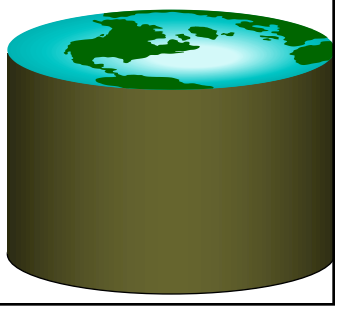
File Organizations and Indexing

Lecture 3
Chapter 8

"If you don't find it in the index, look very carefully through the entire catalogue."

-- Sears, Roebuck, and Co., Consumer's Guide, 1897



1

Review: Memory, Disks, & Files

- **Everything won't fit in RAM (usually)**
- **Hierarchy of storage, RAM, disk, tape**
- **"Block" - unit of storage in RAM, on disk**
- **Allocate space on disk for fast access**
- **Buffer pool management**
  - Frames in RAM to hold blocks
  - Policy to move blocks between RAM & disk
- **Storing records within blocks**

2

Today: File Storage

- **How to keep blocks of records on disk**
- ***but* must support operations:**
  - scan all records
  - search for a record id "RID"
  - insert new records
  - delete old records

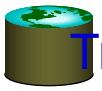
3

Alternative File Organizations

Many alternatives exist, *each good for some situations, and not so good in others:*

- Heap files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best for retrieval in *search key* order, or only a `range' of records is needed.
- Clustered Files (with Indexes): Coming soon…

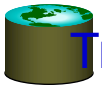
4



Transcript Stored as a Heap File

666666	MGT123	F1994	4.0	
123456	CS305	S1996	4.0	page 0
987654	CS305	F1995	2.0	
717171	CS315	S1997	4.0	
666666	EE101	S1998	3.0	page 1
765432	MAT123	S1996	2.0	
515151	EE101	F1995	3.0	
234567	CS305	S1999	4.0	
				page 2
878787	MGT123	S1996	3.0	

5



Transcript Stored as a Sorted File

111111	MGT123	F1994	4.0	
111111	CS305	S1996	4.0	page 0
123456	CS305	F1995	2.0	
123456	CS315	S1997	4.0	
123456	EE101	S1998	3.0	page 1
232323	MAT123	S1996	2.0	
234567	EE101	F1995	3.0	
234567	CS305	S1999	4.0	
				page 2
313131	MGT123	S1996	3.0	

6

Cost Model for Analysis

We ignore CPU costs, for simplicity:

- **B:** The number of data blocks
- **R:** Number of records per block
- **D:** (Average) time to read or write disk block
- Measuring number of block I/O's ignores gains of pre-fetching and sequential access; thus, even I/O cost is only loosely approximated.
- Average-case analysis; based on several simplistic assumptions.

☛ *Good enough to show the overall trends!*

7

Some Assumptions in the Analysis

- **Single record insert and delete.**
- **Equality selection - exactly one match (what if more or less???).**
- **Heap Files:**
  - Insert always appends to end of file.
- **Sorted Files:**
  - Files compacted after deletions.
  - Selections on search key.

8

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records			
Equality Search			
Range Search			
Insert			
Delete			

9

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	
Equality Search			
Range Search			
Insert			
Delete			

10

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	
Equality Search	0.5 BD	$(\log_2 B) * D$	
Range Search			
Insert			
Delete			

11

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	
Equality Search	0.5 BD	$(\log_2 B) * D$	
Range Search	BD	$[(\log_2 B) +$ #match pg$]*D$	
Insert			
Delete			

12

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	
Equality Search	0.5 BD	$(\log_2 B) * D$	
Range Search	BD	$[(\log_2 B) +$ #match pg$]*D$	
Insert	2D	$((\log_2 B)+B)D$ *(because R,W 0.5)*	
Delete			

13

Cost of

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	
Equality Search	0.5 BD	$(\log_2 B) * D$	
Range Search	BD	$[(\log_2 B) +$ #match pg$]*D$	
Insert	2D	$((\log_2 B)+B)D$	
Delete	0.5BD + D	$((\log_2 B)+B)D$ *(because R,W 0.5)*	

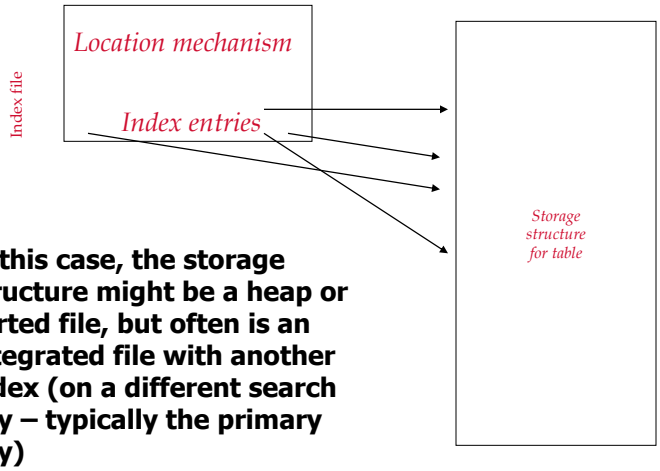
14

Indexes

- **Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,**
  - Find all students in the "CS" department
  - Find all students with a gpa > 3
- **An *index* on a file is a disk-based data structure that speeds up selections on the *search key fields* for the index.**
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is not the same as *key* (e.g. doesn't have to be unique ID).
- **An index contains a collection of *data entries*, and supports efficient retrieval of all records with a given search key value** k**.**

15

Index File With Separate Storage Structure



In this case, the storage structure might be a heap or sorted file, but often is an integrated file with another index (on a different search key – typically the primary key)

16

First Question to Ask About Indexes

- **What kinds of selections do they support?**
  - Selections of form field <op> constant
  - Equality selections (op is =)
  - Range selections (op is one of <, >, <=, >=, BETWEEN)
  - More exotic selections:
    - 2-dimensional ranges ("east of Dufferin and west of Bathurst and North of Front and South of Bloor")
      - Or n-dimensional
    - 2-dimensional distances ("within 2 miles of Soda Hall")
      - Or n-dimensional
    - Ranking queries ("10 restaurants closest to Berkeley")
    - Regular expression matches, genome string matches, etc.
    - One common n-dimensional index: R-tree
      - Supported in most major RDBMS

17

Index Classification

- **What selections does it support**
- **Representation of data entries in index**
  - i.e., what kind of info is the index actually storing?
  - 2 alternatives here
- **Clustered vs. Unclustered Indexes**
- **Single Key vs. Composite Indexes**
- **Tree-based, hash-based, other**

18

Alternatives for Data Entry **k*** in Index

- **Two alternatives:**
  - <**k**, rid of matching data record>
  - <**k**, list of rids of matching data records>
- **Choice is orthogonal to the indexing technique.**
  - Examples of indexing techniques: B+ trees, hash-based structures, R trees, …
  - Typically, index contains auxiliary information that directs searches to the desired data entries
- **Can have multiple (different) indexes per file.**
  - E.g. file sorted by *age*, with a hash index on *salary* and a B+tree index on *name*.

19

Alternatives for Data Entries (Contd.)

Alternative 1

<k, rid of matching data record>

and Alternative 2

<k, list of rids of matching data records>

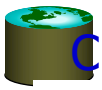
- Support multiple indexes
- Alternative 2 more compact than Alternative 1, but leads to *variable sized data* entries even if search keys are of fixed length.
- Even worse, for large rid lists the data entry would have to span multiple blocks!

21

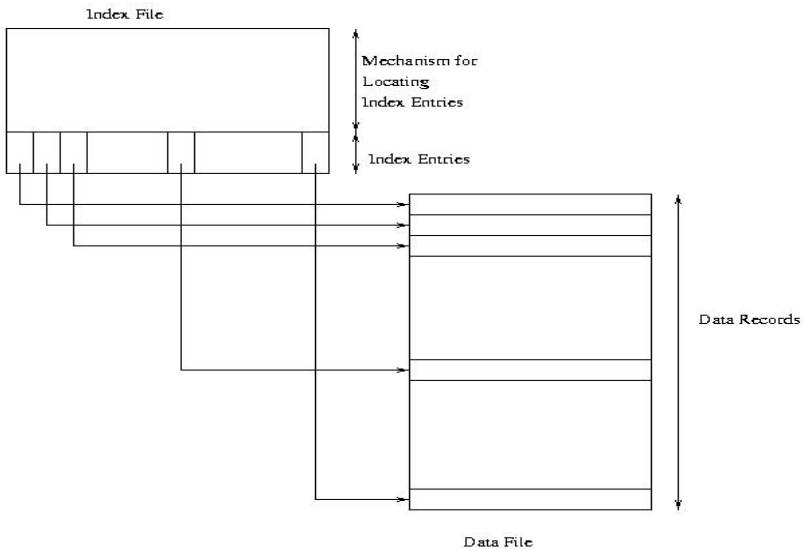
Index Classification

- ***Clustered* vs. *unclustered*: If order of data records is the same as, or `close to', order of index data entries, then called *clustered index*.**
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies *greatly* based on whether index is clustered or not!

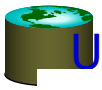
22



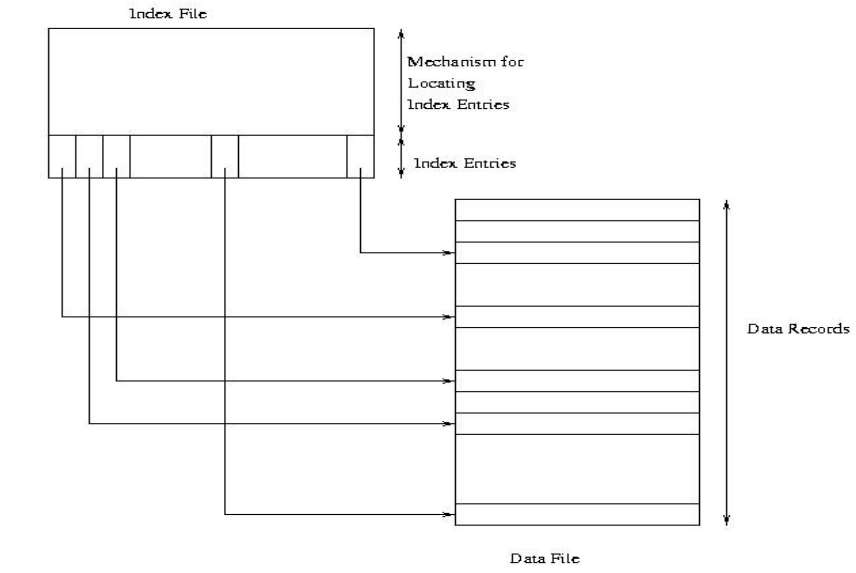
Clustered Index



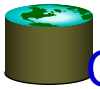
23



Unclustered Secondary Index

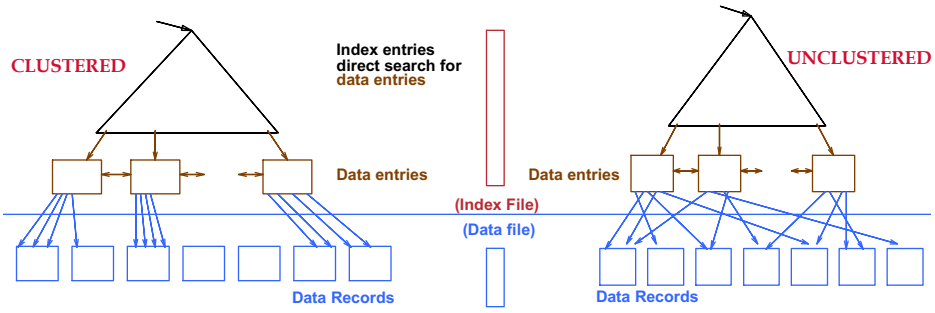


24



Clustered vs. Unclustered Index

- **Suppose that Alternative (1) is used for data entries, and that the data records are stored in a Heap file.**
  - To build clustered index, first sort the Heap file (with some free space on each block for future inserts).
  - Overflow blocks may be needed for inserts. (Thus, order of data recs is `close to', but not identical to, the sort order.)



25

Unclustered vs. Clustered Indexes

- **What are the tradeoffs????**
- **Clustered Pros**
  - Efficient for range searches
  - May be able to do some types of compression
  - Possible locality benefits (related data?)
  - ???
- **Clustered Cons**
  - Expensive to maintain (on the fly or sloppy with reorganization)

26

Cost of

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	1.5 BD
Equality Search	0.5 BD	$(\log_2 B) * D$	$(\log_F 1.5B) * D$
Range Search	BD	$[(\log_2 B) +$ #match pg$]*D$	$[(\log_F 1.5B) +$ #match pg$]*D$
Insert	2D	$((\log_2 B)+B)D$	$((\log_F 1.5B)+1) * D$
Delete	0.5BD + D	$((\log_2 B)+B)D$ *(because R,W 0.5)*	$((\log_F 1.5B)+1) * D$

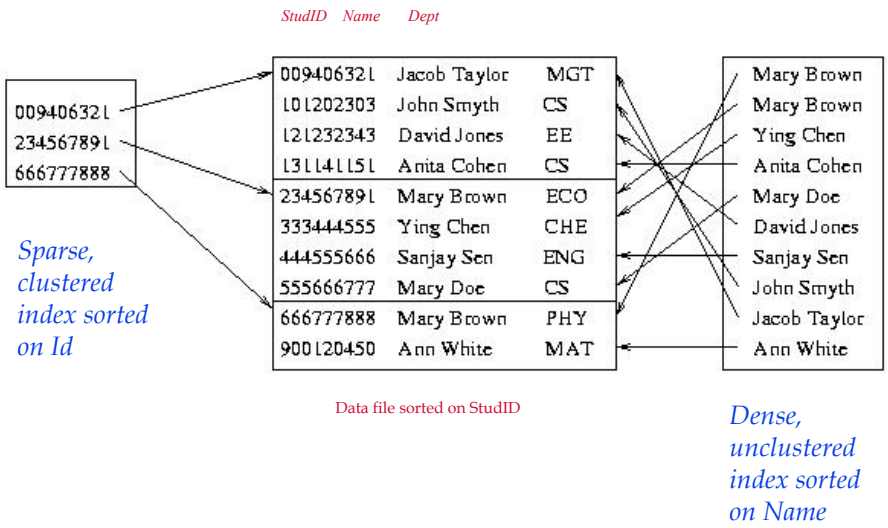
27

Sparse vs. Dense Index

- ***Dense index*: has index entry for each data record**
  - Unclustered index *must* be dense
  - Clustered index need not be dense
- ***Sparse index*: has index entry for each page of data file**

28

Sparse Vs. Dense Index cont'd



29

Sparse Index

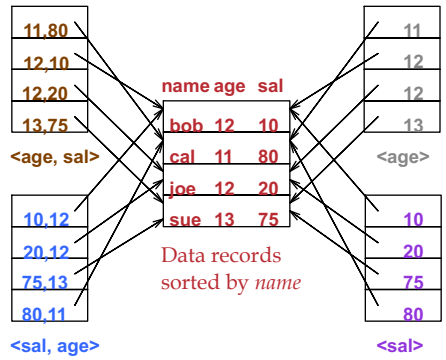
- **To locate a record with search-key value *K* we:**
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points
- **Less space and less maintenance overhead for insertions and deletions.**
- **Generally slower than dense index for locating records.**

30

Composite Search Keys

- **Search on a combination of fields.**
  - Equality query: Every field value is equal to a constant value. E.g. wrt <age,sal> index:
    - age=20 and sal =75
  - Range query: Some field value is not a constant. E.g.:
    - age > 20; or age=20 and sal > 10
- **Data entries in index sorted by search key to support range queries.**
  - Lexicographic order
  - Like the dictionary, but on fields, not letters!

Examples of composite key indexes using lexicographic order.



Data entries in index sorted by <*sal,age*>

Data entries sorted by <*sal*>

31

Summary

- **Many alternative file organizations exist, each appropriate in some situation.**
- **If selection queries are frequent, sorting the file or building an *index* is important.**
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- **Index is a collection of data entries plus a way to quickly find entries with given key values.**

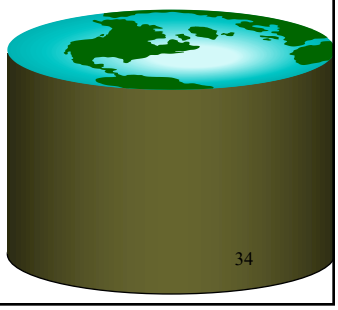
32

Summary (Contd.)

- **Data entries in index can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.**
  - Choice orthogonal to *indexing structure (i.e. tree, hash, etc.)*.
- **Usually have several indexes on a given file of data records, each with a different search key.**
- **Indexes can be classified as**
  - clustered vs. unclustered
  - dense vs. sparse
- **Differences have important consequences for utility/performance.**

33

Spatial Indexing and Searching



34

Consider a classic database…

Name	ID	Type	Phone	Location	Grade
Marios Pizza	1	ITA	888-1212	244, 365	D
Joes Bugers	2	US	848-1298	34, 764	A
Tinas Mexican	3	MEX	878-1333	123, 32	A
Sues Pasta	4	ITA	878-1342	876, 65	B

35

Name	ID	Type	Phone	Location	Grade
Marios Pizza	1	ITA	888-1212	244,365	D
Joes Bugers	2	US	848-1298	34,764	A
Tinas Mexican	3	MEX	878-1333	123,32	A
Sues Pasta	4	ITA	878-1342	876,65	B

Given such a database we can easily answer queries such as

- *List all Mexican restaurants.*
- *List all Grade A restaurants.*

By using our friend SQL

However, classic databases do not allow queries such as

- *List all Mexican restaurants within five miles of UCR*
- *List the pizza restaurant nearest to 91 and 60.*

These kinds of queries are called *spatial queries.*

36

There are 3 kinds of spatial queries

- Nearest neighbor queries
- Range queries
- Spatial joins

37

One possible way to do spatial queries

```
Algorithm One_Nearest_Neighbor_Query(location)
 best_so_far = infinity;

 for i = 1 to  number_items_in_database

 retrieve record(i) from database;

   if distance(location,record(i).location )
     best_so_far = distance(location,record(i).location )
     NNpointer= i;
   end;

 end;

 disp('The nearest neighbor is',record(NNpointer ).name );

end;
```

But this does not work well because…

38

…so we need to *index* the data

My informal definition of indexing. Organizing the data such that queries can be answered in sub linear time.

39

So, we call always index 1-dimensional data (*if you can sort it, you can index it*), such that we can answer 1-nearest neighbor queries by accessing just O(log(*n*)) of the database. (*n* is the number of items in the database).
(i.e. the B+tree)

But we can't sort 2 dimensional data…

This is an important problem which has attracted a great deal of research interest….

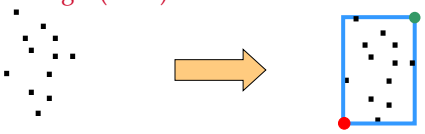
We can in fact index 2 (and higher) dimensional reasonably efficiently with special data structures known as Spatial Access Methods (SAM) or Multidimensional Access Methods.

Although there are many variations we will focus on the R-Tree, introduced by Guttman in the 1984 SIGMOD conference.

40

Suppose we have a cluster of points in 2-D space...

We can build a "box" around points. The smallest box (which is axis parallel) that contains all the points is called a Minimum Bounding Rectangle (MBR)

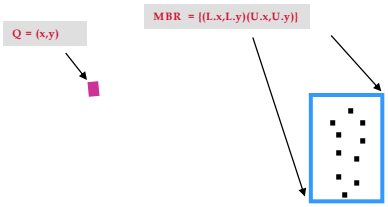


MBR = {(L.x,L.y)(U.x,U.y)}

Note that we only need two points to describe an MBR, we typically use lower left, and upper right.

41

The formula for the distance between a point and the closest possible point within an MBR

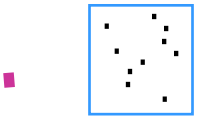


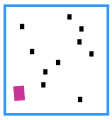
MINDIST(Q,MBR)

if L.x < x < U.x **and** L.y < y < U.y **then** 0
elseif L.x < x < U.x **then** min($(L.y - y)^2$, $(U.y - y)^2$)
elseif ....

42

Two examples of **MINDIST(*point*, *MBR*)**, calculations…





MINDIST(*point*, *MBR*) = 5

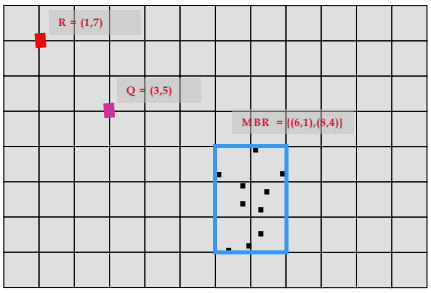
MINDIST(*point*, *MBR*) = 0

43

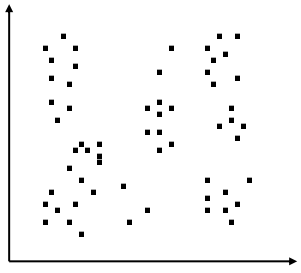
We will find it useful to have a formula for the distance between a point and the closest possible point within an Minimum Bounding Rectangle MBR…

Suppose we have a query point **Q** and one known point **R**

Could any of the points in the MBR be closer to **Q** than **R** is?



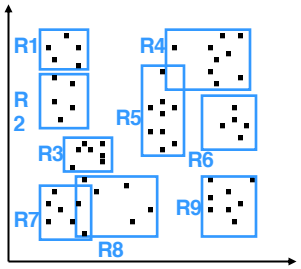
44



We can visualize the locations of interest simply as points in space...

45

We can group clusters of datapoints into "boxes", called Minimum Bounding Rectangles (MBRs).

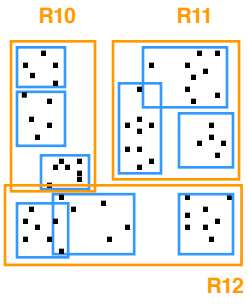


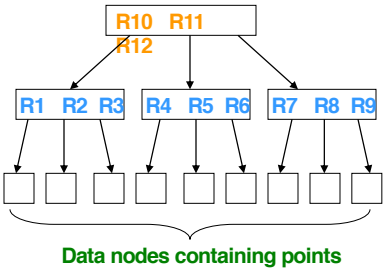
Each MBR can be represented with just two points. The lower left corner, and the upper right corner.

We can further recursively group MBRs into larger MBRs….

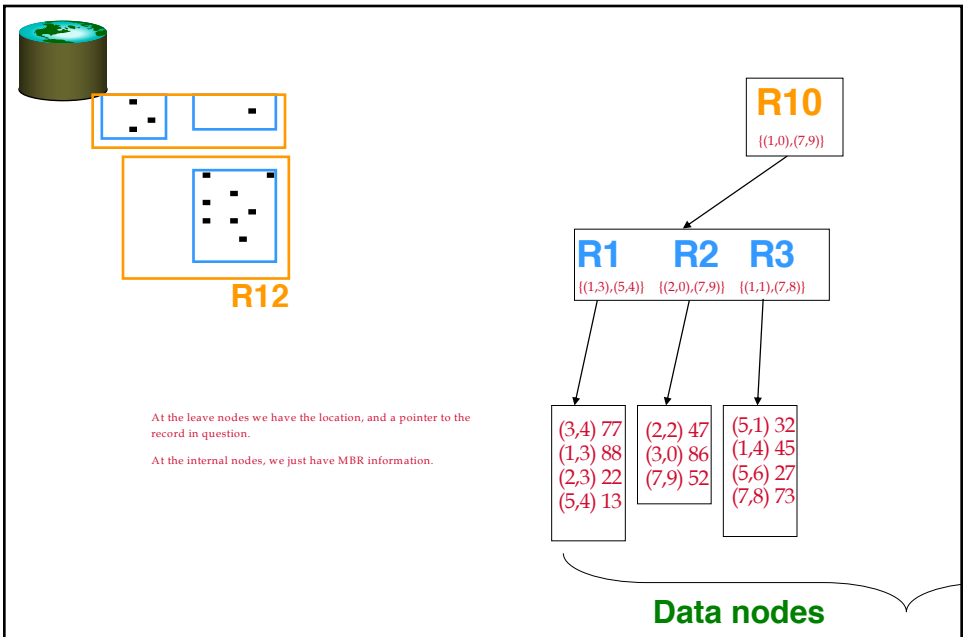
46

…these nested MBRs are organized as a tree (called a spatial access tree or a multidimensional tree).





47



48

R-Trees

- **R-trees are a N-dimensional extension of B$^+$-trees, useful for indexing sets of rectangles and other polygons.**
- **Supported in many modern database systems, along with variants like R$^+$ -trees and R*-trees.**
- **Basic idea: generalize the notion of a one-dimensional interval associated with each B+ -tree node to an N-dimensional interval, that is, an N-dimensional rectangle.**
- **Will consider only the two-dimensional case ($N = 2$)**
  - generalization for $N > 2$ is straightforward, although R-trees work well only for relatively small N

49

R Trees (Cont.)

- **A rectangular bounding box is associated with each tree node.**
  - Bounding box of a leaf node is a minimum sized rectangle that contains all the rectangles/polygons associated with the leaf node.
  - The bounding box associated with a non-leaf node contains the bounding box associated with all its children.
  - Bounding box of a node serves as its key in its parent node (if any)
  - *Bounding boxes of children of a node are allowed to overlap*
- **A polygon is stored only in one node, and the bounding box of the node must contain the polygon**
  - The storage efficiency or R-trees is better than that of k-d trees or quadtrees since a polygon is stored only once

50

Search in R-Trees

- **To find data items (rectangles/polygons) intersecting (overlaps) a given query point/region, do the following, starting from the root node:**
  - If the node is a leaf node, output the data items whose keys intersect the given query point/region.
  - Else, for each child of the current node whose bounding box overlaps the query point/region, recursively search the child
- **Can be very inefficient in worst case since multiple paths may need to be searched**
  - but works acceptably in practice.
- **Simple extensions of search procedure to handle predicates *contained-in* and *contains***

51