎

使用的存储引擎 myisam / innodb/ memory

myisam 存储: 如果表对事务要求不高，同时是以查询和添加为主的，我们考虑使用myisam存储引擎. ,比如 bbs 中的 发帖表，回复表.

INNODB 存储: 对事务要求高，保存的数据都是重要数据，我们建议使用INNODB,比如订单表，账号表.

MyISAM 和 INNODB的区别

1. 事务安全（MyISAM不支持事务，INNODB支持事务）

2. 查询和添加速度（MyISAM批量插入速度快）

3. 支持全文索引（MyISAM支持全文索引，INNODB不支持全文索引）

4. 锁机制（MyISAM时表锁，innodb是行锁）

5. 外键 MyISAM 不支持外键， INNODB支持外键. (在PHP开发中，通常不设置外键，通常是在程序中保证数据的一致)

Memory 存储，比如我们数据变化频繁，不需要入库，同时又频繁的查询和修改，我们考虑使用memory, 速度极快. （如果mysql重启的话，数据就不存在了）



# Myisam注意事项

如果你的数据库的存储引擎是myisam,请一定记住要定时进行碎片整理

举例说明:

create table test100(id int unsigned ,name varchar(32))engine=myisam;

insert into test100 values(1,’aaaaa’);

insert into test100 values(2,’bbbb’);

insert into test100 values(3,’ccccc’);

insert into test100 select id,name from test100;

我们应该定义对myisam进行整理

optimize table test100;