MySQL——B+树索引

Icodeer

一、没有索引时是如何查找数据的

- 一共可以分为**两种**解决方案。

1.1> 方案1：在一个页中查找

- 比如数据较少，都存储在一个页中。在查找记录时，是要根据搜索条件的不同分为**两种**情况的。
- 以**主键列**作为搜索条件

> Step 1> 在页目录（`Page Directory`）中使用**二分法**快速定位到对应的槽（`Slot`）。
>
> Step 2> 然后再遍历该槽对应**分组**中的记录。

- 以**其他列**作为搜索条件

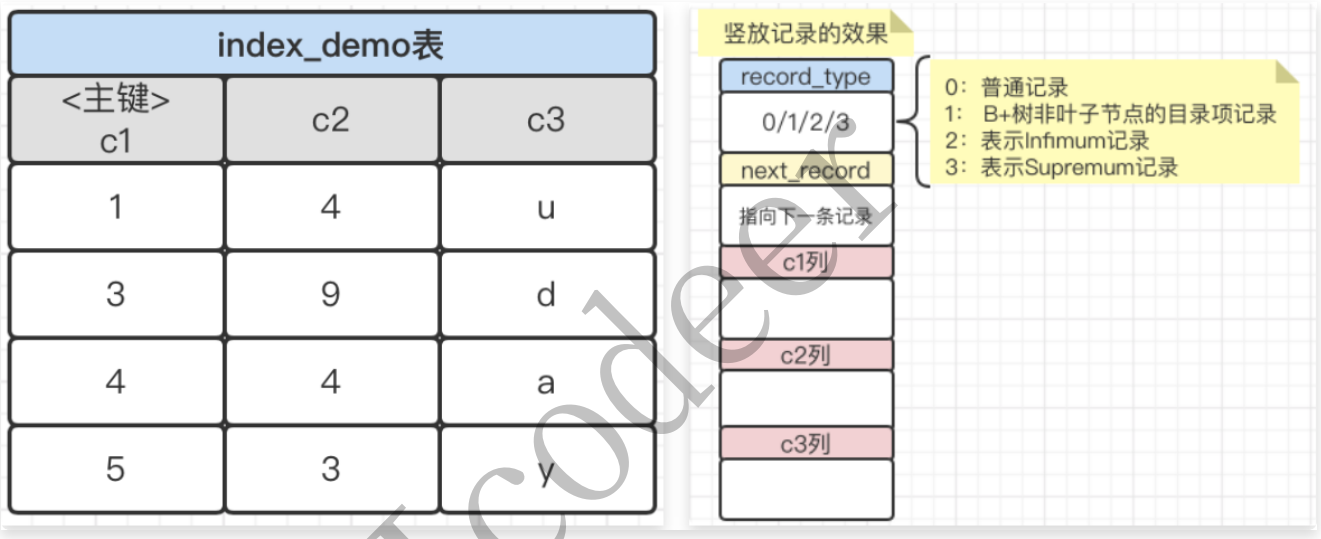
> 因为在数据页中，没有为非主键建立 `Page Directory` 。只能**从Infimum记录**开始依次遍历 `单向链表` 中的每条记录，然后对比每条记录是否符合搜索条件。

1.2> 方案2：在多个页中查找

- 大多情况，数据量都是很多的，那么就会涉及到多个页中的查找。由于没有索引，我们无法定位到记录所在的页，所以**只能从第1页开始**，利用【在1个页中查找】的方式，遍历所有的数据页来查找，非常耗时。

二、索引

- 数据准备阶段，创建表 `index_demo`




```
create table index_demo {
  c1 INT,
  c2 INT,
  c3 CHAR(1),
  PRIMARY KEY(c1)
} ROW_FORMAT=COMPACT;
```

2.1> 一个简单的索引方案的思考

- 准备工作——在 `index_demo` 表中插入 `12` 条数据

Icoder

index_demo表		
<主键> c1	c2	c3
1	4	u
3	9	d
4	4	a
5	3	y
8	7	a
10	4	o
12	7	d
20	2	e
100	9	x
209	5	b
220	6	i
300	8	a

> *前提：假设一个Page数据页中，最多只能存放 `3` 条记录*

- 由于遍历链表查询数据效率太低了，我们思考，能否**借用Page Directory**的思想，来创建简单的目录索引，在创建这个目录的过程中必须完成两件事：

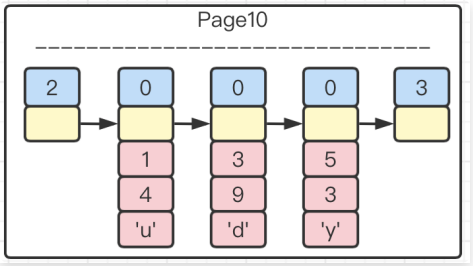
> 【首先】下一个数据页中用户记录的**主键值必须大于**上一个页中用户记录的主键值。
>
> 【其次】给所有的页建立一个**目录项**。

2.1.1> 步骤1：下一个数据页中用户记录的主键值必须大于上一个页中用户记录的主键值。

- 插入三条数据（Page页满了）

```
insert into index_demo values(1, 4, 'u');
insert into index_demo values(3, 9, 'd');
insert into index_demo values(5, 3, 'y');
```

- 则数据页中数据结构如下所示：



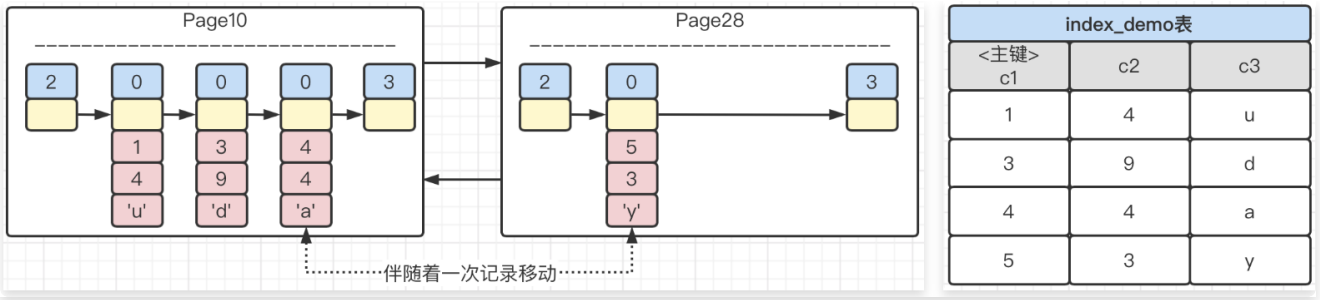
index_demo表		
<主键> c1	c2	c3
1	4	u
3	9	d
5	3	y

- 再插入一条记录（需要申请新的页面来存放数据）

```
insert into index_demo values(4, 4, 'a');
```

- 则数据页中数据结构如下所示：

Icodeer



- 由于新插入的主键为 `4` ，小于 `5` ，所以，新数据会插入到 `Page10` 中，c1=5这条数据，会插入到新申请的页 `Page28` 中。这个过程也叫**页分裂**。
- 那么，*为什么新申请的页不是 `Page11` ？*

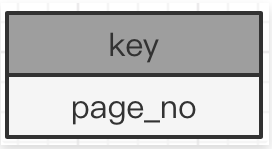
> 因为新分配的数据页编号**可能并不是连续**的，但是InnoDB也会**尽量让这些页面相邻**（这个问题，会在表空间章节中介绍）。

2.1.2> 步骤2：给所有的页建立一个目录项

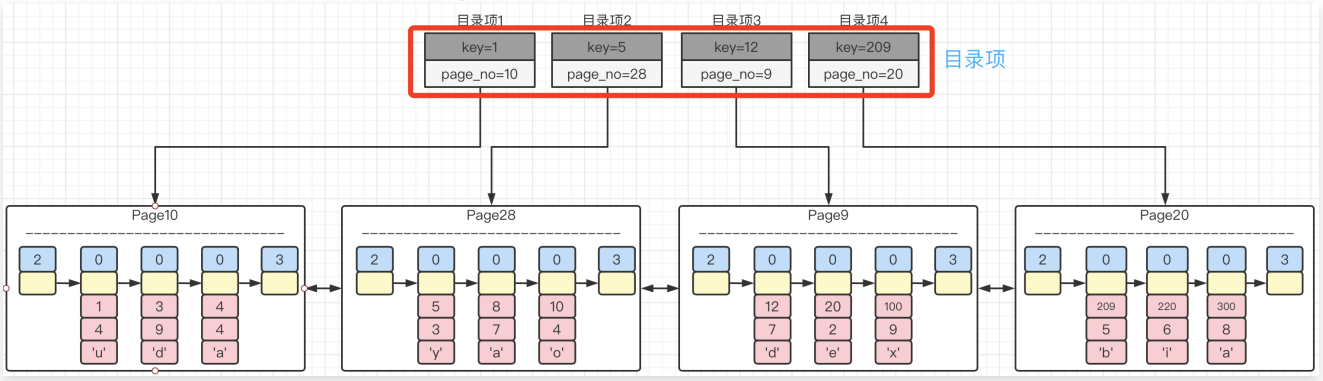
- 给每个页建立**目录项**，包含如下两个内容：

> 【key】页的用户记录中 `最小的主键值` 。
> 【page_no】记录 `页号` 。

- 目录项



- 为页编制目录



- 我们现在来尝试的去查找 `主键c1` 为 `20` 的记录，看一下具体的处理过程：

> 【步骤1】先从目录项中根据**二分法**快速确定出 `key=20` 的记录在 `目录项3` 中，找到对应的 `page_no=9`

Icodeer

> 【步骤2】再去 `Page9` 中，采用 **【二分法Page Directory来定位组 + 组内遍历链表查找记录】** 的方式定位具体记录。

2.2> InnoDB中的索引方案

2.2.1> 简单的索引方案存在的问题

- 上面的简单的索引方案还是**存在一些问题**，比如：为了使用**二分法**快速定位具体的目录项，就要求所有目录项都可以在**物理存储器上连续存储**。这样会导致两个问题：

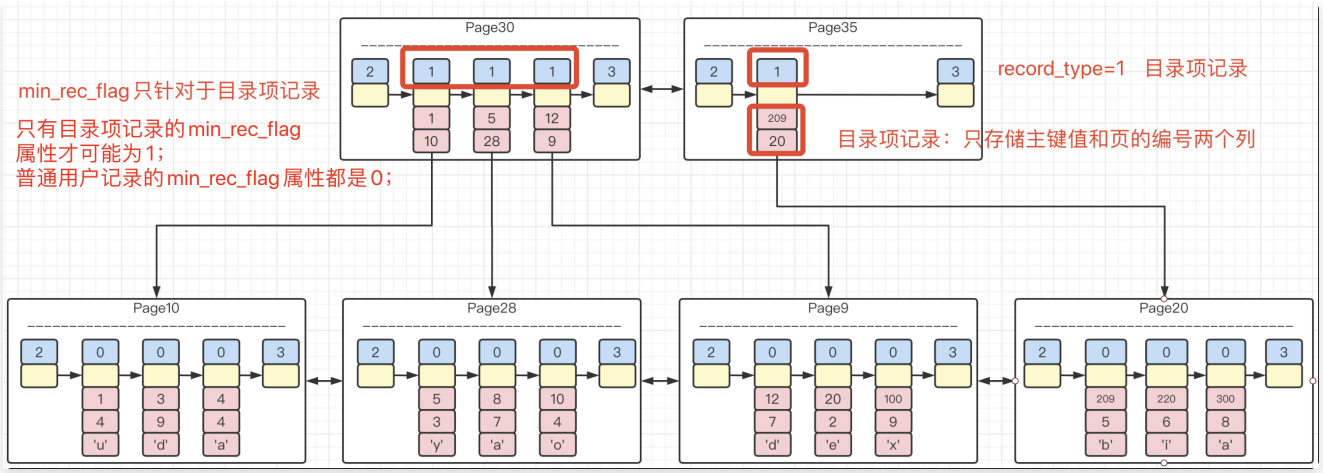
> 【**问题1**】由于 `页` 是InnoDB管理存储空间的基本单位，也就是说最多只能保证**16KB**的连续存储空间。所以，如果数据越来越多，那么16KB是不够的。换句话说：就是当数据量越来越大的时候，是**无法保证目录项连续存储**的，也就无法使用二分法快速定位了。
>
> 【**问题2**】假设删除了 `Page28` 里面的数据，那么 `目录项2` 也就没有了意义。为了目录项是**连续存储**的，我们有两种处理方式：
>
> - **方式1**：删除目录项2，然后将它后面所有的目录项都向前移动，填补空缺空间。（缺点：有性能损耗，数据量越大，性能损耗越大）
> - **方式2**：不去删除项目2，这样就不用移动其他目录项。（缺点：没有了性能损耗，但是却因为垃圾数据造成空间浪费）

- 解决办法：

> 采用**数据页**来存储目录项，将**目录项作为记录保存**到数据页中。为了与用户记录进行区分，把这些用来表示目录项的记录称之为**目录项记录**。*如何区分一条记录是普通记录还是目录项记录呢?*
>
> `答：采用记录头信息中的record_type属性：`
>
> `0：普通的用户记录。`
>
> `1：目录项记录。`
>
> `2：Infimum记录。`
>
> `3：Supremum记录。`

- 将目录项放到数据页中的效果，如下图所示：

Icodeer



- *普通用户记录和目录项的区别有哪些?*

【不同点】

> - `目录项记录` 的**record_type=1**，而 `普通用户记录` 的**record_type=0**。
> - `目录项记录` 只包含**主键值**和**页的编号**两个列；而 `普通用户记录` 的列是**用户自己定义**的，另外还有InnoDB自己添加到**隐藏列**。
> - `min_rec_flag` 属性，只有目录项记录的这个属性才可能为**min_rec_flag=1**，普通用户记录的 `min_rec_flag` 都是0。

【相同点】

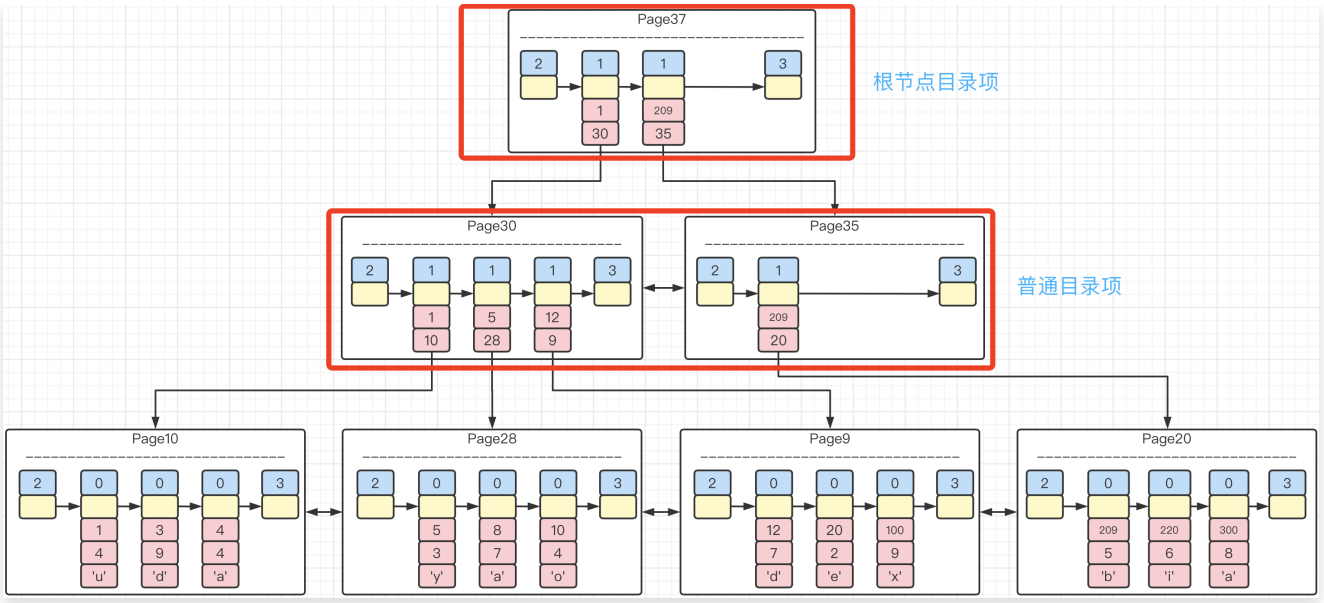
> - 它们用的都是一样的**数据页**。
> - 它们页的**组成结构**也是一样的（就是我们前面介绍的那 `7` 个部分）。
> - 都会**为主键生成页目录Page Directory**。
> - 按**主键值**进行查找的时候，可以使用**二分法**来加快查询速度。

2.2.2> B+树

- 由于一个页我们假设只能存储 `3条` 记录，那么目录项就生成了两个—— `Page30` 和 `Page35` 。
  那如果数据增多，目录项也会扩展很多，那怎么快速定位目录项呢?

> 答案：再往上层建立节点；而这种结果，就是 `B+树` 了。如下所示：

Icodeer



- 我们真正的**用户记录**其实都存放在**B+树最底层**的节点上，即：`叶子节点`。
- 其他用来存放**目录项记录**的节点称为`非叶子节点`或`内节点`。
- B+树**最上面**的节点叫`根节点`。
- 其中**最下层**为`第0层`，往上层一次递增。

- *如果是按照上面的说法，往上面建立更高层的节点，那么会不会实际情况下，B+树会有好多层呢？*
  我们可以大致计算一下，假设存储目录项的页可以存放`1000`条目录项记录；存储用户记录的页可以存放`100`条记录；

> 【第1层】那么也就是说，只有1个用于存放用户记录的页节点（leaf Page），那么最多可以存储`100`条用户记录。
> 【第2层】那么最多可以存放 100 * 1000 = `10万`条记录。
> 【第3层】那么最多可以存放 100 * 1000 * 1000 = `1亿`条记录。
> 【第4层】那么最多可以存放 100 * 1000 * 1000 * 1000 = `1000亿`条记录。

- 所以，综上所述，一般情况下，我们用到的B+树都**不会超过4层**。数据页的`Page Header`部分，介绍过一个名为**PAGE_LEVEL**的属性，它就代表着这个数据页作为节点在B+树中的层级。

2.2.3> 聚簇索引

- 在InnoDB存储引擎中，聚簇索引就是数据的存储方式，也就是所谓的“**索引即数据，数据即索引**”。
- 聚簇索引只能在**搜索条件是主键值**时才能发挥作用，原因是B+树中的数据都是按照主键进行排序的。
- 聚簇索引有如下两个特点：

Icodeer

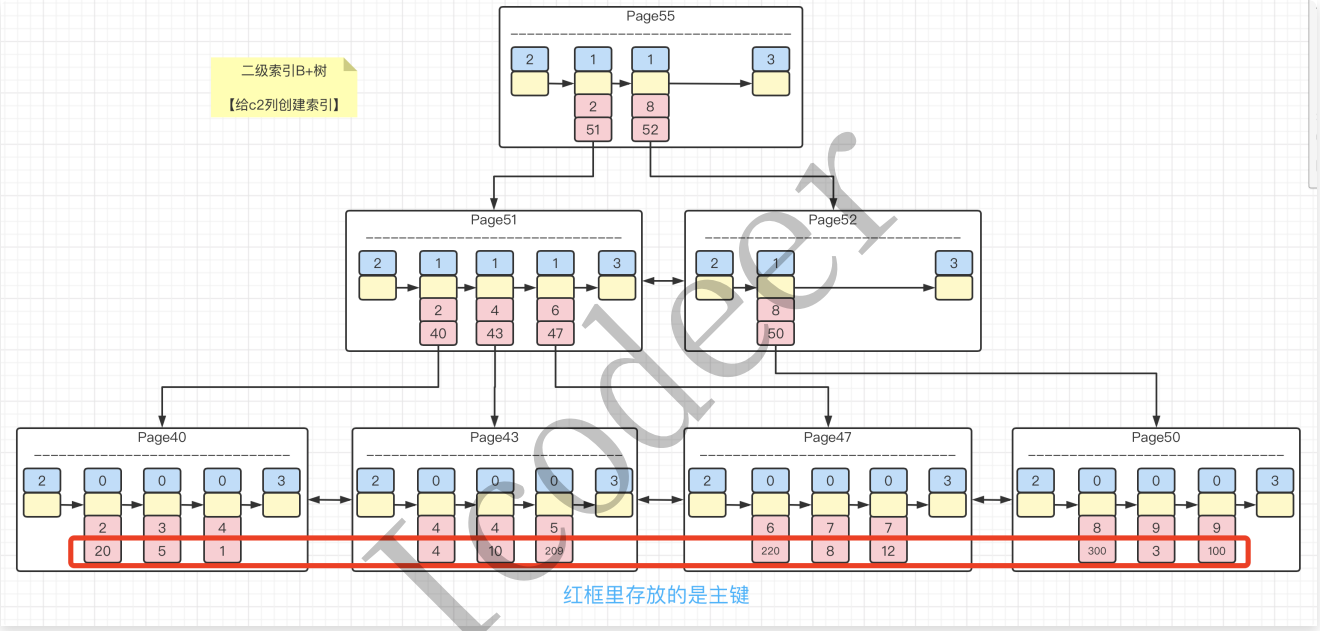
1> 记录 & 页 都是按照**主键值**的大小进行**排序**的。

- **记录**——按照主键的大小顺序排成一个**单向链表**；页内的记录被划分成若干个组，每个组中主键值最大的记录在页内的偏移量会被当作槽依次存放在 页目录 中。
- **页**——按照主键的大小顺序排成一个**双向链表**；存放目录项记录的也分为不同的层级，同层页中的 目录项 记录的主键大小顺序排成一个双向链表。

2> B+树的 叶子节点 存储的是**完整**的**用户记录**。

2.2.4> 二级索引

- 当我们要提高搜索**非主键列**的查询速度时，就涉及到给这个列创建 二级索引 了。如下所示，给 c2 创建索引：



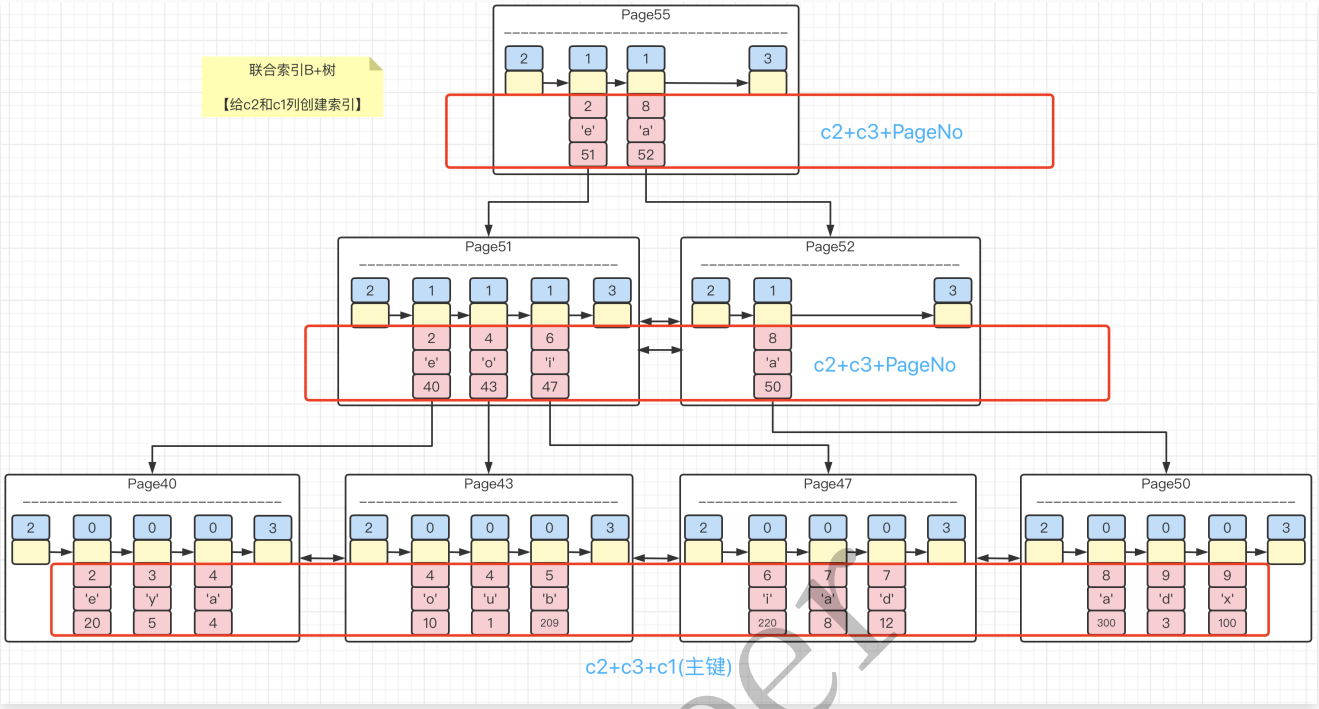
- **叶子节点：** 包含了 c2 列 + c1 列（主键）。
- **目录项节点：** 包含了 c2 列 + 页号 。
- 这个索引，是**将c2列进行了排序**。区别是，叶子节点存储的**不是完整的用户记录**，而只是 c2 列 + 主键列 着两个列的值。
- 目录项记录中不再是 主键 + 页号 的搭配，而变成了 c2列 + 页号 的搭配。
- 由于二级索引的叶子节点并没有完整的用户记录，所以还需要通过**携带主键信息**到聚簇索引中重新定位完整的用户记录的过程也成为**回表**。
- *为什么采用回表去取完整的用户记录，而不是在二级索引里也存放完整的用户记录呢?*

  答：如果把完整的用户记录放到叶子节点就太占空间了，每当给非主键列创建索引的时候，都需要复制一份完整的用户记录。太浪费空间了。

2.2.5> 联合索引

Looper

- 我们也可以同时**为多个列建立索引**
- 比如创建 `c2` 和 `c3` 的联合索引，会先把记录和页按照c2列进行排序，如果当c2列中的记录相同的情况下，在采用c3列进行排序。如下图所示：



2.3> InnoDB中B+树索引的注意事项

2.3.1> 根页面万年不动窝

- 真实的B+树形成过程是这样的

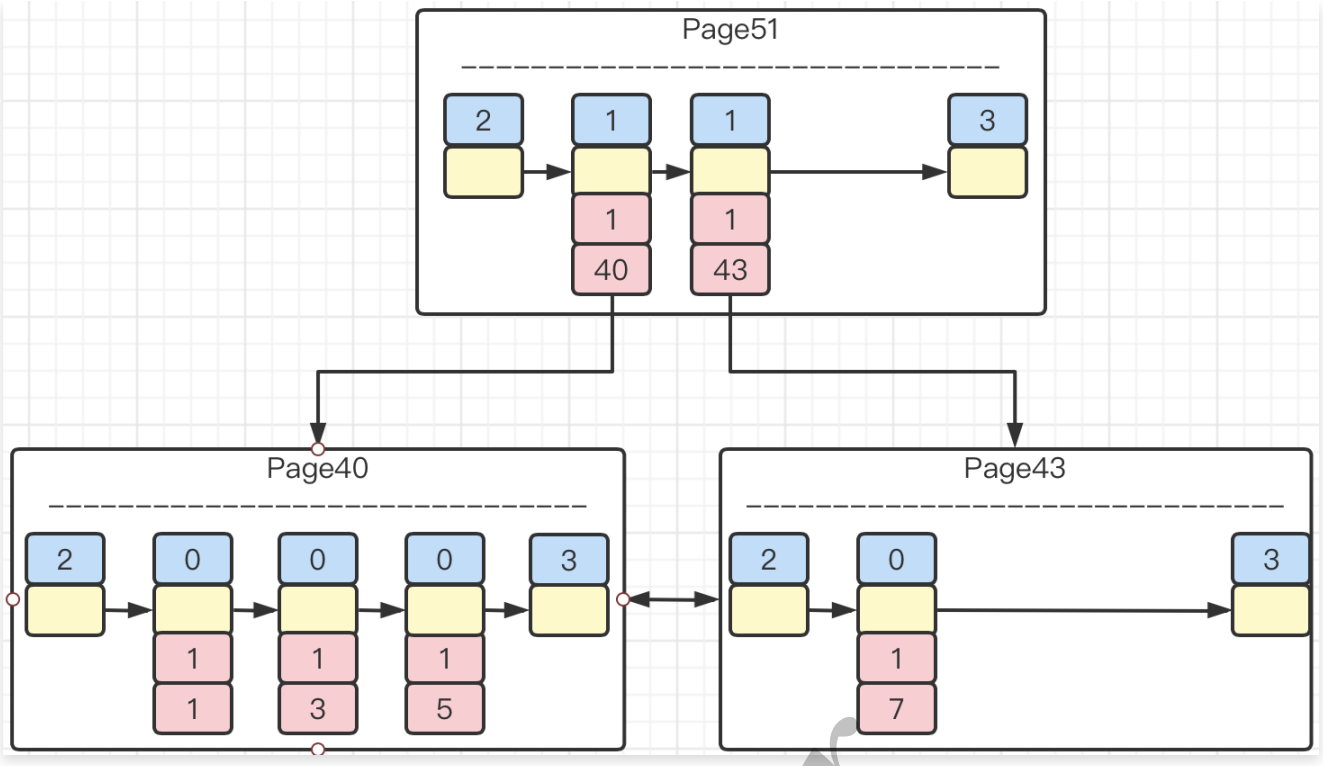
> 首先：当我们创建一张表的时候，InnoDB会自动为这张表创建一个B+树索引的根节点页面，此时没有数据，那么页里没有记录也没有目录项。
> 然后：随着记录的插入，先把用户记录都存储到这个根节点中。
> 最后：当根节点页面里的空间用完了，会先把根节点中的数据，都复制到一个新分配的页（例如：页a）中，然后对这个页a进行分裂操作，得到另一个新分配的页（例如：页b），此时，根节点便升级为了存储目录项记录的页，可以把页a和页b的记录项记录插入到根节点中。

- 所以，综上所述：一棵B+树索引的根节点自创建之日起便不会再移动了（即：页号不再改变）。

2.3.2> 内节点中目录项记录的唯一性

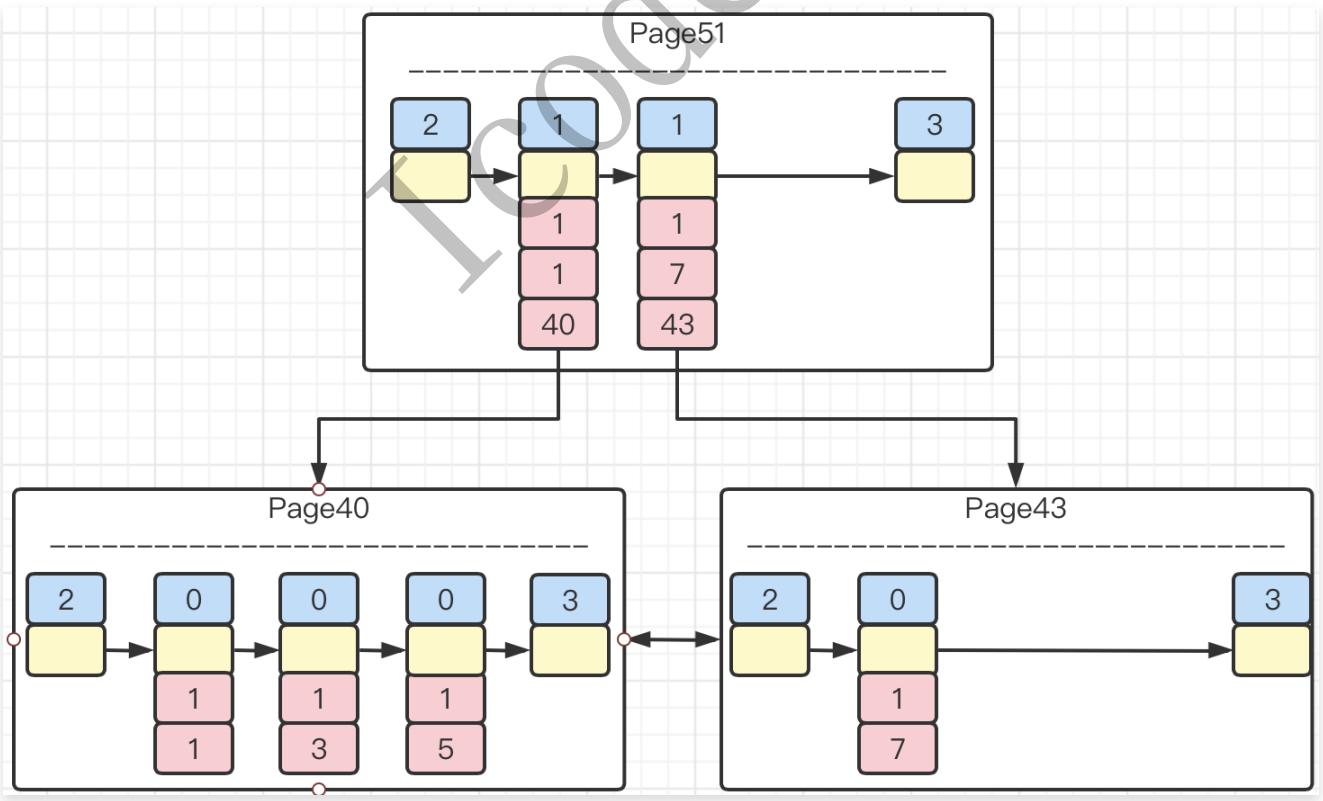
- 假设有如下二级索引（c2）：

Icoder



当我们想再插入c1=9，c2=1时，我们无法确定是插入到Page40还是Page43。

- 为了让新插入的记录能找到自己在哪个页中，就需要保证B+树同一层内节点的目录项记录**除页号字段外是唯一的**。所以二级索引的内节点的**目录项记录**的内容实际是有3部分构成的：



Toocer

> - 1> **索引列的值**。即：c2
> - 2> **主键值**。即：c1

> - 3> 页号。即：pageNo

- 这样，如果c2列的值相同，那么可以接着比较主键值。所以，归其根源，我们可以认为，为**c2列建立的二级索引其实相当于为（c2，c1）列建立了一个联合索引**。

吾尝终日而思矣，不如须臾之所学也；
吾尝跂而望矣，不如登高之博见也。
登高而招，臂非加长也，而见者远；
顺风而呼，声非加疾也，而闻者彰。
假舆马者，非利足也，而致千里；
假舟楫者，非能水也，而绝江河。
君子生非异也，善假于物也。
------ 摘自《劝学》

愿本文可以成为大家的“山”、“风”、“马”、“舟”，助大家在技术之路上乘风破浪，大展宏图~~

同时，也欢迎大家关注我的公众号“爪哇缪斯” ~\(^o^)/~「干货分享，每天更新」

Iceer