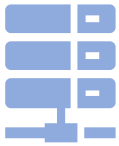
北京航空航天大學
BEIHANG UNIVERSITY

数据管理技术

北京航空航天大学

周号益

2024年



第四章　存储与索引

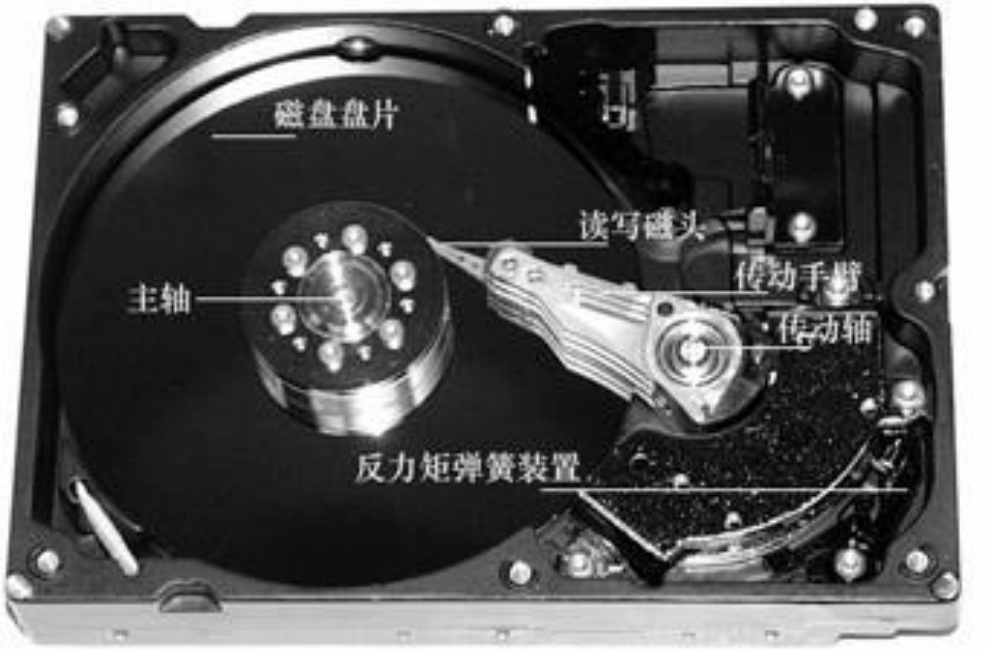
存储介质

物理存储

- ✓存储器分类
  - 一级存储 - - 内存
  - 二级存储 - - 磁盘 ——→ 非易失性存储器
  - 三级存储 - - 大容量存储器（磁带机、DVD等）——→
- ✓存取速度
  - 内存 > > 磁盘 > 大容量存储器
  - 0.01μ秒 0.0025秒 0.01秒 (一次读取时间)
- ✓数据存储原则
  - 在内存中存放常用数据
  - 在磁盘中存放业务数据
  - 在三级存储器中存放归档数据

存储介质

- 磁盘存储
  - ✓数据库技术构建在磁盘存取原理之上
  - ✓磁盘结构



存储介质

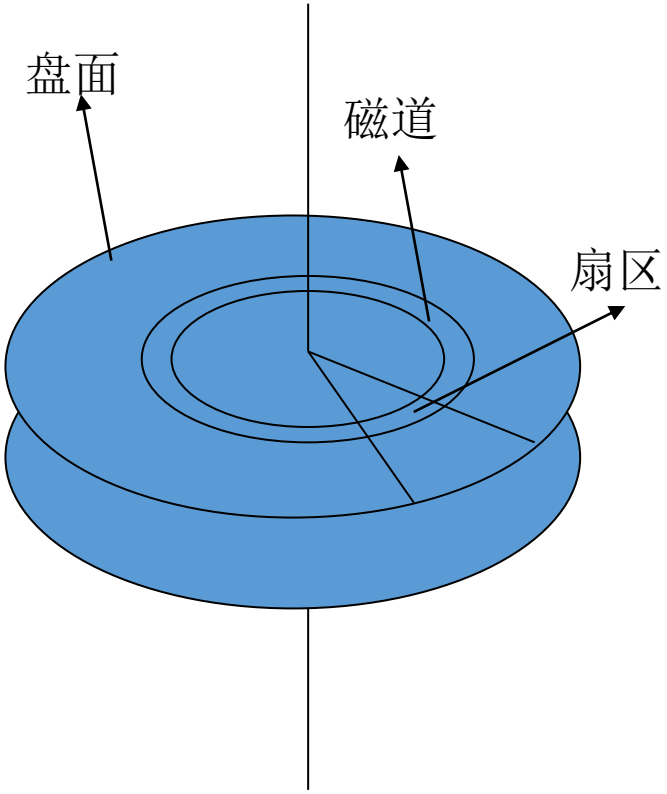
磁盘结构

- ✓盘面（100多个）
- ✓磁道（每个盘面大约1000多个）
- ✓扇区（一个磁道分为若干扇区）
- ✓扇区通常包含512字节

  (越内圈的扇区存储密度越大)
- ✓磁盘页面由若干个连续的、

  可一次性读取的扇区组成，

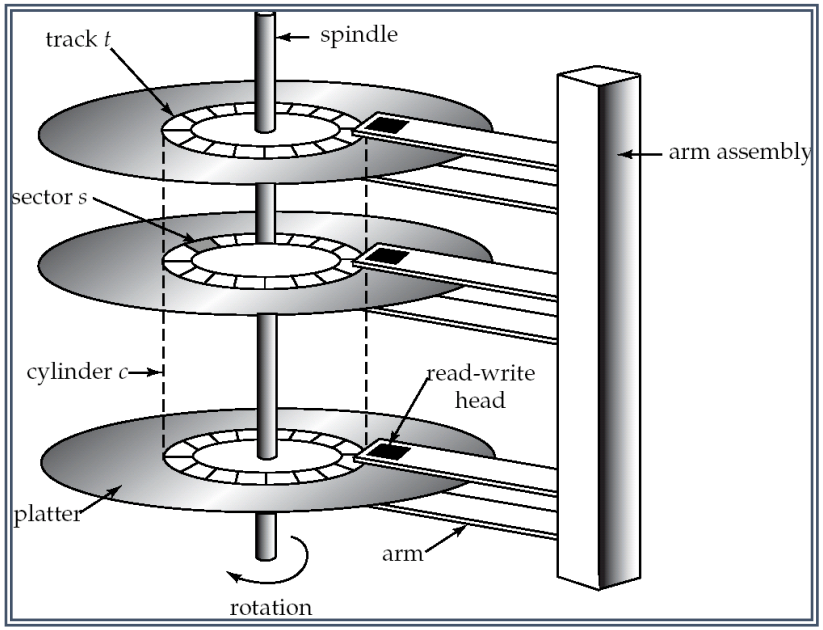
  一般一个页面大小为4KB、8KB等



存储介质

- 磁盘读写时延
  - ✓寻道时延：磁头移动时延10ms
  - ✓旋转时延：平均时延约4ms
  - ✓数据读写时延：
    - 读写4kb（一个页面）约0.04毫秒
  - ✓顺序读写：快（只有读写时延）
  - ✓随机读写：慢（寻道+旋转+读写）
- 磁盘存取的特性对表存储、索引结构等产生重要影响
  - ✓**1. 在内存中存放常用数据**
  - ✓**2. 尽量减少磁盘IO**
  - ✓**3. 磁盘最小存取单元是页面**

一般磁盘数据访问是面向页面的，一次读取整个页面的数据并放在缓存中

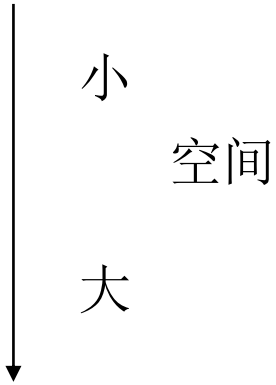


4. 局部性原理与磁盘预读

要尽量将相关的或常用的数据放在同一个页面上；由于磁头顺序读取效率很高，因此通常还预先读取临近的页面

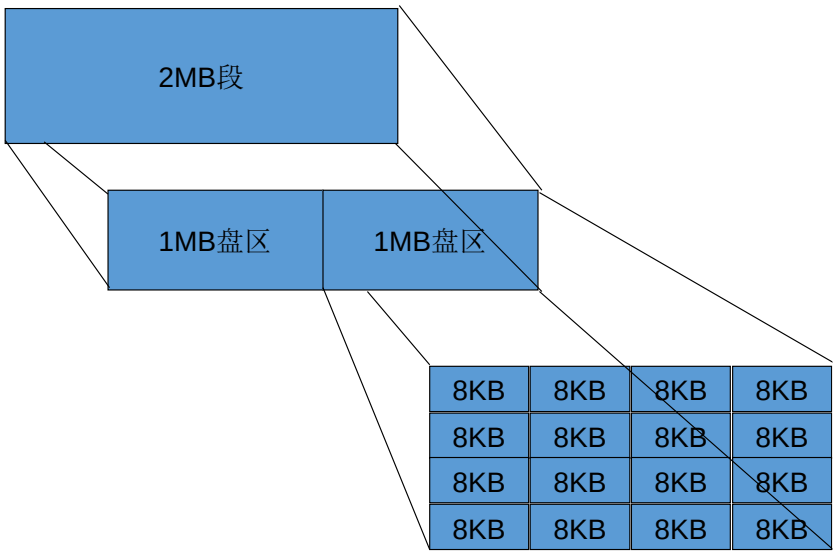
数据的存储结构

- 数据库逻辑存储结构
  - ✓为不同层次的管理目标设置不同级别的磁盘管理单元
  - ✓Oralce的逻辑存储结构：
    - 块
    - 区
    - 段
    - 表空间
    - 数据库



数据的存储结构

- 块
  - ✓相当于磁盘页面，通常为2K~32K固定大小的空间
  - ✓块是数据库中最小的分配单元。一次I/O将读写一整个块。
- 区
  - ✓是在磁盘上连续的块的组合。
- 段
  - ✓由一个或多个盘区组成
  - ✓一个段只存储一类数据对象。例如有表段、索引段、回滚段等
    - 通常数据库中每个表对应于一个段

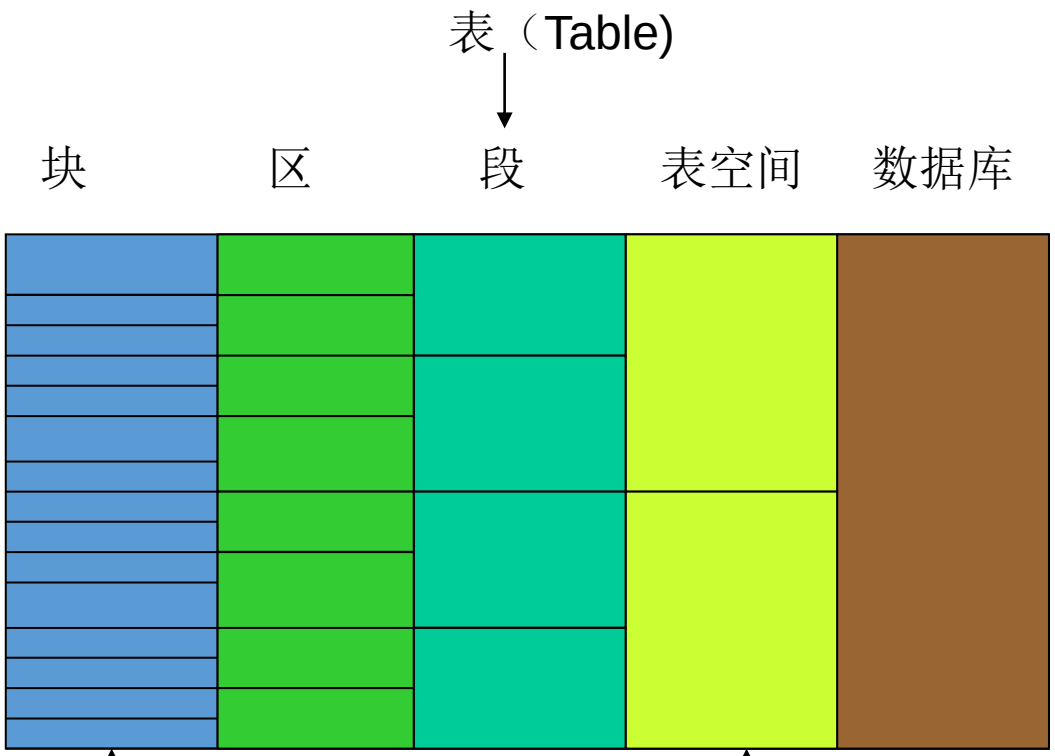


数据的存储结构

- 表空间
  - ✓逻辑上表空间可由0个或多个段组成
  - ✓物理上表空间由一个或多个数据文件组成
  - ✓一个段不能跨越一个表空间,但可跨越表空间内的文件
  - ✓设置表空间的作用是将数据对象的逻辑结构和物理结构统一起来
- 数据库
  - ✓包含了一个或多个表空间(如用户表空间、临时表空间、系统表空间等)
  - ✓组成数据库的表空间和数据文件是一对多关系。

数据的存储结构

逻辑结构：



物理结构：

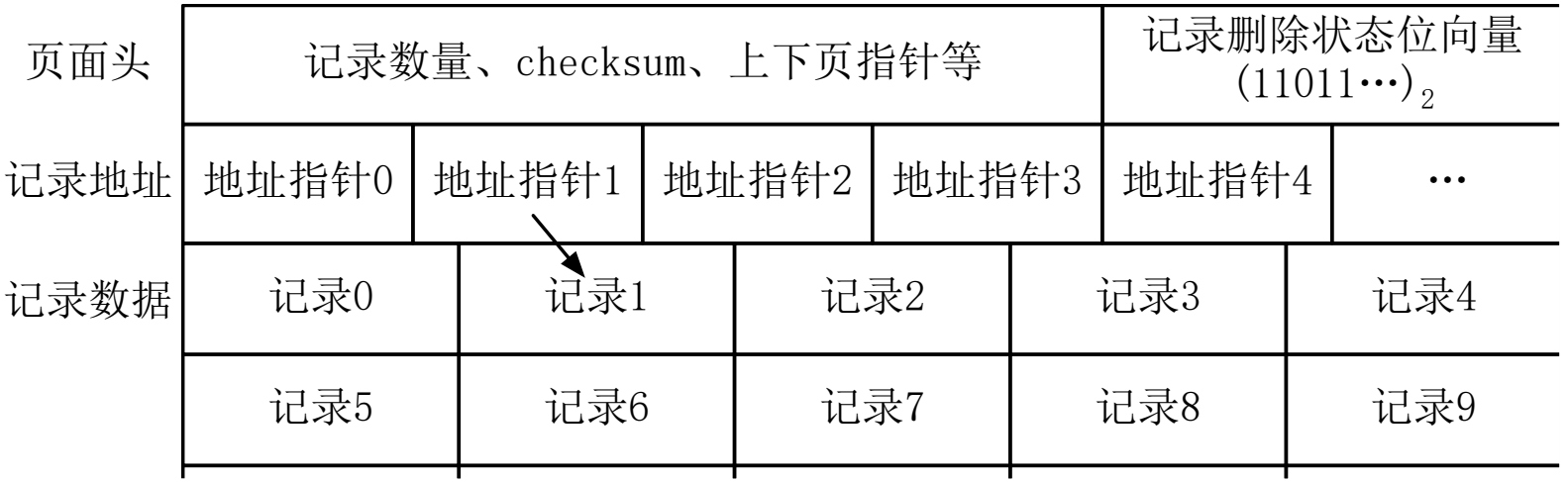
操作系统块

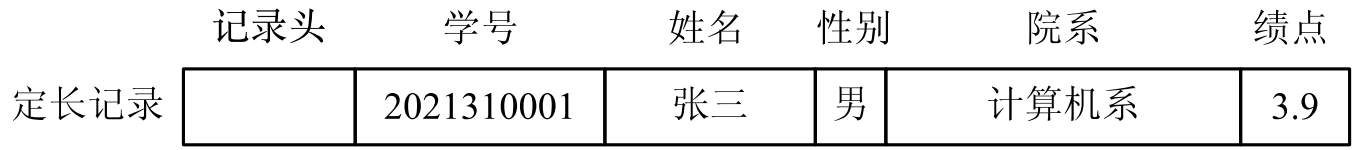
OS Block

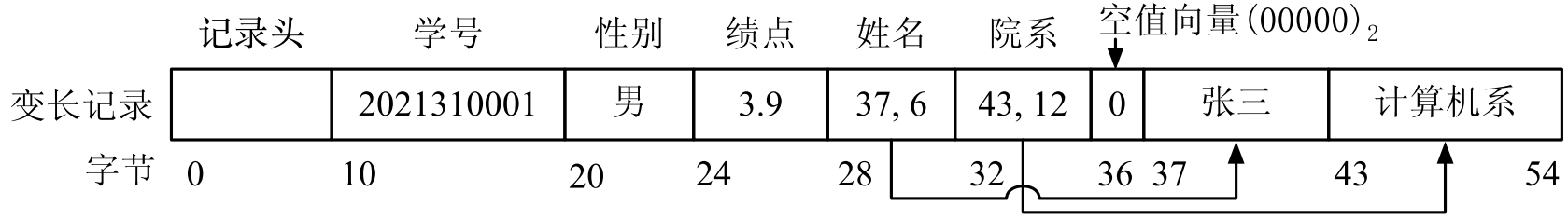
数据文件

数据的存储结构

物理页面组织结构





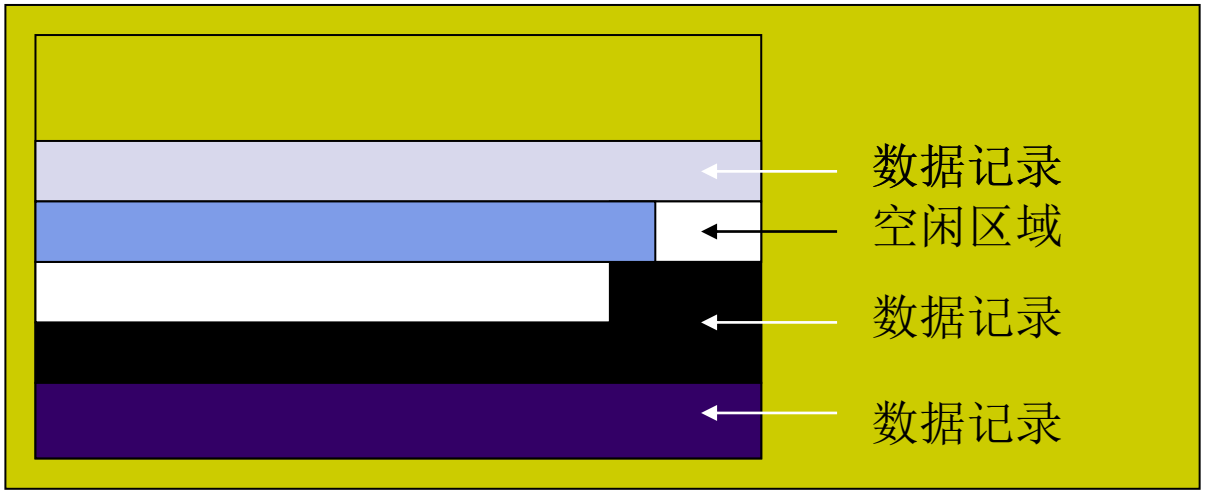


数据的存储结构

- 文件组织结构
  - ✓数据文件中页面之间的组织方式
- 常见文件结构类型
  - ✓堆组织表
  - ✓索引组织表
  - ✓聚簇表
    - ●索引聚簇表
    - ●散列聚簇表

数据的存储结构

- 堆组织表
  - ✓记录的顺序没有限制，将记录简单排列在文件中
  - ✓数据排列顺序是不可预测的



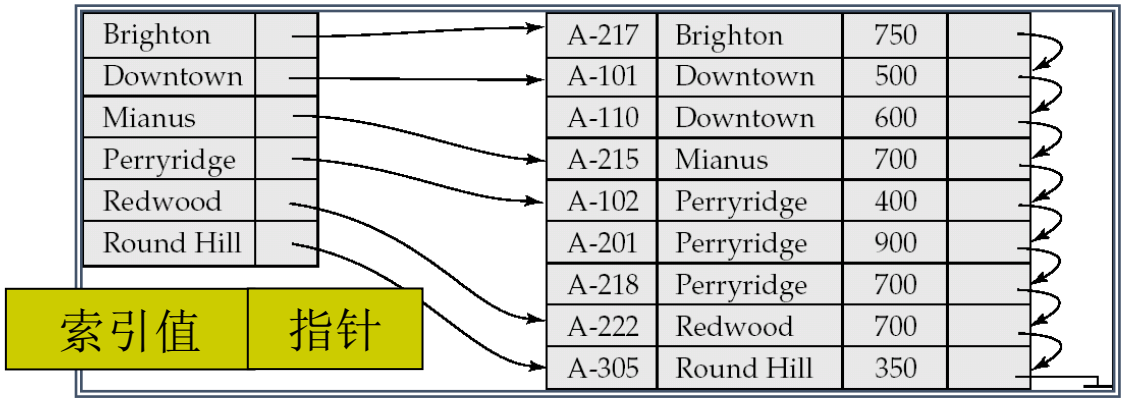
如何在海量数据中寻找符合条件的记录?

索引

■什么是索引

- 是主表上的一种辅助数据结构
- 对数据库表中一个或多个列的值进行排序
- 用于提高主表的查询速度
- 如果想按特定列的值来查找数据，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

二分法查找



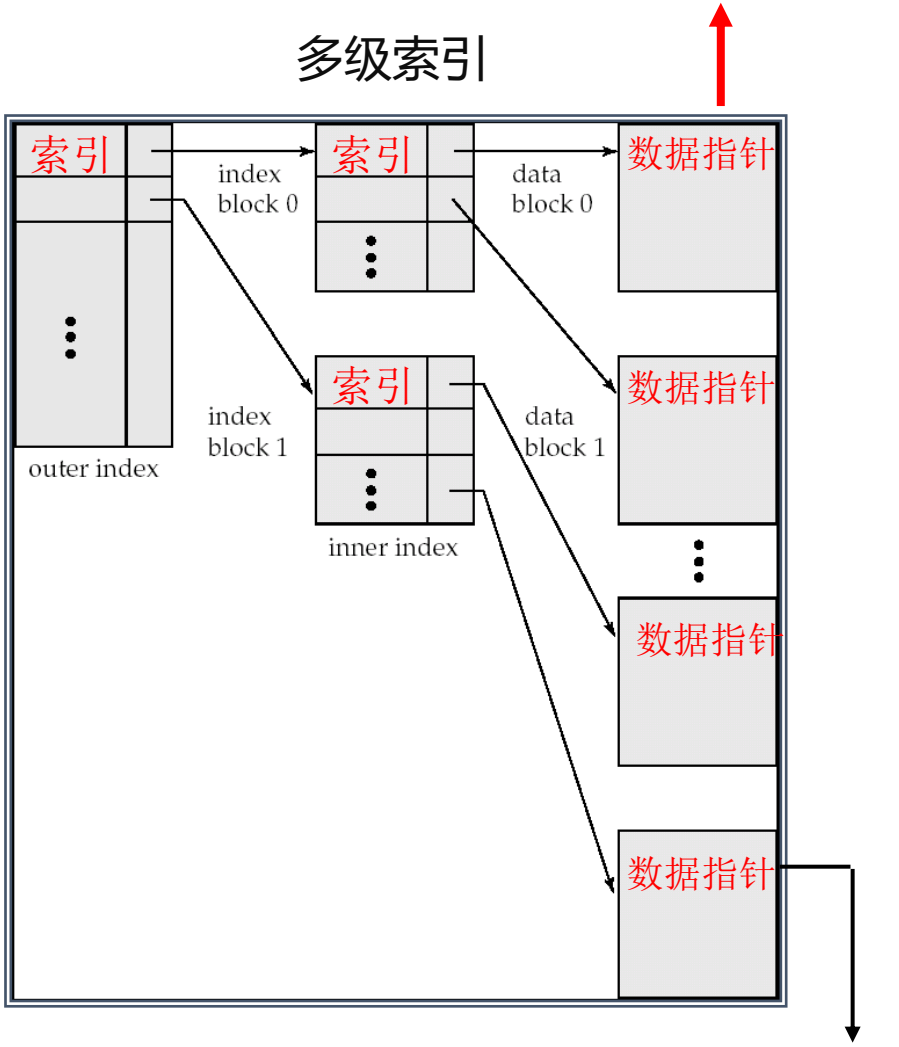
单级索引

索引

每个块是一个页面

■多级索引

- 数据记录数量庞大的情况下，单级索引效率太低
  - 二分法查找时间复杂度：$O(\log_2 n)$
  - 1百万条记录，需要20次磁盘IO
- 改进：
  - 引入多级索引：B+树索引
  - B+树以多叉平衡树存储索引，每个节点可以有多个子节点（多至上千个）
  - 百万记录的查找只需3次IO



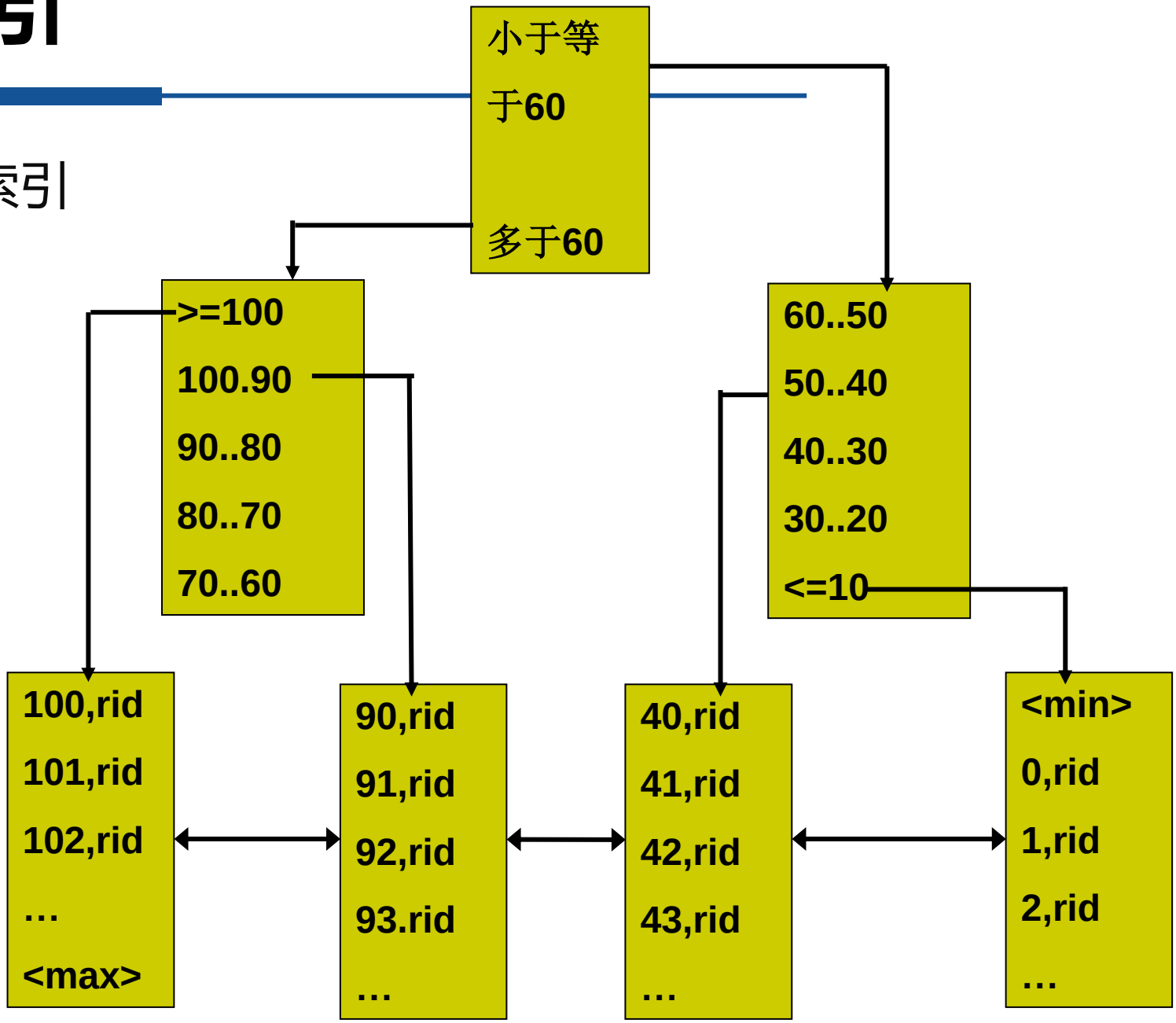
索引

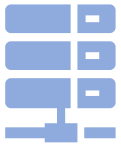
■B+树索引

- 数据库系统中使用最广泛的多级索引
- 特点
  - ✓ 将索引键组织成一棵平衡树， 即从树根到树叶的所有路径一样长
  - ✓ 数据（指向基本表记录存储位置的指针）存储在叶结点
  - ✓ 最底层的叶节点包含每个索引键和指向被索引行的指针（行id）
  - ✓ 叶节点之间有通道可供平行查询
  - ✓ 每一个叶节点都和磁盘页面大小一致
  - ✓ 查询的时间复杂度：$O(\log_m n)$ (m为分叉数，即B+树的阶)

索引

■B+树索引

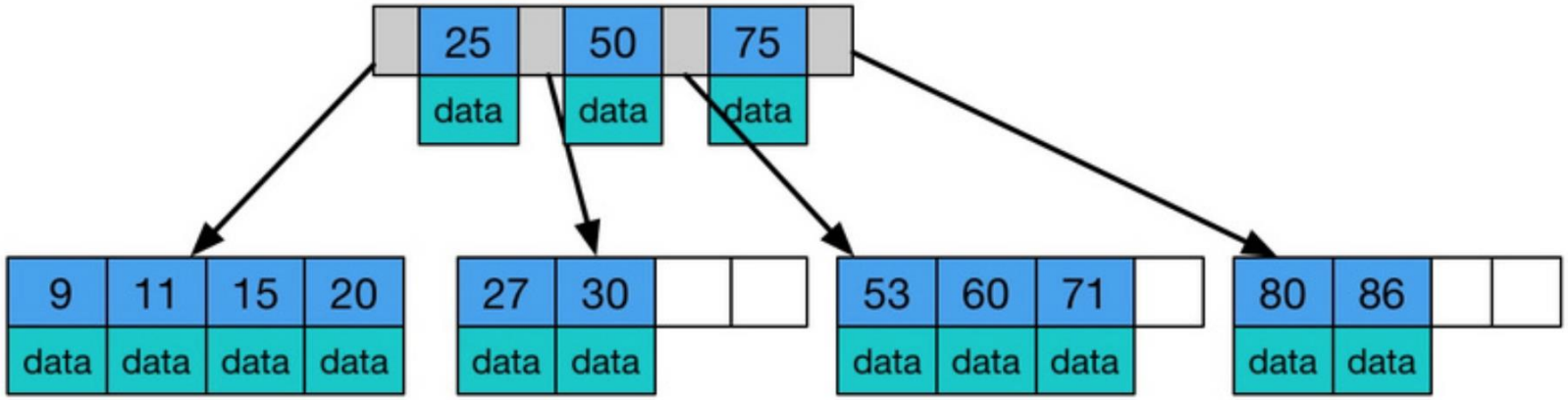

小于等
于60
多于60
>=100
100.90
90..80
80..70
70..60
60..50
50..40
40..30
30..20
<=10
100,rid
101,rid
102,rid
…
<max>
90,rid
91,rid
92,rid
93.rid
…
40,rid
41,rid
42,rid
43,rid
…
<min>
0,rid
1,rid
2,rid
…

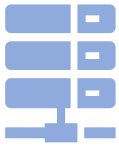


索引

■多级索引

- 为什么不用B树索引?
  - B树在每个节点（不仅是叶结点）中都存储数据
  - B树对于离根节点近的数据的查询会更快
  - 然而...





索引

■多级索引

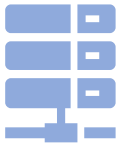
- 为什么不用B树索引?
  - B树将数据（行id）放到非叶节点中，导致每个节点能存放的索引项减少，树的层级更多，检索时需要更多磁盘IO
  - B+树将所有数据在叶节点中有序存放并构成一个链表
    - 范围查询的效率更高
    - 缓存命中率更高 (空间局部性原理)
  - B+树的更新维护代价更小

索引

■ 为基本表附加索引后，基本表的索引字段数据更新会引起索引的重构，造成数据库更新性能下降。因此索引的重构代价是考察索引性能的重要指标。

■B+树索引的更新

- B+树的插入删除不会引起过多I/O操作
- 插入过程 (见下页)
- 插入结果：B树仍然是平衡的，并保持了很好的性能



索引

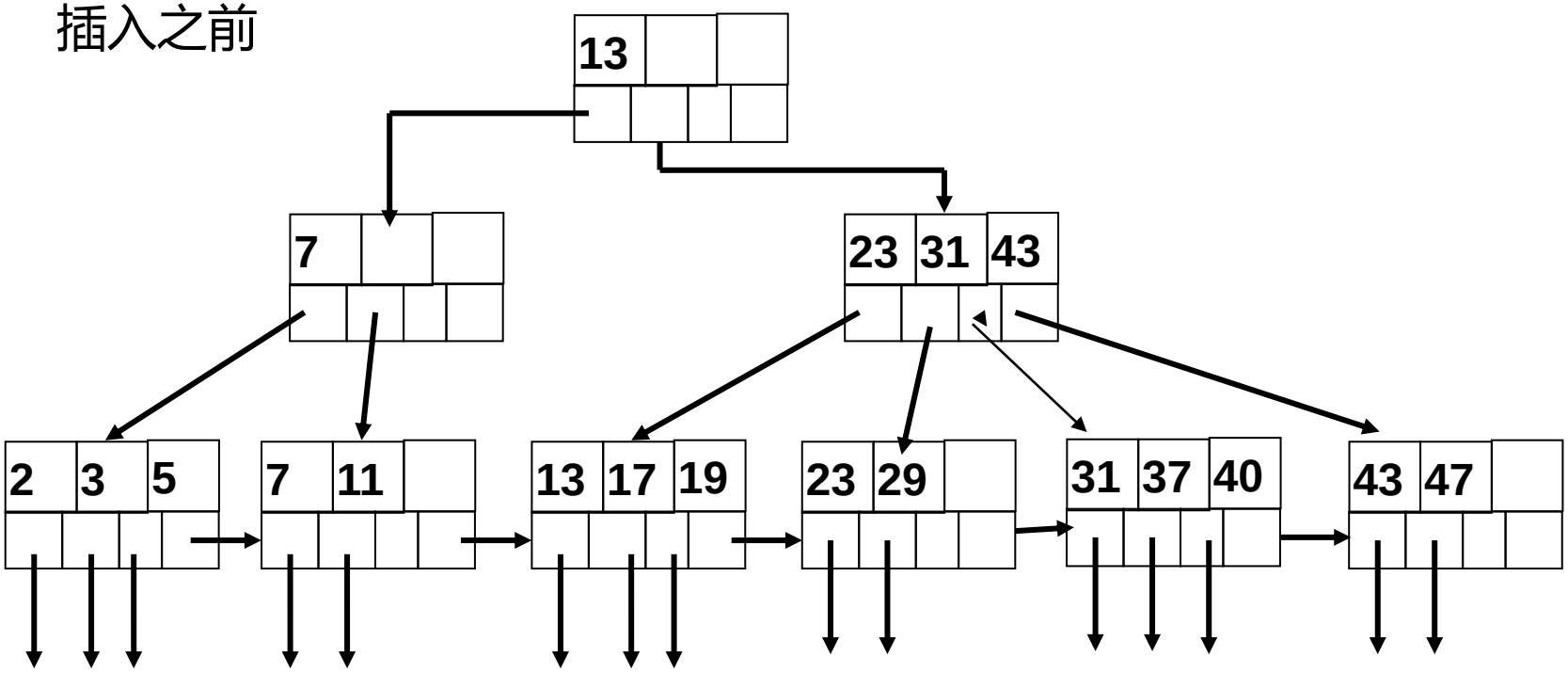
■B+树索引插入过程

- 设法在适当的叶结点中为新键找到空闲空间，如果有的话，就把键放在那里
- 如果在适当的叶结点中没有空间，就把该叶结点分裂成两个，并且把其中的键分到这两个新结点中，使每个新节点有一半或刚好超过一半的键。
- 某一层的结点分裂在其上一层看来，相当于是要在这一较高的层次上插入一个 新的键 - 指针对。因此，我们可以在这一较高层次上逆规地使用这个插入策略；如果有空间，则插入；如果没有，则分裂这个父结点且继续向树的高层推进。
- 例外的情况是，如果试图插入键到根结点中并且根结点没有空间，那么我们就分裂根结点成两个结点，且在更上一层创建一个新的结点。这个新的根结点有两个刚分裂成的结点作为它的子结点。

索引

■B+树索引插入过程

插入之前

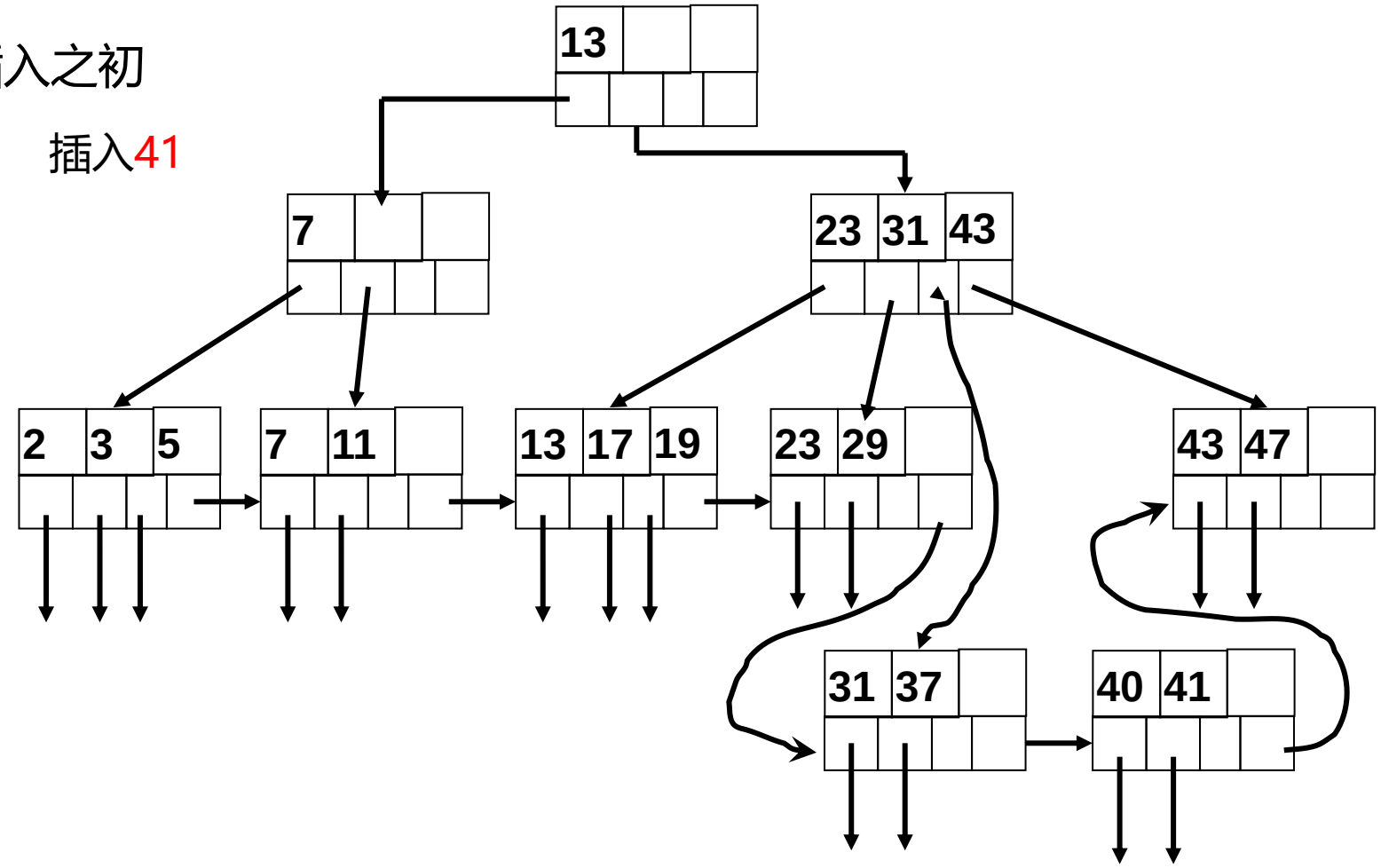


索引

■B+树索引插入过程

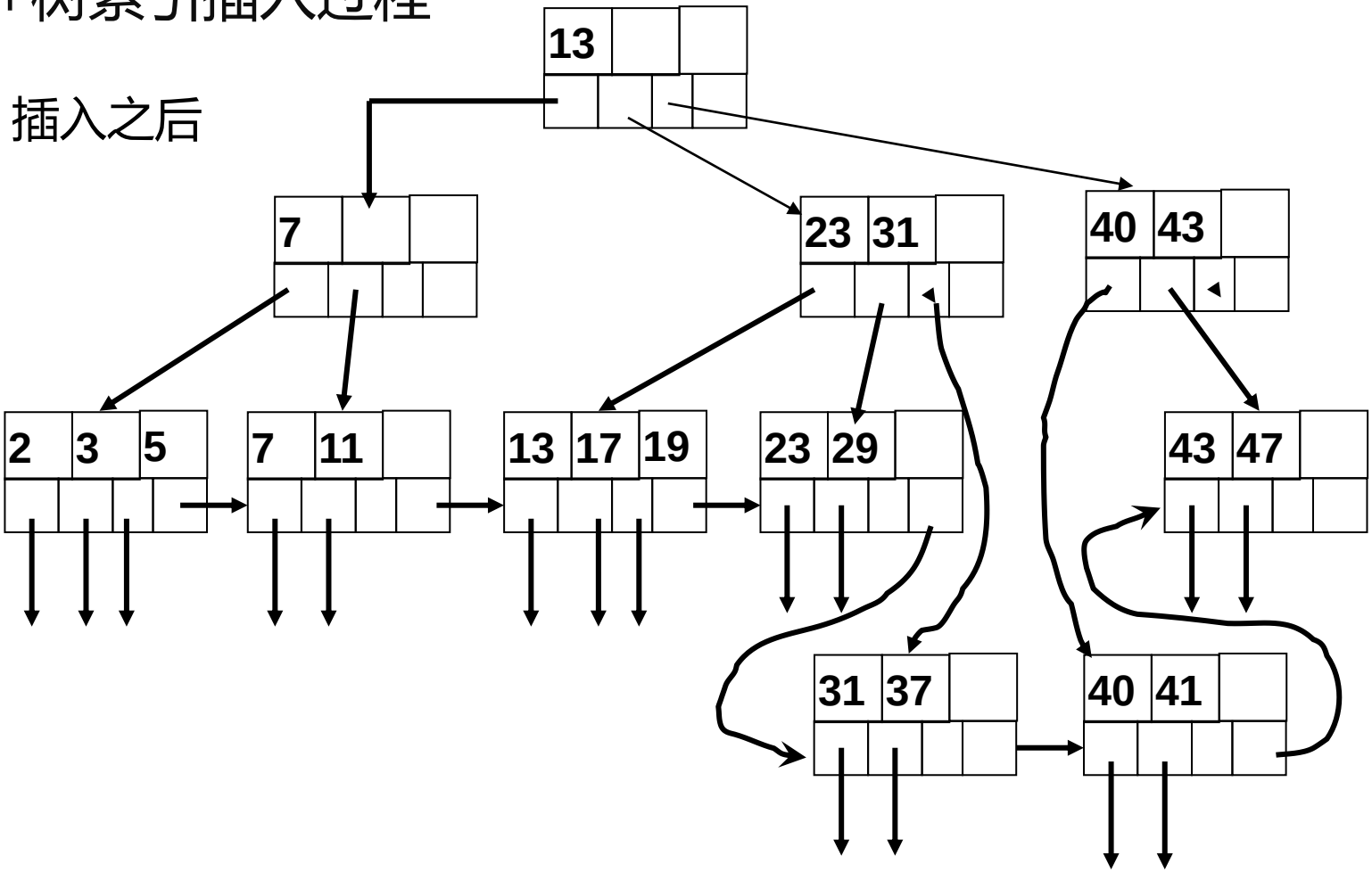
插入之初

插入41



索引

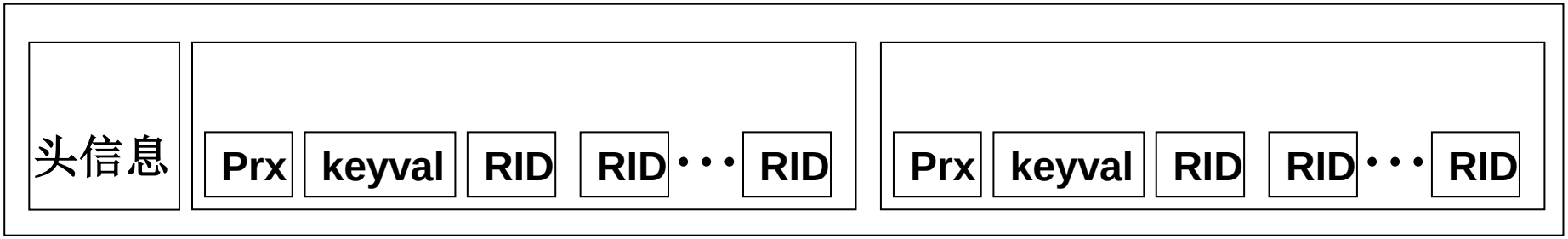
■B+树索引插入过程



索引

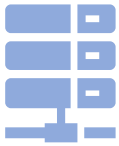
■B+树索引|重复键的处理

用一个链表存储重复的键值对应的记录行的RID值



在不同行有重复键值的叶节点的布局

Rowid：记录了行的物理地址

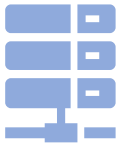


索引

■B+树索引的效率

- 一般B+树保持在3层，这意味着只需3次磁盘I/O即可获得数据的物理存储地址
- 若将B+树的根节点和中间节点存入缓存（这是完全可以的），则只需1次磁盘I/O就能读取数据

- 何时使用B+树索引
  - 大部分情况下B+树索引都能工作得很好
  - 当要查询的记录数占记录总数的百分比非常大的时候，不用索引将比用索引更快



索引

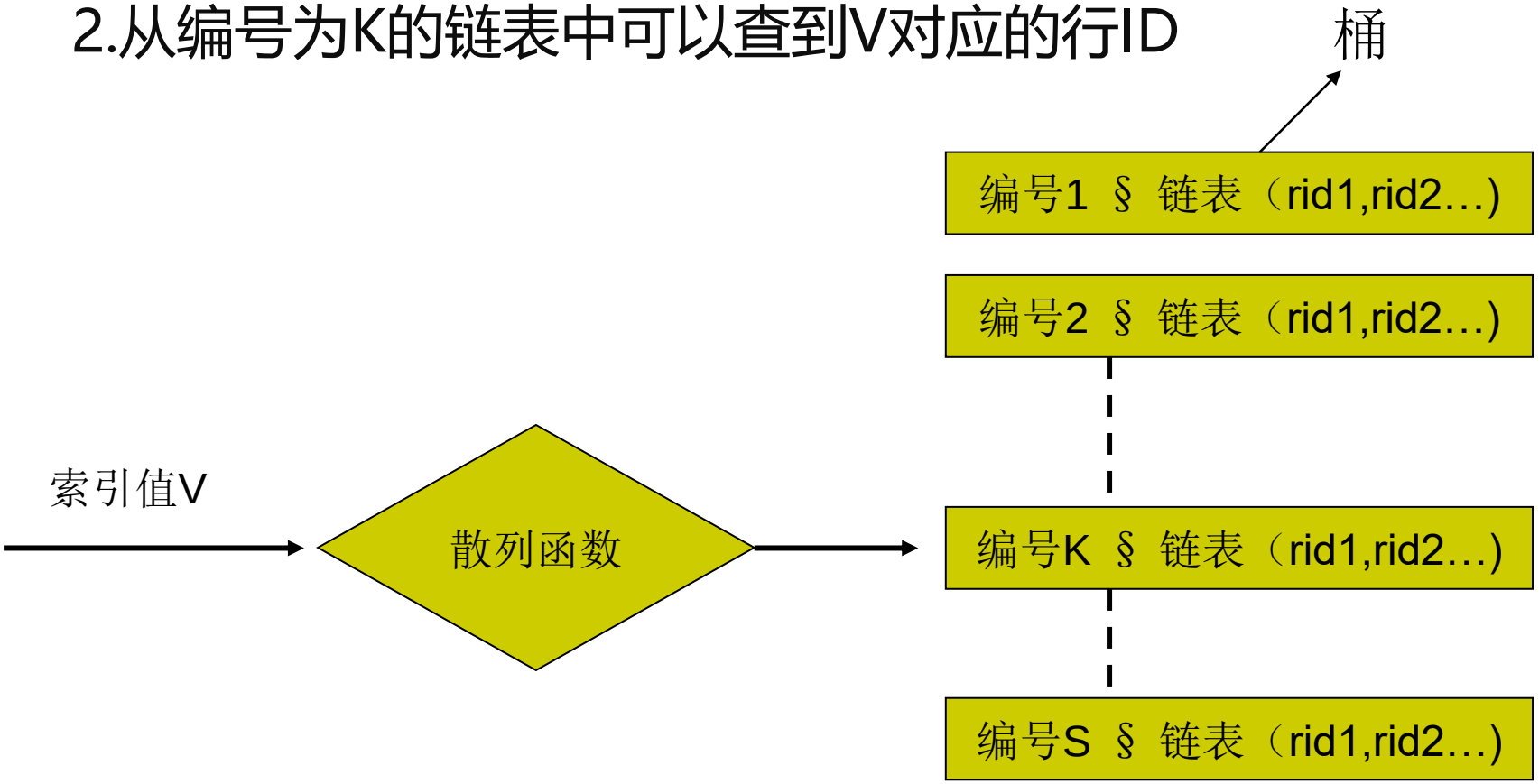
- 散列索引（Hash Index）
  - B+树索引需要3次左右磁盘IO才能查到数据记录
  - 散列索引只需一次磁盘IO就可以查到数据记录
  - 基本思想是：根据给定索引值，用一种算法将记录分散存储到多个“桶”中（一般一个桶就是一个数据块，块中内容用一次磁盘操作就可以读取到内存中）。当要查找记录时，用相同算法算出该记录所在的桶，读取整个桶的数据到内存中，然后在桶中顺序查找要找的纪录

索引

- 散列索引直观理解

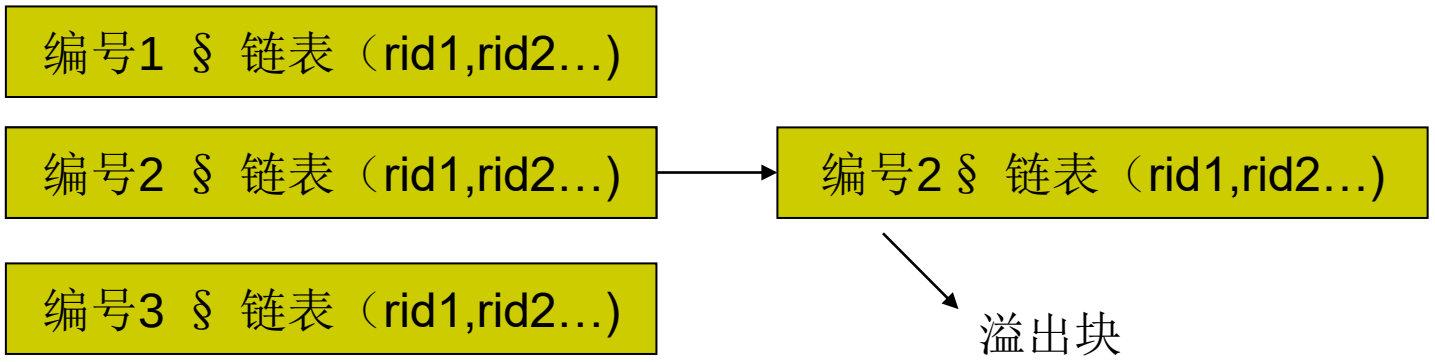
  1.索引值V按照散列函数计算出一个整数K

  2.从编号为K的链表中可以查到V对应的行ID

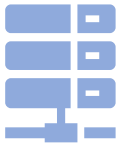


索引

- 散列索引溢出块
  - 如果桶的数量足够多，则每个桶通常占用一个磁盘页面（块）
  - 如果记录数很多，则会出现一个块中容纳不下新记录的情况，这时可以增加一个溢出块到桶的链上

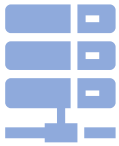


若溢出块太多，将大大增加磁盘I/O次数



索引

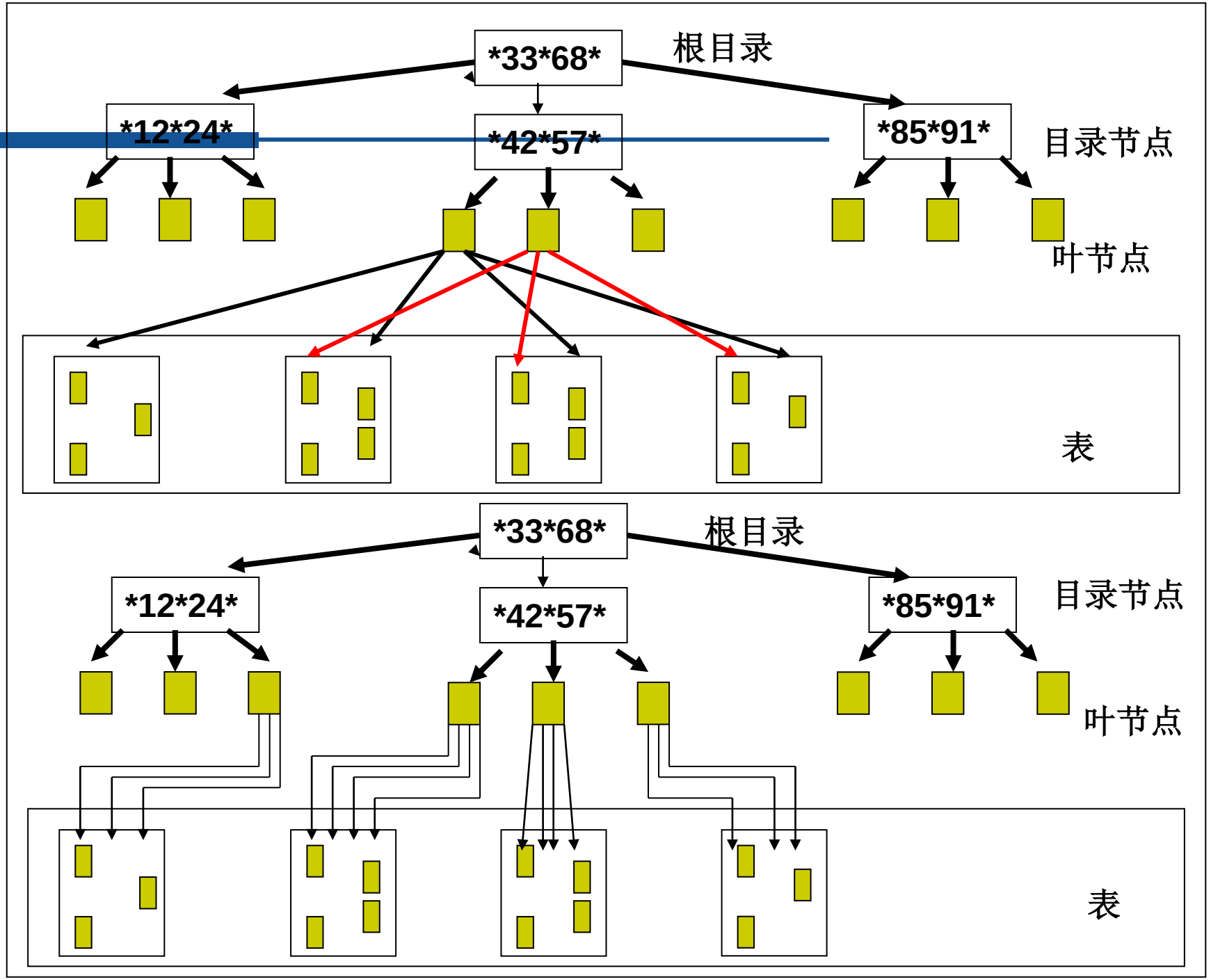
- 散列索引特点
  - 散列索引是CPU密集型的，B+树索引是I/O密集型的（I/O次数多于散列索引）
  - 散列索引在进行等值查找时速度很快
  - 散列索引无法用于范围查找
  - 不适合在重复值很多的列上建立哈希索引
  - 哈希索引重构代价很大，不适合在更新频繁的表中建立哈希索引

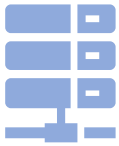


索引

- 聚簇索引 (Cluster Index)
  - 大多数关系表以堆组织表的形式存放
  - 建立聚簇索引后，数据在物理文件中的存放位置不再是无序的，而是根据索引中键值的逻辑顺序决定了表中相应行的物理顺序，即形成索引组织表

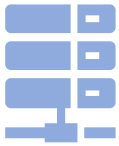
非聚簇索引（上）和聚簇索引（下）的比较





索引

- 聚簇索引特点
  - 物理顺序只有一个，因此一张表只能有一个聚簇索引
  - 在聚簇索引列上的查询速度比B+树索引快
  - 数据在物理上按顺序排在数据页上，重复值也排在一起，因而在使用包含范围检查(between、<、<=、>、>=)或使用group by或order by的查询时，可以大大提高查询速度
  - DML频繁的表中不要建立聚簇索引，因为会带来大量索引数据维护的开销
  - MySQL在表的主键上建立聚簇索引



索引

- 联合索引
  - 若要加速如下的查询，该怎么建立索引?
    - Select * from tb_a where a>1 and b=1
  - 通过在（a，b）字段上建立联合索引，可以获得比单独建立a字段索引和b字段索引更快的查找速度
  - 最左前缀原则:
    - 只有在查询条件中使用了联合索引的最左前缀（左边字段）时，该联合索引才回生效
    - 上例中若执行下述查询语句则联合索引不生效
      - Select * from tb_a where b=1

索引

■查看如下SQL语句：

Select * from Emp where

age / 2>20

■即使在age字段上建立了索引，但索引对上述SQL语句不起作用

■解决办法：

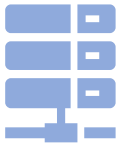
- 改写表达式，使表达式左边不包含计算式

Select * from Emp where age > 20 * 2

■ 另一个例子：

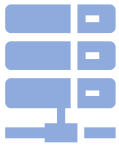
Select * from Emp where to_char(birth_day, 'YYYY-MM-DD') = '2022-11-10'，怎么改写?

需要用到函数索引：将to_char(birth_day, 'YYYY-MM-DD ') 定义为索引



索引

- 还有许多其它有趣的索引类型：
  - 位图索引； 反序索引；基于深度学习的模型...
- 索引会带来维护开销，只在经常被做为查询条件的字段上，或是确实需要提升查询性能的字段上建立索引
- 针对不同的数据情况选择合适的索引类型，考虑因素如：
  - 重复值占比
  - 列值是否被频繁更新
  - 是否范围查询或分组查询



索引

- 索引按照字段特性还可分为：

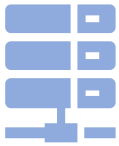
  主键索引、唯一索引和普通索引

- 主键索引

  - 建立在主键字段上的索引
  - 一张表最多只有一个主键索引
  - 索引列的值不允许有空值

- 唯一索引

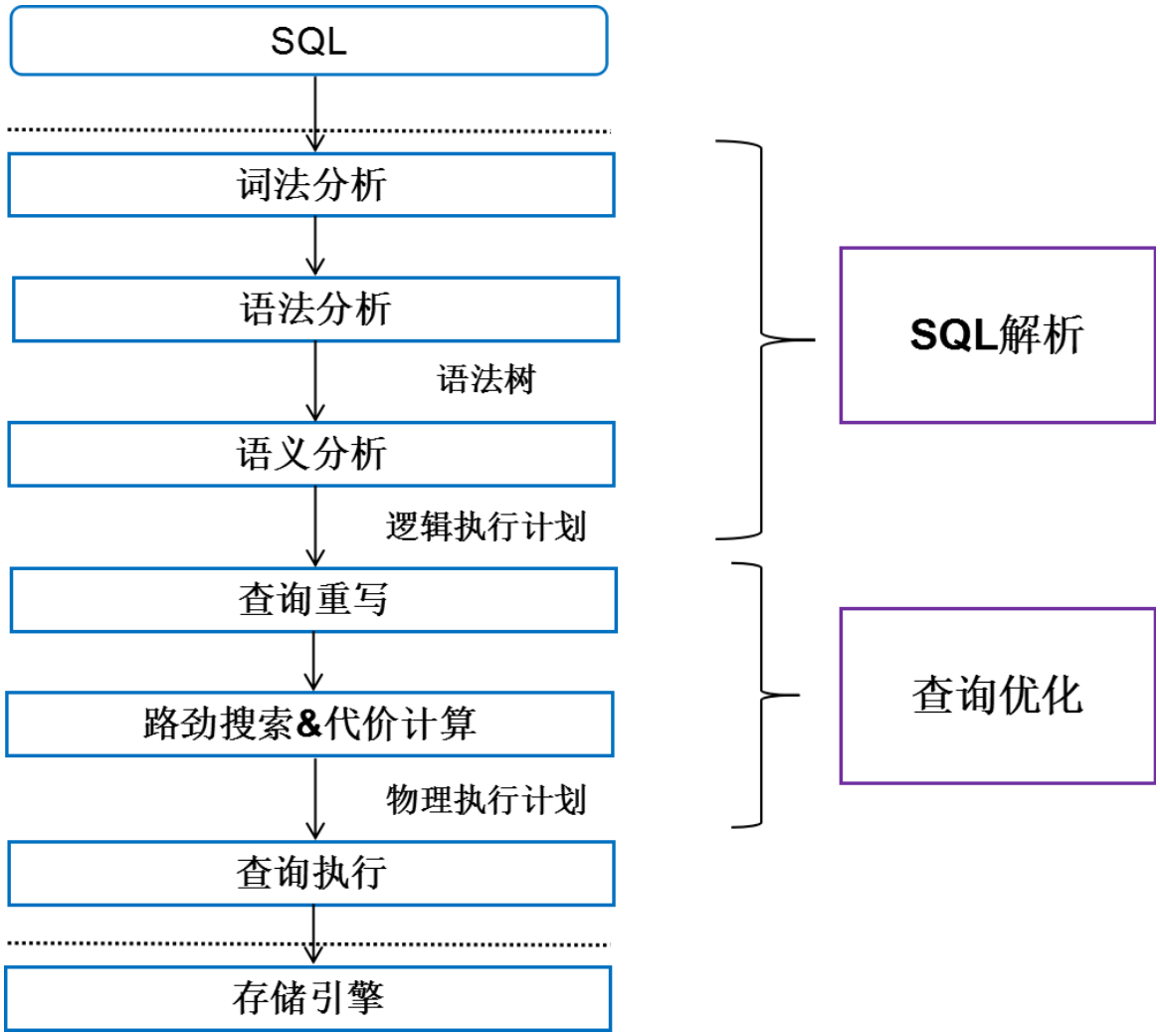
  - 建立在 UNIQUE 字段上的索引
  - 一张表可以有多个唯一索引
  - 索引列的值必须唯一，但是允许有空值

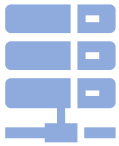


第五章 查询优化

查询优化

SQL语句执行过程





查询优化

- SQL语句不同的执行方案对性能影响巨大
  - 查询优化问题分析

例：求选修了课程C2的学生姓名

```
SELECT   Student.Sname
FROM       Student, SC
WHERE    Student.Sno=SC.Sno
      AND      SC.Cno='2';
```

此查询的IO和CPU处理代价是多少?

查询优化

- 查询优化问题分析

假设1：外存：

Student:1000条,SC:10000条, 选修2号课程:50条

假设2：内存：

一个内存块装元组:10个Student, 或100个SC, 内存中一次可以存放：5块Student元组, 1块SC元组和若干块连接结果元组

假设3：读写速度：20块/秒

查询优化

- 查询优化问题分析

$Q1 = \Pi_{S\ name}(\sigma_{Student.Sno=SC.Sno\ \wedge SC.Cno='2'}(Student \times SC))$

① Student×SC

读取总块数= 读Student表块数 + 读SC表遍数 *每遍块数

=1000/10+(1000/(10×5)) ×(10000/100)

=100+20×100=2100

读数据时间=2100/20=105秒

查询优化

■ 查询优化问题分析

中间结果大小 = 1000*10000 = 10^7 (1千万条元组)

写中间结果时间 = 10000000/10/20 = 50000秒

② б

读数据时间 = 50000秒

③ П

总时间 =105 + 50000 + 50000秒 = 100105秒

= 27.8小时

查询优化

- 查询优化问题分析

2. $Q2 = \Pi_{S\ name}(\sigma_{SC.Cno='2'}(Student \bowtie SC))$

① ⋈

读取总块数= 2100块

读数据时间=2100/20=105秒

中间结果大小=10000　(减少1000倍)

写中间结果时间=10000/10/20=50秒

② б

读数据时间=50秒

③ Π

总时间 = 105 + 50 + 50秒 = 205秒=3.4分

查询优化

- 查询优化问题分析

3. $Q2 = \Pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$

① б

读SC表总块数= 10000/100=100块

读数据时间=100/20=5秒

中间结果大小=50条 不必写入外存

② ⋈

读Student表总块数= 1000/10=100块

读数据时间=100/20=5秒

③ Π

总时间 = 5 + 5秒 = 10秒

查询优化

- 查询优化问题分析

4. $Q2 = \Pi_{S\,name}(Student \bowtie б_{SC.Cno='2'}(SC))$

假设SC表在Cno上有索引，Student表在Sno上有索引

① б

读SC表索引=

读SC表总块数= 50/100<1块

读数据时间

中间结果大小=50条 不必写入外存

查询优化

- 查询优化问题分析

② ⋈

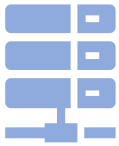
读Student表索引|=

读Student表总块数= 50/10=5块

读数据时间

③ Π

总时间<10秒



查询优化

■ 查询优化的一般准则

- ●选择运算应尽可能先做

  目的：减小中间关系

- ●在执行连接操作前对关系适当进行预处理

  按连接属性排序

  在连接属性上建立索引

- ●投影运算和选择运算同时做

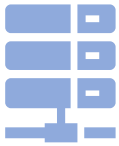
  目的：避免重复扫描关系

- ●将投影运算与其前面或后面的双目运算结合

  目的：减少扫描关系的遍数

- ●提取公共子表达式

- ●...



查询优化

- 查询优化的一般过程
  1. 将查询转换成某种内部表示，通常是语法树
  2. 根据一定的等价变换规则把语法树转换成标准（优化）形式 （代数优化）
  3. 选择低层的操作算法（物理优化）
     - 对于语法树中的每一个操作计算各种执行算法的执行代价
     - 选择代价小的执行算法
  4. 生成查询计划(查询执行方案)

查询优化

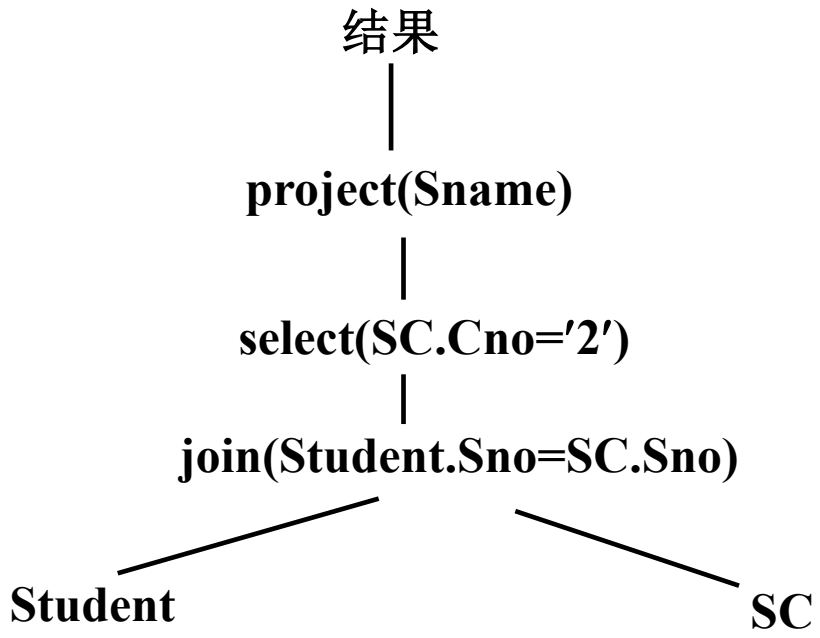
■ 查询优化

例：求选修了课程C2的学生姓名

```
SELECT  Student.Sname
FROM    Student, SC
WHERE   Student.Sno=SC.Sno
AND     SC.Cno='2';
```

（1）把查询转换成某种内部表示

语法树



查询优化

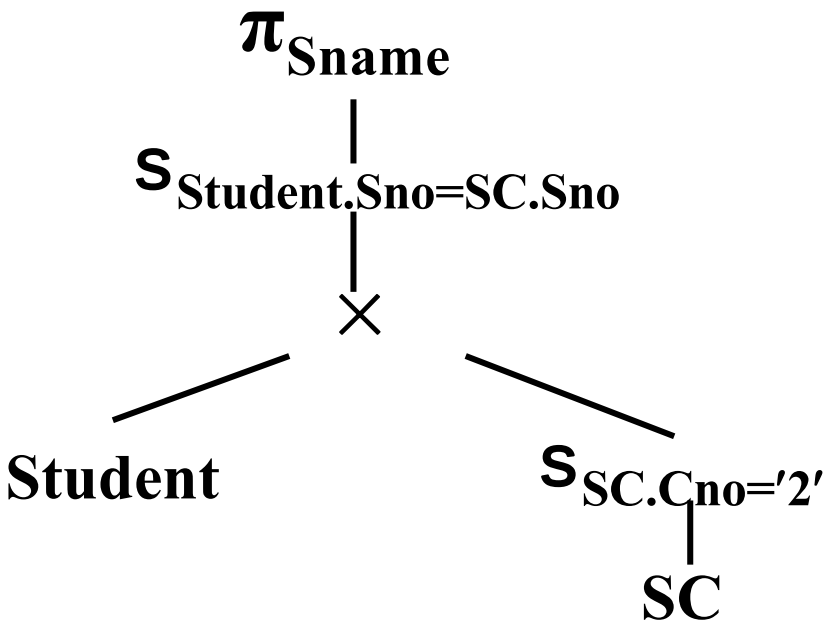
■ 查询优化

例：求选修了课程C2的学生姓名

```
SELECT  Student.Sname
FROM    Student, SC
WHERE   Student.Sno=SC.Sno
AND     SC.Cno='2';
```

（2）代数优化

利用优化算法把语法树转换成标准（优化）形式



查询优化

■ 查询优化

（3）物理优化，选择低层的存取路径

- 优化器查找数据字典获得当前数据库状态信息

•选择字段上是否有索引

•连接的两个表是否有序

•连接字段上是否有索引

–然后根据一定的优化规则选择存取路径

如本例中若SC表上建有Cno的索引，则应该利用这个索引，而不必顺序扫描SC表。

查询优化

■ 查询优化

(4) 生成查询计划，选择代价最小的

-在作连接运算时，若两个表(设为R1，R2)均无序，连接属性上也没有索引，则可以有下面几种查询计划：

- 对两个表作排序预处理
- 对R1在连接属性上建索引
- 对R2在连接属性上建索引
- 在R1，R2的连接属性上均建索引

-对不同的查询计划计算代价，选择代价最小的一个。

作业

- 什么是索引?
- 简述B树索引、Hash索引及聚簇索引的实现原理和优缺点。
- 提交时间：下次上课之前