索引

1. 基本介绍

作用

- 提升查询效率
- 会降低增删改效率

语法

创建索引

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX <索引名> ON <表名> <列名1,列名2,...>
```

删除索引

```
DROP INDEX <索引名> ON <表名>
```

查看索引

```
SHOW INDEX FROM <表名>
```

类型

1. 普通索引：经常使用
2. 唯一索引：当创建unique唯一约束的使用，数据库会自动创建，没有重复项，可以有一个null
3. 主键索引：创建primary key主键的时候，会自动创建，没有重复项，没有null
4. 组合索引：同时创建在多列上
5. 全文索引：全库

6. 空间索引：全磁盘空间

数据库B+树的实现原理

详见b站视频 为什么要使用B+ Tree

在主键约束中，即使乱序插入，最后也会按照主键顺序排序，这是因为主键自带索引

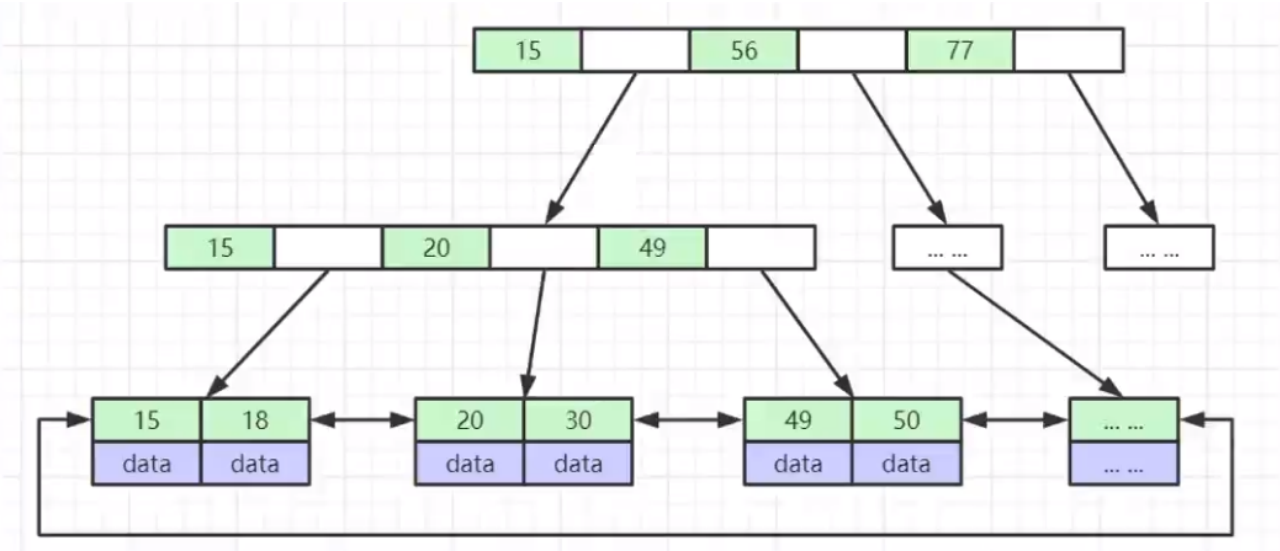
B+树介绍

全部的 **索引 + 数据段** 存在最底层（叶子节点），叶子节点以链表形式相连

中间节点只记录部分叶子节点的索引，不包含**数据段**（这里是B树和B+树的区别）

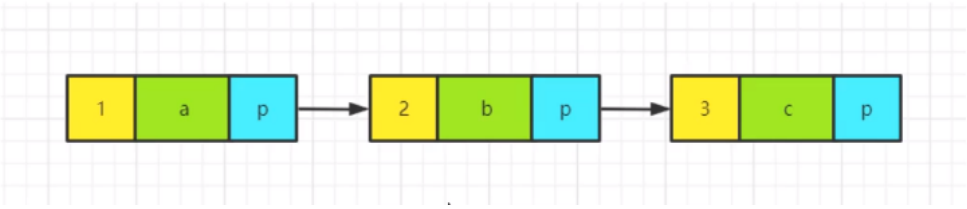
每个中间节点会包含多个索引，按照索引**升序排列**，也就B+树每个节点会指向多个子节点

以此往上增长B+树



初始数据存储结构

数据库存储的内部结构类似于一个链表的形式，通过指针关联不同的数据

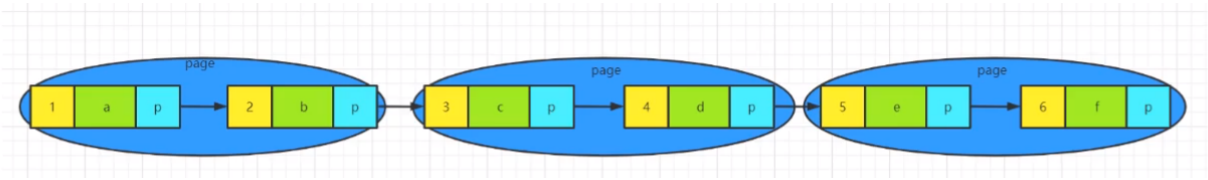


- 黄色：id
- 绿色：内容段
- 蓝色：指针，指向下一个节点

如果进行数据查找，特别是在大数据量的情况下，查找速度会非常慢

Page

MySQL会给数据进行分页，把一部分数据存入page中



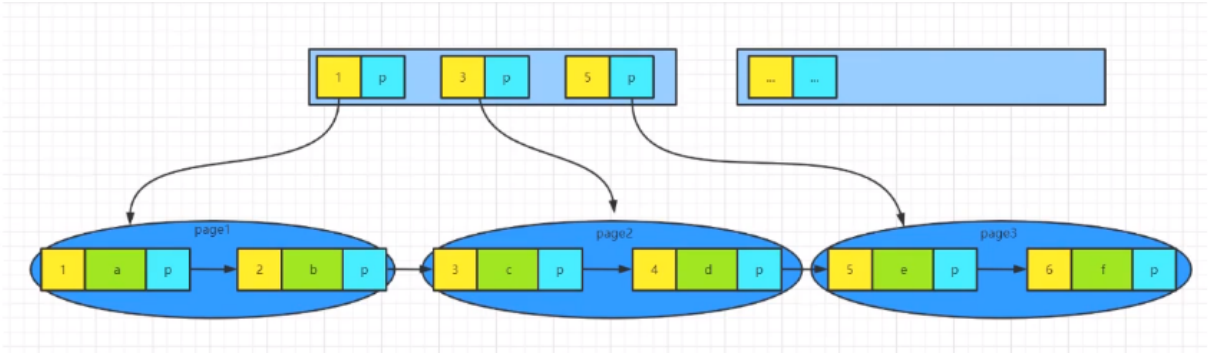
每个 page 可以存储 16KB 的数据

这就相当于给数据建立了上层目录，先找大目录，再找具体的数据

MySQL给 page 也提供了快速查询的 page 目录

Page目录

存每个 page 的第一个数据的 id 和指针存入 page 目录里面

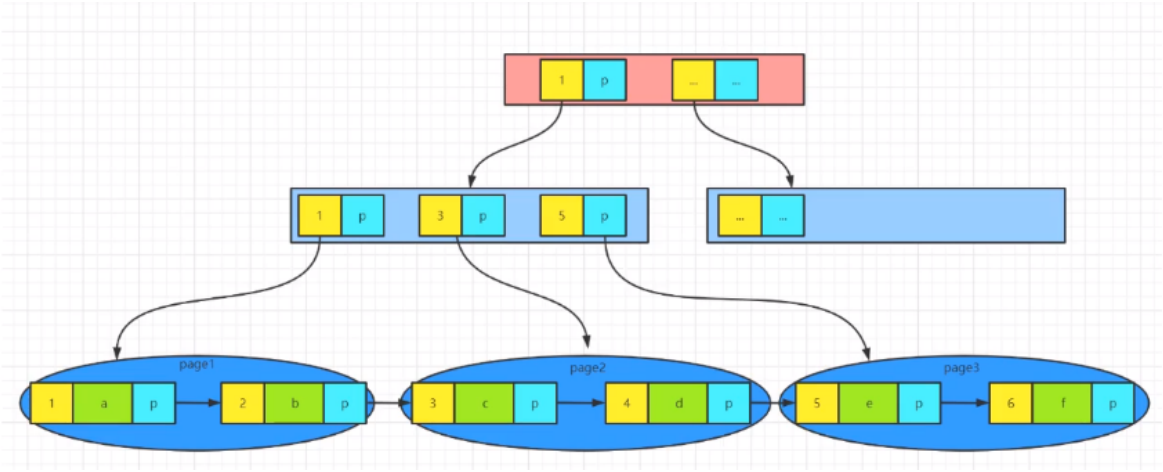


查找数据时，首先找到它的 page，再进入 page 中查找具体的数据

一个 page 目录中也可以存储 16KB 的数据

Page目录的目录

为了提高查询效率，MySQL会给 page 目录再加一层目录



三层的结构就是 B+ 树的结构了

B+树的存储大小推算

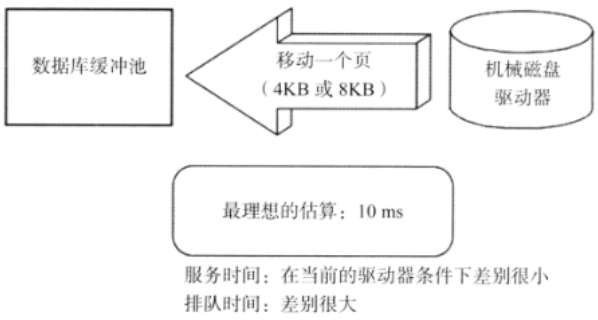
- 假设一条记录的空间是 `32 byte`
- 那么一个 page 可以存储 `16*1024/32 = 512` 条记录
- 一个 page 目录存储 page 的第一个 id 和指针，大约占据 `6 byte`，所以一个 page 目录可以存储 `16*1024/6 ≈ 2730` 个 page
- page 目录的目录同样存储 page 目录的第一个 id 和指针，同样可以存储 `2730` 个 page 目录
- 所以最终可以存储的数据为 `512*2730*2730 ≈ 38亿`

三层 B+ 树可以存储亿/千万级别的数据

2. 缓冲池和磁盘I/O

从磁盘驱动器进行随机I/O

从磁盘驱动器中加载每一页的成本都是一样的，最理想的估算是 10ms，如果磁盘驱动器被过去重复使用，那么这将大福降低查询速度

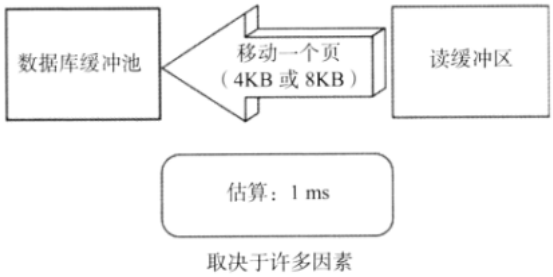


缓冲池作用

为了确保表或者索引中的数据是随时可用的，我们使用内存中的**缓冲池**来**最小化磁盘活动**

缓冲池处理器将尽力确保**经常使用的数据被保存在池中**

索引和表页在或不再缓冲池中，会**影响访问成本**



磁盘服务器会判断该页是否在数据库缓冲池中，如果 在磁盘服务器的读缓冲区中，那么**花费时间将从 10ms 降低到 1ms**

顺序读取

顺序读取页有非常重要的优势

- 同时读取多个页意味着平均读取每个页的时间将减少，当前磁盘服务器的条件下的，对于 4KB 大小的页而言，这一值可以降低到 0.1ms
- 由于 DBMS 事先知道要读取哪些页，所以在页被真正请求之前就应该将其读取进来，被称为**预读**

同步 I/O 和异步 I/O

同步 I/O：在进行I/O操作，DBMS不能进行其它操作，必须等待完成

异步 I/O：当前一步的页尚在处理时被提前发起了，这一处理时间和 I/O 时间之间可能有很大一部分重叠，每页都以这种方式被预读案后再处理

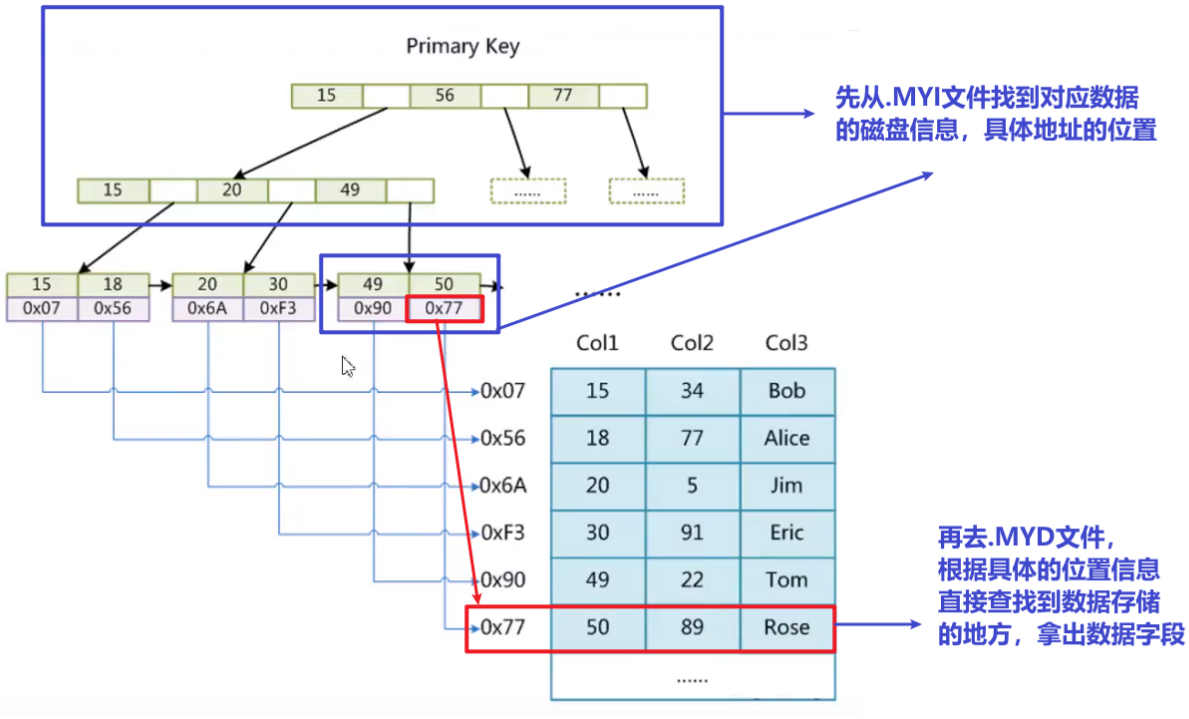
2. MySQL数据库具体的存储文件

打开MySQL内部存储地段，我们可以发现

在 `data` 里面存储着各个数据库（schema）

点击进去后有各种表的信息，其中包含

- `.frm`：表框架信息
- `.MYD`：数据信息
- `.MYI`：索引信息



3. 索引设计

创建索引的目的是在硬件容量的限制的前提下把证所有的数据库调用运行得足够快

系统化索引设计

1. 找到由于索引不合适而导致运行太慢的查询语句
   最差输入：导致执行时间最长的变量值
2. 设计索引，使所有查询语句都运行的足够快
   表的维护（插入、更新、删除也必须足够快）

根据谓词确定索引

WHERE 子句由一个或者多个谓词（搜索参数）组成，例如

```
SELECT ...
FROM ...
WHERE sex = 'M' AND (weight > 90 OR height > 190)
```

谓词表达式是索引设计的入手点，如果一个索引能满足 SELECT 查询语句的所有谓词表达式，那么优化器很有可能建起一条高效的访问路径

有时候，有些列可能既存在于 WHERE 子句中，也存在索引中，但这个列却不能参与索引片的定义，不过这些列仍然能够减少回表进行同步读的操作，我们称这些列为**过滤列**

假设我们拥有索引 (A,B,C,D)

那么匹配方法如下（按 WHERE 顺序匹配）

1. 在WHERE子句，该**列是否至少拥有一个足够简单的谓词与之对应**
   - 如果有，那么这个列是**匹配列**
   - 如果没有，那么这个列以及其后的索引列都是**非匹配列**
2. 如果该谓词是一个**范围谓词**，那么剩余的索引列都是**非匹配列**
3. 对于最后一个匹配列之后的索引列，如果拥有一个**足够简单的谓词与其对应**，那么该列为**过滤列**

```
WHERE
    sex = 'M' AND        -- 等值谓词，是匹配列
    weight > 90          -- 范围谓词，是匹配列，但是剩下的列不能参与匹配过程，它不能定能一索引片
    OR height > 190      -- 非匹配列，但是有一个简单的谓词对应，是过滤列

```

根据目标确定索引的建立规范

索引的建立标准 CURSOR41

1. 取出所有等值谓词的列 **WHERE <列名> = ...**，把这些列作为索引最开头的列——以任意顺序都可以
2. 将 **ORDER BY** 列添加到索引中，不要该变这些列的顺序，忽略那些在第一步已经添加到索引的列
3. 将查询语句中剩余的列添加到索引中去，列在索引中添加的顺序对查询语句的性能没有影响，但是将易修改的列放在最后可以降低更新的成本

注意 CURSOR41 在下面的方面比较挑剔

- WHERE 条件**不包含范围谓词 BETWEEN、>、>=**等
- FROM 语句仅涉及**单表**
- 所有谓词对于优化器来说够简单

索引建立标准 CURSOR43

当查询语句中涉及到范围谓词的时候，很多时候三个条件不能满足，这时候根据优先 ORDER BY 列还是优先范围谓词列会产生两种建立索引的方法

第一种

1. 取出对于优化器来说不过分复杂的**等值谓词列**，将这些列作为索引的前导列——以任意顺序皆可
2. 将选择性最好的**范围谓词**作为索引的下一个列，如果存在的话。最好的选择性是指对于最差的输入值有最低的过滤因子。只考虑"对于优化器来说不过分复杂的范围谓词即可
3. 以正确的顺序添加**ORDER BY列** (如果ORDER BY列有DESC的话,加上DESC)，**忽略在第1步或第2步中已经添加的列**，以任意顺序将SELECT语句中**其余的列添加至索引中**（但是需要以**不易变**的列开始）

第二种

1. 取出对于优化器来说不过分复杂的**等值谓词列**，将这些列作为索引的前导列——以任意顺序皆可
2. 以正确的顺序添加**ORDER BY**列(如果ORDER BY列有DESC的话,加上DESC)。忽略在第1步中已经添加的列
3. 以任意顺序将**SELECT语句中其余的列**添加至索引中(但是需要以不易变的列开始)

如果**第一种方法**使用该索引时**不需要涉及排序**，那么它是最优索引

如果实在无法满足 CURSOR41 中的三个条件，且需要排序，那么**第二种方法**将是相应查询语句的最佳索引

创建标准

按照下列标准选择建立索引的列

- 频繁搜索的列
- 经常用作查询选择的列
- 经常排序、分组的列
- 经常用作连接的列（主键/外键）

不要使用下面情况的列创建索引

- 仅包含几个不同值的列
- 表中仅包含几行

索引的注意

- 查询时减少使用`*`返回全部列，不要返回不需要的列
- 索引应该尽量小，在字节数小的列上建立索引
- WHERE子句中有多个条件表达式的时候，包含索引列的表达式应置于其它条件表达式之前
- 避免在ORDER BY子句中使用表达式

索引的误区

- 现在 B+ 树的所有非叶子页通常会留在内存或读缓存中，通常只有叶子页需要从磁盘驱动器中读取
- 索引层级可以超过5层，对索引的层数做强制限制没有什么意义
- 单表中包含的索引数可以超过6个
- 可以索引不稳定的列

索引不被执行的情况

1. 数据库认为使用索引没有遍历速度快的时候
2. 建立了联合索引，不按顺序/缺少部分列的情况
3. 使用 LIKE/SUBSTRING/UPPER/LOWER/日期函数

4. 使用函数
5. 使用数据类型转换

最好在插入数据的时候就进行规范化，避免大小写/日期规范问题

如果想要用函数查询，我们插入的时候就可以多建一列，该列插入了可能会使用函数的数据

大多数产品实际上允许您通过索引表达式（有时称为“生成的”或“虚拟的”）来做一些更简洁的事情，这可能是一个函数的结果，只有当表达式或函数是确定的（如果一个输入对应着一个确定的输出）才可以使用

冗余索引

完全冗余

如果有

```
WHERE a = 1 AND b = 2
WHERE b = 1 AND a = 2
```

两条语句，而你为它设计了两条索引

index(A,B) 和 index(B,A)

那么这两条索引中一条是完全多余的

可能冗余

如果有一个索引，它的所有列的顺序被另一个索引所包含，那么这个索引是多余的索引

4. 执行计划

如果在列上有索引，则DBMS是否使用它呢？不一定。优化器可能决定不使用索引。有没有办法知道它是否会引用？

```
EXPLAIN <查询语句>
```

所有 DBMS 产品都实现了一种显示所谓的“执行计划”的方法（语法稍有不同），即看到优化器会选择如何运行查询