Lecture 12

Indexes Continued

Summary – last week

- Indexes
  - Tree based indexes

This week:

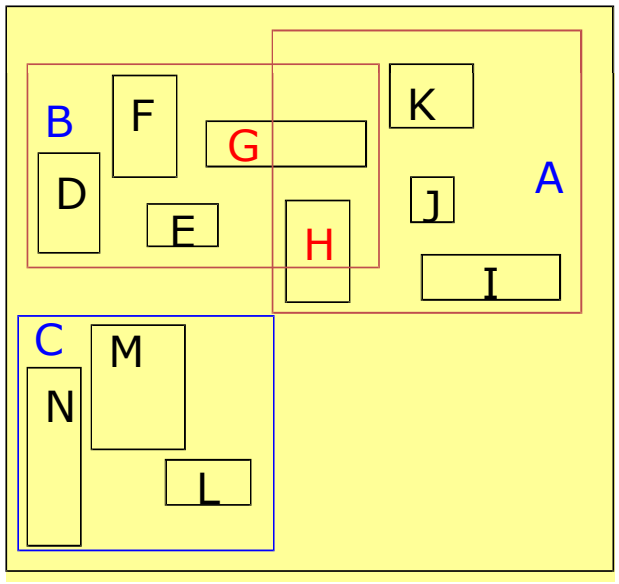
- Indexes
  - R+ Tree
  - Bitmap indexes

Improved Version of R-Tree

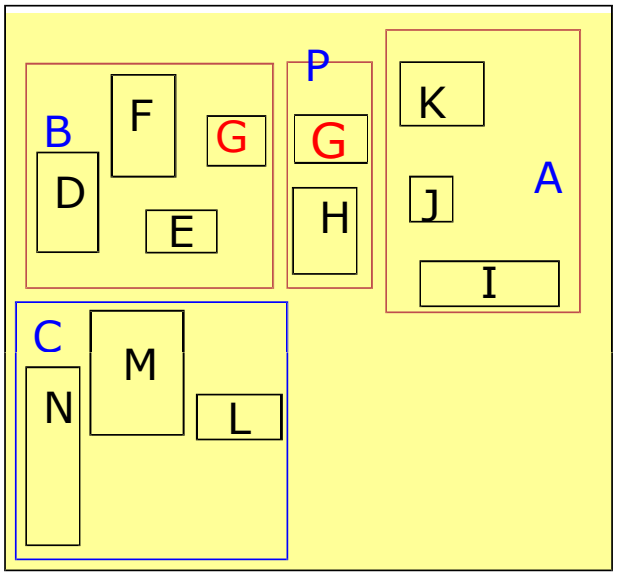
- Where are R-trees inefficient?
  - They **allow overlapping** between neighboring MBRs
- R^+-Trees (Sellis ua, 1987)
  - **Overlapping** of neighboring MBRs are prohibited
  - This may lead to identical leafs occurring more than once in the tree
  - Improve search efficiency, but similar scalability as R-trees

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

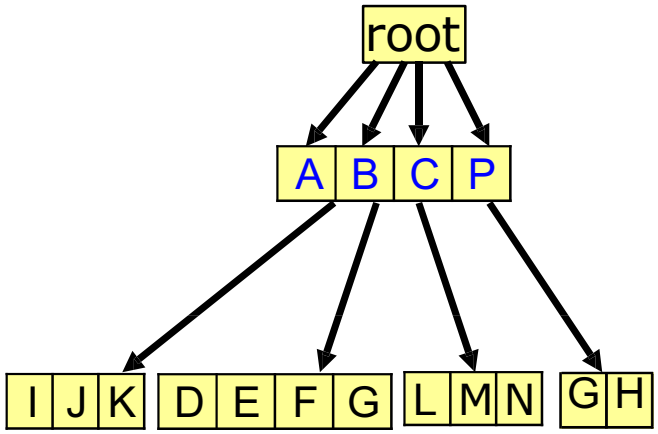
R^+ -Tree



Old



New



- Overlaps are not permitted (A and B)
- Data rectangles are divided and may be present (e.g., G) in several leafs

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Performance

Advantages

- The main advantage of R+-trees is to improve the search performance
- Especially for point queries, this saves 50% of access time

Disadvantages

- Drawback is the low occupancy of nodes resulting through many splits
- R+-trees often degenerate with the increasing number of changes

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes

- Bitmap Indexes
  - Lets assume a relation Expenses with three attributes: Nr, Shop and Sum
  - A bitmap index for attribute Shop looks like this

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Value	Vector
P&C	001000
Real	010101
Saturn	100010

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
~~5~~	~~Saturn~~	~~350~~
6	Real	80

- Handling modification
  - Assume record numbers are not changed
- Deletion
  - **Tombstone** replaces deleted record (6 doesn't become 5!)
  - Corresponding bit is set to 0

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	001000
Real	010101
Saturn	100000

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

- Inserted record is assigned the next record number
  - A bit of value 0 or 1 is appended to each bit vector
  - If new record contains a new value of the attribute, add one bit-vector
    - E.g., insert new record with REWE as shop

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80
7	REWE	23

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	0010000
Real	0101010
Saturn	1000100
REWE	0000001

Bitmap Indexes (cont'd.)

- Update
  - Change the bit corresponding to the old value of the modified record to 0
  - Change the bit corresponding to the new value of the modified record to 1
  - If the new value is a new value of A, then insert a new bit-vector: e.g., replace Shop for record 2 to REWE

Nr	Shop	Sum
1	Saturn	150
2	REWE	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	001000
Real	000101
Saturn	100010
REWE	010000

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

- Select
  - Basic bit operations:
    - E.g., select the sums we have spent in Saturn and P&C

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Value	Vector
P&C	001000
Real	010101
Saturn	100010

Saturn	OR	P&C	=	Result
1		0		1
0		0		0
0		1		1
0		0		0
1		0		1
0		0		0

  - Bitmap indexes should be used when selectivity is **high**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

- Advantages
  - Operations are efficient and easy to implement (directly supported by hardware)
- Disadvantages
  - For each new value of an attribute a new bitmap-vector is introduced
    - If we bitmap index an attribute like birthday (only day) we have 365 vectors: 365/8 bits ≈ 46 Bytes for a record, just for that
    - Solution to such problems is multi-component bitmaps
  - Not fit for range queries where many bitmap vectors have to be read
    - Solution: range-encoded bitmap indexes

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Summary

- Bitmap indexes
  - Great for indexing tables with set-like attributes e.g., Gender: Male/Female
  - Operations are efficient and easy to implement (directly supported by hardware)
  - Multi component reduce the storage while range encoded allow for fast range queries

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Next Lecture

- Optimization
  - Partitioning
  - Joins
  - MaterializedViews

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides