# 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:

1. 在mysql中fulltext 索引只针对 myisam生效
2. mysql自己提供的fulltext针对英文生效->sphinx (coreseek) 技术处理中文
3. 使用方法是 match(字段名..) against(‘关键字’)
4. 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(

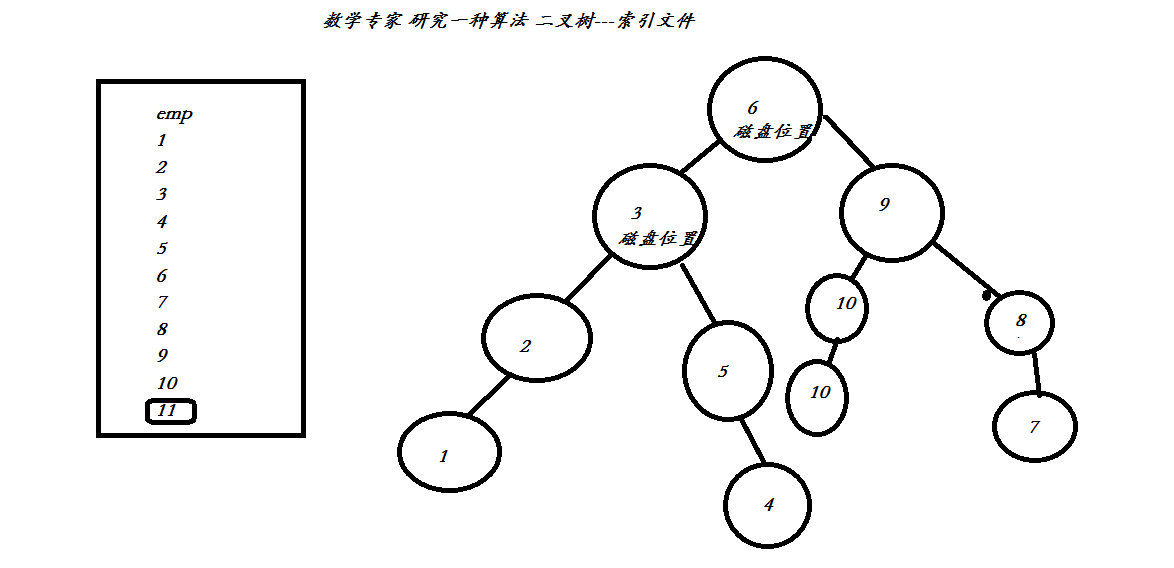
id int unsigned,

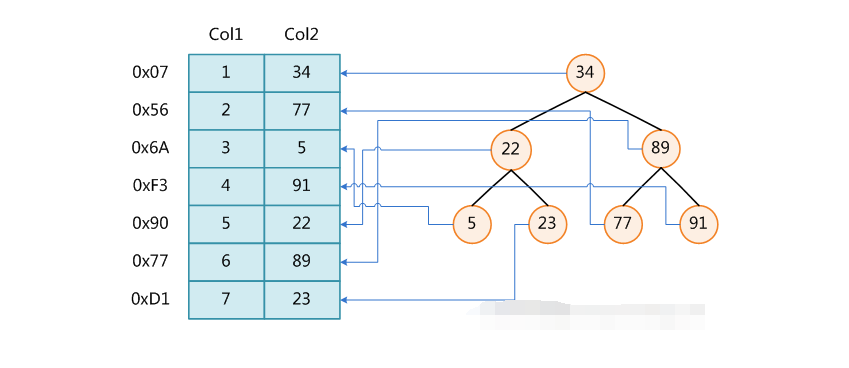
name varchar(32)

)

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**。

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。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

上图展示了一种可能的索引方式。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（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。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 O(log2n)的复杂度内获取到相应数据。

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。**一般来说，应该在这些列上创建索引：**在经常需要搜索的列上，可以加快搜索的速度；在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

同样，对于有些列不应该创建索引。**一般来说，不应该创建索引的的这些列具有下列特点：**

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，**修改性能和检索性能是互相矛盾的**。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

根据数据库的功能，可以在数据库设计器中创建三种索引：**唯一索引、主键索引和聚集索引**。

**唯一索引**

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在 employee 表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。**主键索引**数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。**聚集索引**在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

局部性原理与磁盘预读

由于存储介质的特性，磁盘本身存取就比主存慢很多，再加上机械运动耗费，磁盘的存取速度往往是主存的几百分分之一，因此为了提高效率，要尽量减少磁盘 I/O。为了达到这个目的，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。这样做的理论依据是计算机科学中著名的**局部性原理**：**当一个数据被用到时，其附近的数据也通常会马上被使用。程序运行期间所需要的数据通常比较集中。**

由于磁盘顺序读取的效率很高（不需要寻道时间，只需很少的旋转时间），因此对于具有局部性的程序来说，预读可以提高 I/O 效率。

预读的长度一般为页（page）的整倍数。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（在许多操作系统中，页得大小通常为 4k），主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后异常返回，程序继续运行。

B-/+Tree 索引的性能分析

到这里终于可以分析 B-/+Tree 索引的性能了。

上文说过一般使用磁盘 I/O 次数评价索引结构的优劣。先从 B-Tree 分析，根据 B-Tree 的定义，可知检索一次最多需要访问 h 个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次 I/O 就可以完全载入。为了达到这个目的，在实际实现 B-Tree 还需要使用如下技巧：

每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个 node 只需一次 I/O。

**B-Tree 中一次检索最多需要 h-1 次 I/O（根节点常驻内存），渐进复杂度为 O(h)=O(logdN)。**一般实际应用中，出度 d 是非常大的数字，通常超过 100，因此 h 非常小（通常不超过 3）。

而红黑树这种结构，h 明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性，所以红黑树的 I/O 渐进复杂度也为 O(h)，效率明显比 B-Tree 差很多。

**综上所述，用 B-Tree 作为索引结构效率是非常高的。**

**应该花时间学习 B-树和 B+ 树数据结构**

=============================================================================================================

1）B 树

B 树中每个节点包含了键值和键值对于的数据对象存放地址指针，所以成功搜索一个对象可以不用到达树的叶节点。

成功搜索包括节点内搜索和沿某一路径的搜索，成功搜索时间取决于关键码所在的层次以及节点内关键码的数量。

在 B 树中查找给定关键字的方法是：首先把根结点取来，在根结点所包含的关键字 K1,…,kj 查找给定的关键字（可用顺序查找或二分查找法），若找到等于给定值的关键字，则查找成功；否则，一定可以确定要查的关键字在某个 Ki 或 Ki+1 之间，于是取 Pi 所指的下一层索引节点块继续查找，直到找到，或指针 Pi 为空时查找失败。

2）B+ 树

B+ 树非叶节点中存放的关键码并不指示数据对象的地址指针，非也节点只是索引部分。所有的叶节点在同一层上，包含了全部关键码和相应数据对象的存放地址指针，且叶节点按关键码从小到大顺序链接。如果实际数据对象按加入的顺序存储而不是按关键码次数存储的话，叶节点的索引必须是稠密索引，若实际数据存储按关键码次序存放的话，叶节点索引时稀疏索引。

B+ 树有 2 个头指针，一个是树的根节点，一个是最小关键码的叶节点。

所以 B+ 树有两种搜索方法：

一种是按叶节点自己拉起的链表顺序搜索。

一种是从根节点开始搜索，和 B 树类似，不过如果非叶节点的关键码等于给定值，搜索并不停止，而是继续沿右指针，一直查到叶节点上的关键码。所以无论搜索是否成功，都将走完树的所有层。

B+ 树中，数据对象的插入和删除仅在叶节点上进行。

这两种处理索引的数据结构的不同之处：  
a，B 树中同一键值不会出现多次，并且它有可能出现在叶结点，也有可能出现在非叶结点中。而 B+ 树的键一定会出现在叶结点中，并且有可能在非叶结点中也有可能重复出现，以维持 B+ 树的平衡。  
b，因为 B 树键位置不定，且在整个树结构中只出现一次，虽然可以节省存储空间，但使得在插入、删除操作复杂度明显增加。B+ 树相比来说是一种较好的折中。  
c，B 树的查询效率与键在树中的位置有关，最大时间复杂度与 B+ 树相同(在叶结点的时候)，最小时间复杂度为 1(在根结点的时候)。而 B+ 树的时候复杂度对某建成的树是固定的。可以扫描2的次方。

## 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

1. 查询作为查询条件字段应该创建索引
2. 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’

1. 频繁更新字段，也不要定义索引。
2. 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

## 索引的注意事项

### 创建一张表

新增dept 数据

|  |
| --- |
| create PROCEDURE insert\_dept(in start int(10),in max\_num int(10))  BEGIN  declare i int DEFAULT 0;  set autocommit=0;  REPEAT  set i=i+1;  insert into dept values ((start+i),rand\_string(10),rand\_string(8));  UNTIL i =max\_num  end REPEAT;  commit;  END  执行  call insert\_dept(100,10); |

创建主键索引

alter table 表名 add primary key (列名);

创建一个联合索引

alter table dept add index my\_ind (dname,loc); // dname 左边的列,loc就是右边的列

注意:

1.对于创建的多列索引，如果不是使用第一部分，则不会创建索引。

explain select \* from dept where loc='aaa'\G

就不会使用到索引

2.模糊查询在like前面有百分号开头会失效。

3. 如果条件中有or，即使其中有条件带索引也不会使用。换言之，就是要求使用的所有字段，都必须建立索引, 我们建议大家尽量避免使用or 关键字

4.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来。否则不使用索引。(添加时,字符串必须’’), 也就是，如果列是字符串类型，就一定要用 ‘’ 把他包括起来.

5.如果mysql估计使用全表扫描要比使用索引快，则不使用索引。

## 查询所用使用率

show status like ‘handler\_read%’;

大家可以注意：  
handler\_read\_key:这个值越高越好，越高表示使用索引查询到的次数。

handler\_read\_rnd\_next:这个值越高，说明查询低效。