第二讲：

关系数据库的特点：用表来存储数据、表是互相关联的。

属性-字段

元组-记录

记录集合-存储块

块的集合-存储关系

模式schema就是表头。Example：SC{S，C，G，G’}

每个属性有一个数据类型-一个domain（一个特定数据类型元素的集合）

R是多个domains的笛卡尔积的子集

schema associates labels to damains，example：SC{S：string，C：string，G：integer，G’：long integer}

超键：一个属性组合，没有重复记录。

候选键：最小个数的超键

主键：get

外键：一个表中的属性要参考另一个表中的主键，

实体完整性：主键非空、无重复值。

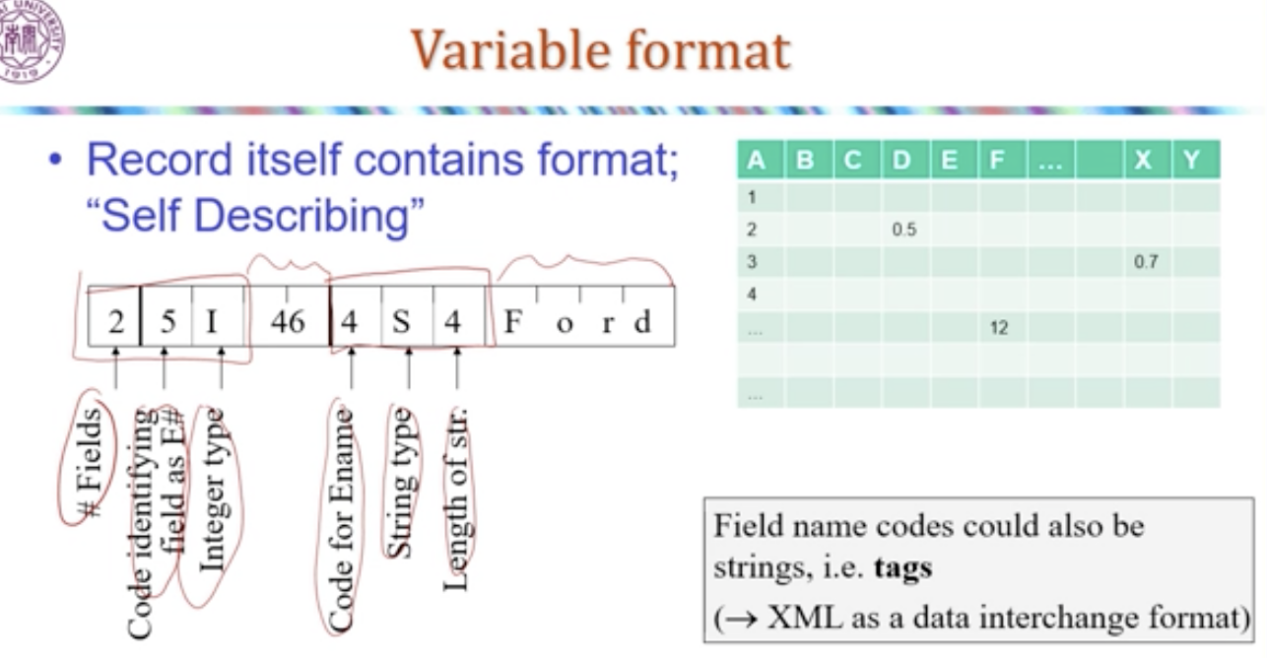
参照完整性：外键如果取值，就要取参考主键中出现的值。（可以空值）

数据项的存储：按相应大小的字节存储，字符编码是ASCII，布尔类型（true全1，false全0；枚举类型（编号后存储）日期类型YYYYMMDD/YYYYDDD/YYYY-MM-DD；时间类型整数或者字符；字符串：null terminated/length given/fixed length）

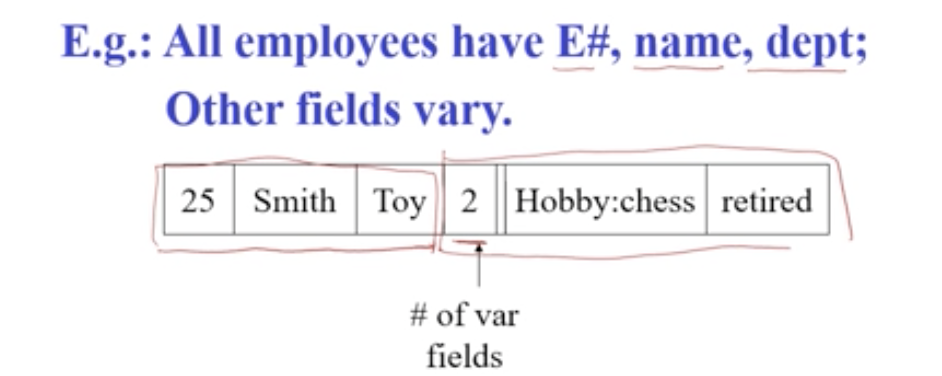
**一个记录的存储**：

1固定模式：模式固定，每个记录都会存储该表模式的属性组合，如果在每一行记录中的每个属性数据的长度也是固定的，则每条记录的长度也是固定的。

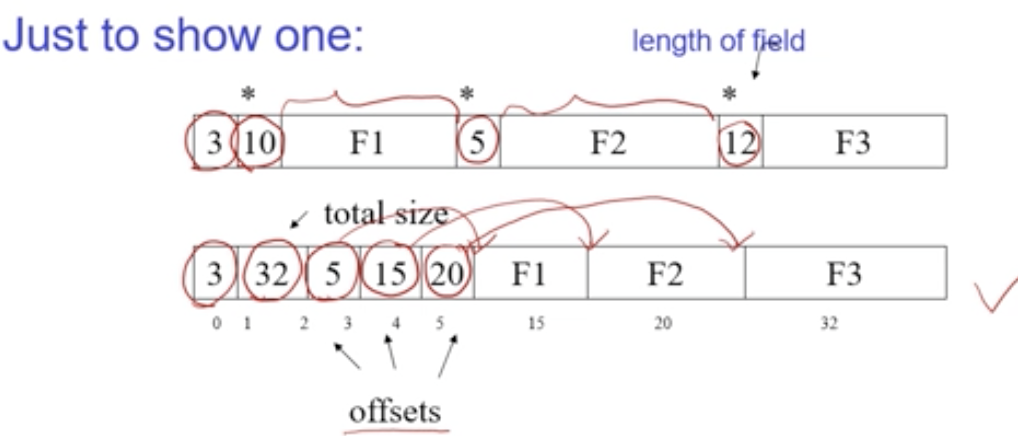
2变化模式：更多应用在一个稀疏表中，具体如下图：



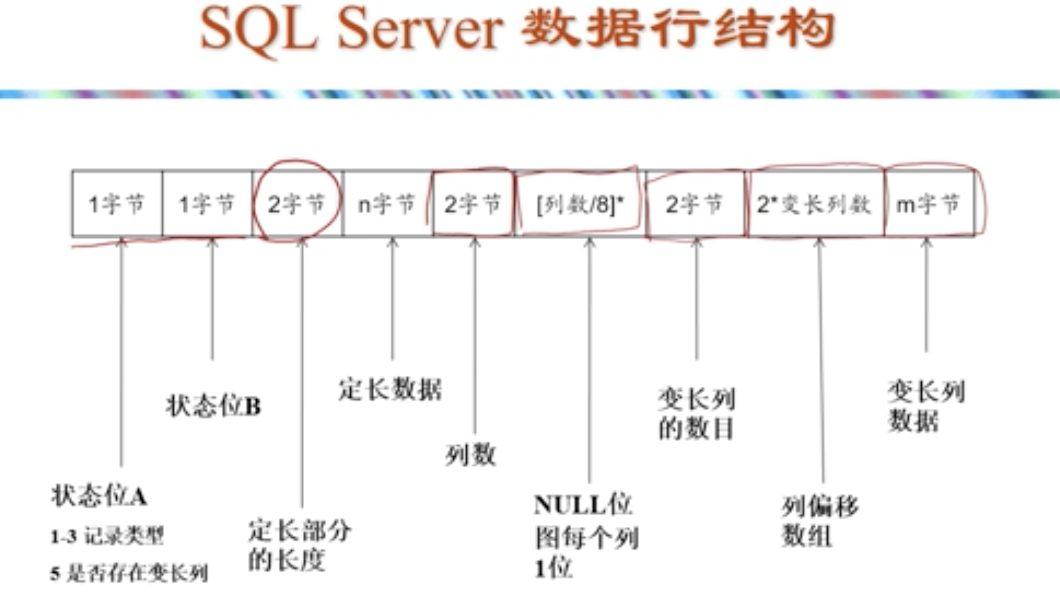
3混合模式：



记录的两种存储方式：



一个数据行的结构：

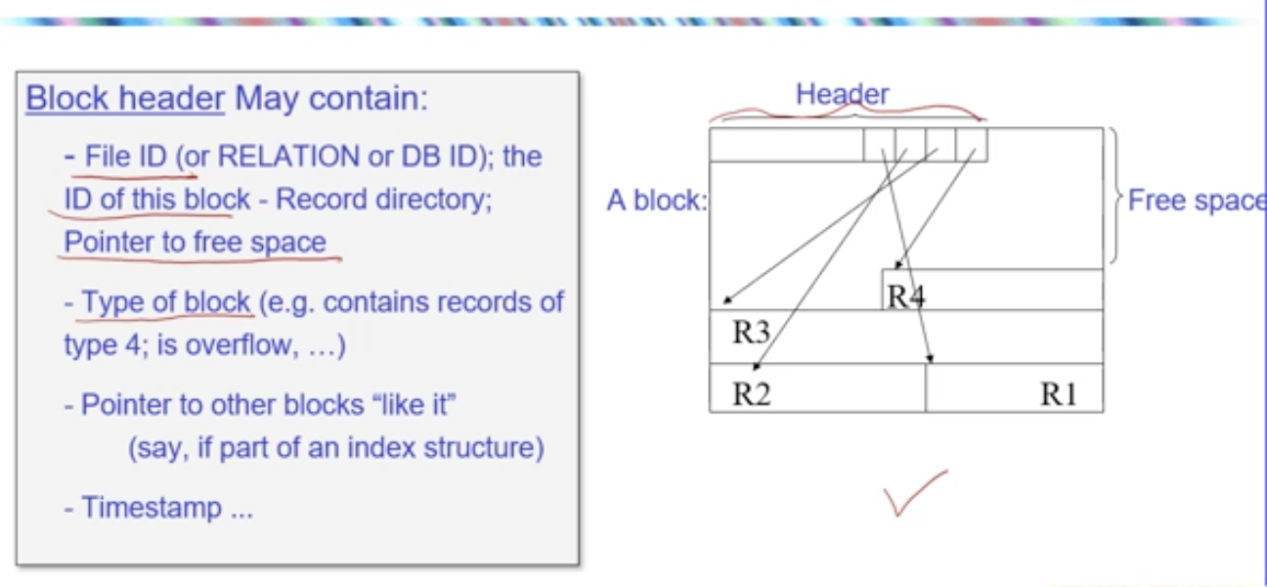


**记录放入块的不同方法**：

1. 依次放入：如何分割：固定长度不需要分割/特殊标记/给出偏移量
2. Spanned/unspanned：不跨块存储：一个记录只能属于一个块。（简单但是浪费空间）跨块存储：一个记录可以跨块。（不受块大小的约束，当记录大小大于块的时候就必须跨块存储）
3. Mixed record tpyes：在一个数据块总放入不同数据类型的数据，让频繁访问的数据在一个数据块中加快效率。
4. Split records：记录的定长部分在一个块中，变长部分在另一个块中。

数据放入块中的存储顺序方式：

1. 物理邻接
2. Linked，用指针链接起来。
3. Overflow area：溢出块，要定期重组。





列存储：将主键与每个属性值单独绑定，到时候连接用主键id来连接。

我们学的都是行存储，适合经常进行事务操作。

**课堂习题**：

候选码不唯一。

**Fixed-size records should always be fixed-format：false**

**课后练习**：

1.Suppose we have blocks of size 4096 bytes that we use to store fixed-length records. Each block has a 128 byte header used to store information including the number of records in the block.

(1) Suppose we have records consisting of a 12 byte header, and 3 fields of size 16 bytes, 32 bytes and 70 bytes respectively. Within each record, fields can start at any byte. How many records can we put in a block?

答：4096-128=3968，12+16+32+70=130，3968/130=30.

(2) Suppose that we have 5 records, each with a 24 byte header and 1300 bytes of data.

(i) How many blocks will we need to store these 5 records if no spanning is allowed?

答：1300+24=1324，3968/1324=2，5/2+1=3，需要3个blocks。

(ii) How much total free space is there in the blocks if no spanning is allowed? (Assume we are not storing anything else)

答：3968×3-1324×5=5284

(iii) How many blocks will we need to store these 5 records if spanning is allowed? In addition to the record header, each time a record is split into fragments, each fragment needs an **extra header of 12 bytes**.

答：3968-1324×2=1320，1320-12=1308，第一个块存了第三个数据的1308bytes，第二个块要存第三个块的16个bytes，3968-4-12-12=3940bytes，3940/1324>2，所以一共需要两个块就够了。

2.Now consider blocks of size 4096 bytes that we use to store variable-length records. Each block has a fixed 128 byte header used to store information including the number of records in the block. In addition to this fixed header, the header contains variable number of **4 byte pointers** to each record in the block. Records can start at any byte oﬀset and are packed as densely as possible. Which of these following combinations of records can be stored in **a single block**? Circle all that apply.

(i) 59 records of 63 bytes each

4096-128-59×63-59×4>0

(ii) 30 records of 94 bytes each and 10 records of 100 bytes each

4096-128-30×94-10×100-4×40<0

(iii) 50 records of 11 bytes, 10 records of 13 bytes and 20 records of 148 bytes

同理

(iv) 2 records of size 1982

同理

第三讲：

磁盘结构：磁头（读取）、盘片（存储媒介）、磁道（盘片上的同心圆）、轴、扇区（对磁道512字节等分），块。

时间=寻道时间+旋转延迟时间+块传输时间（磁头经过一个块的时间）

寻道时间：磁头从开始移动到数据所在柱面所在的时间。

旋转延迟时间：从寻到所在柱面到磁头找到数据块起始的时间。

块传输时间。

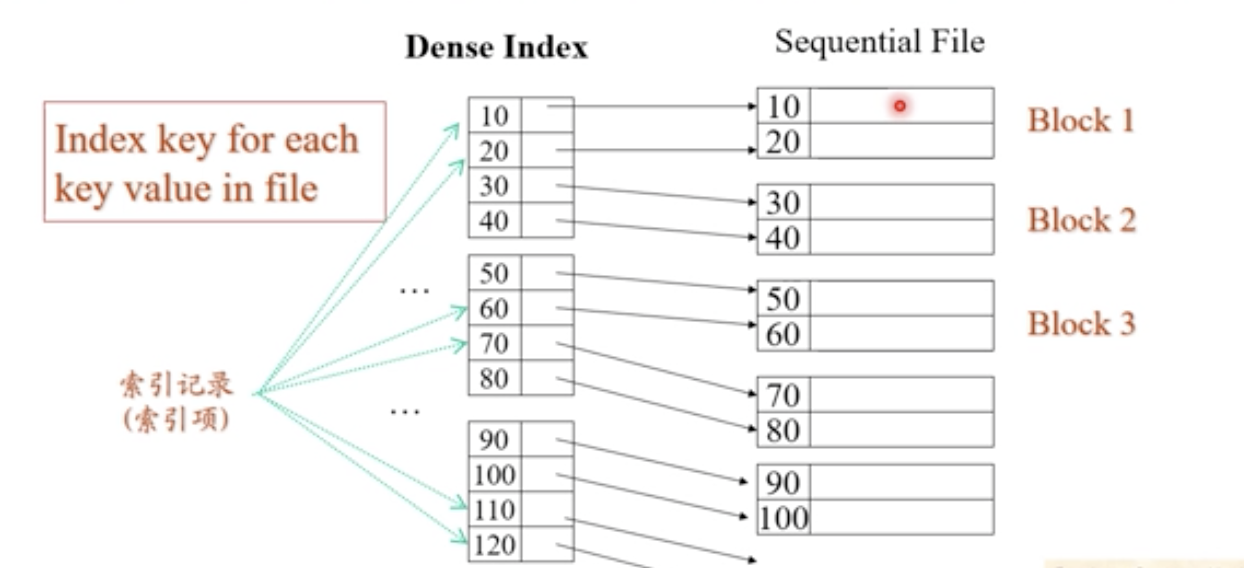
1常规索引：

a稠密索引

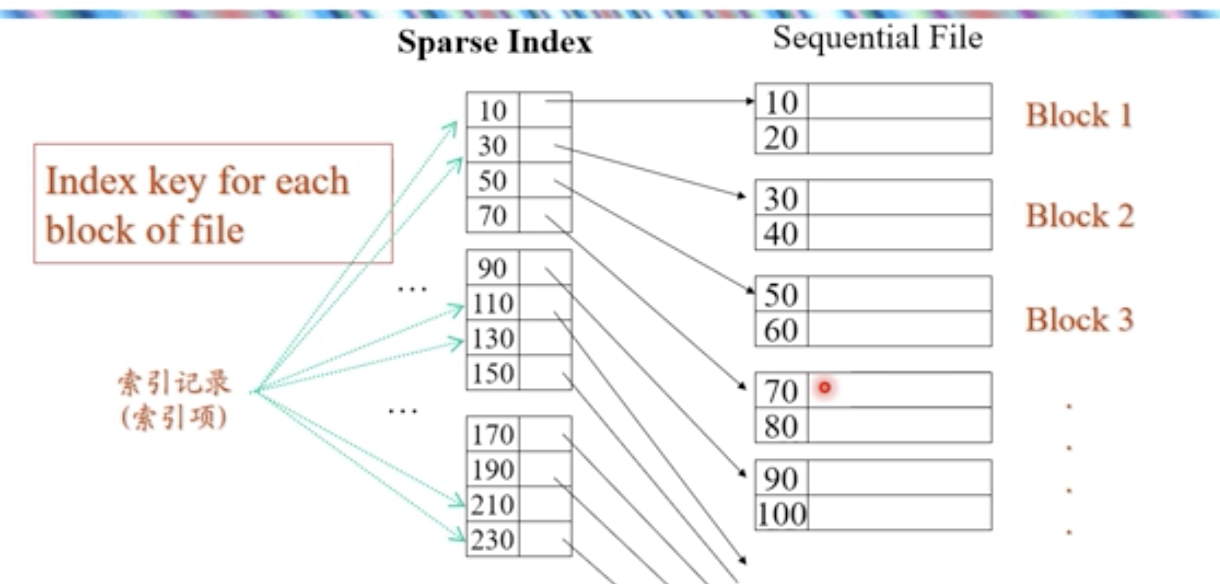
索引包括一个匹配的key值和一个指向对应数据位置的指针。

要求对每个都建立索引，同时要排序。

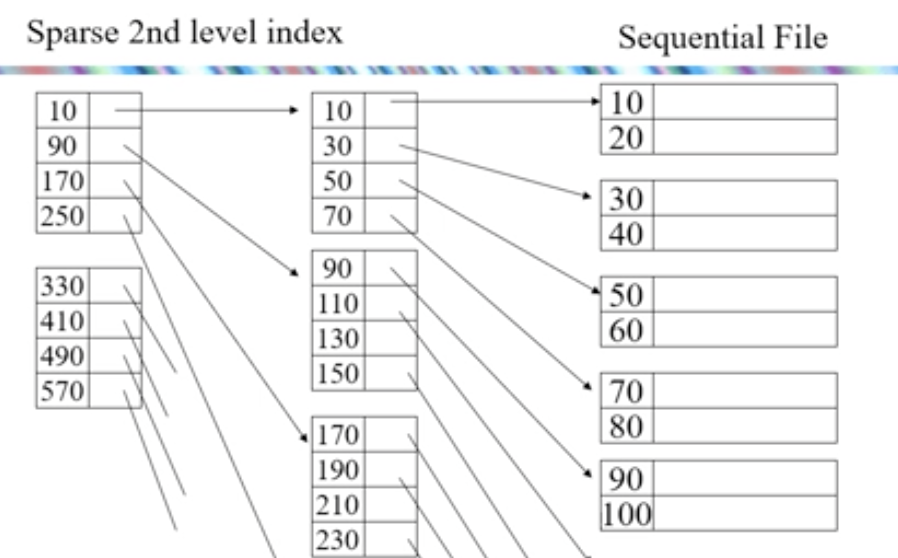
可以通过仅查找索引项就确实数据是否在。



b稀疏索引：为每个数据块建立索引，每个索引项取对应的数据块的最小key值。同时这个要排序。要查找到底层数据块才能确定一个目标key值的存在性。



c多级索引：为索引文件再建立索引，嵌套多级索引，



主索引：**指针直接指向数据块**。

辅助索引：除了主索引，另外建的索引。

**预习题**：

稀疏索引只能建立在排序的属性上面，只能主索引可以是稀疏索引。稠密索引可以在多个属性上建立。

当查询主要是查询那些存在的key值的时候，稀疏索引更好。

identical相同的

重复key值：

1稠密索引，逐个遍历。

2稀疏索引，可能会出现重复key值，但是这时候的搜索的算法要进行改变，搜索时不止要搜索当前块还要检索上一个指针指向的块。（另法：块有重复则跳过，指针指下个块，搜索时要自动搜索当前块的下一个块）（要求原始块排序）

索引更新操作：

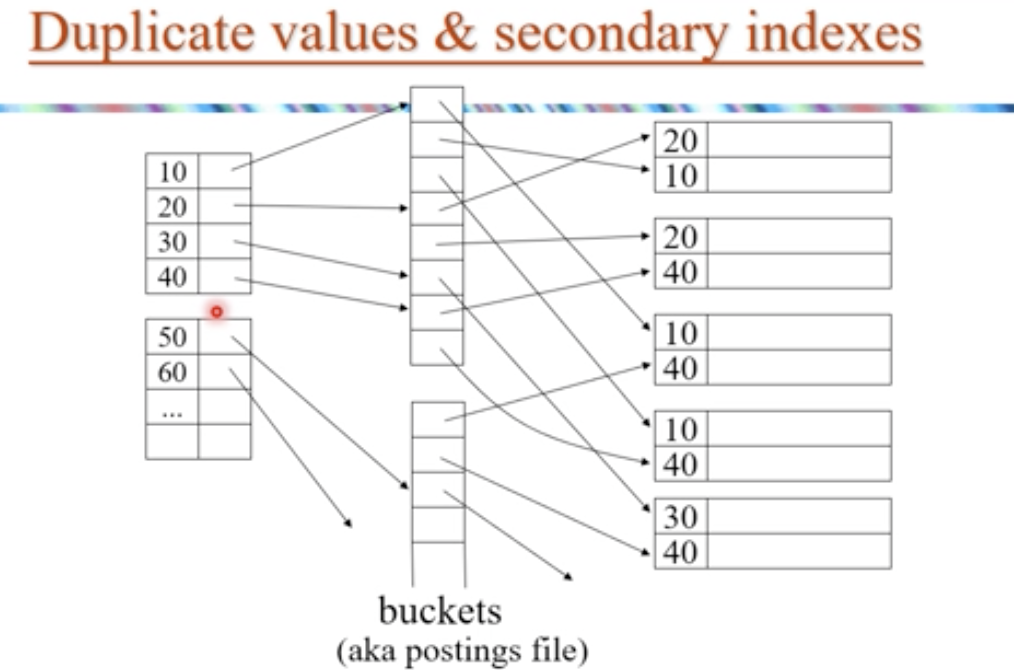
1基于稀疏索引的删除操作：删除不是索引中的指针指向的那个记录，不变无影响。否则将要对对应的索引项更改成块当前的第一个记录。如果一整个数据块上的都被删没了，那么也要对应调整索引项。

2稠密索引删除：俩边同时删除。

3稀疏索引的插入：恰好有剩余空间的时候，索引项可以不进行维护。无空余看相邻块有无空余，如果有进行调整，同时要对索引项进行变化。（DBMS当前技术是使用溢出块来进行插入，用指针链接好顺序，当溢出链很长时，定期重构顺序文件，重构索引）

辅助索引：在非排序上构建索引。第一级索引（指针指向每一个数据）一定是稠密索引，其他级的索引可以是稀疏索引。

为了节省空间（存在重复的key）我们构建指针筒，如下图：（还可以应用在单词定位等）



以上都是常规索引。

**B+树技术**：随数据的变动自动调整结构。

**平衡树**、不需要重组、最少是半空，

非叶节点：n个key，n+1个指针指向下一级索引块。

叶子结点：每个key**直接指向记录**，还有一个邻接指针指向按顺序链接的索引块

非叶子结点指针数不能低于n+1/2向上取整

叶子结点指针数不低于n+1/2向下取整（不包括指向邻接块的指针）

根结点key指从1到n都可

最底层是一个稠密索引。索引树会根据数据的变动同步自动进行调整。

插入算法：从最顶层到最底层搜索位置，如果目标位置有空余则插入。如果目标位置没有空余，需要再申请一个索引块，将原索引块的key值和新加入的一起平均分到两个索引块中，然后再将新的索引块链接到上层索引块中。如果上一种情况中的上层索引块也满了，需要申请一个新的非叶子节点索引块，将这层的索引项平分，然后再链接到上一级的索引块当中，类似的情况都可以进行上述的嵌套操作。最后一种情况就是由于新的索引项的插入导致索引树增加了一层，操作类似。

删除算法：第一种情况，不影响树结构；第二种情况，叶子节点不满足最小指针数要求，要从邻接节点去借key值，然后调整父节点。第三种情况，如果邻接节点无法借，则合并两个索引块，然后相应对父节点进行调整。第四种情况，让整个树结构少一层，正常操作即可。

用墓碑标记删除的key而不是真正的删除从而导致整个树结构的变化，为了节省访问效率，这是一种折中的方法。

这种结构的优势是自动调整结构，不需要重组。读取次数是B+树层数，效率很高。

哈希变换

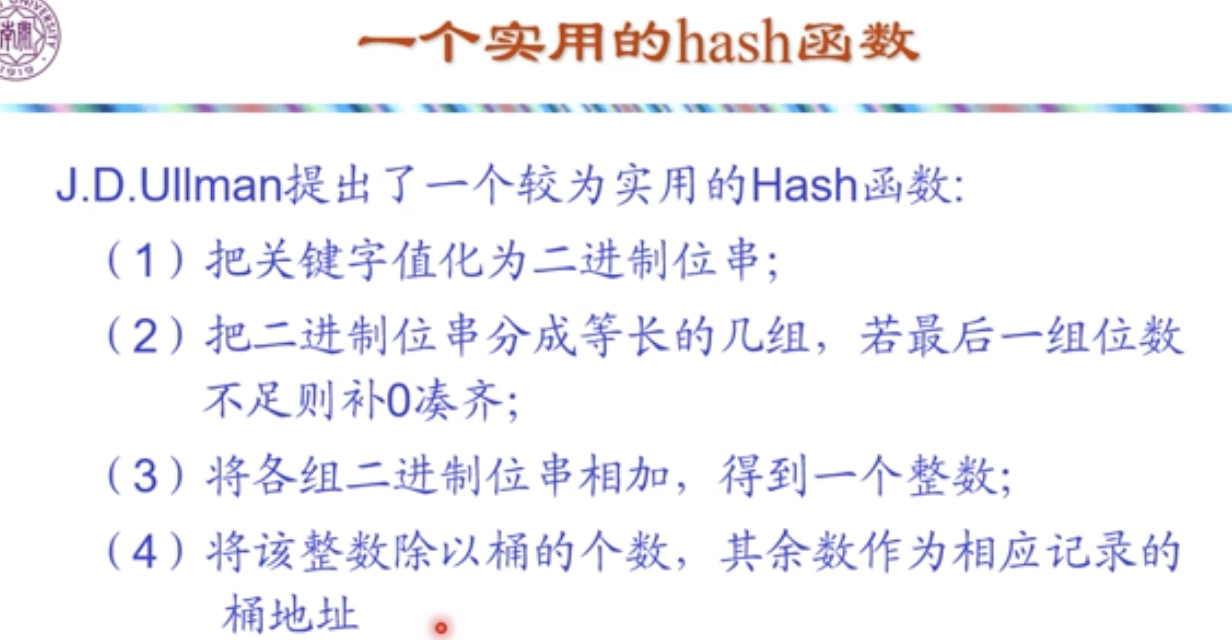
key-》hash（key）

1**主索引**，直接通过key值算H（key）找到记录位置，即根据key值直接定位到记录位置。

2哈希函数输出映射到一个索引桶（辅助索引）中（保留指向所有目标记录的指针），然后再访问到记录位置。

一个好的哈希函数设计：尽量保证key分配要均衡，哈希函数的运算要快。

例子：hash函数

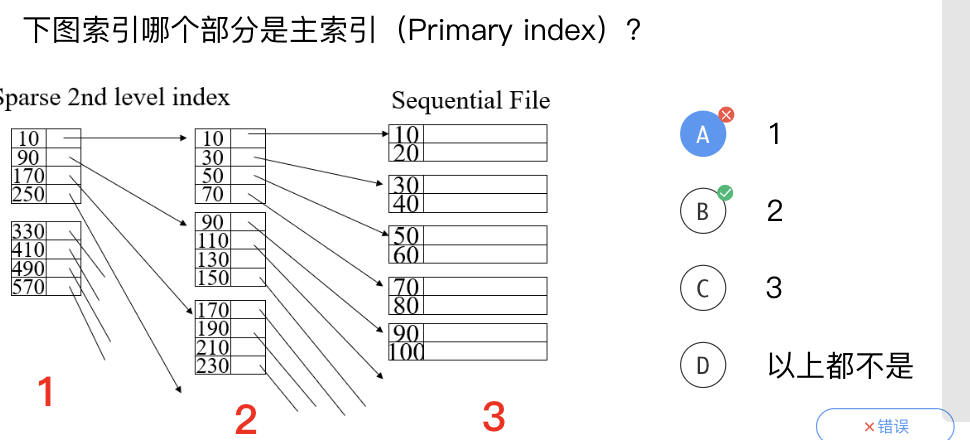


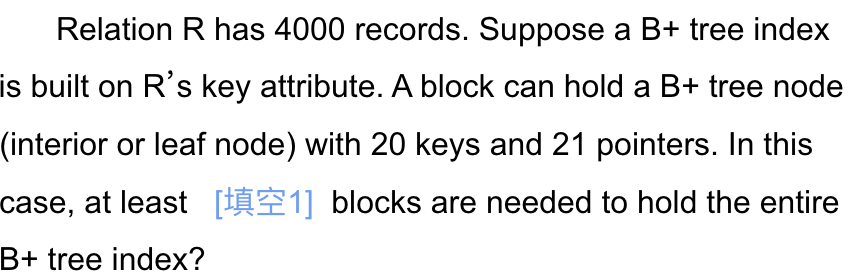
插入：空间不够可以插入溢出桶

删除：如果删除到有链接溢出块的桶需要进行相应的调整

空间利用率（占位的key值/总位置key值）保持在0.5-0.8。

**课堂习题**：





每个数据块最多20个索引项，所以最底层主索引是4000/20=200个数据块，然后200/20是10个，再加1，至少一共211个数据块。