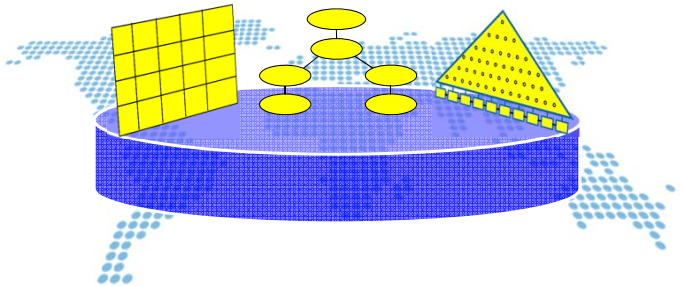
数据库系统

数据存储与访问路径(3)

陈世敏

（中科院计算所）



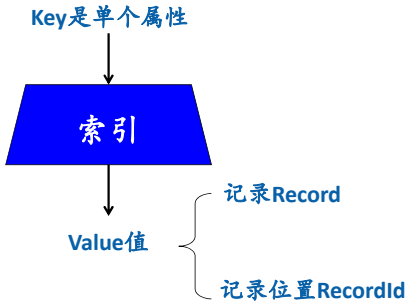
上节内容

- 索引的概念
- 树结构索引
  - B+-Trees
  - 索引访问代价
  - 内存优化的B+-Tree
- 哈希索引
  - Hash function
  - Chained hashing
  - Extendible hashing
- 其他索引
  - ISAM
  - 位图索引
  - 倒排索引

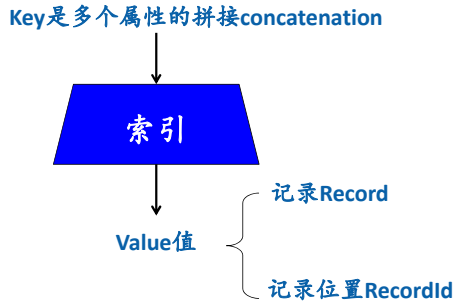
Outline

- 多维索引
  - 概念
  - 多维散列索引
  - 多维树结构索引
- 物理数据库设计

单个列上的索引

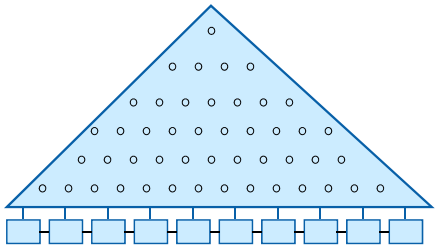


多个列上的索引 (!= 多维索引)



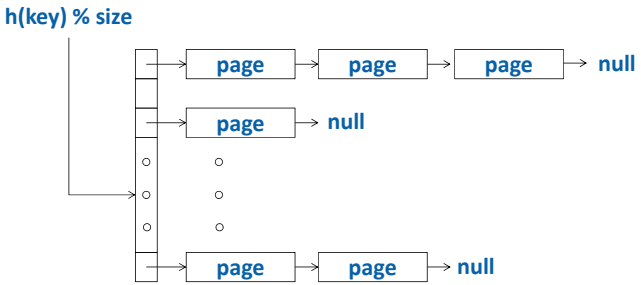
- 例如：（姓名，年龄）
- 可以使用上节讲到的各种索引
- 把多个属性的整体看作一个key，按顺序比较每个部分

在B+-Tree中，key可以是多列拼接组成



- 例如：key = (姓名，年龄)
  - 先比较第一部分，完全相同时再比较第二部分
- 从本质上看，这仍然是单维的索引
  - 不同列的重要性不一样

哈希表中的key也可以是多列拼接组成



- 例如：key = (姓名，年龄)
- 从本质上看，这仍然是单维的索引

单维索引的问题

- key = (姓名，年龄)

- 如果只有年龄上的条件，怎么处理？
  - 年龄=70，年龄>15
  - 哈希表不能支持，B+-Tree很难支持

单维索引vs.多维索引

- 从本质上讲，上述索引是单维的
  - B^+-tree是按照一个顺序排列的
  - Hash table是按照一种方法来计算桶的下标的
  - 不同维度的重要性是不同的
- 多维索引
  - 数据被认为存在于二维或更高维的空间中
  - 每个维度具有相似的重要性

多维索引应用场合

- 地理信息系统（Geographic Information System, GIS）
  - 通常是二维空间
  - 包含房屋，道路，桥梁，湖泊等
- 集成电路设计
  - 每层电路是一个二维空间
  - 多层组成三维空间
- 高维空间
  - 可以把文本的每个单词作为一个维度
  - 那么每个文本对应于高维空间的一个点
  - 我们这里介绍的多维索引主要适用于2维、3维等低维空间

GIS中的典型查询类型

- 部分匹配
  - 指定一维或多维上的值，查找匹配的对象
- 范围查询
  - 给出一维或多维上的范围，查找匹配的对象
- 最近邻查询
  - 查找与给定点最近的点
- 下面，我们考虑不同索引结构如何支持这些查询

传统索引表达多维空间

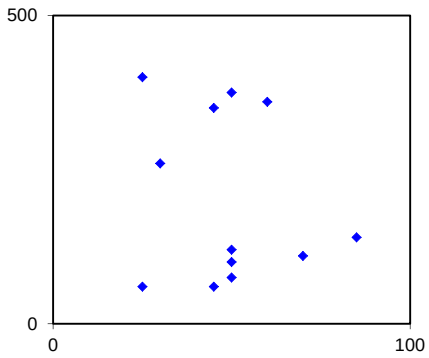
- 例如：用B^+-Tree对(x,y)坐标进行索引
  - （x, y）=（经度，纬度）
- 有什么问题？
  - 部分匹配
    - 指定一维或多维上的值，查找匹配的对象
    - 如果指定y上的值，而x不指定，那么会怎么样？
  - 范围查询
    - 给出一维或多维上的范围，查找匹配的对象
    - 如果仅给定y上的范围，那么会怎么样？
  - 最近邻查询
    - 查找与给定点最近的点？

Outline

- 多维索引
  - 概念
  - 多维散列索引
    - 网格文件（Grid File）
    - 分段散列（Partitioned Hashing）
  - 多维树结构索引
- 物理数据库设计

举例

- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 如何建立索引？

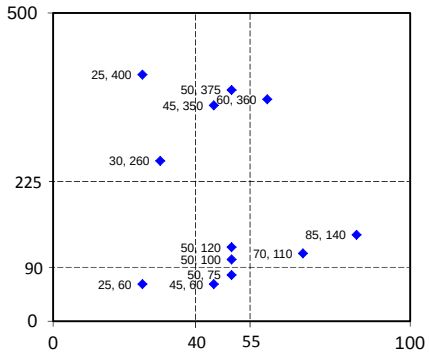


网格文件（Grid File）

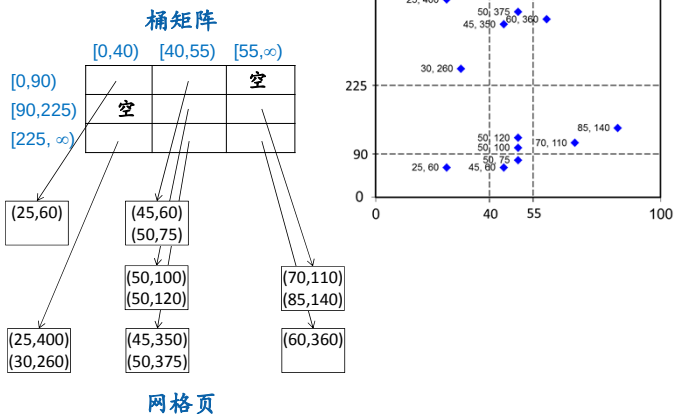
- 基本思想
  - 把整个空间切分成一个个长方形的网格
  - 每个网格➔一个桶
  - 用一个数据结构记录桶的位置

网格文件举例

- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 划分x和y
  - 如图所示
  - 每个网格包含基本相似数量的记录



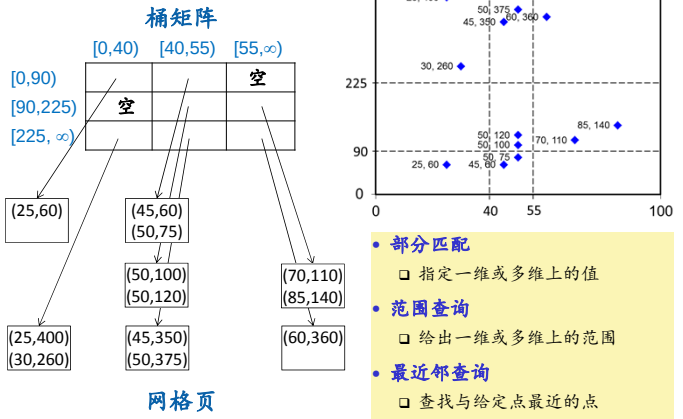
网格文件举例



网格文件（Grid File）

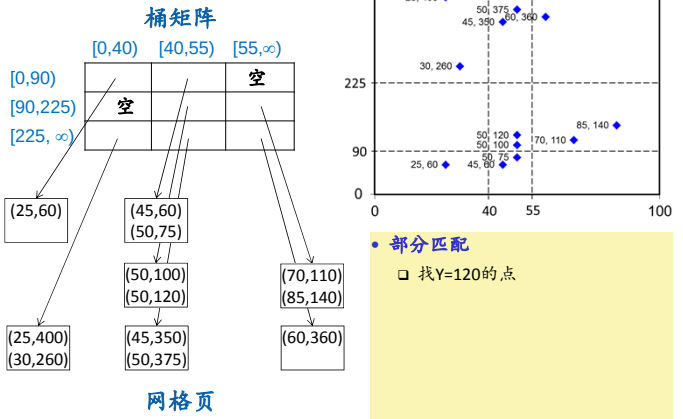
- 网格文件没有用hash function
- 网格：多维空间中的一个矩形空间
- 每维属性
  - 从小到大顺序排列，切分为多个区间
    - 第1维：有S_1个区间；
    - 第2维：有S_2个区间；
    - 。。。。。。。；
    - 第K维：有S_K个区间
- 网格文件由下面两个数据结构组成
  - 桶矩阵：K维矩阵，$S_1 X S_2 X \ldots X S_k$个元素
    - 每个矩阵元素对应一个网格
  - 每个矩阵元素记录网格的存储位置
    - 该数据页包含这个网格中所有的索引项
    - 如果数据页不够大，那么采用溢出链

如何查询网格文件？



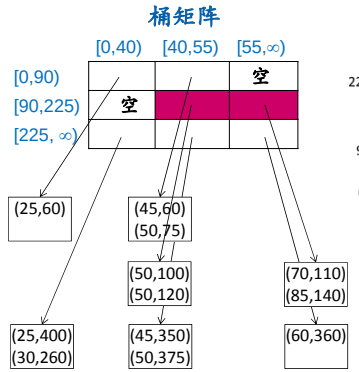
- **部分匹配**
  - 指定一维或多维上的值
- **范围查询**
  - 给出一维或多维上的范围
- **最近邻查询**
  - 查找与给定点最近的点

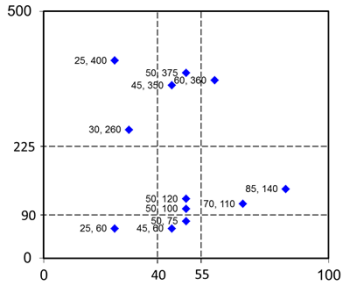
如何查询网格文件？



- **部分匹配**
  - 找Y=120的点

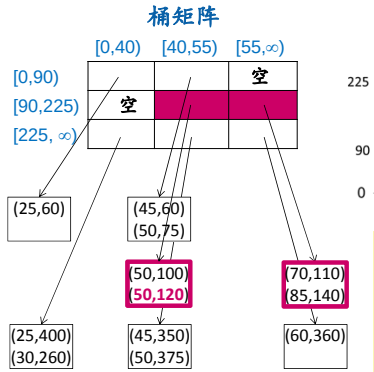
如何查询网格文件?

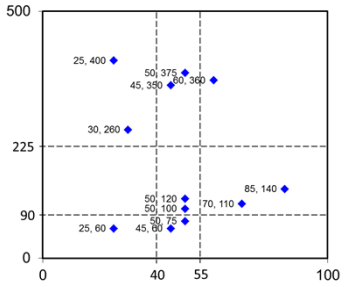




- **部分匹配**
  - 找Y=120的点
  - 先找到相关的桶

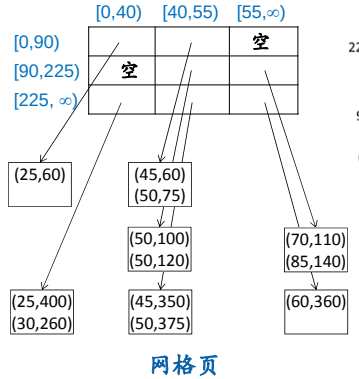
如何查询网格文件?

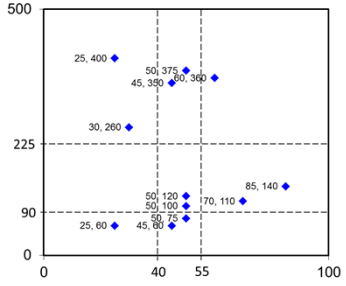




- **部分匹配**
  - 找Y=120的点
  - 先找到相关的桶
  - 然后在桶网格页内搜索

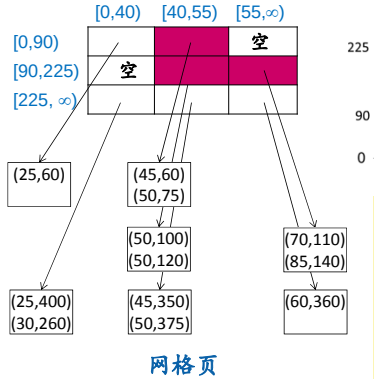
如何查询网格文件?

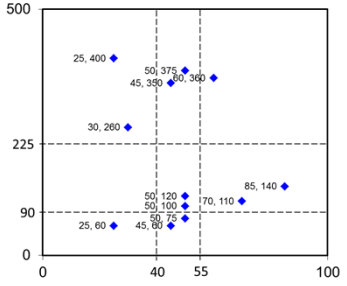




- **范围查询**
  - 50<=x<=60 and 70<=y<=100

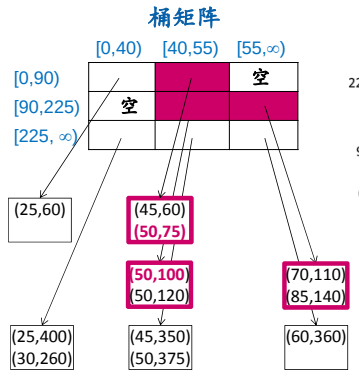
如何查询网格文件?

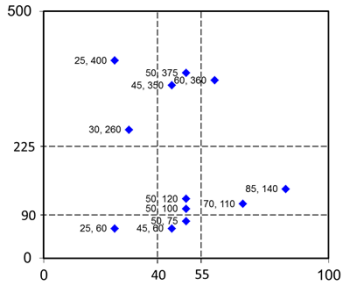




- **范围查询**
  - 50<=x<=60 and 70<=y<=100
  - 找到相关的网格

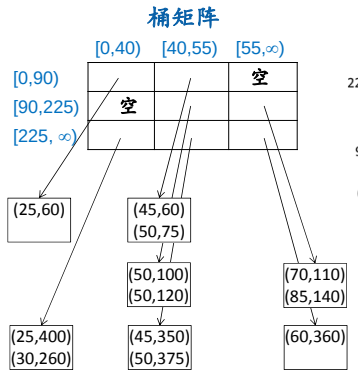
如何查询网格文件？

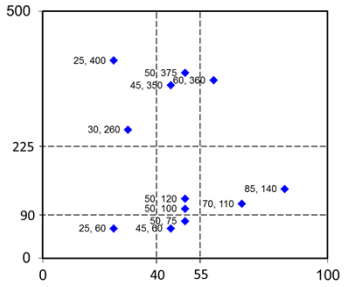




- **范围查询**
  - 50<=x<=60 and 70<=y<=100
  - 找到相关的网格
  - 然后读取网格页，进一步搜索

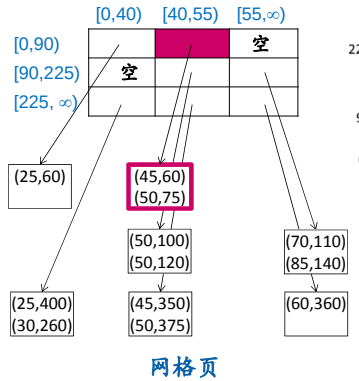
如何查询网格文件？

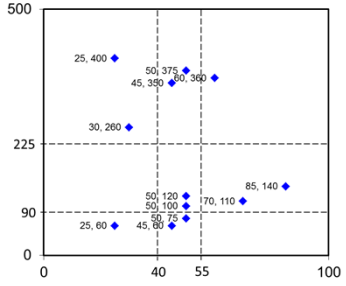




- **最近邻查询**
  - (50,50)的最近邻？

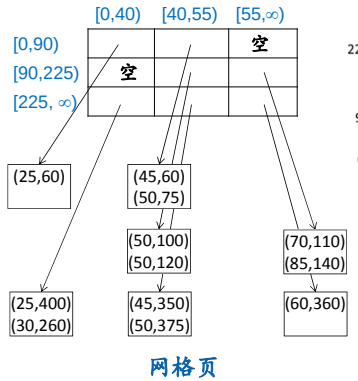
如何查询网格文件？

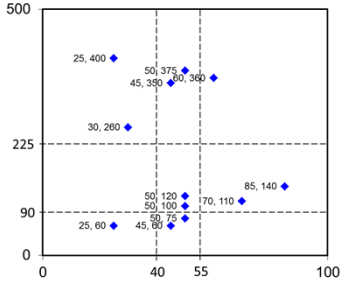




- **最近邻查询**
  - (50,50)的最近邻？
  - 先搜索所在网格
    - 确定当前最近邻距离
    - 与（45,60）距离11.2
  - 再搜索周围的网格？
    - 距离更远，可以停止了

如何查询网格文件？

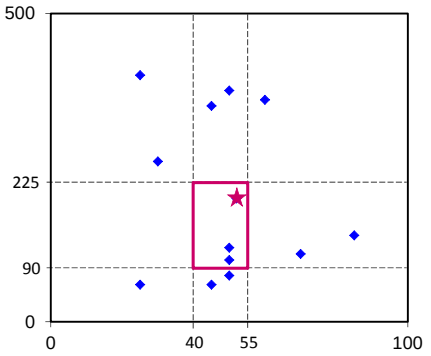




- **根据已知条件，查桶矩阵**
  - 找到相关的网格
- **然后读取网格页，进一步搜索**

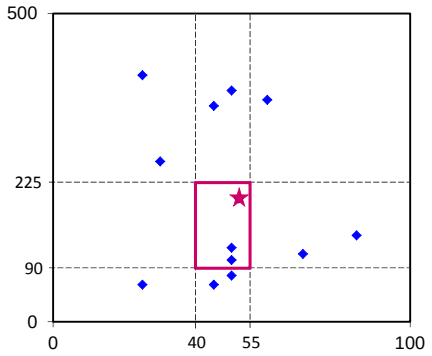
网格文件的插入

- 插入(52,200)
- 假设每个页只可以放两条记录，那么中心网格一个页放不下了
- 解决方案?



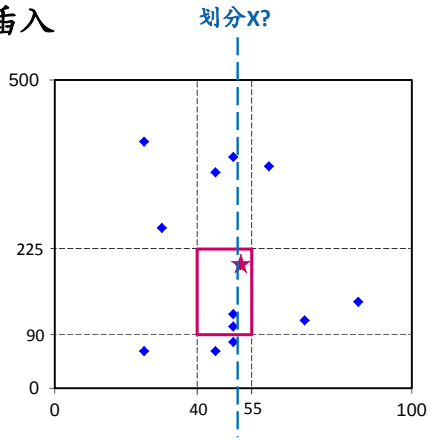
网格文件的插入

- 插入(52,200)
- 假设每个页只可以放两条记录，那么中心网格一个页放不下了
- 方案1：溢出页
  - 分配一个溢出页
  - 可以存放新记录



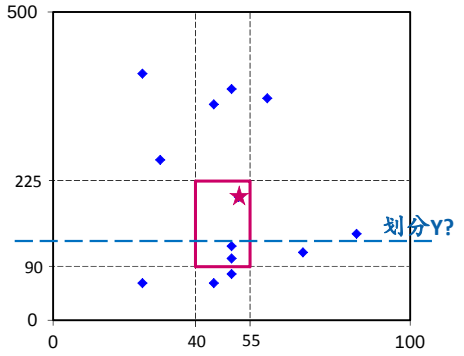
网格文件的插入

- 插入(52,200)
- 假设每个页只可以放两条记录，那么中心网格一个页放不下了
- 方案2：划分网格
  - 划分X?



网格文件的插入

- 插入(52,200)
- 假设每个页只可以放两条记录，那么中心网格一个页放不下了
- 方案2：划分网格
  - 划分X?
  - 划分Y?
- 目标：改变的网格中数据分布尽可能均匀



网格文件

- 网格文件算法的关键难点是需要确定
  - 给定一组数据，如何计算网格？
  - 插入数据时，如何划分网格？
- 我们这里不仔细介绍
  - 思路：是否可以使用统计直方图？

Outline

- 多维索引
  - 概念
  - 多维散列索引
    - 网格文件（Grid File）
    - 分段散列（Partitioned Hashing）
  - 多维树结构索引
- 物理数据库设计

分段散列（Partitioned Hashing）

- 基本思想
  - 思考计算哈希函数、映射到桶这个过程
  - 使各个维度与桶能够分别对应？
    - 这样可以支持所需要的多维查找

分段散列（Partitioned Hashing）

- 对于多维
  - 第1维，属性值key_1，计算哈希值$h_1=hash_1(key_1)\ \%\ Size_1$
  - 第2维，属性值key_2，计算哈希值$h_2=hash_2(key_2)\ \%\ Size_2$
  - …
  - 第n维，属性值key_n，计算哈希值$h_k=hash_n(key_n)\ \%\ Size_n$
- 注意：Size都是2的幂，即取h(key)的一些bit
- 最终的哈希值为所有上述多个哈希值的拼接

h_1	h_2	…	h_n

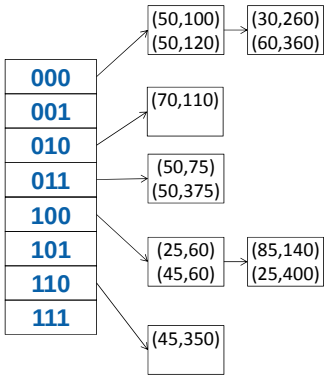
分段散列举例

- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- hx= x % 2　　取x的1位
  hy= y % 4　　取y的2位
- h= (hx << 2) | hy

计算

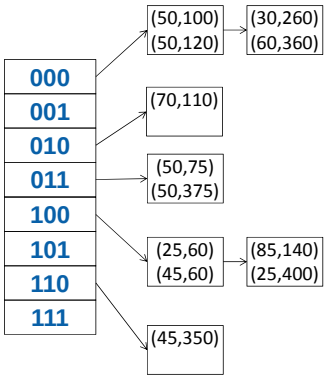
X	Y	hx=X%2	hy=Y%4	hx<<2|hy
25	60	1	0	4
45	60	1	0	4
50	75	0	3	3
50	100	0	0	0
50	120	0	0	0
70	110	0	2	2
85	140	1	0	4
30	260	0	0	0
25	400	1	0	4
45	350	1	2	6
50	375	0	3	3
60	360	0	0	0

分段散列举例



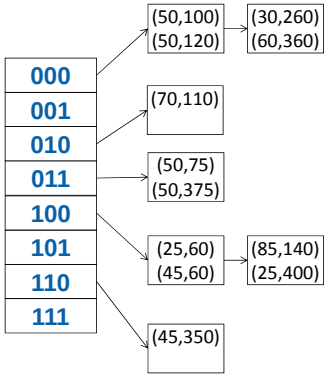
X	Y	hx=X%2	hy=Y%4	hx<<2|hy
25	60	1	0	4
45	60	1	0	4
50	75	0	3	3
50	100	0	0	0
50	120	0	0	0
70	110	0	2	2
85	140	1	0	4
30	260	0	0	0
25	400	1	0	4
45	350	1	2	6
50	375	0	3	3
60	360	0	0	0

如何查询分段散列？



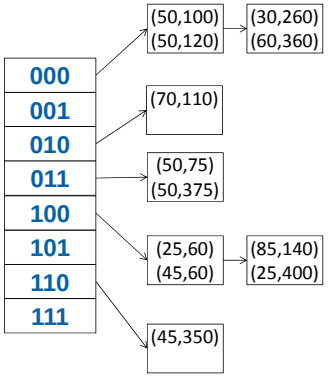
- **部分匹配**
  - 指定一维或多维上的值
- **范围查询**
  - 给出一维或多维上的范围
- **最近邻查询**
  - 查找与给定点最近的点

如何查询分段散列？



- **部分匹配**
  - 指定一维或多维上的值
- **例如：给定了Y=100**
  - 那么可以计算hy=00
  - 而hx可能为0或1
  - 那么就可以在000，100两个桶里找
- **给定部分维，可以穷举其它维，选择相应的桶**

如何查询分段散列？



- **范围查询？**
  - 给出一维或多维上的范围
- **最近邻查询？**
  - 查找与给定点最近的点
- **无法有效支持**
  - **哈希本身不能支持不等的比较**

Grid File vs. Partitioned Hashing

- 查询
  - Partitioned Hashing 不支持范围查询和最近邻查询
- 数据结构
  - Partitioned Hashing很容易实现均匀地分布数据到桶中
  - Grid File则比较困难，尤其是当维数很高时，会出现大量的网格为空的情况

Outline

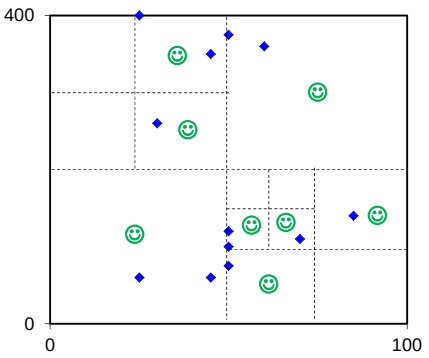
- 多维索引
  - 概念
  - 多维散列索引
  - 多维树结构索引
    - 四叉树（Quad Tree）
    - R树（R-Tree）
- 物理数据库设计

四叉树（Quad Tree）

- 以二维为例
  - 每个树结点代表二维空间中的一个长方形
  - 内部结点等分为4个子长方形，对应4个孩子结点
  - 叶子节点？
    - 这个节点内部包含的记录点可以存放在一个页中
  - 内部节点？
    - 如果放不下，那么这个结点是内部结点
    - 需要进一步分成4个孩子节点

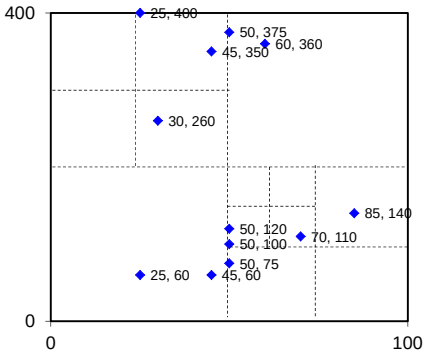
四叉树举例

- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 假设Page可以放最多2条记录
- 注意：一个长方形包括其左边和下边，而不包括其右边和上边

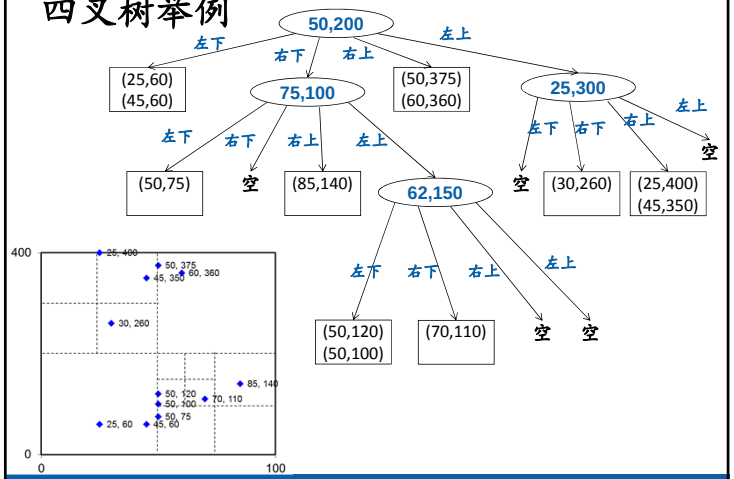


四叉树举例

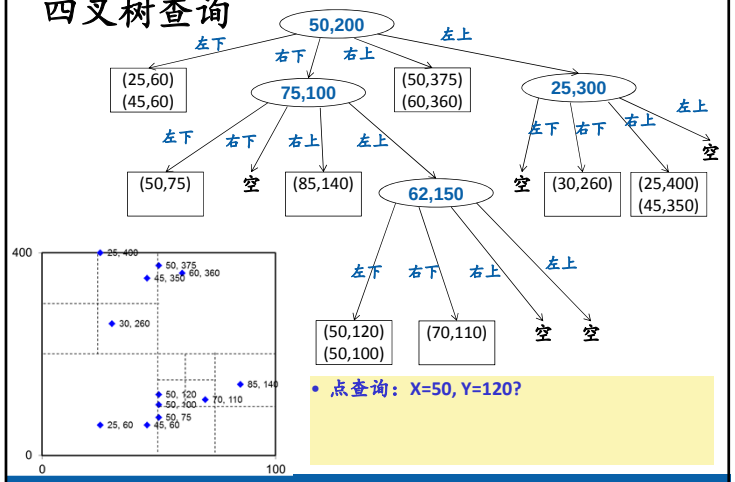
- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 假设Page可以放最多2条记录
- 注意：一个正方形包括其左边和下边，而不包括其右边和上边



四叉树举例

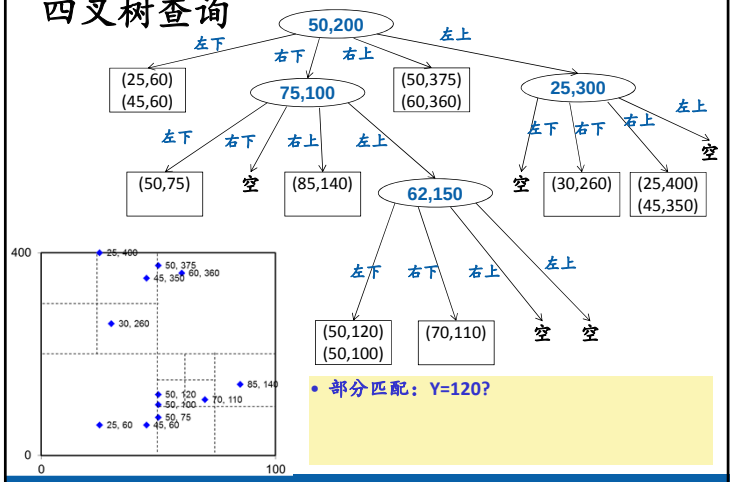


四叉树查询



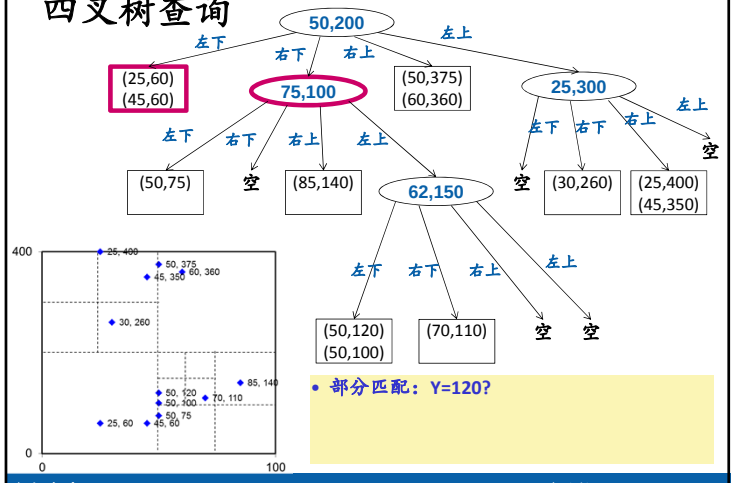
- 点查询：X=50, Y=120?

四叉树查询



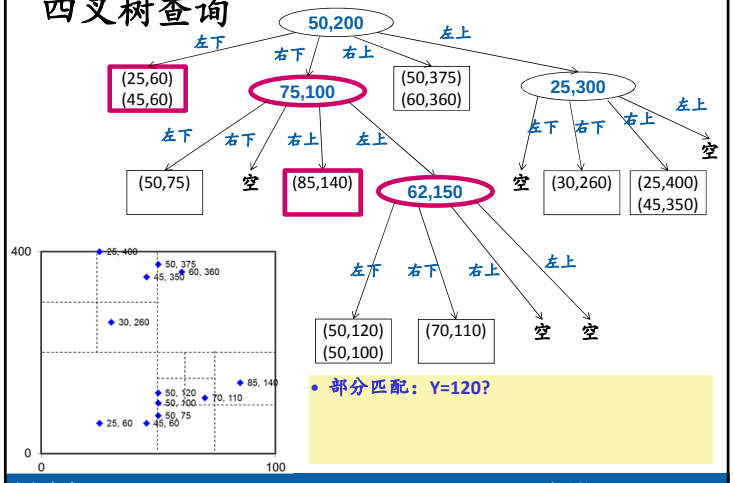
- 部分匹配：Y=120?

四叉树查询



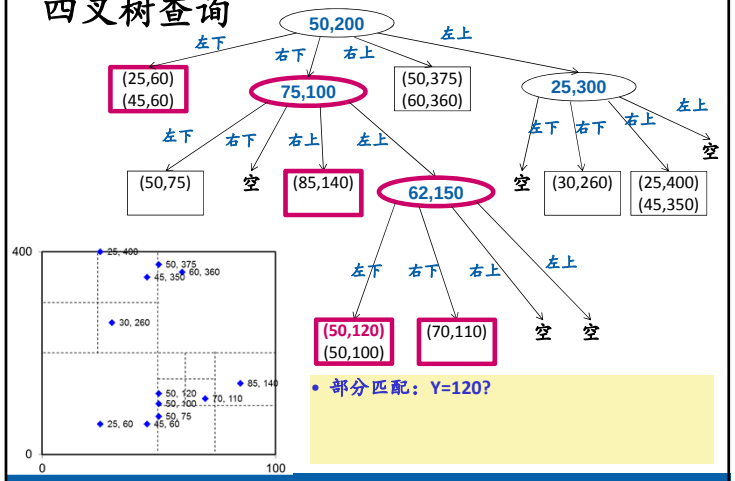
- 部分匹配：Y=120?

四叉树查询



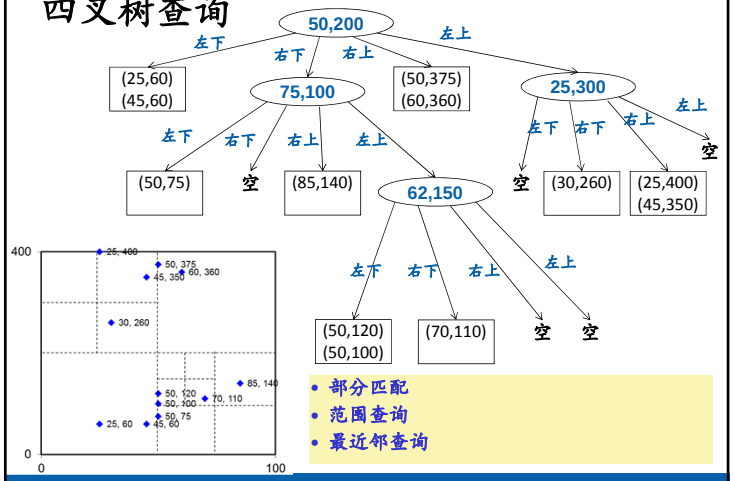
- 部分匹配：Y=120?

四叉树查询



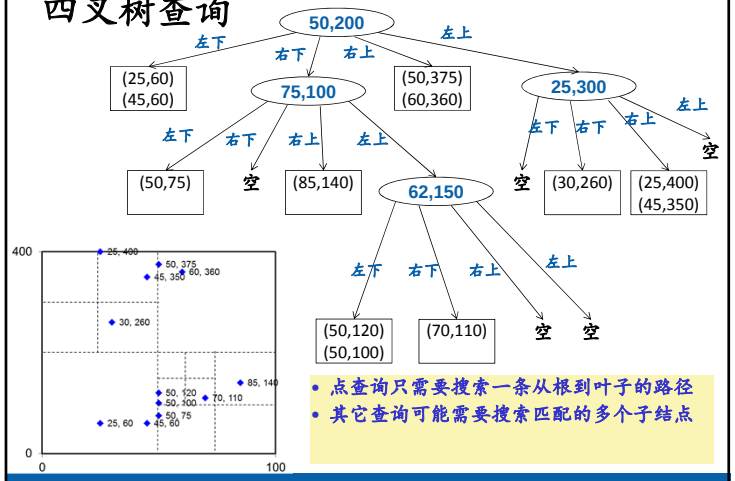
- 部分匹配：Y=120?

四叉树查询



- 部分匹配
- 范围查询
- 最近邻查询

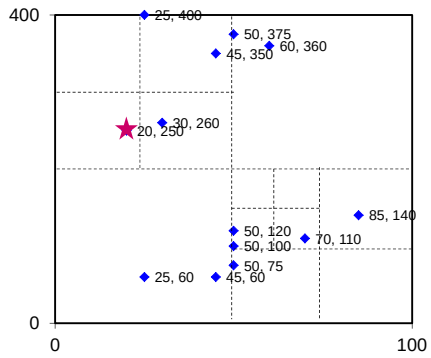
四叉树查询



- 点查询只需要搜索一条从根到叶子的路径
- 其它查询可能需要搜索匹配的多个子结点

四叉树插入 insert(20,250)

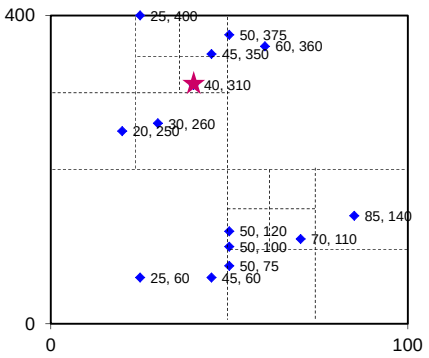
- 找到叶结点
- 如果不满，直接插入



四叉树插入

insert(40,310)

- 找到叶结点
- 如果不满，直接插入
- 如果已满，那么分裂为4部分



k维四叉树

- k维
  - 每个树结点代表k维空间中的一个长方形
  - 内部结点长方形等分为2^k个子正方形，对应2^k个孩子结点

Outline

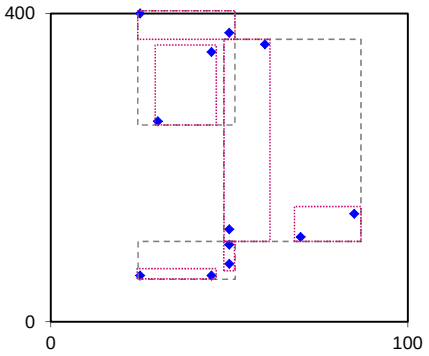
- 多维索引
  - 概念
  - 多维散列索引
  - 多维树结构索引
    - 四叉树（Quad Tree）
    - R树（R-Tree）
- 物理数据库设计

R树(R-Tree)

- 每个树结点代表一个区域
  - 称作MBR（Minimum Bounding Rectangle）
  - 是包含子树中所有对象的最小的外接矩形
- 两个树结点的MBR可能有重叠的区域
  - 希望使重叠的区域很小
  - 而B^+-Tree的每个结点代表一个区间，区间之间不重叠（当没有重复key时）

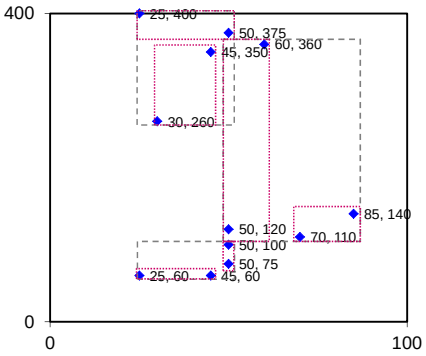
R-Tree举例

- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 假设Page可以放最多2条记录，或者3个MBR孩子结点

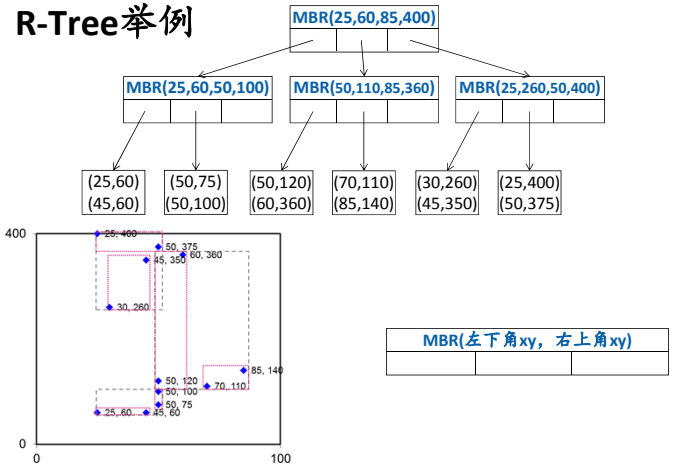


R-Tree举例

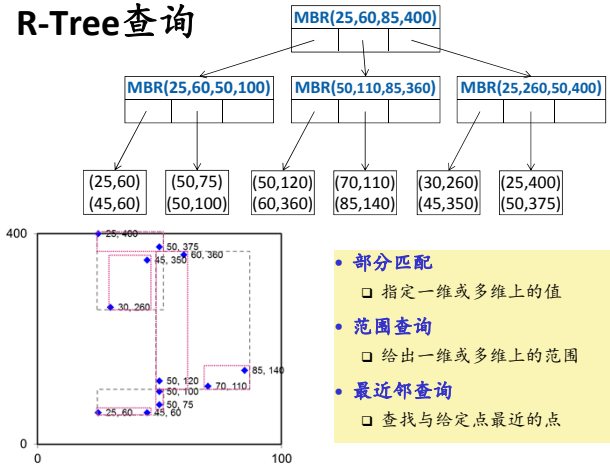
- 我们有下述（x,y）
  - (25,60) (45,60)
  - (50,75) (50,100)
  - (50,120) (70,110)
  - (85,140) (30,260)
  - (25,400) (45,350)
  - (50,375) (60,360)
- 假设Page可以放最多2条记录，或者3个MBR孩子结点



R-Tree举例

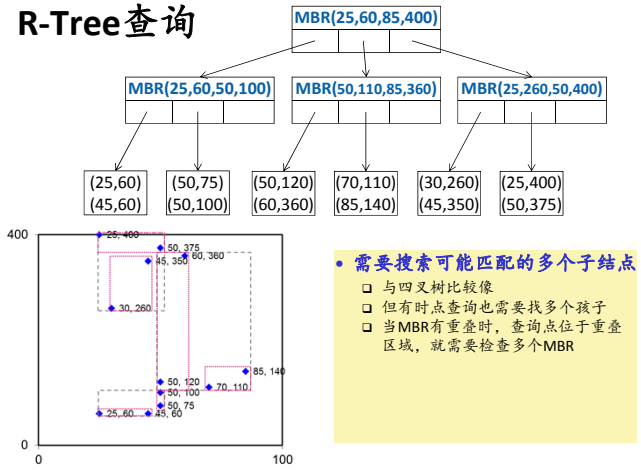


R-Tree查询



- **部分匹配**
  - 指定一维或多维上的值
- **范围查询**
  - 给出一维或多维上的范围
- **最近邻查询**
  - 查找与给定点最近的点

R-Tree查询



- **需要搜索可能匹配的多个子结点**
  - 与四叉树比较像
  - 但有时点查询也需要找多个孩子
  - 当MBR有重叠时，查询点位于重叠区域，就需要检查多个MBR

R-Tree建立和插入

- 关键问题是如何减少重叠区域？
- 我们这里不进一步仔细讲解
- 思路：采用启发式规则来减少重叠区域

多维树结构与B⁺-Tree比较

- 叶子可能处于不同层次
- 搜索可能需要访问同一层的多个孩子结点

Outline

- 多维索引
- 物理数据库设计
  - 索引的选择
  - 只需索引的查询
  - 索引选择的辅助工具

方针1：是否建立索引？

- 数据库系统自动建立的索引
  - Primary key, foreign key
- 用户建立索引
  - 目的：快速获取满足某种过滤条件的记录
  - 只有where语句中涉及的列才需要索引
- 所以
  - 需要分析应用的需求，分析有哪些常见的查询
  - 最好一个索引有助于多个查询

方针2：索引Key的选择、索引的种类

- 索引Key是Where语句中出现的属性
  - 等值条件：哈希索引、B^+-Tree等树结构索引
  - 范围选择：B^+-Tree等树结构索引

（如果系统支持其他的索引结构）

- 当数据不进行update时，可以考虑Bitmap index
- 对于文本数据，可以考虑Inverted index
- 对于多维数据，可以考虑多维索引

方针3：多属性索引Key

- 当Where语句中包含同一个表的多个属性
- 可以考虑单维索引+多个属性拼接为Key
- 注意索引中属性的次序

方针4：是否聚簇（Clustered）索引

- 基本原则
  - 一个表只能按照一个索引进行聚簇
  - 其它索引都是二级索引
- 范围查询在聚簇上的收益最大

方针5：平衡索引的开销

- 收益
  - 提高查询速度
- 开销
  - 空间开销：外存开销、内存开销
  - 时间开销：维护开销，当insert/delete/update时，相应的索引也必须被修改
- 选择恰当的索引和恰当的索引数量使在可以容忍的开销下，获得最大收益

只需要索引就可以执行的查询

- 如果查询的所有列都在索引中可以找到
- 那么这个查询实际可以用索引来完成（MS SQL Server）
- 例如

```
select count(*)
from Student
where major= “计算机”;
```

如果在Student表上的major列上已经建立了索引，那么就可以直接在索引上完成count(*)的操作

辅助工具

- 可以选择的索引数量是指数级的
- 如何选择好的索引？
  - 人工：通过经验，建立索引，然后测试
  - 辅助工具：例如, DB2 index advisor, Microsoft SQL Server Index Tuning Wizard
    - 通过一定的启发式规则，搜索可能的索引集合，估计其对查询的影响和开销，推荐一组索引

小结

- 多维索引
  - 概念
  - 多维散列索引
    - 网格文件（Grid File）
    - 分段散列（Partitioned Hashing）
  - 多维树结构索引
    - 四叉树（Quad Tree）
    - R树（R-Tree）
- 物理数据库设计
  - 索引的选择
  - 只需索引的查询
  - 索引选择的辅助工具