

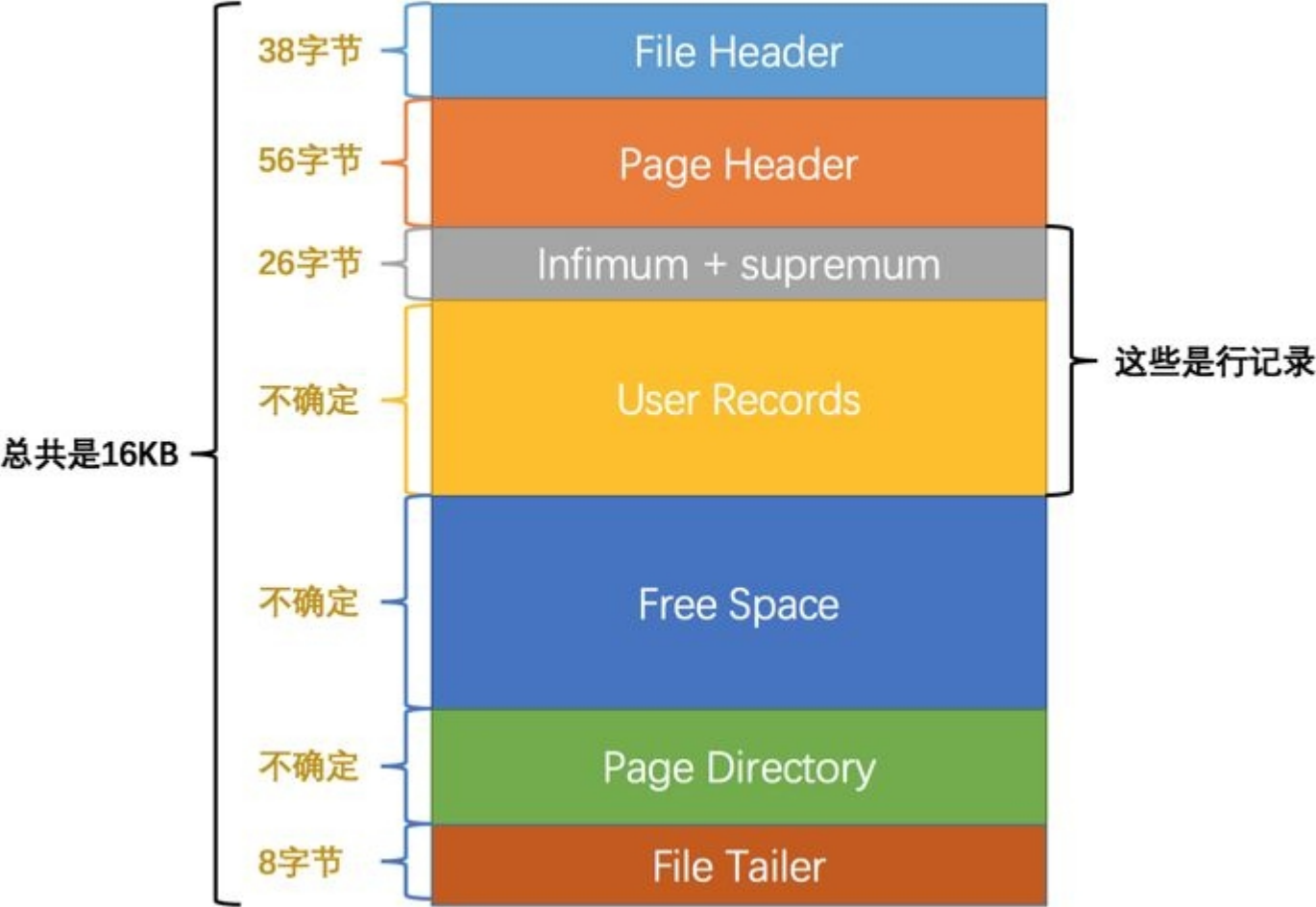
7 第七讲 索引

教学内容

- 7.1 MySQL的数据存储
- 7.2 索引的概念和作用
- 7.3 使用SQL语句创建和管理索引
- 7.4 使用Navicat创建和管理索引

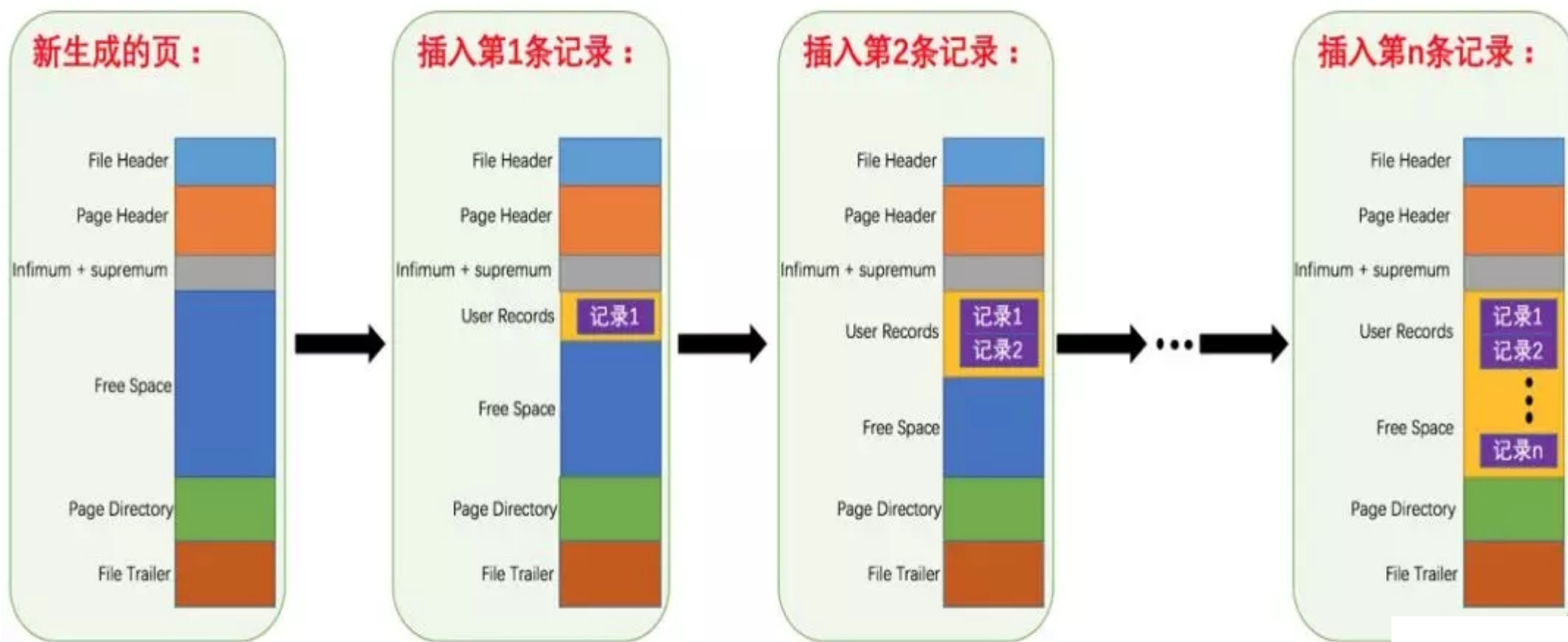
7.1 MySQL的数据存储

InnoDB页结构示意图

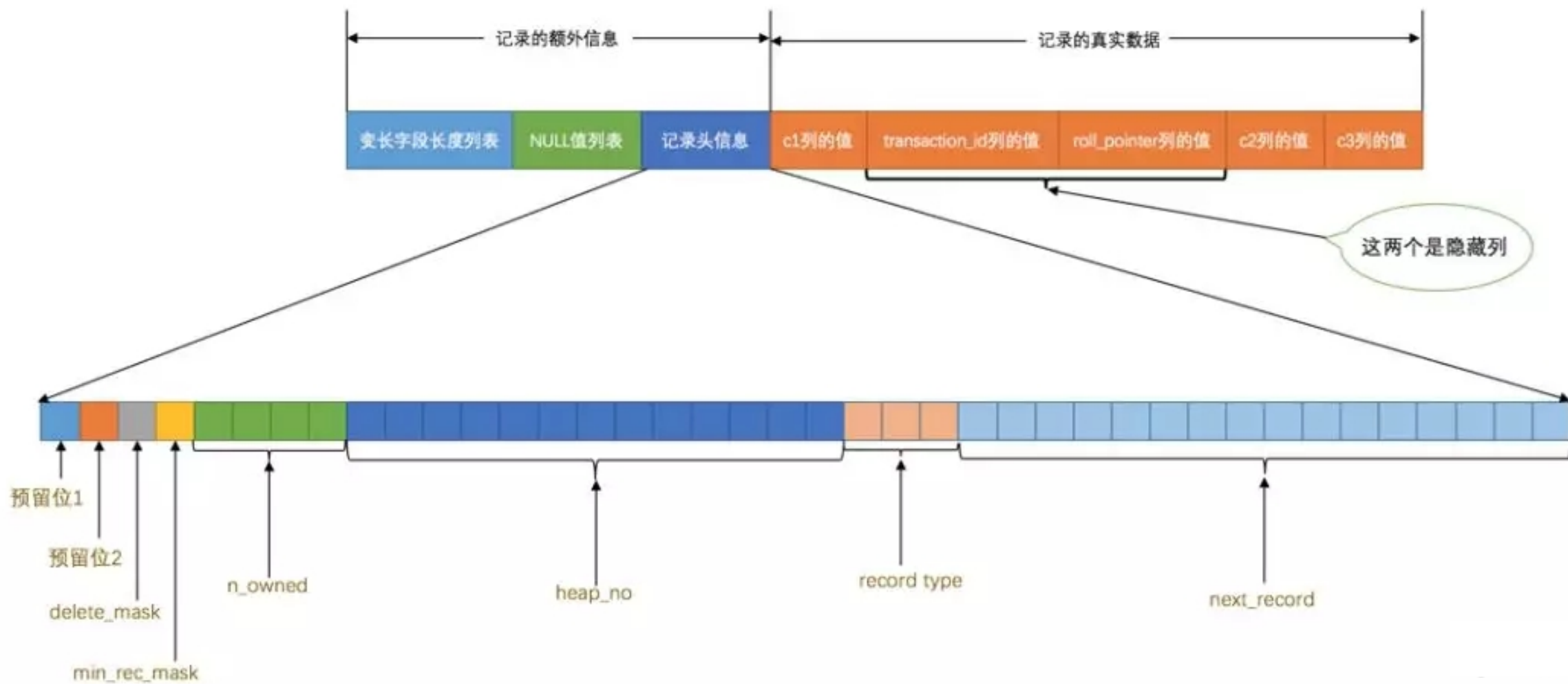


中文名	占用空间大小	简单描述
文件头	38字节	一些描述页的信息
页头	56字节	页的状态信息
最小记录和最大记录	26字节	两个虚拟的行记录
用户记录	不确定	实际存储的行记录内容
空闲空间	不确定	页中尚未使用的空间
页目录	不确定	页中的记录相对位置
文件结尾	8字节	校验页是否完整


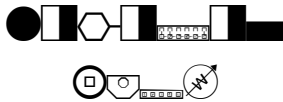
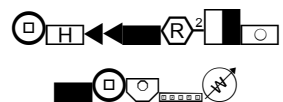


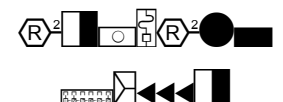
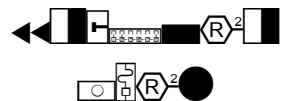
记录在页中的存储



记录头信息的秘密



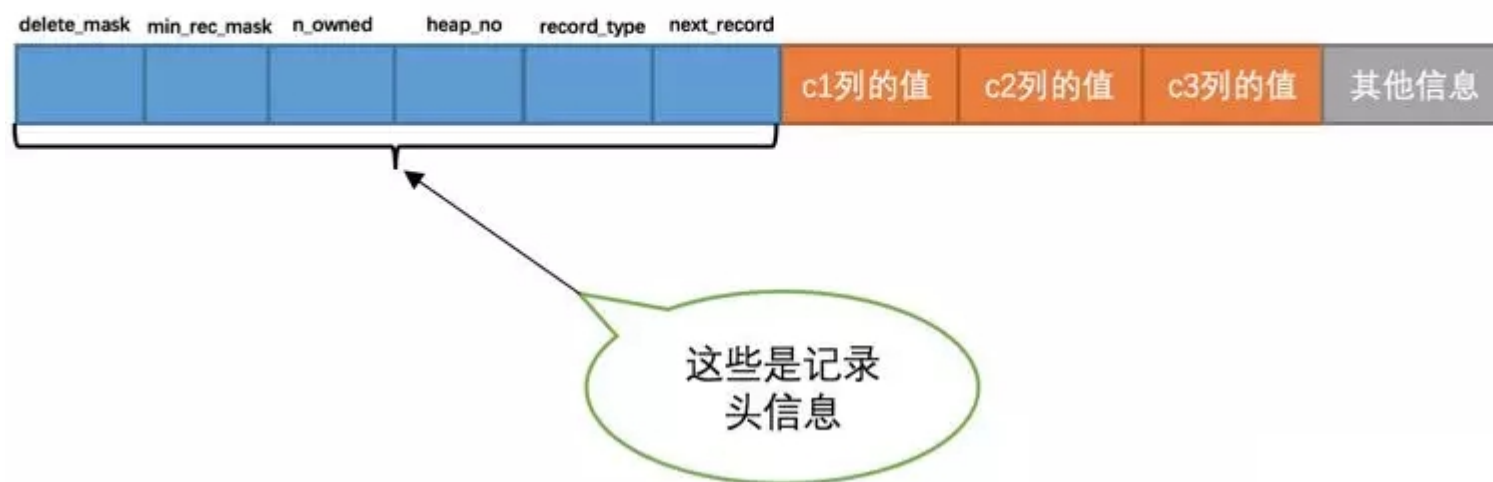
记录头信息

名称	大小（单位：  ）	描述
预留位 ∇	∇	没有使用
预留位 \bigcirc	∇	没有使用
	∇	标记该记录是否被删除
	∇	标记该记录是否为 \blacktriangleright 树的非叶子节点中的最小记录
	\bullet	表示当前槽管理的记录数
	$\nabla \nabla$	表示当前记录在记录堆的位置信息
	∇	表示当前记录的类型 \blacktriangle 表示普通记录， ∇ 表示 \blacktriangleright 树非叶节点记录， \bigcirc 表示最小记录， ∇ 表示最大记录
	$\nabla \bullet$	表示下一条记录的相对位置

案例

- **CREATE TABLE page_demo(**
- **c1 INT,**
- **c2 INT,**
- **c3 VARCHAR(10000),**
- **PRIMARY KEY (c1)**
- **) CHARSET=ascii**
ROW_FORMAT=Compact;

page_demo表的行格式简化图



添加记录

- **INSERT INTO page_demo VALUES(1, 100, 'aaaa'), (2, 200, 'bbbb'), (3, 300, 'cccc'), (4, 400, 'dddd');**

第1条记录：

delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record				
0	0	0	2	0	32	1	100	'aaaa'	其他信息

第2条记录：

delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record				
0	0	0	3	0	32	2	200	'bbbb'	其他信息

第3条记录：

delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record				
0	0	0	4	0	32	3	300	'cccc'	其他信息

第4条记录：

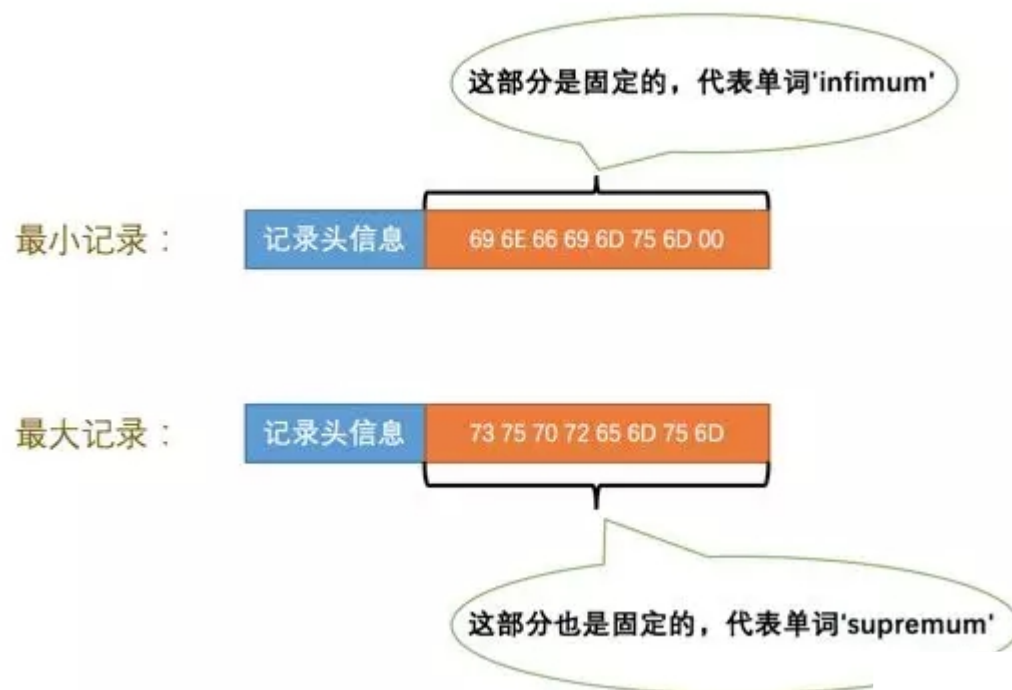
delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record				
0	0	0	5	0	-111	4	400	'dddd'	其他信息

这是 User Records部分

- **delete_mask**:标记着当前记录是否被删除， 占用1个二进制位， 值为0 未删除， 1删除

Infimum + Supremum的部分

- 不存放在页的User Records部分



记录属性

- heap_no: 最小记录和最大记录分别是0和1, 表明位置最靠前
- record_type: 4种记录类型, 0为普通记录, 1为B+树非叶节点记录, 2为最小记录, 3为最大记录
- next_record: 当前记录到下一条记录的地址值

delete_mask min_rec_mask n_owned heap_no record_type next_record							delete_mask min_rec_mask n_owned heap_no record_type next_record						
最小记录:							最大记录:						
0	0	1	0	2	28	'infimum'	0	0	5	1	3	0	'supremum'

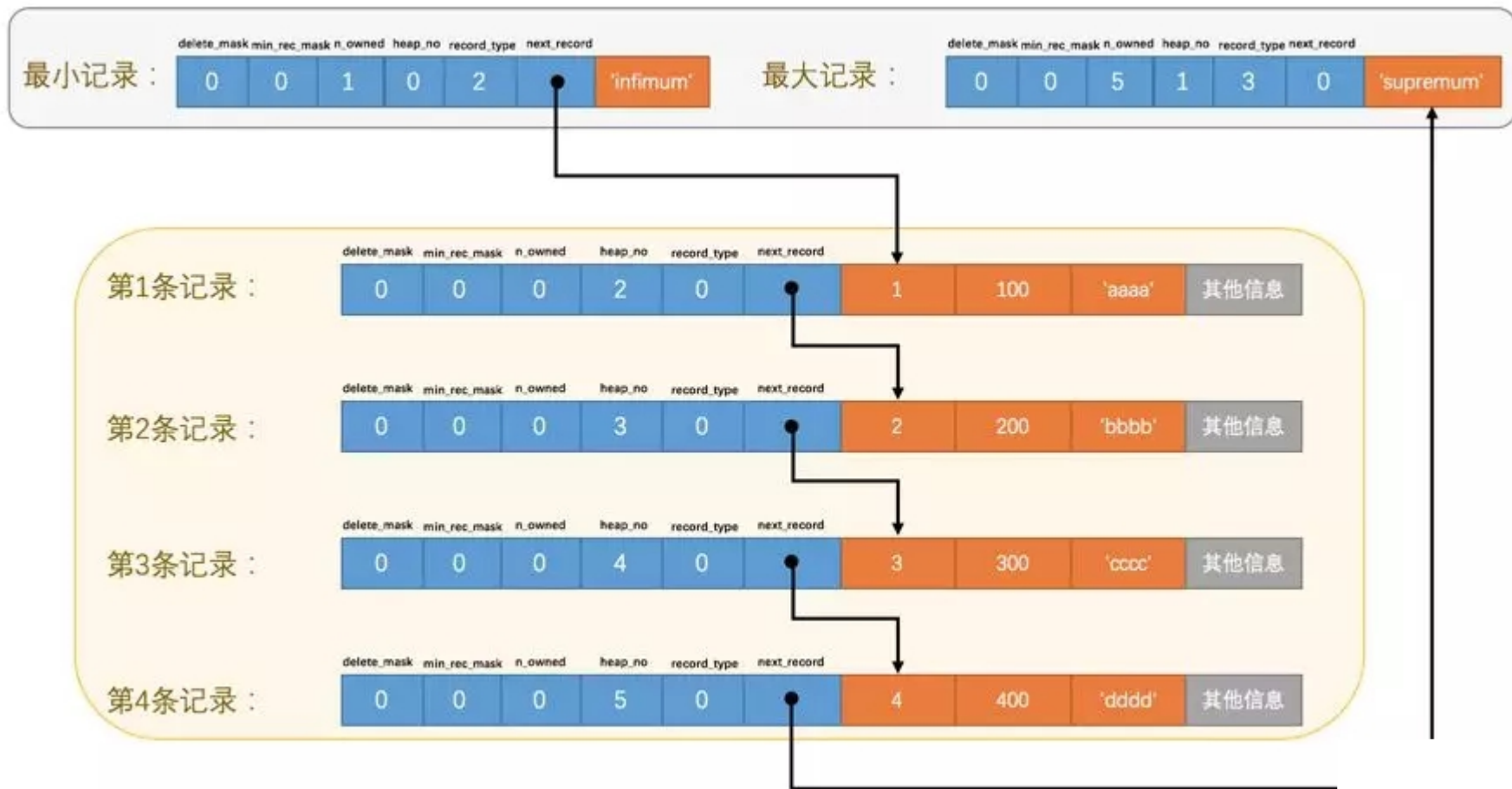
第1条记录:	delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record	1	100	'aaaa'	其他信息
	0	0	0	2	0	32				
第2条记录:	delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record	2	200	'bbbb'	其他信息
	0	0	0	3	0	32				
第3条记录:	delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record	3	300	'cccc'	其他信息
	0	0	0	4	0	32				
第4条记录:	delete_mask	min_rec_mask	n_owned	heap_no	record_type	next_record	4	400	'dddd'	其他信息
	0	0	0	5	0	-111				

这是Infimum +Supremum部分

这是 User Records部分

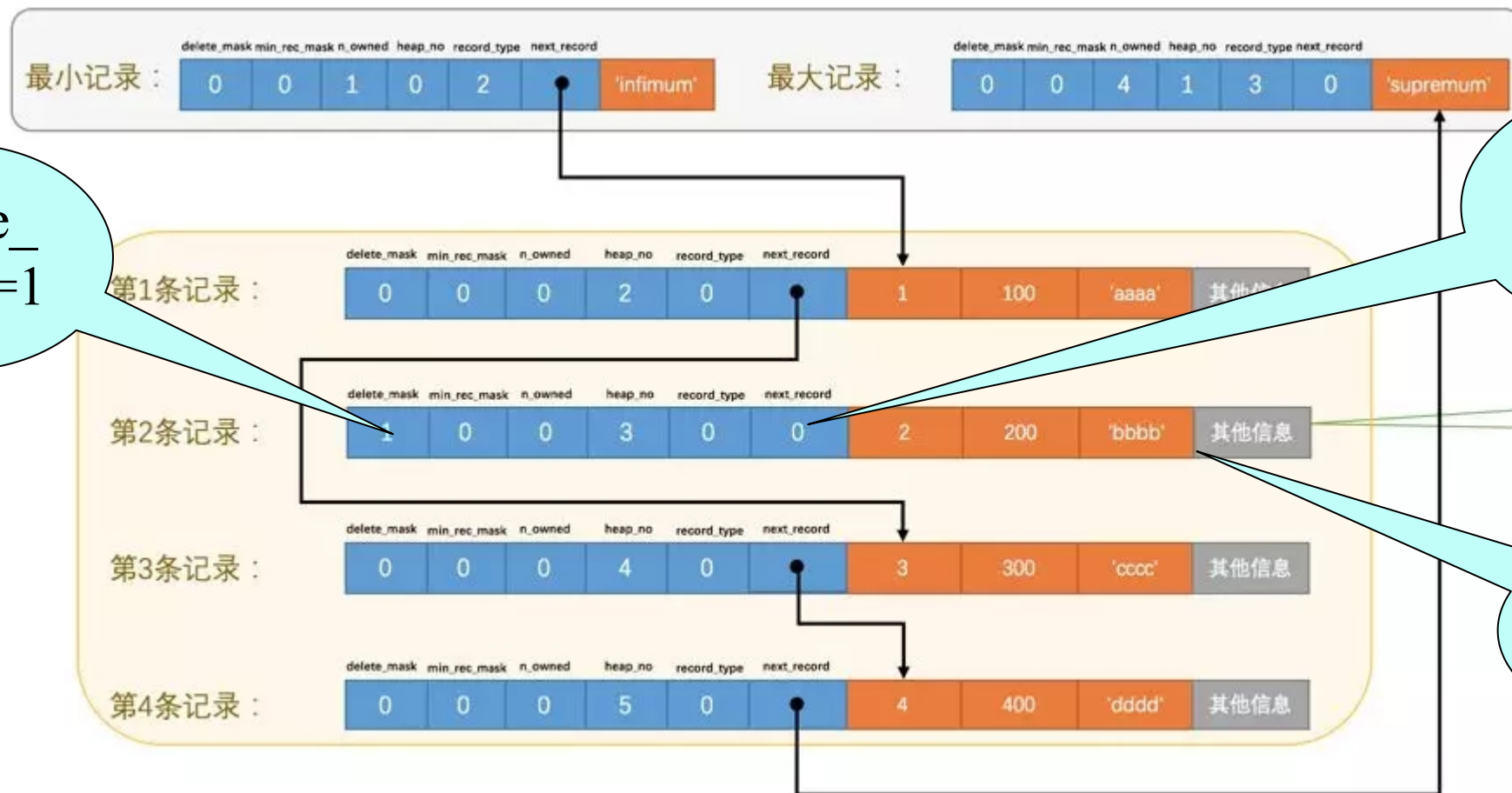
一个单链表

- 记录按照从小到大的顺序形成了一个单链表
- **next_record: 0**，即最大记录是没有下一条记录



删除记录

■ DELETE FROM page_demo WHERE c1 = 2;



delete_mask=1

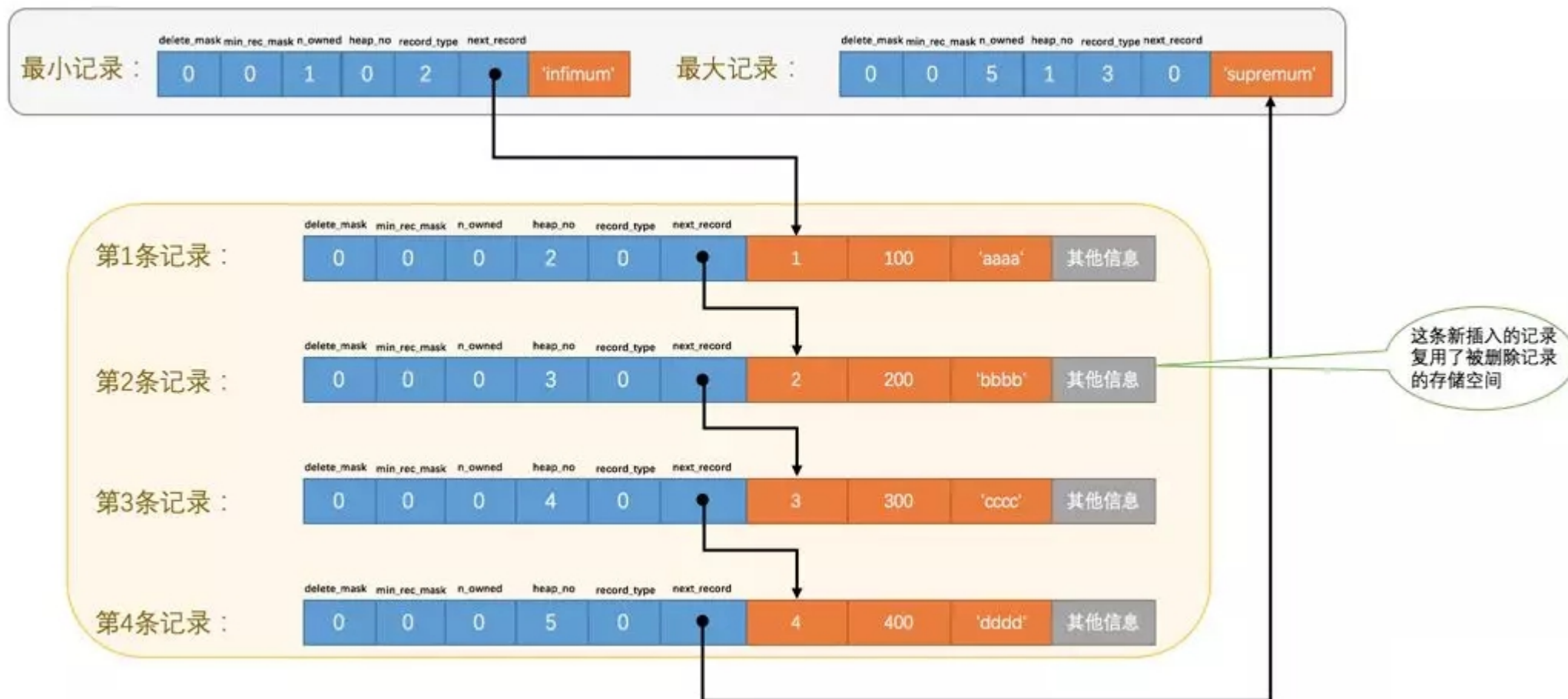
Next_record=0 无下一条记录

这条记录被删了

存储空间不回收

添加记录

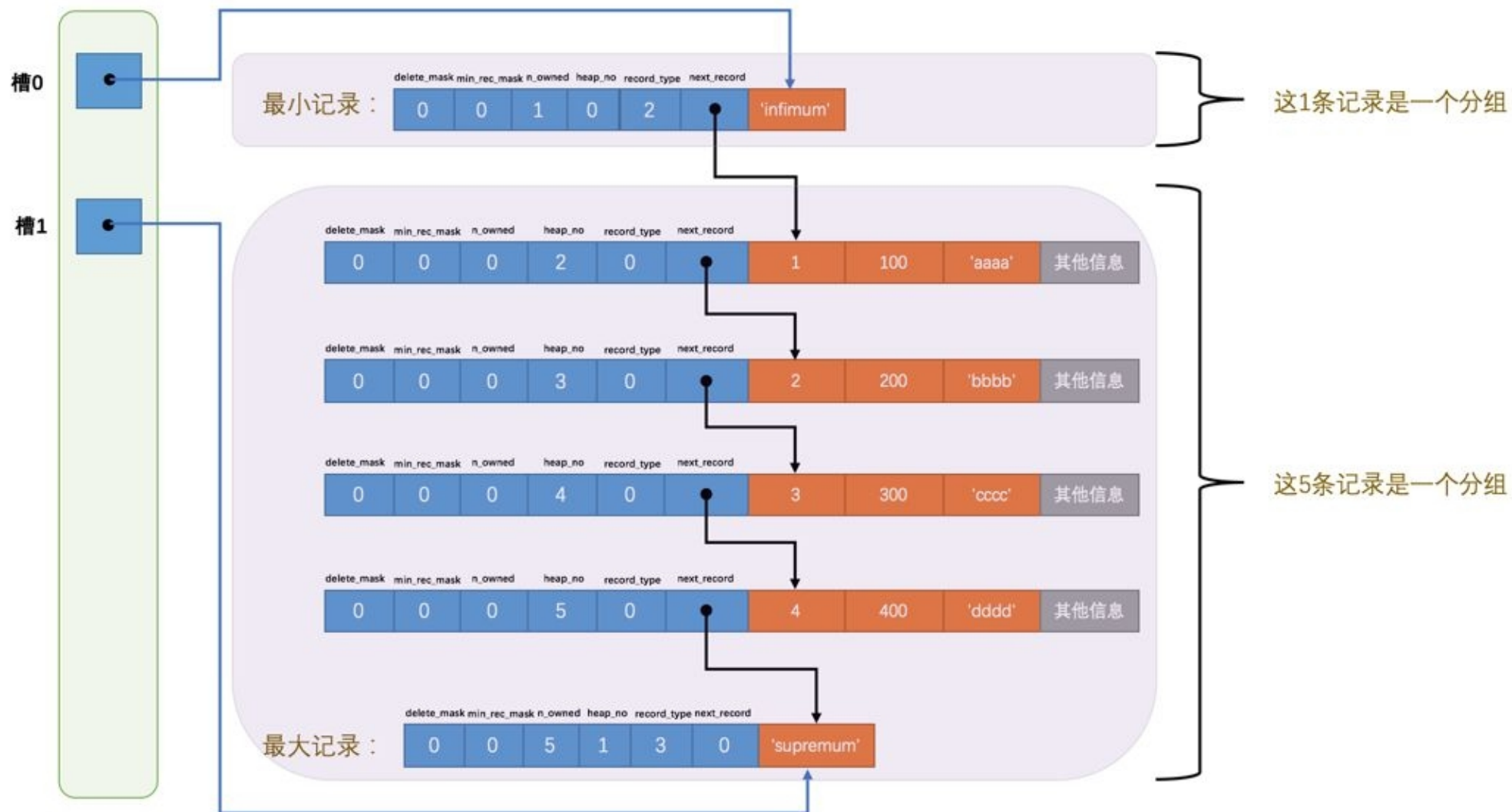
- `INSERT INTO page_demo VALUES(2, 200, 'bbbb');`



页目录

- 将所有正常的记录（包括最大和最小记录，不包括标记为已删除的记录）划分为几个组。
- 每个组的最后一条记录的头信息中的n_owned属性表示该组内共有几条记录。
- 将每个组的最后一条记录的地址偏移量按顺序存储起来，每个地址偏移量也被称为一个槽（英文名：**Slot**）。这些地址偏移量都会被存储到靠近页的尾部的地方，页中存储地址偏移量的部分也被称为**Page Directory**。

页目录——槽



InnoDB分组规则

■ 分组规则

- 最小记录所在的分组只能有 1 条记录
- 最大记录所在的分组拥有的记录条数只能在 1~8 条之间
- 剩下的分组中记录的条数范围只能在是 4~8 条之间

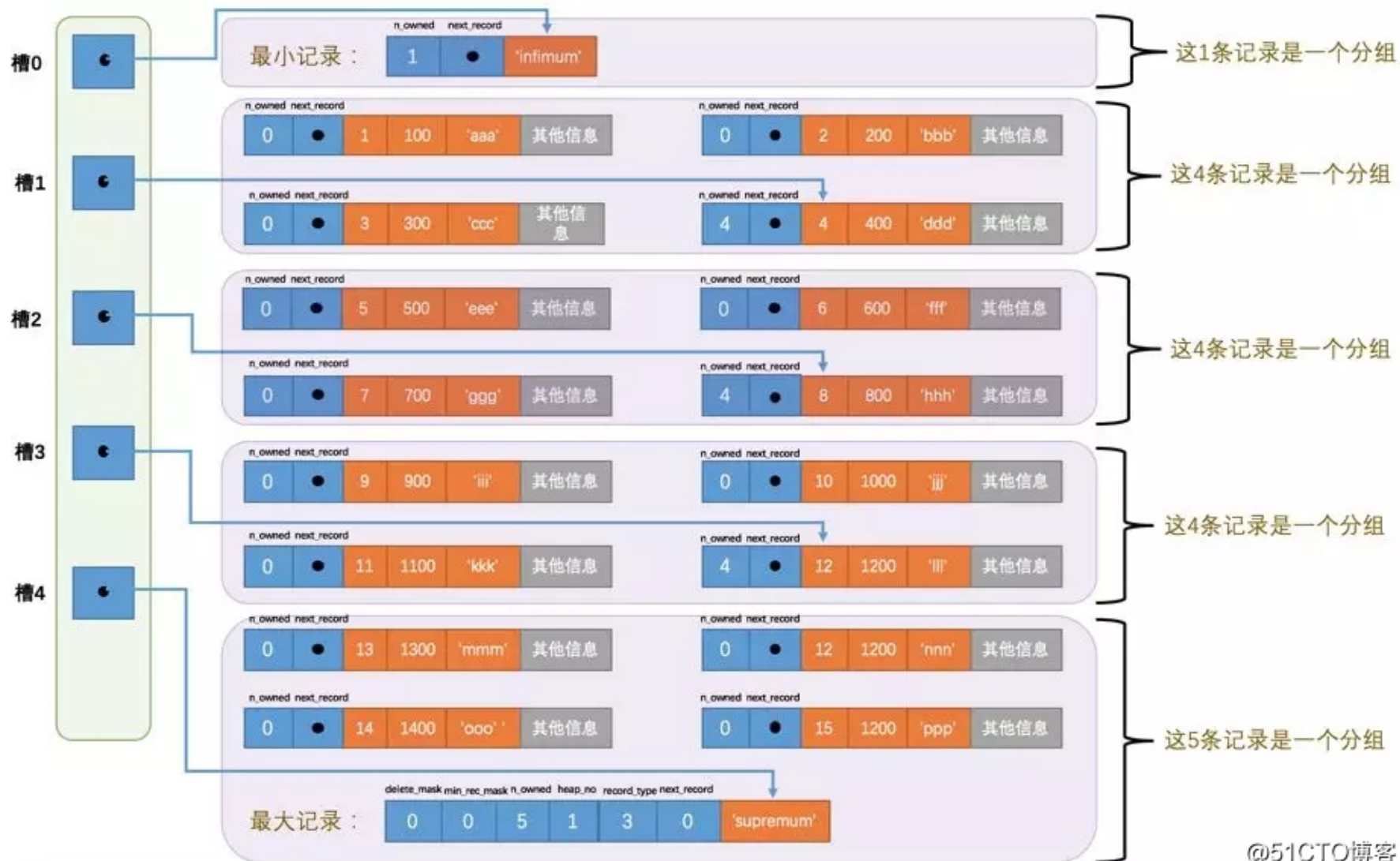
■ 分组步骤

- 初始情况下一个数据页里只有最小记录和最大记录两条记录，分属于两个分组
- 之后每插入一跳记录都把这条记录放到最大记录所在的组，直到最大记录所在组中的记录数等于8个。
- 在最大记录所在组中的记录数等于8个的时候再插入一条记录时，将最大记录所在组平均分裂成2个组，最大记录所在的组剩下4条记录，然后就可以把即将插入的那条记录放到该组中。

添加16条记录

- **INSERT INTO page_demo**
- **VALUES**
- **(5, 500, 'eeee'), (6, 600, 'ffff'), (7, 700, 'gggg'),**
- **(8, 800, 'hhhh'), (9, 900, 'iiii'), (10, 1000, 'jjjj'),**
- **(11, 1100, 'kkkk'), (12, 1200, 'llll'), (13, 1300, 'mmmm'),**
- **(14, 1400, 'nnnn'), (15, 1500, 'oooo'), (16, 1600, 'pppp');**

16条记录的槽



@51CTO博客

二分查找

- 各个槽记录的主键值从小到大排序，可使用二分法来快速查找。
- 4个槽的编号：0、1、2、3、4，最低的槽low=0，最高的槽high=4
- 找主键值为5的记录：
 - 计算中间槽的位置： $(0+4)/2=2$ ，查看槽2对应记录的主键值为8，因 $8 > 5$ ，设置high=2，low保持不变。
 - 重新计算中间槽的位置： $(0+2)/2=1$ ，查看槽1对应的主键值为4。设置low=1，high保持不变。
 - 因 $high - low = 1$ ，所以确定主键值为5的记录在槽1和槽2之间，遍历链表查找即可。
- 在一个数据页中查找指定主键值的记录的过程分为两步：
 - 通过二分法确定该记录所在的槽
 - 通过记录的next_record属性组成的链表遍历查找该槽中的各个记录

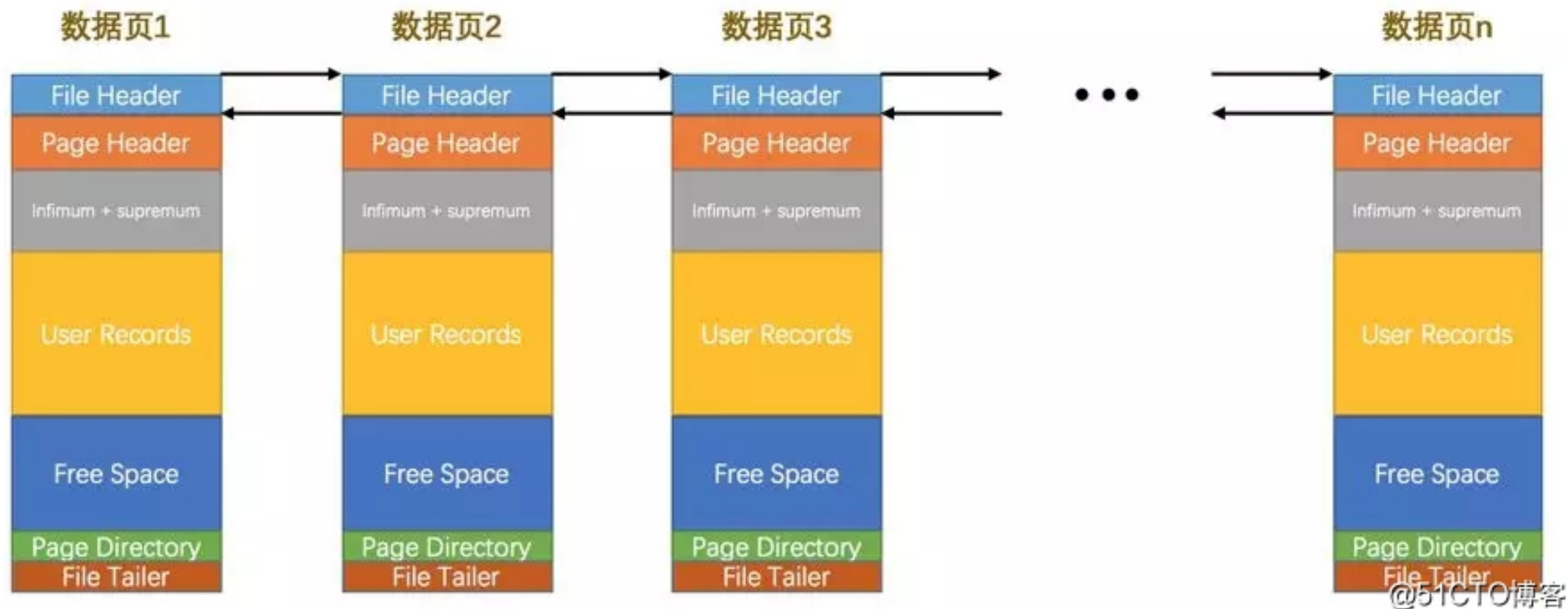
Page Header 56字节

名称	占用空间大小	描述
	1字节	在页目录中的槽数量
	1字节	第一个记录的地址
	1字节	本页中的记录的数量（包括最小和最大记录以及标记为删除的记录）
	1字节	指向可重用空间的地址（就是标记为删除的记录地址）
	1字节	已删除的字节数，行记录结构中 \bullet 为 ∇ 的记录大小总数
	1字节	最后插入记录的位置
	1字节	最后插入的方向
	1字节	一个方向连续插入的记录数量
	1字节	该页中记录的数量（不包括最小和最大记录以及被标记为删除的记录）
	1字节	修改当前页的最大事务 ϕ ，该值仅在二级索引中定义
	1字节	当前页在索引树中的位置，高度
	1字节	索引 ϕ ，表示当前页属于哪个索引
	1字节	非叶节点所在段的地址 ϕ ，仅在B+树的M页定义

File Header 38字节

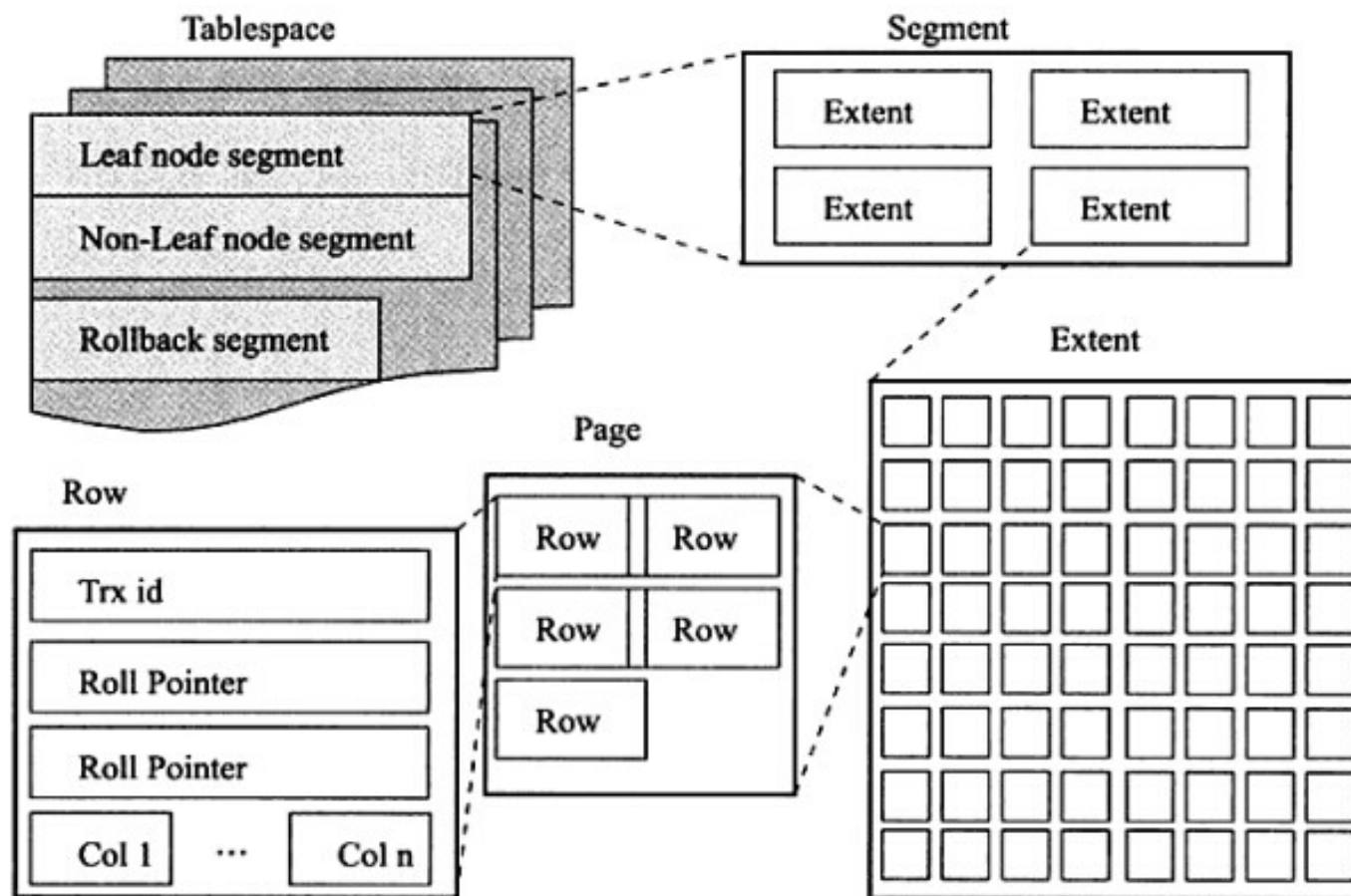
名称	占用空间大小	描述
	2字节	页的校验和 (CRC32值)
	2字节	页号
	2字节	上一个页的页号
	2字节	下一个页的页号
	1字节	最后被修改的日志序列位置 (英文名是: LSN (Log Sequence Number))
	1字节	该页的类型
	1字节	仅在系统表空间的一个页中定义，代表文件至少被更新到了该值，独立表空间中都是0
	2字节	页属于哪个表空间

MySQL数据页



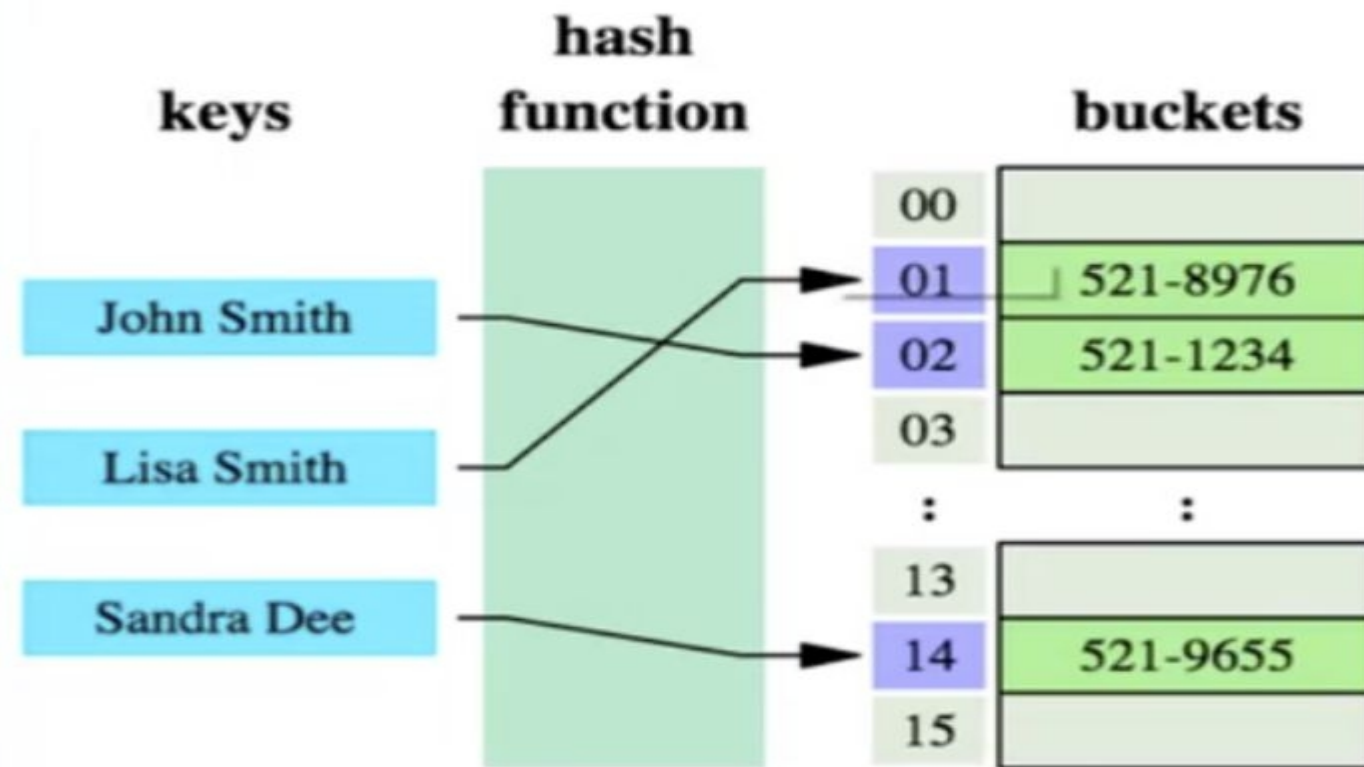
Innodb数据页结构分析

- Innodb 逻辑存储结构图，从上往下依次为：Tablespace、Segment、Extent、Page 以及 Row



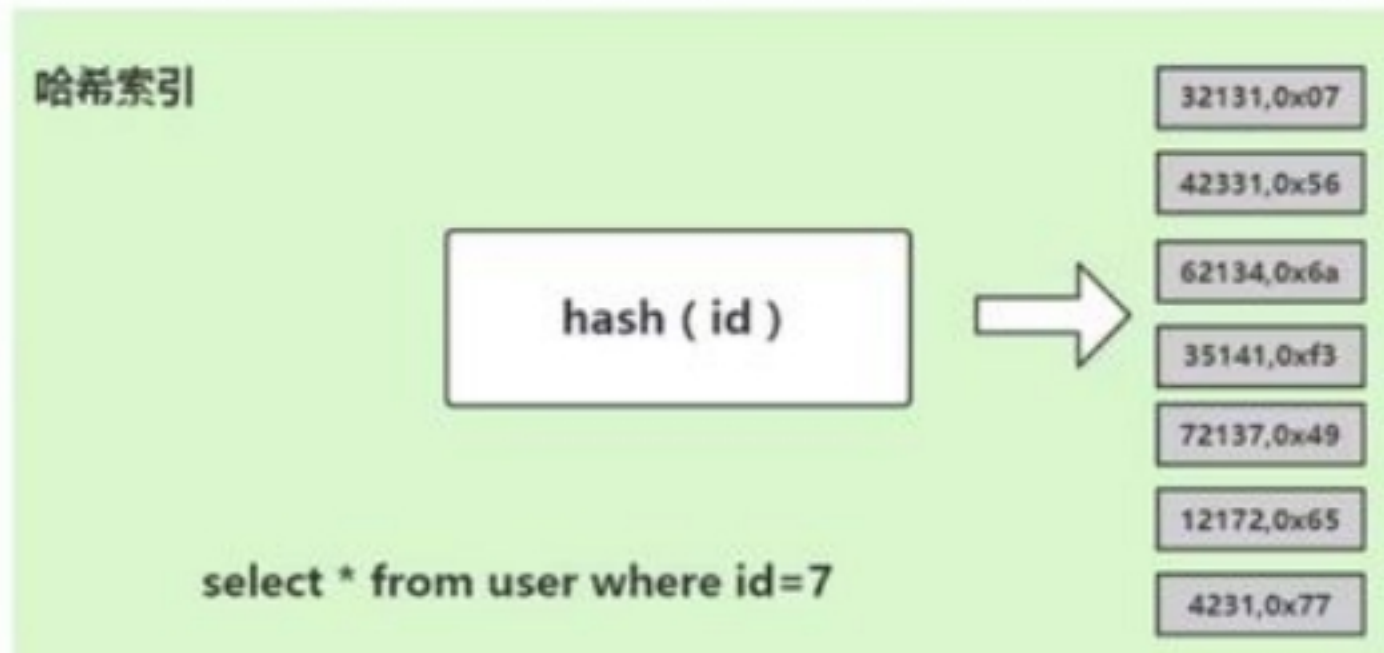
哈希表（Hash）

- 哈希算法：也叫散列算法，就是把任意值(key)通过哈希函数变换为固定长度的 **key** 地址，通过这个地址进行具体数据的数据结构。



哈希算法

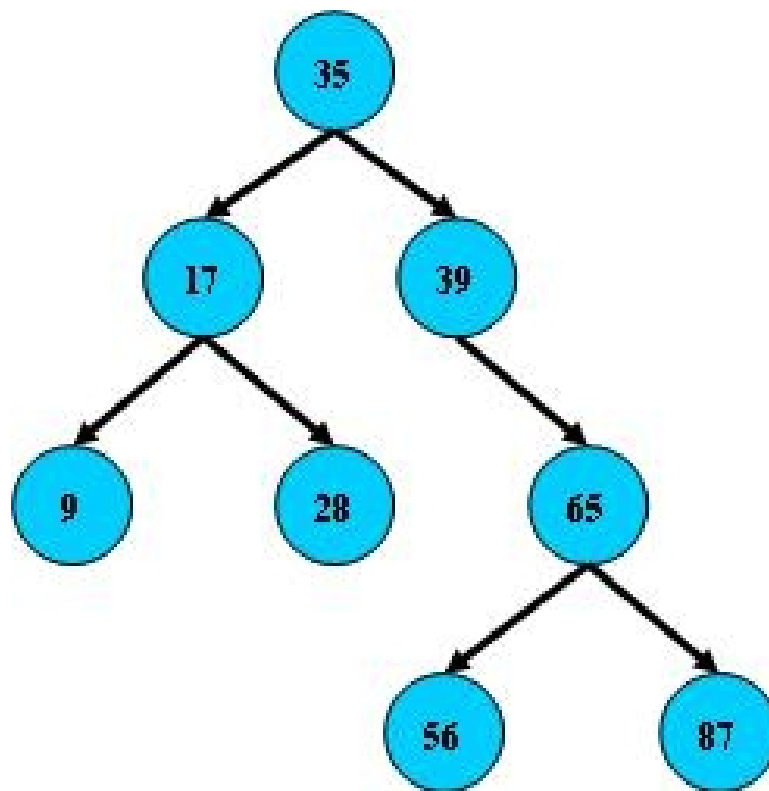
- 哈希算法首先计算存储 $\text{id}=7$ 的数据的物理地址 $\text{addr}=\text{hash}(7)=4231$
- 4231 映射的物理地址是 $0x77$ ， $0x77$ 就是 $\text{id}=7$ 存储的数据的物理地址，通过该地址可找到对应 $\text{user_name}='g'$
- 缺点：范围查找



id(pk)	user_name
1	a
2	b
3	c
4	d
5	e
6	f
7	g

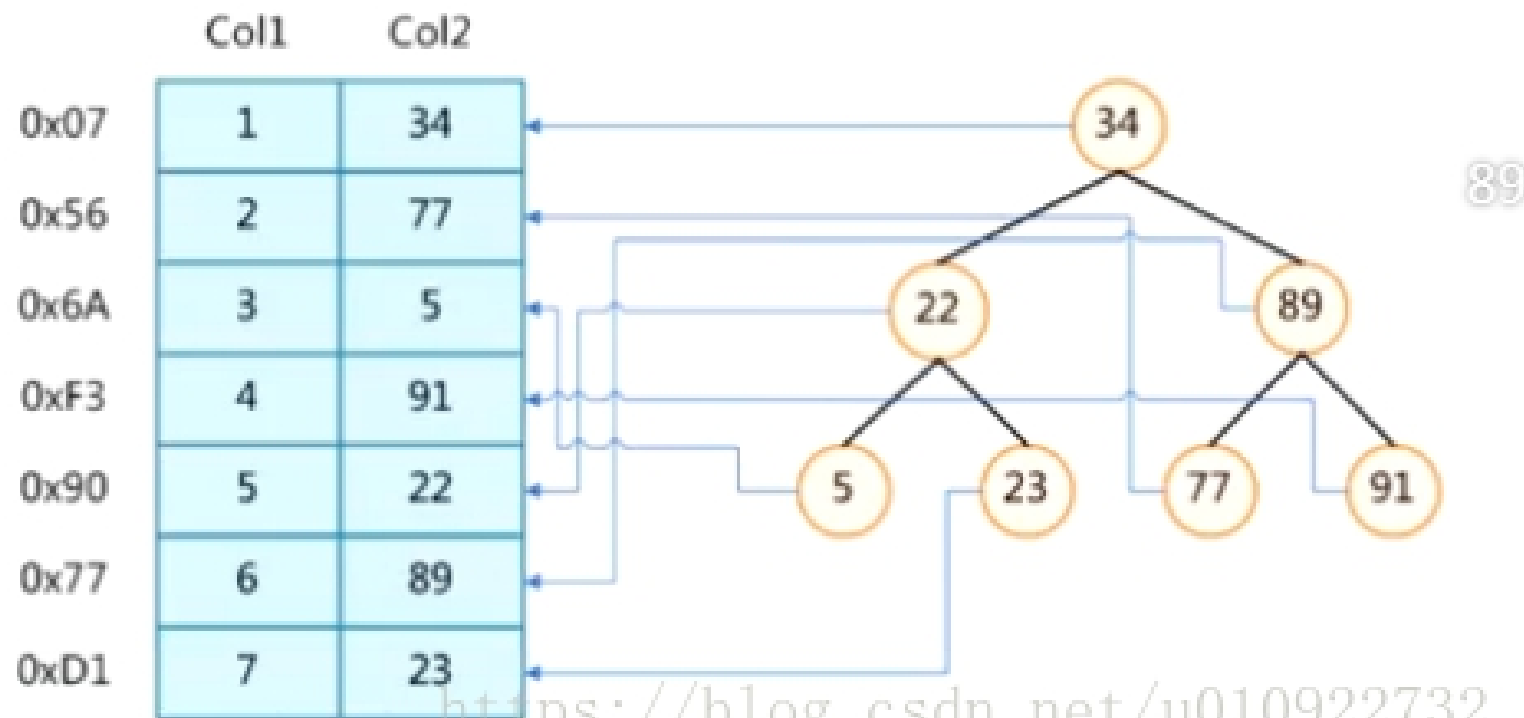
B树（二叉搜索树）

- 所有非叶子结点至多拥有两个儿子（Left和Right）
- 所有结点存储一个关键字；
- 非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；



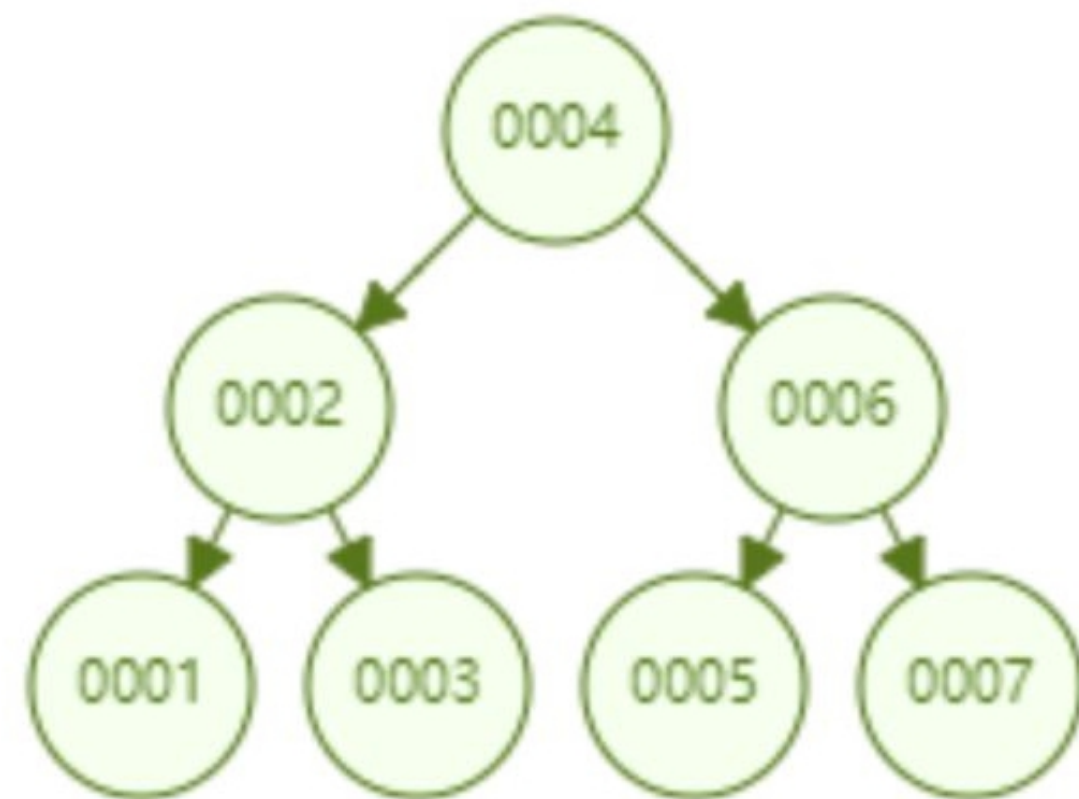
二叉树

- where Col2 = 22
 - 无索引 5次；二叉树索引2次

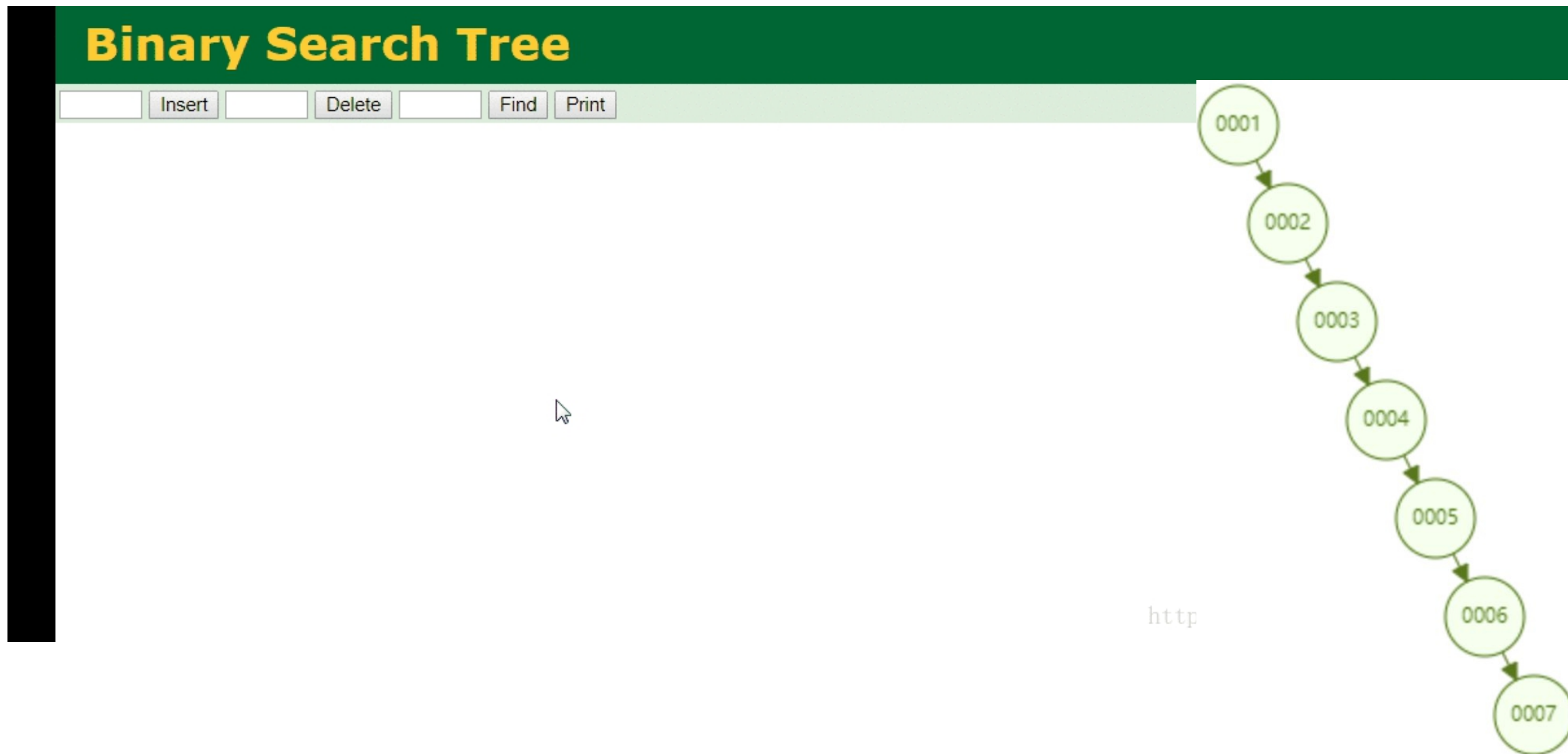


二叉树

Binary Search Tree

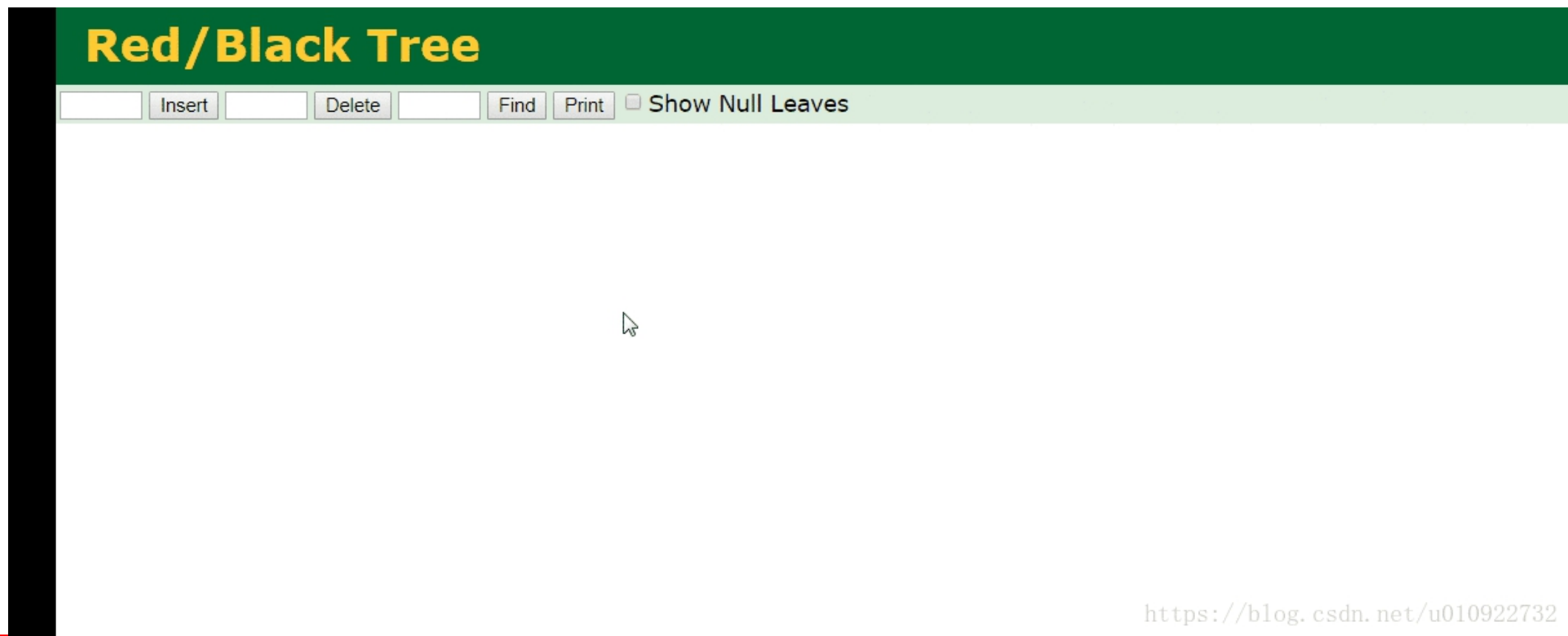
 

二叉树缺点



红黑树

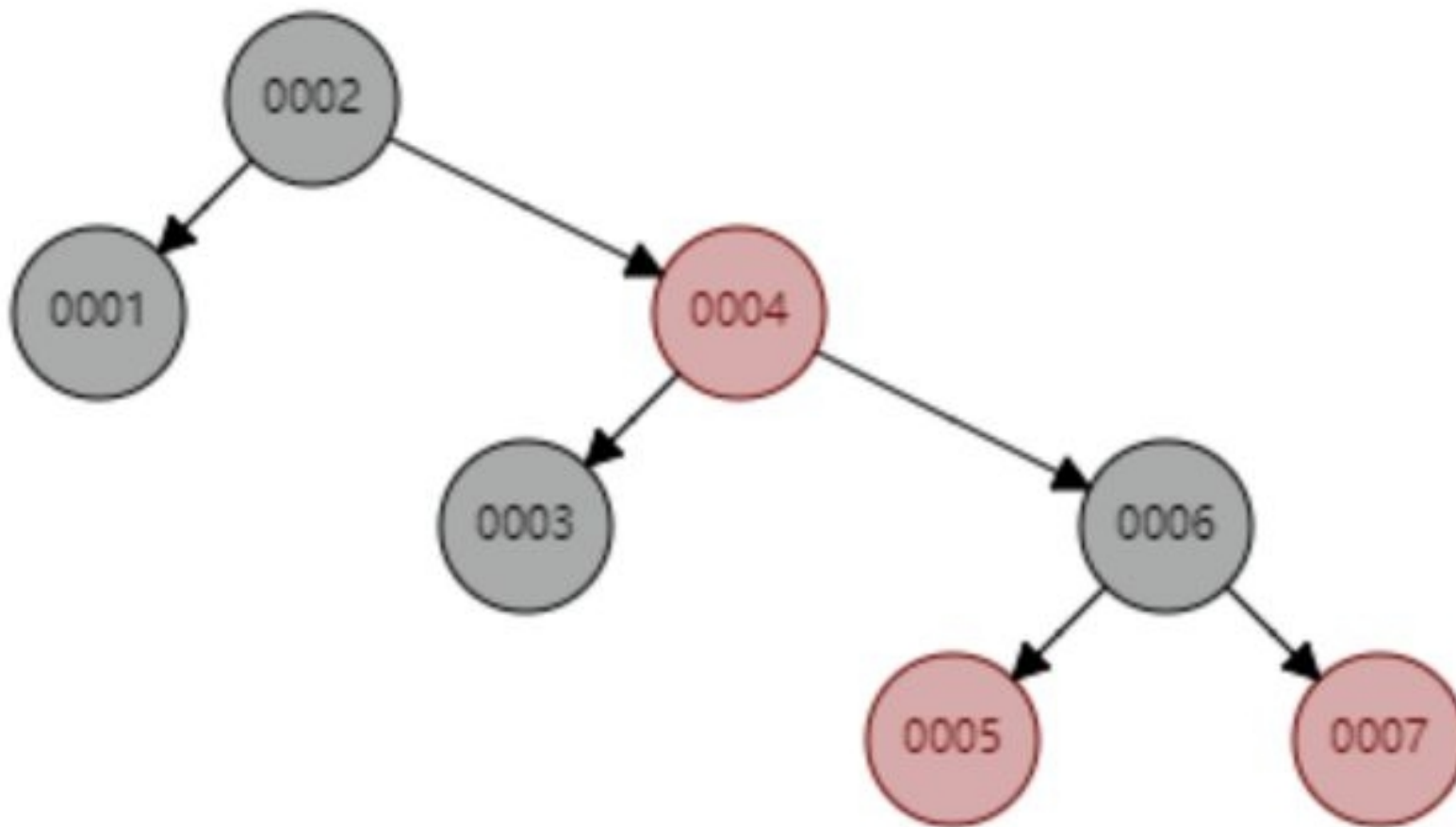
- 红黑树也叫平衡二叉树，不仅继承了二叉树的优点，而且解决了二叉树遇到的自增整形索引的问题
- 红黑树会左旋、右旋对结构进行调整，始终保证左子节点数 < 父节点数 < 右子节点数的规则。



<https://blog.csdn.net/u010922732>

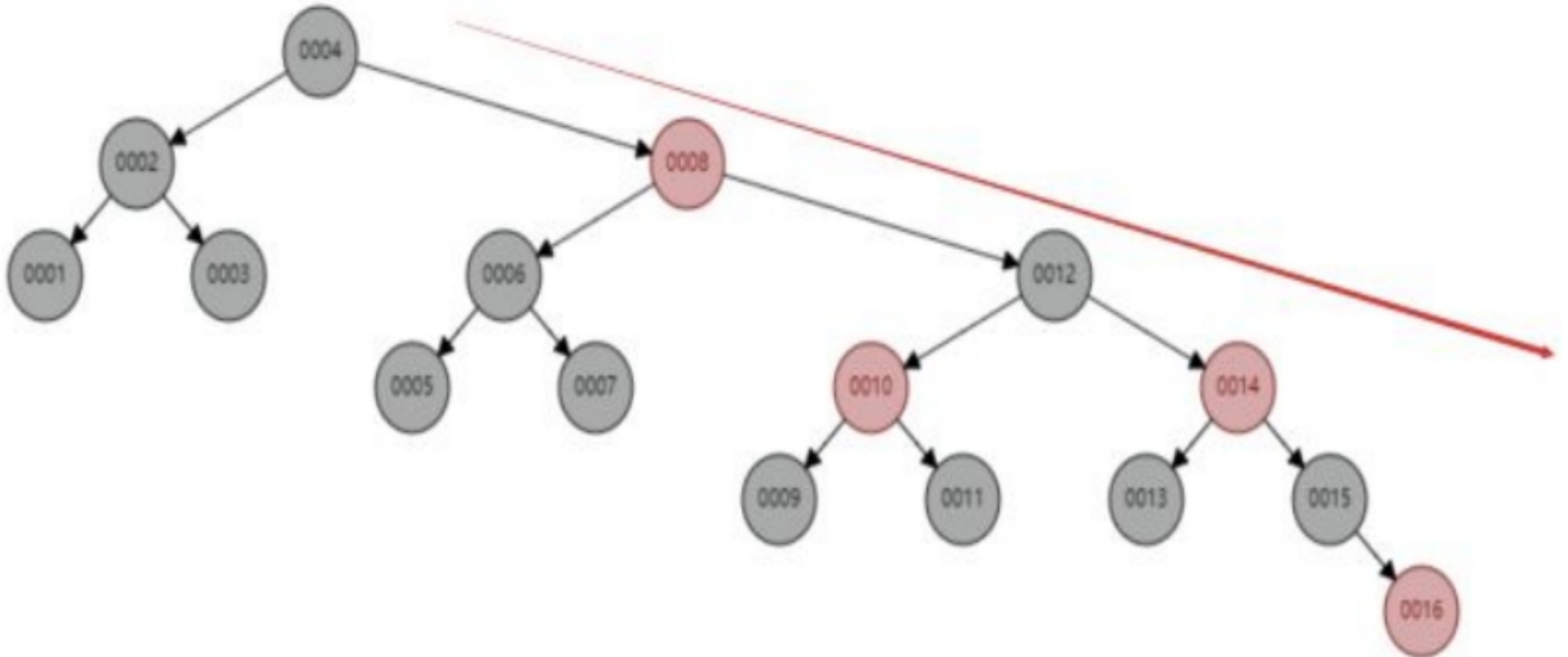
红黑树

- 红黑树顺序插入 1~7 个节点，查找id=7 时需要计算的节点数为 4



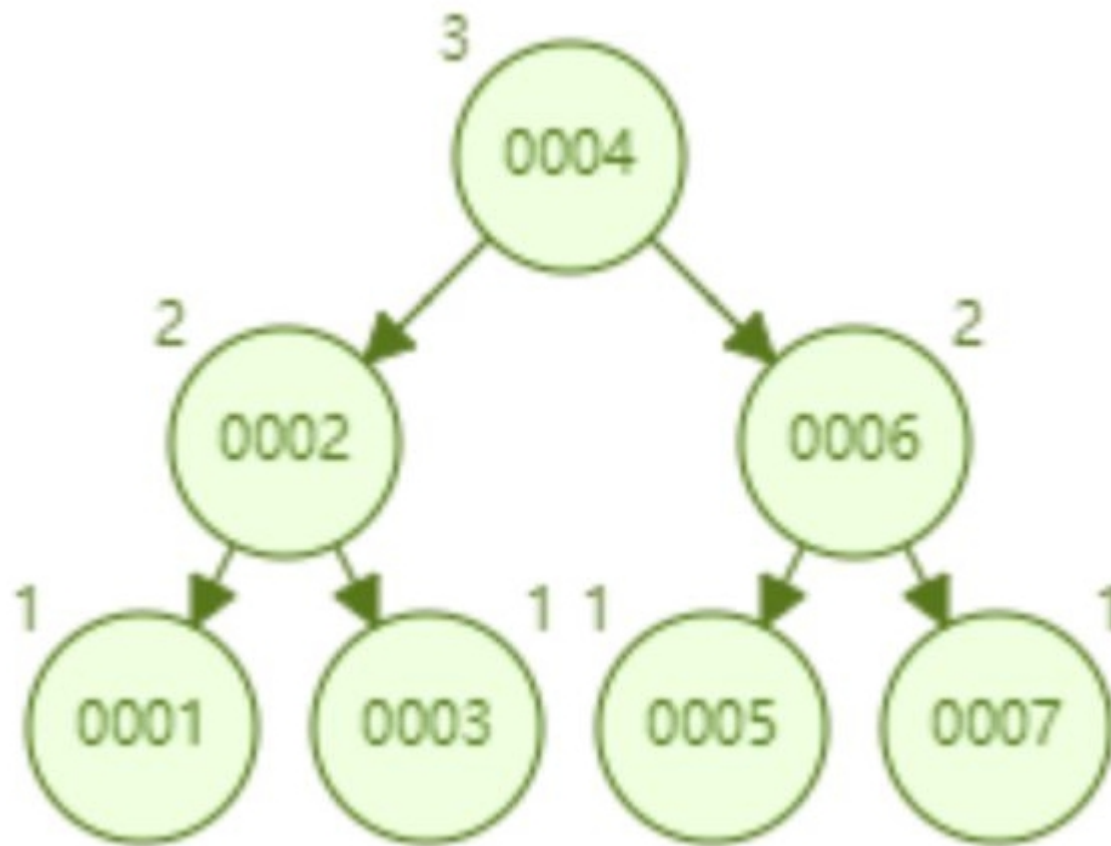
红黑树16个节点

- 红黑树顺序插入 1~16 个节点，查找 id=16 需要比较的节点数为 6 次



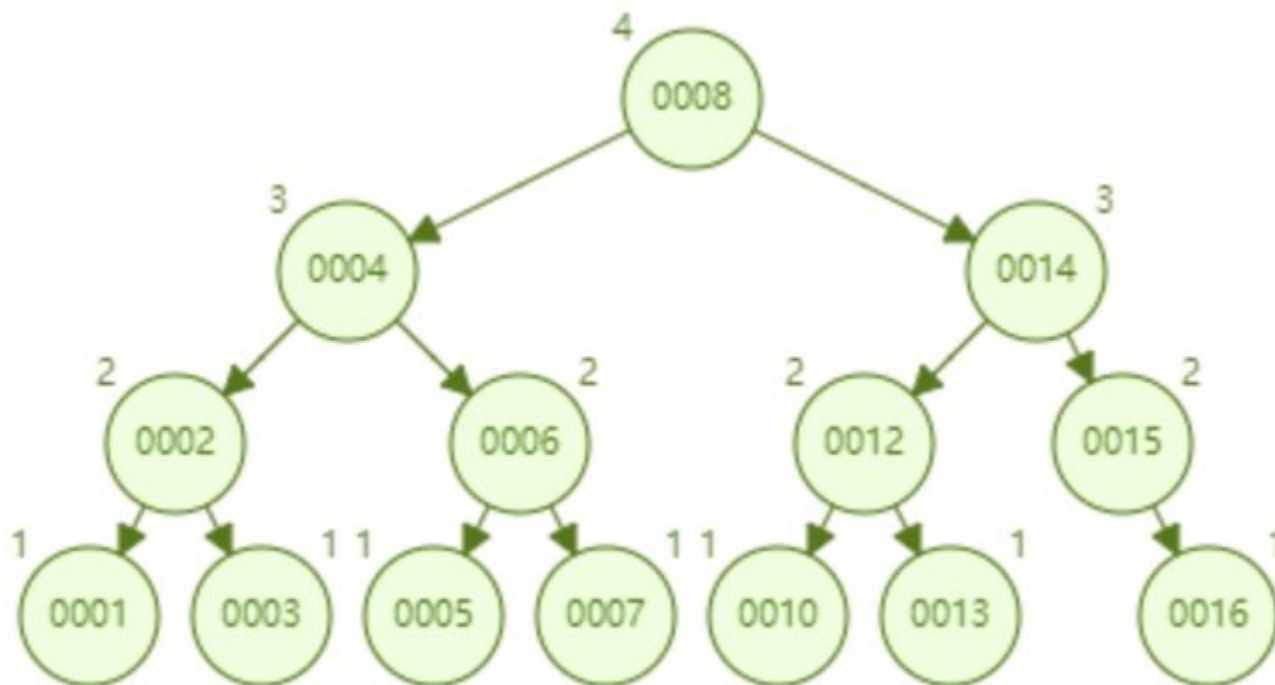
自平衡二叉树 AVL 树

- AVL 树顺序插入 1~7 个节点，查找 id=7 所要比节点次数为 3



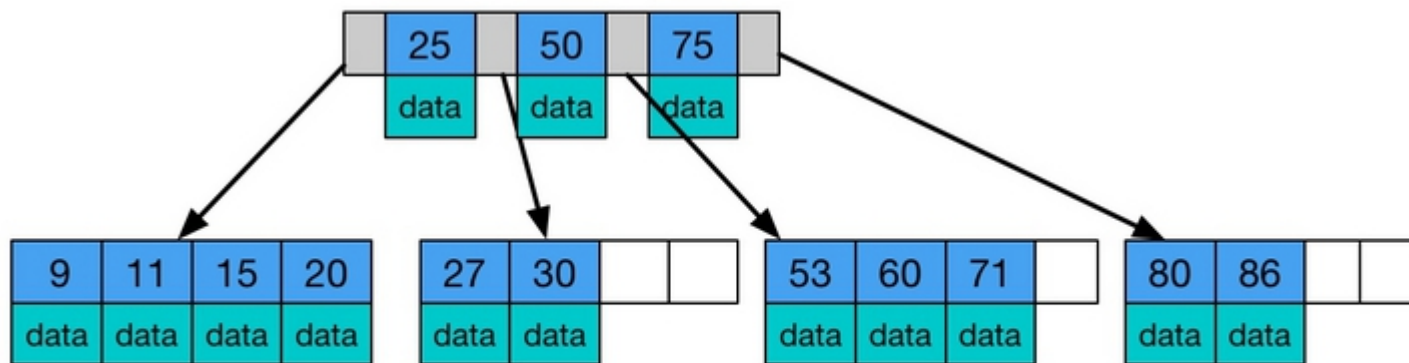
AVL 树

- AVL 树顺序插入 1~16 个节点，查找 id=16 需要比较的节点数为 4
- 缺点：一个树节点只存储一个数据，一次磁盘 IO 只取一个节点



B-树

- B-树,这里的 B 表示 **balance**(平衡的意思),B-树是一种多路自平衡的搜索树
- 它类似普通的平衡二叉树，不同的一点是B-树允许每个节点有更多的子节点。



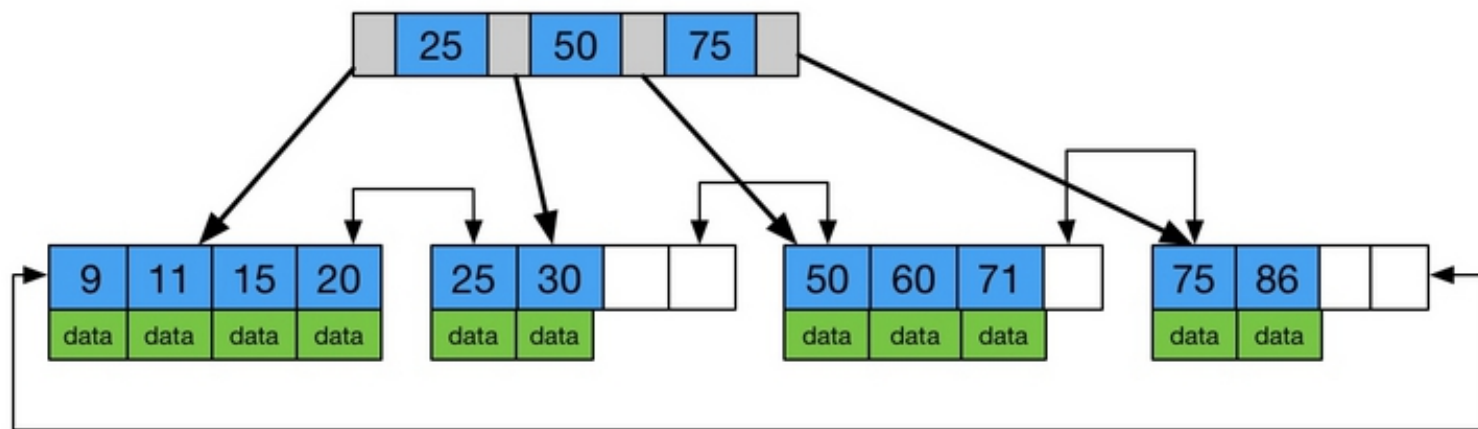
简化 B- 树

B-树特点

- 所有键值分布在整颗树中；
- 任何一个关键字出现且只出现在一个结点中；
- 搜索有可能在非叶子结点结束；
- 在关键字全集内做一次查找,性能逼近二分查找；

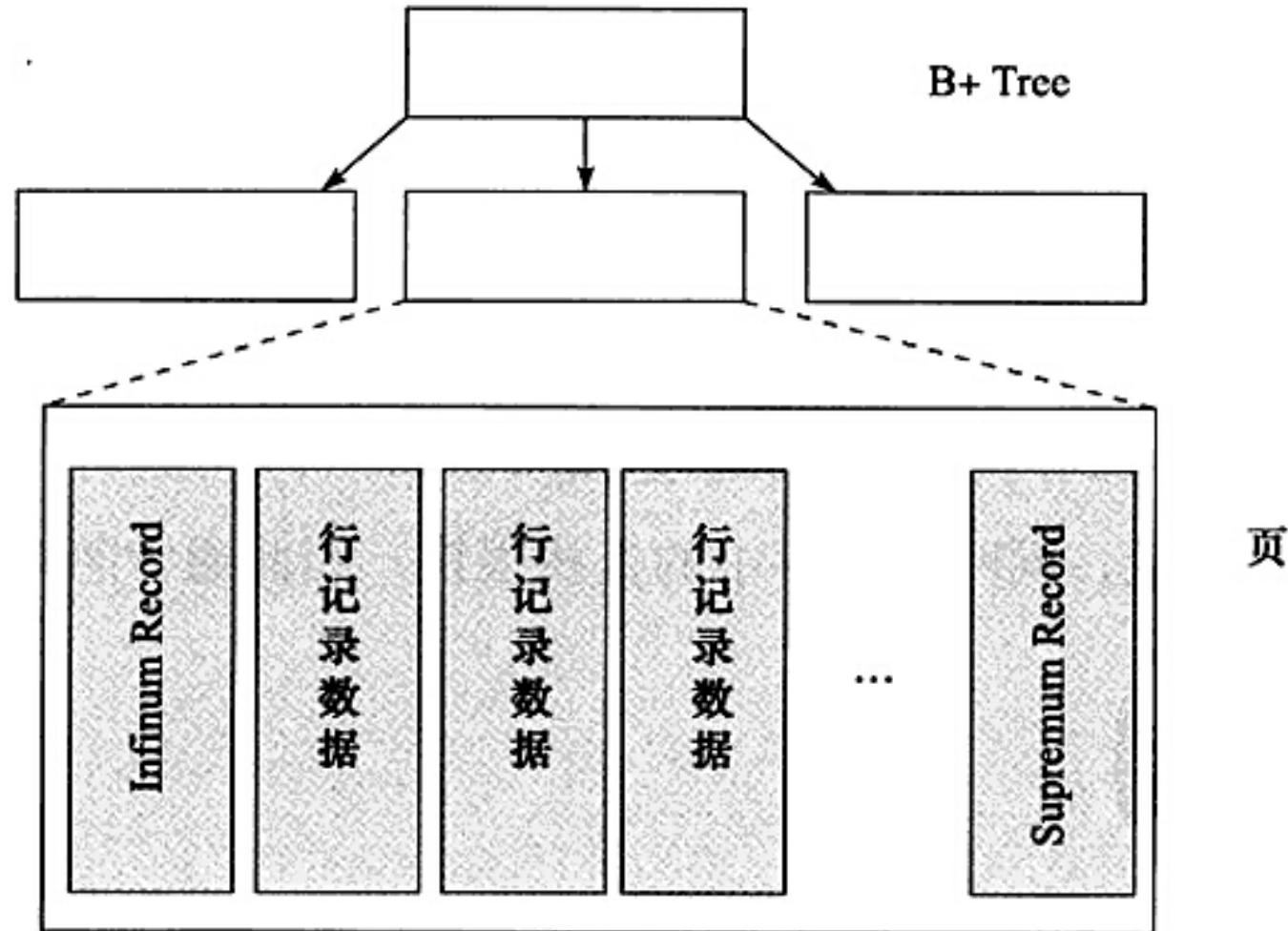
B+ 树

- **B+树是B-树的变体，也是一种多路搜索树, 与 B- 树的不同之处在于:**
 - 所有关键字存储在叶子节点出现,内部节点(非叶子节点并不存储真正的 data)
 - 为所有叶子结点增加了一个链指针

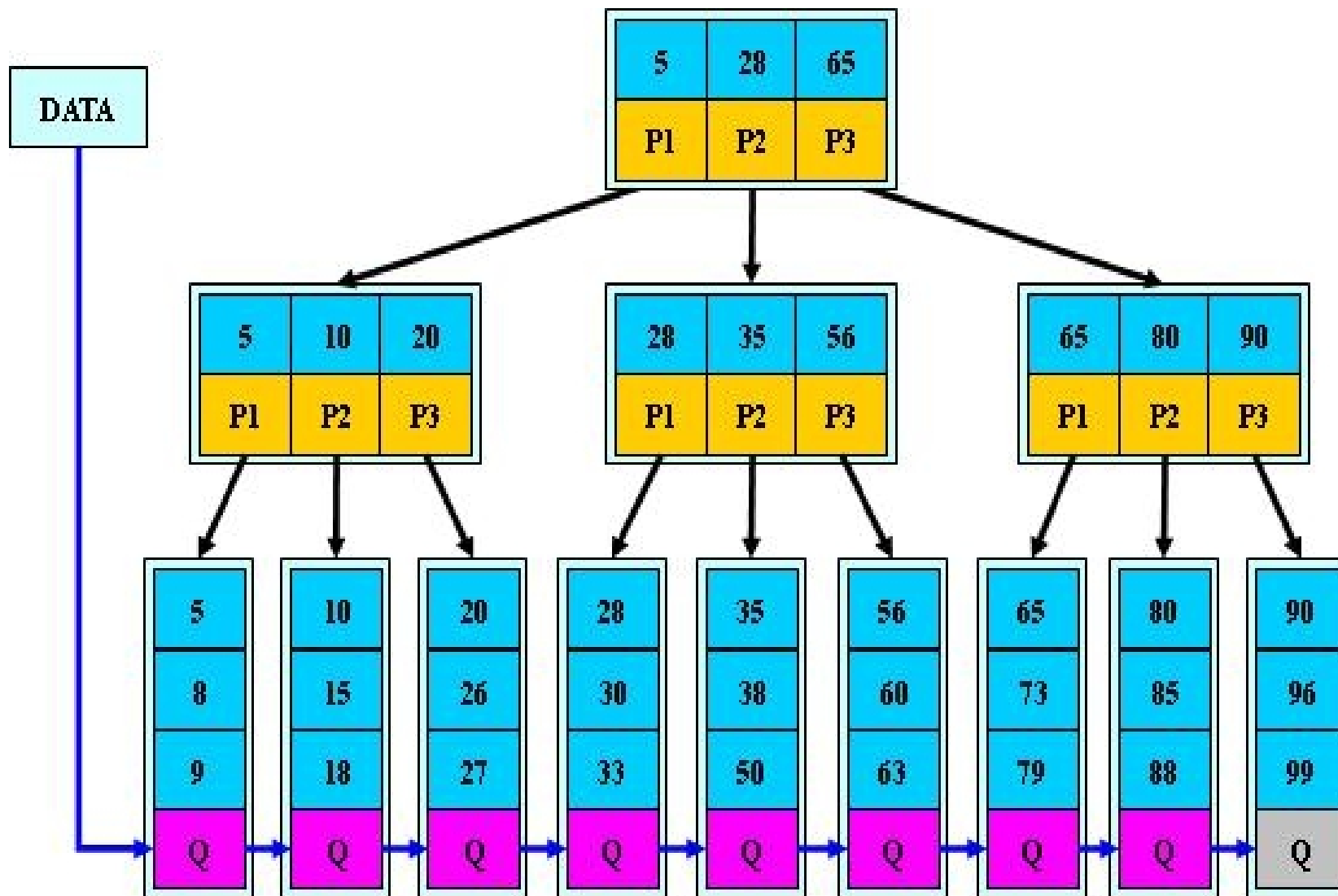


简化 B+ 树

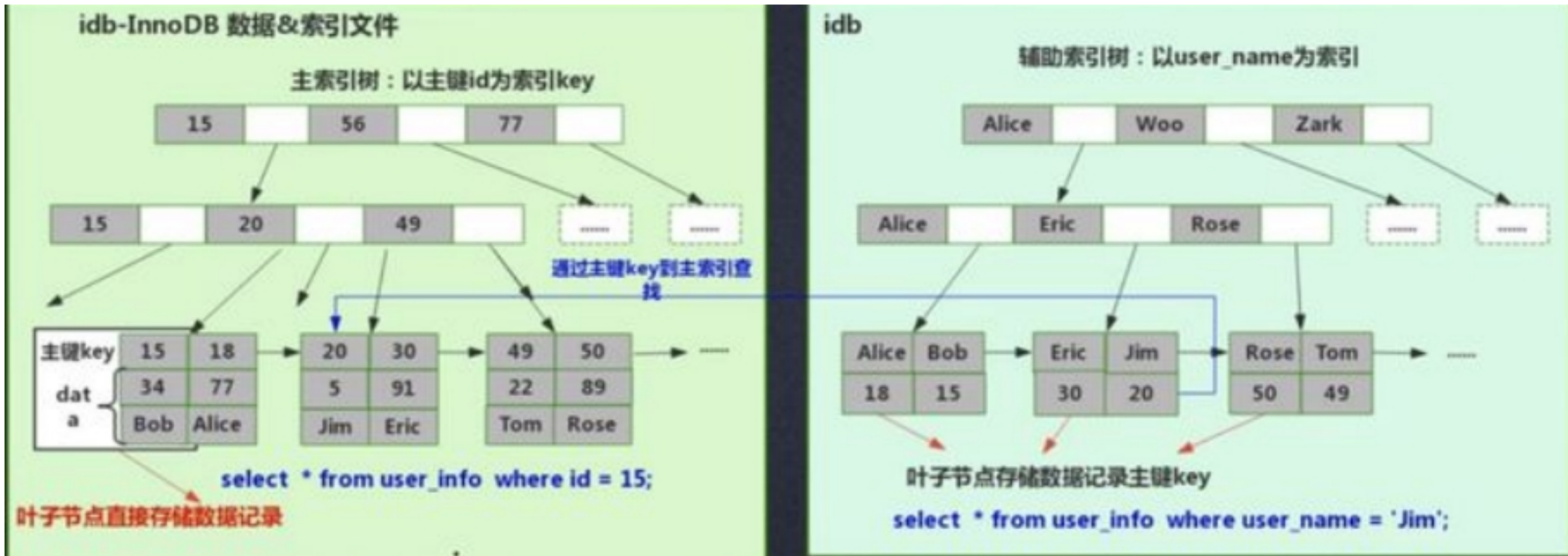
B+树



B+ 树的结构图



InnoDB数据&索引

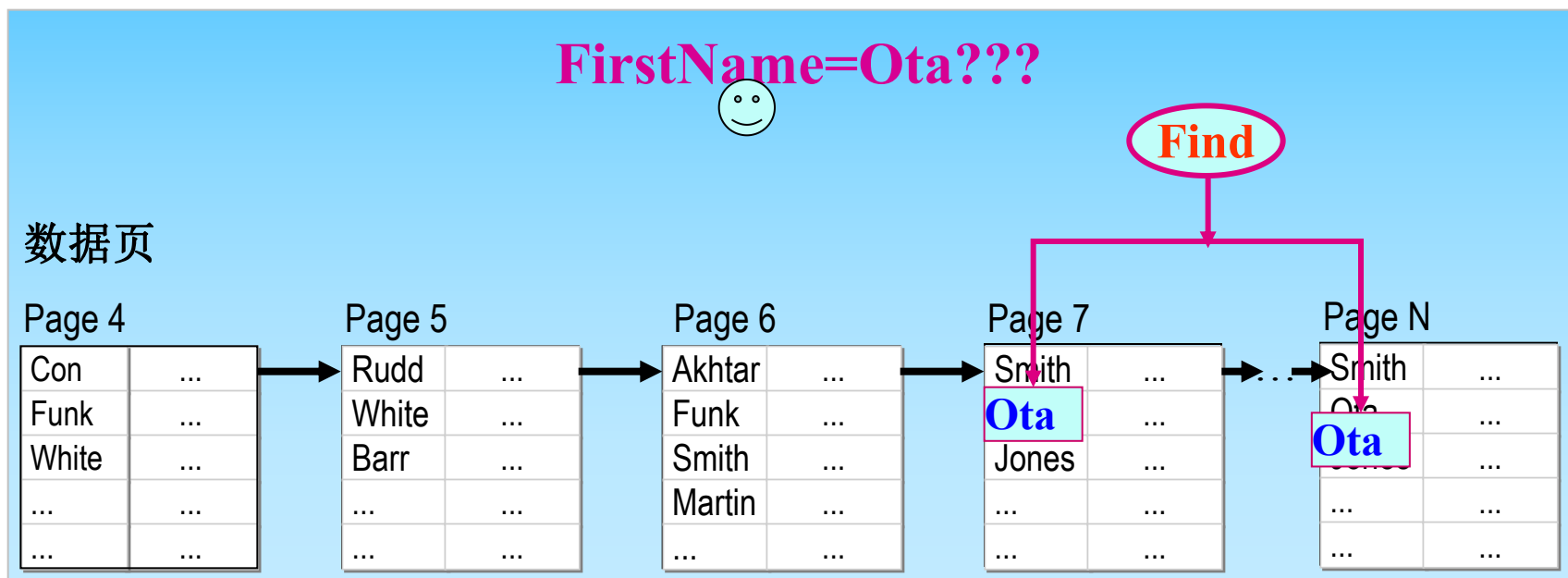


MySQL的数据访问

- 在一个表中扫描所有的数据页

查找姓名为Ota的记录

```
Select FirstName From Member  
Where FirstName='Ota'
```

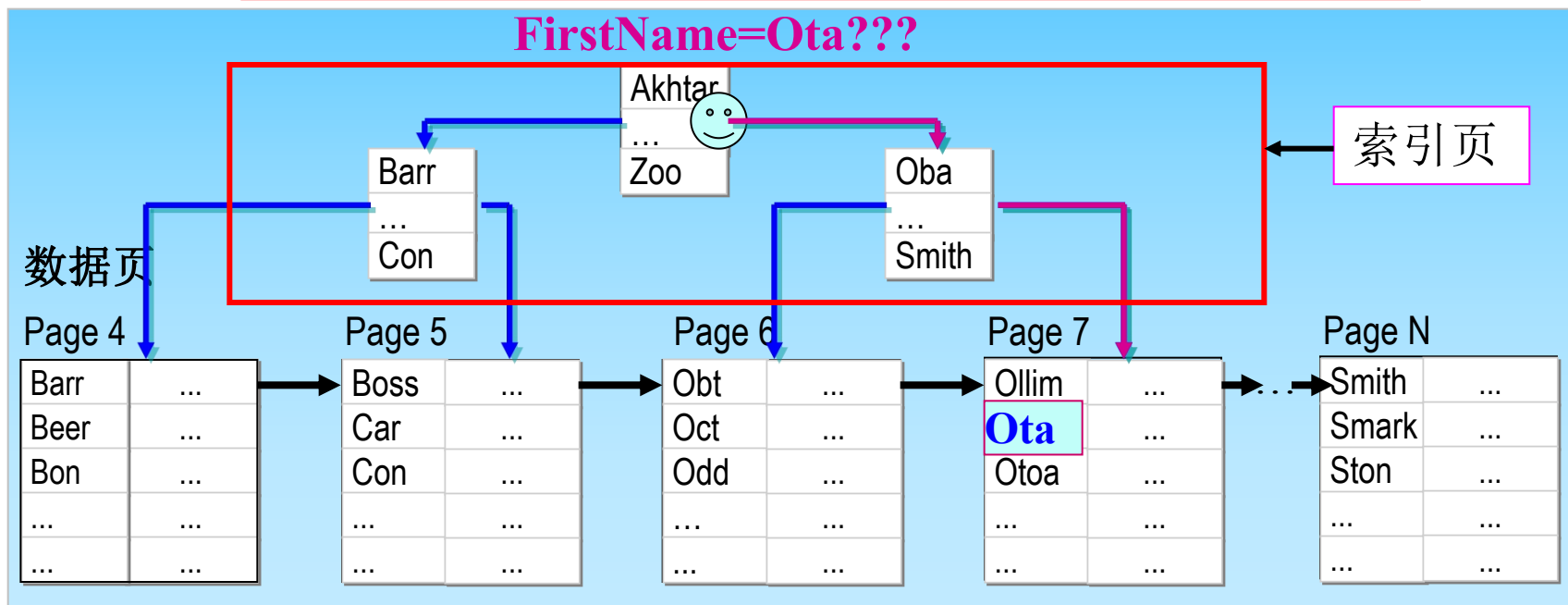


MySQL的数据访问

- 用指向数据页数据的索引

查找姓名为Ota的记录,FirstName为聚集索引

Select FirstName From Member Where
FirstName='Ota'



7.2 索引

- 数据库中的索引与书籍中的索引类似，在一本书中，利用索引可以快速查找所需信息，无须阅读整本书。
- 在数据库中，索引使数据库程序无须对整个表进行扫描，就可以在其中找到所需数据。
- 书中的索引是一个词语列表，其中注明了包含各个词的页码。
- 数据库中的索引是某个表中一列或者若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。

7.2.1 索引的概念

- 索引是对数据表中一个或多个字段的值进行排序的结构。
- 表中的一个索引就是一个列表，列表包含了一些值，以及包含这些值的记录在数据表中的存储位置。
- 索引键可以是单个字段、多个字段的组合字段
- 索引自身也需要进行维护，并占用一定的资源。一般只有经常用来检索的字段上建立索引，例如经常在**WHERE**子句中引用的字段。

7.2.2 索引的作用

- 通过创建唯一索引，可以保证数据记录的唯一性。
- 可以大大加快数据检索速度。
- 可以加速表与表之间的连接，这一点在实现数据的参照完整性方面有特别的意义。
- 在使用ORDER BY和GROUP BY子句中进行检索数据时，可以显著减少查询中分组和排序的时间。
- 使用索引可以在检索数据的过程中使用优化器，提高系统性能。

7.3 使用SQL语句创建和管理索引

■ 索引分类

- 普通索引和唯一索引
- 单列索引和组合索引
- 全文索引
- 空间索引

■ 索引创建

- 创建表时创建索引
- CREATE INDEX创建索引
- ALTER TABLE语句创建索引

索引的设计原则

- 索引并非越多越好。
- 避免对经常更新的表进行过多的索引，并且索引中的列尽可能少。
- 数据量小的表最好不要使用索引。
- 在条件表达式中经常用到的不同值较多的列上建立检索，在不同值少的列上不要建立索引。
- 当唯一性是某种数据本身的特征时，指定唯一索引。
- 在频繁进行排序或分组（即进行**group by**或**order by**操作）的列上建立索引

创建表时创建索引

■ 语法:

- CREATE TABLE
- table_name [col_name data_type]
[UNIQUE|FULLTEXT|SPATIAL] [INDEX|KEY]
- [index_name] (col_name [length])
- [ASC | DESC]

CREATE TABLE 语句添加索引

■ 主键索引

- CONSTRAINT PRIMARY KEY [索引类型] (<列名>,...)

■ 普通索引

- KEY | INDEX [<索引名>] [<索引类型>] (<列名>,...)

■ 唯一索引

- UNIQUE [INDEX | KEY] [<索引名>] [<索引类型>] (<列名>,...)

■ 外键索引

- FOREIGN KEY <索引名> <列名>

创建普通索引 / 唯一索引

- **CREATE TABLE tb_stu_info(**
- **id INT NOT NULL,**
- **name CHAR(45) DEFAULT NULL,**
- **dept_id INT DEFAULT NULL,**
- **age INT DEFAULT NULL,**
- **height INT DEFAULT NULL,**
- **INDEX(height),**
- **UNIQUE INDEX(name)**
- **);**

使用CREATE INDEX创建索引

■ 语法

- CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
- [index_type]
- ON tbl_name (index_col_name,...)
- [index_type]

■ 参数

- index_col_name: col_name [(length)] [ASC | DESC]
- index_type: USING {BTREE | HASH | RTREE}

索引示例

- -- 创建无索引的表格

```
create table testNoPK (  
    id int not null,  
    name varchar(10)  
);
```

- -- 创建普通索引

```
create index IDX_testNoPK_Name on testNoPK (name);
```

使用ALTER TABLE语句创建索引

■ 语法

- ADD INDEX [<索引名>][<索引类型>](<列名>,...)
- ADD PRIMARY KEY [<索引类型>](<列名>,...)
- ADD UNIQUE [INDEX | KEY][<索引名>][<索引类型>](<列名>,...)
- ADD FOREIGN KEY [<索引名>](<列名>,...)

■ 示例

- alter table test add index(t_name);

删除索引

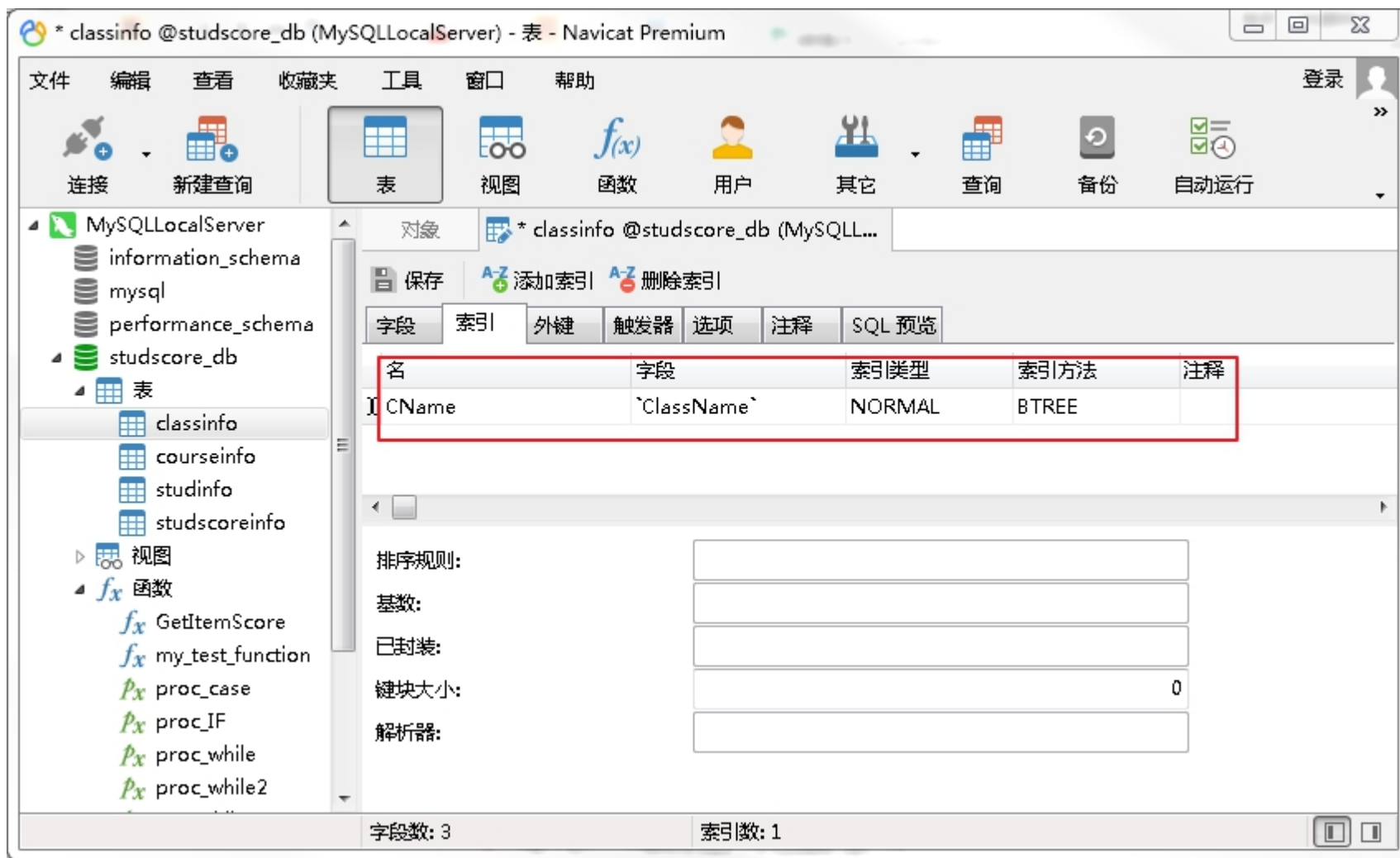
■ 使用**ALTER TABLE**删除索引

- ALTER TABLE table_name DROP INDEX index_name;

■ 使用**DROP INDEX**语句删除索引

- DROP INDEX index_name ON table_name;

7.4 使用Navicat创建和管理索引



下次课内容

- 常量和变量
- 运算符和表达式
- 选择结构
- 循环结构while
- LOOP、REPEAT
- course管理系统实例