Advanced Indexing Techniques

Knowledge Objectives

1. Explain which index suits best depending on the selectivity of the selection predicate, the kind of comparison and the volatility of the table
2. Name three situation where an index is useless
3. Explain what a bitmap index is
4. Explain the conditions where a bitmap suits better than a B+ index and vice-versa
5. Explain what a join-index is
6. Explain the benefit of bitmap-join-indexes in multidimensional queries
7. Explain what makes the difference between a join-index and a clustered structure, from the query time point of view

Understanding Objectives

1. Know the factors involved in the choice between rebuilding an index or making individual insertions in the case of massive insertions
2. Calculate the approximate size of a bitmap index
3. Estimate the cost of a selection with a complex predicate using a bitmap index
4. Estimate the cost of a join (or semi-join) operation using a join index (either bitmap, B+, hash or cluster)
5. Given a simple query (not mixing selection and join operations) and the structures of the tables, decide whether it can be solved by accessing only the indexes
6. Given the attributes in a multi-attribute index and a complex selection predicate, decide whether the index can be used to solve the query or not

Bitmap-index

Ballpoint	Pencil	Pen	Rubber	A4 paper	A3 paper	Chalk	Eraser
1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0

Catalunya	León	Madrid	Andalucía
1	0	0	0
1	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1
0	1	0	0
1	0	0	0
1	0	0	0

Querying with bitmaps

SELECT COUNT(*)

…

WHERE articleName IN [‘Ballpoint’,‘Pencil’] AND region=‘Catalunya’

Ballpoint		Pencil				Catalunya		
1		0		1		1		1
0		0		0		1		0
0		1		1		0		0
0		0		0		0		0
0		0		0		0		0
1	OR	0	=	1	AND	1	=	1
0		0		0		0		0
0		0		0		0		0
0		0		0		1		0
0		1		1		1		1

Updating bitmaps

- Two cases of insertion:
  - Without domain expansion:
    - Add "1"
  - With domain expansion:
    - Add a new vector
- One case of deletion:
    - Change "1" for "0"

Catalunya	León	Madrid	Andalucía
1	0	0	0
1	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1
0	1	0	0
1	0	0	0
1	0	0	0

Updating bitmaps

- Two cases of insertion:
  - Without domain expansion:
    - Add “1”
  - With domain expansion:
    - Add a new vector
- One case of deletion:
    - Change “1” for “0”

Catalunya	León	Madrid	Andalucía
1	0	0	0
1	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1
0	1	0	0
1	0	0	0
1	0	0	0
0	0	1	0

Updating bitmaps

- Two cases of insertion:
  - Without domain expansion:
    - Add "1"
  - With domain expansion:
    - Add a new vector

- One case of deletion:
    - Change "1" for "0"

Catalunya	León	Madrid	Andalucía	Euskadi
1	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	0
1	0	0	0	0
1	0	0	0	0
0	0	1	0	0
0	0	0	0	1

Updating bitmaps

- Two cases of insertion:
  - Without domain expansion:
    - Add “1”
  - With domain expansion:
    - Add a new vector
- One case of deletion:
    - Change “1” for “0”

Catalunya	León	Madrid	Andalucía	Euskadi
0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	0
1	0	0	0	0
1	0	0	0	0
0	0	1	0	0
0	0	0	0	1

Probabilities with a bitmap

- Probability of a tuple fulfilling P

  SF

- Probability of a tuple NOT fulfilling P

  1-SF

- Probability of none of the tuples in a block fulfilling P

  $(1\text{-}SF)\cdot(1\text{-}SF)\cdot \ldots \cdot(1\text{-}SF) = (1\text{-}SF)^R$

- Probability of some tuple in a block fulfilling P

  $1\text{-}(1\text{-}SF)^R$

Cost of bitmap per operation

bits: bits per index block
ndist: different values
v: number of queried values

- Table scan
  - $ndist \cdot \lceil |T|/bits \rceil \cdot D + B \cdot D$
- Search for some tuples
  - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-(1-SF)^R)) \cdot D$
  - Examples:
    - Search for one tuple
      - $\lceil |T|/bits \rceil \cdot D + D$
    - Search for several tuples (given one value)
      - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot D$
    - Search for several tuples (given several values)
      - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-v)/ndist)^R)) \cdot D$
- Insertion of one tuple (in the last table block)
  - Existing value: $ndist \cdot 2D + 2D$
  - New value: $ndist \cdot 2D + 2D + \lceil |T|/bits \rceil \cdot D$
- Deletion of all tuples with a given value
  - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot 2D$

Cost of bitmap per operation

bits: bits per index block
ndist: different values
v: number of queried values

- Table scan
  - **Useless**
- Search for some tuples
  - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-(1-SF)^R)) \cdot D$
  - Examples:
    - Search for one tuple
      - $\lceil |T|/bits \rceil \cdot D + D$
    - Search for several tuples (given one value)
      - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot D$
    - Search for several tuples (given several values)
      - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-v)/ndist)^R)) \cdot D$
- Insertion of one tuple (in the last table block)
  - Existing value: $ndist \cdot 2D + 2D$
  - New value: $ndist \cdot 2D + 2D + \lceil |T|/bits \rceil \cdot D$
- Deletion of all tuples with a given value
  - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot 2D$

Cost of bitmap per operation

bits: bits per index block
ndist: different values
v: number of queried values

- Table scan
  - **Useless**
- Search for some tuples
  - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-(1-SF)^R)) \cdot D$
  - Examples:
    - Search for one tuple
      - **Useless?**
    - Search for several tuples (given one value)
      - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot D$
    - Search for several tuples (given several values)
      - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-v)/ndist)^R)) \cdot D$
- Insertion of one tuple (in the last table block)
  - Existing value: $ndist \cdot 2D + 2D$
  - New value: $ndist \cdot 2D + 2D + \lceil |T|/bits \rceil \cdot D$
- Deletion of all tuples with a given value
  - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot 2D$

Comparison

- Better than B-tree and hash for multi-value queries
- Optimum performance for several conditions over more than one attribute (each with a low selectivity)
- Orders of magnitude of improvement compared to a table scan (specially for SF<1%)
- May be useful even for range queries
- Sometimes used to manage NULL values
- Useful for non-unique attributes (specially for ndist<|T|/100, i.e. hundreds of repetitions)
- Bad performance for concurrent INSERT, UPDATE and DELETE
- Use more space than RID lists for domains of 32 values or more (may be better with compression), assuming uniform distribution and 4 bytes per RID

Bitmap indexes in Oracle 11g

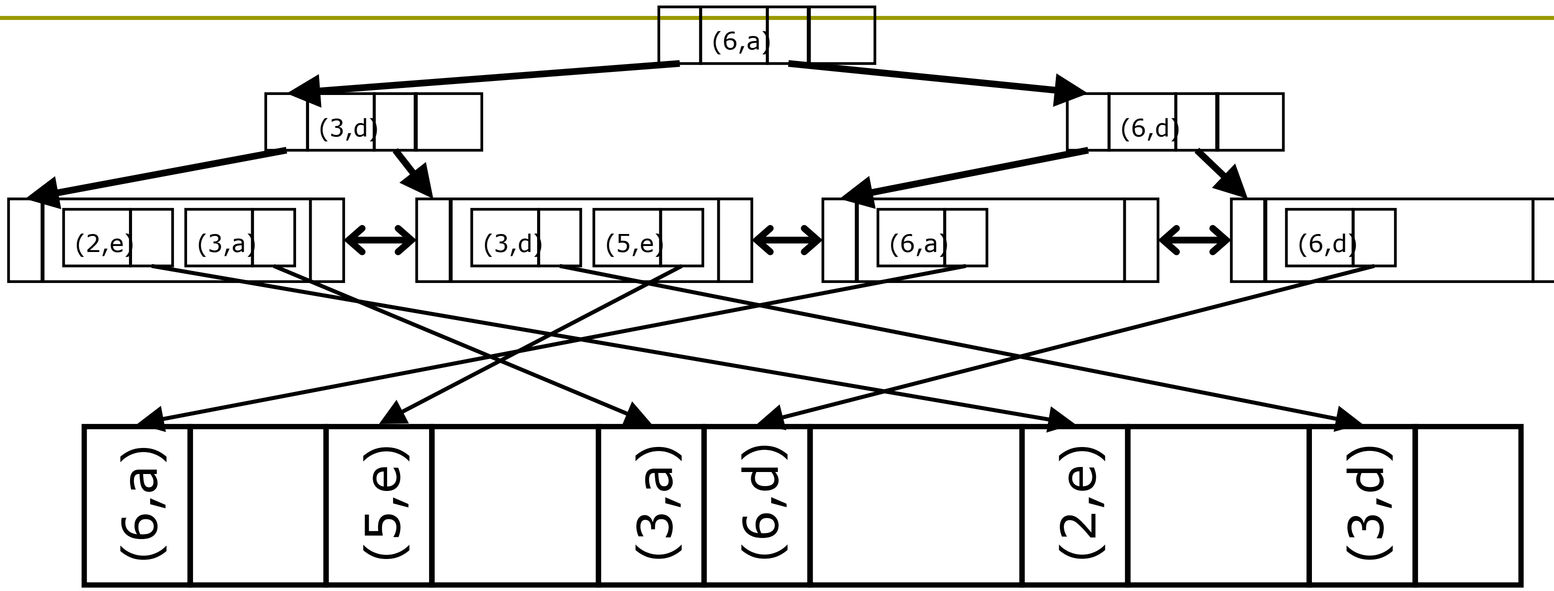
```
CREATE
[{UNIQUE|BITMAP}] INDEX <name>
ON <table> (<column>[,column]*);
```

- Allowed even for unique attributes
- Does not allow to check uniqueness

Usefulness of multi-attribute trees

- Need more space
  - For each tuple, keeps attributes A_1, .., A_k
  - May result in more levels, worsening access time
- Modifications are more frequent
  - Every time one of the attributes in the index is modified
- It is much more efficient than intersecting RID lists (to evaluate conjunctions)
- Can be used to solve several kinds of queries
  - Equality of all first *i* attributes
  - Equality of all first *i* attributes and range of *i+1*
- The order of attributes in the index matters
  - We cannot evaluate condition over A_k, if there is no equality for A_1, .., A_{k-1}

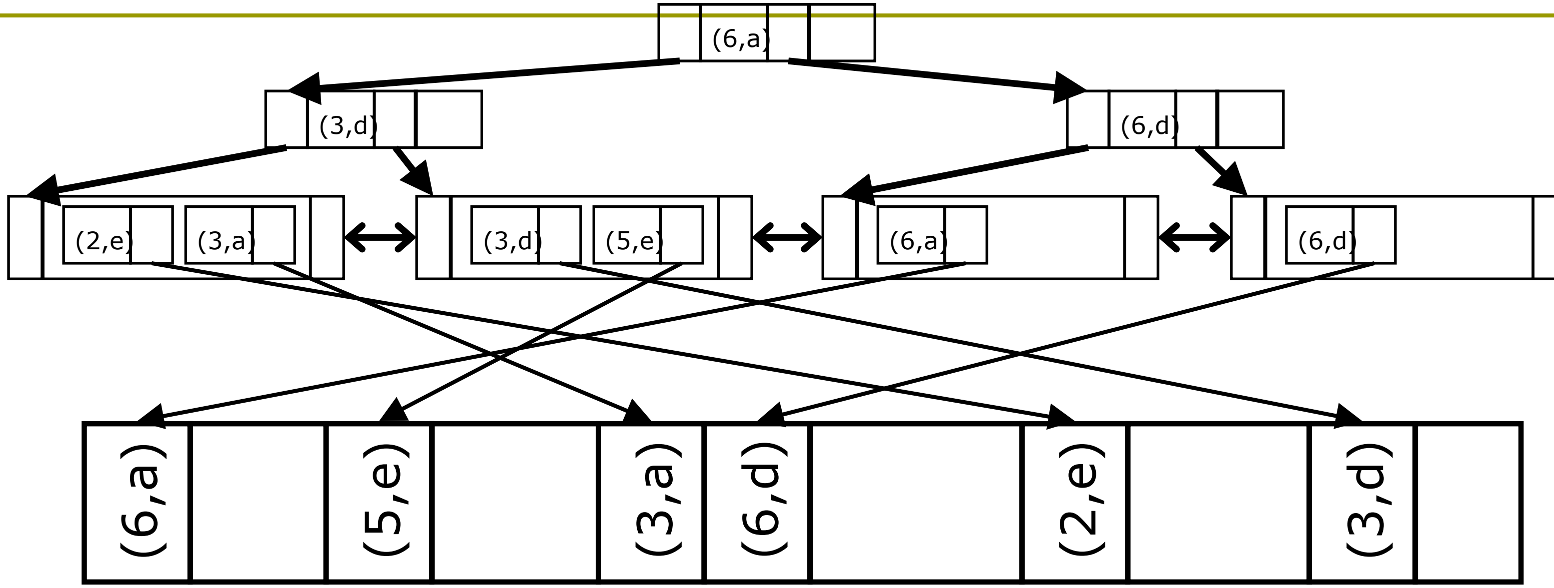
Multi-attribute tree



- Queries:
  - Num='3' AND Let='d'
  - Num='3' AND Let>'b'
  - Num='3'
  - Num>'3' AND Let='a'
  - Num>'3' AND Let>'b'
  - Num>'3'
  - Let='e'
  - Let>'b'
  - Num='3' OR Let='a'

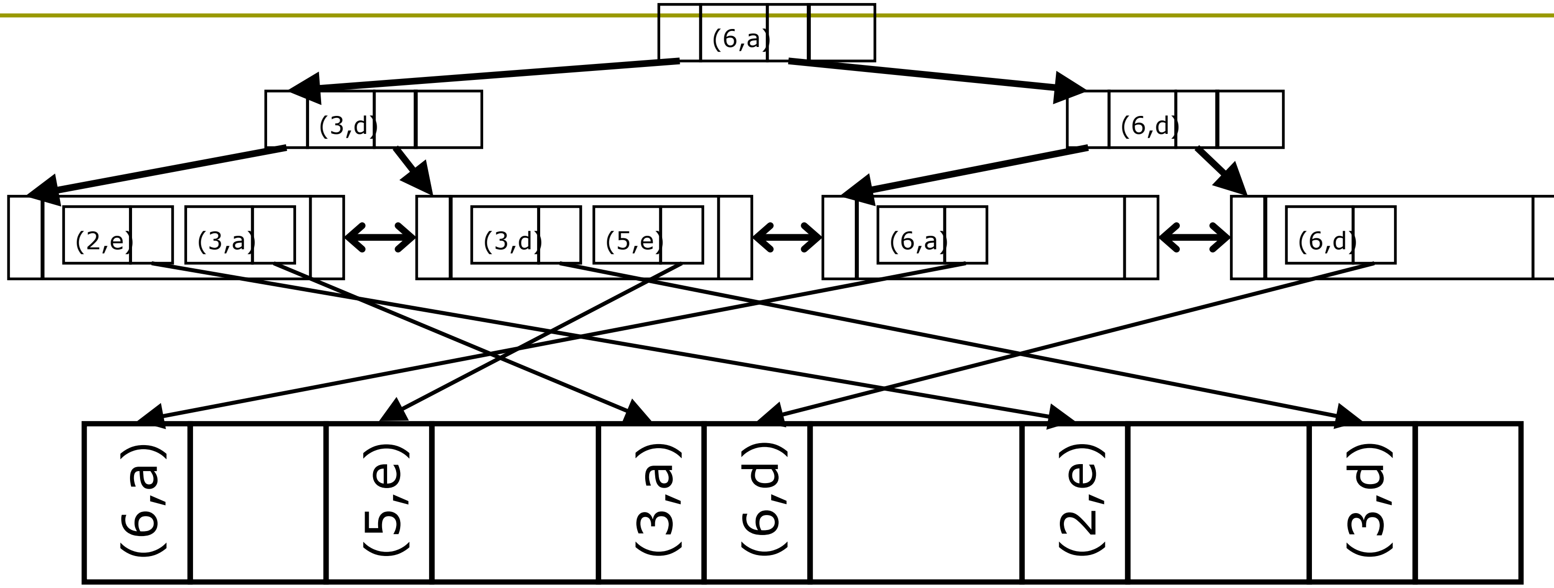
Could these queries be answered by means of the above multi-attribute + index?

Multi-attribute tree



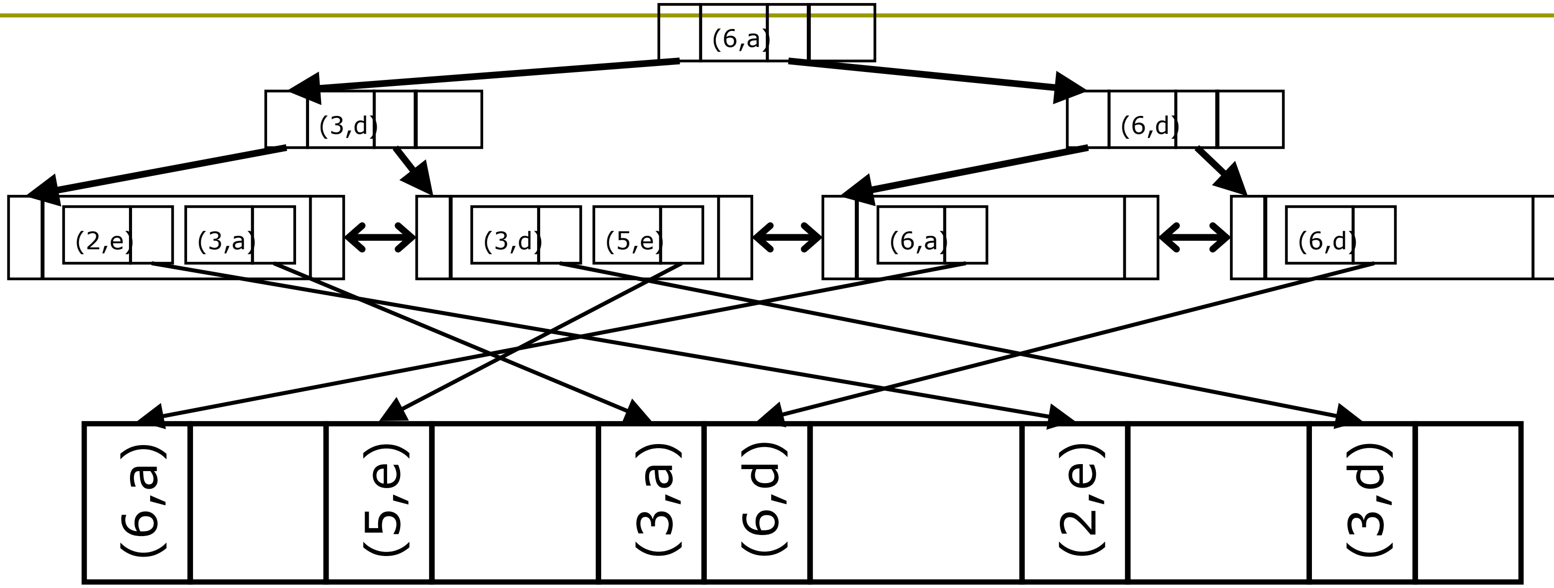
- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b'
  - Num='3'
  - Num>'3' AND Let='a'
  - Num>'3' AND Let>'b'
  - Num>'3'
  - Let='e'
  - Let>'b'
  - Num='3' OR Let='a'

Multi-attribute tree



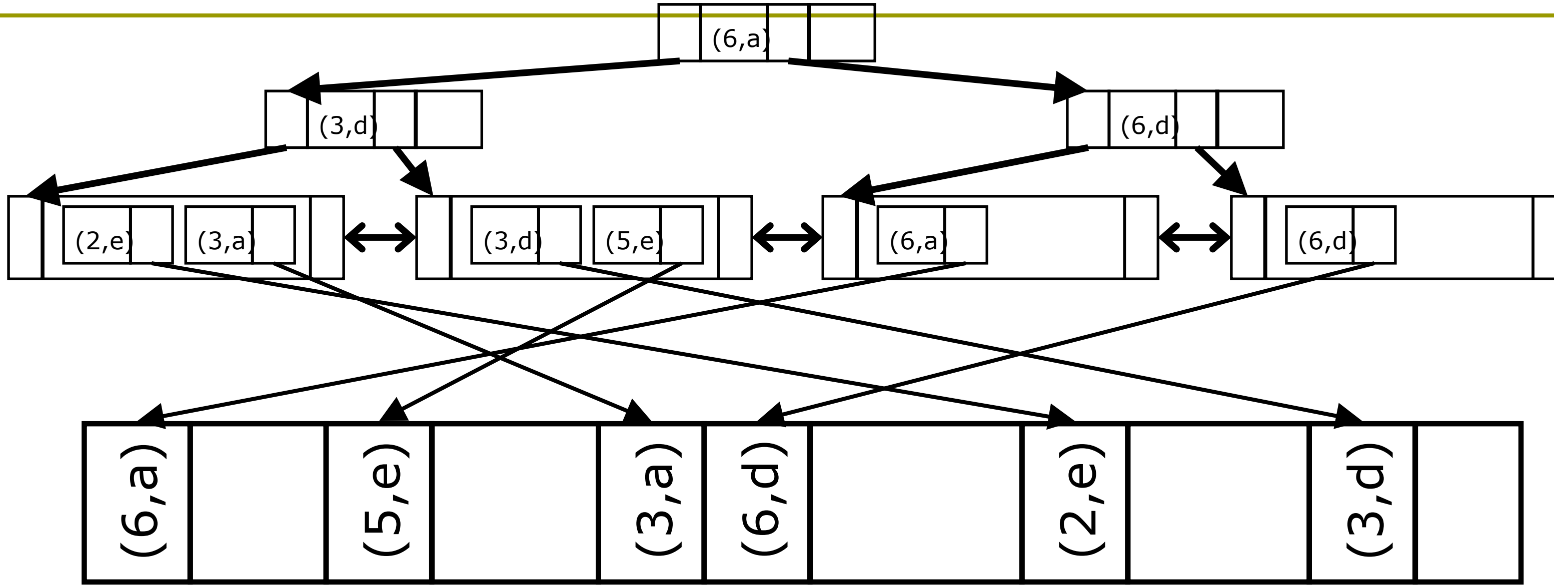
- Queries:
  - Num=‘3’ AND Let=‘d’ YES
  - Num=‘3’ AND Let>‘b’ YES
  - Num=‘3’
  - Num>’3’ AND Let=‘a’
  - Num>’3’ AND Let>‘b’
  - Num>’3’
  - Let=‘e’
  - Let>’b’
  - Num=‘3’ OR Let=‘a’

Multi-attribute tree



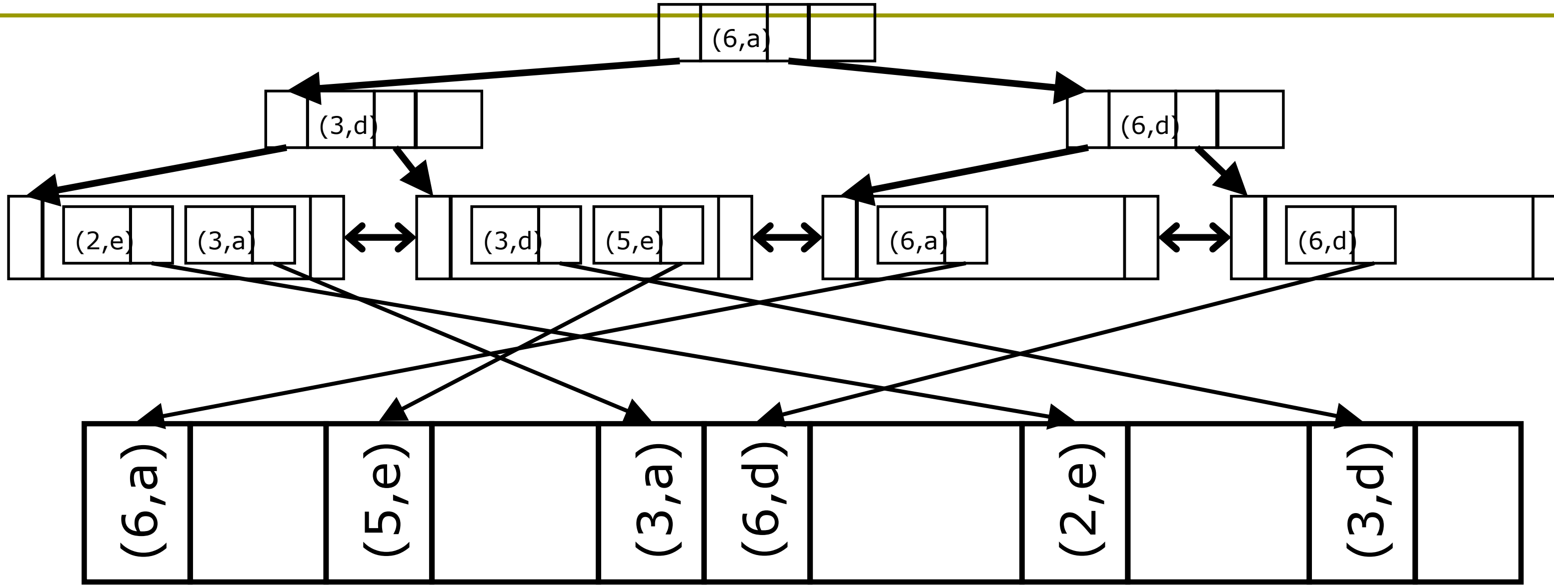
- Queries:
  - Num=‘3’ AND Let=‘d’ YES
  - Num=‘3’ AND Let>‘b’ YES
  - Num=‘3’ YES
  - Num>’3’ AND Let=‘a’
  - Num>’3’ AND Let>‘b’
  - Num>’3’
  - Let=‘e’
  - Let>’b’
  - Num=‘3’ OR Let=‘a’

Multi-attribute tree



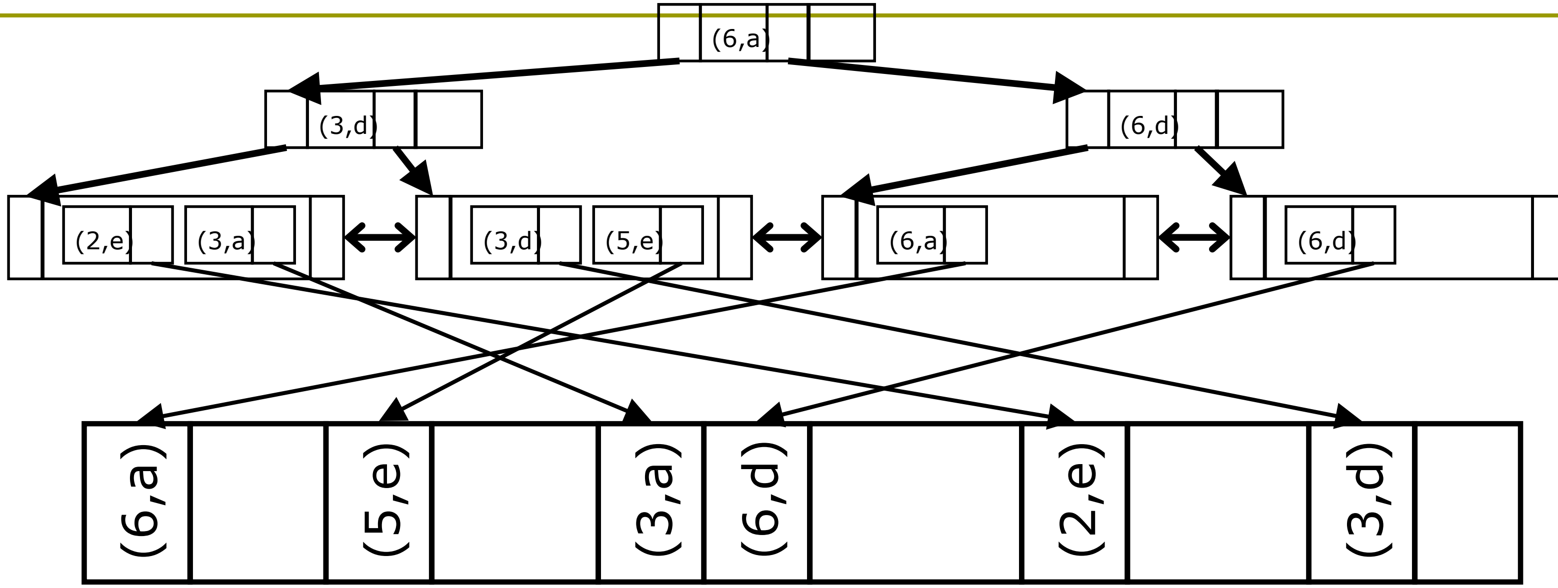
- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b' YES
  - Num='3' YES
  - Num>'3' AND Let='a' NO
  - Num>'3' AND Let>'b'
  - Num>'3'
  - Let='e'
  - Let>'b'
  - Num='3' OR Let='a'

Multi-attribute tree



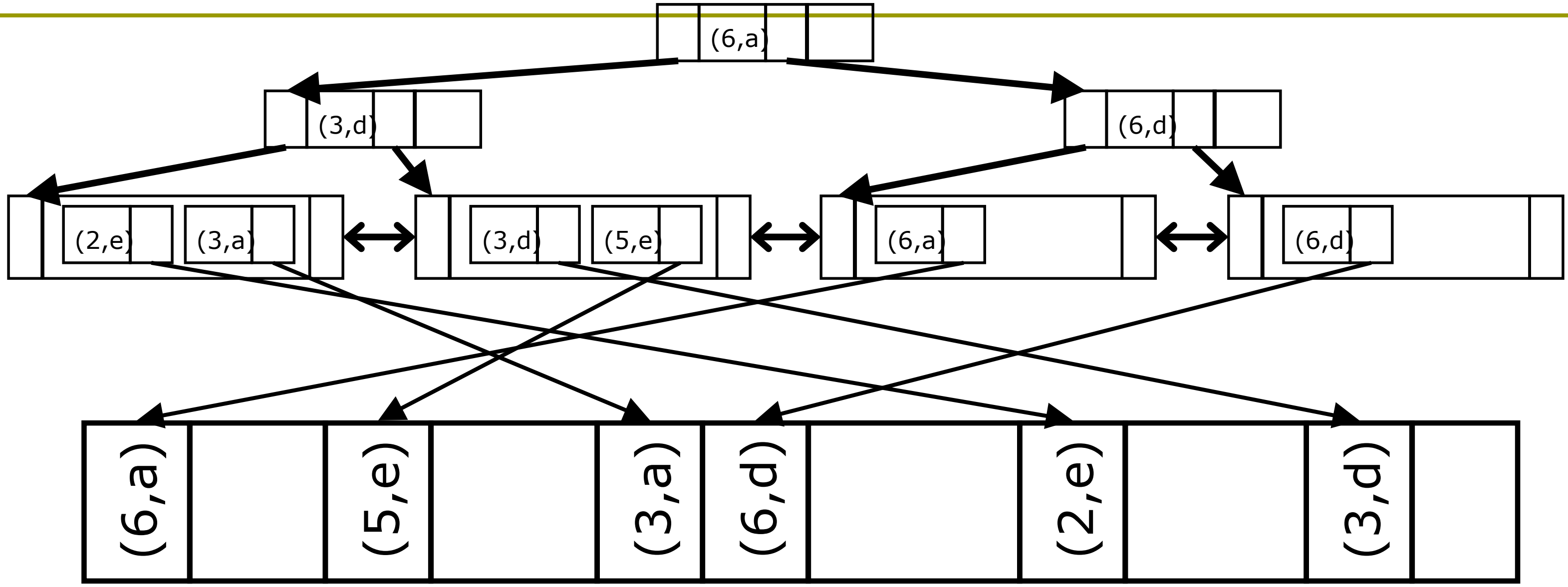
- Queries:
  - Num=‘3’ AND Let=‘d’ YES
  - Num=‘3’ AND Let>‘b’ YES
  - Num=‘3’ YES
  - Num>’3’ AND Let=‘a’ NO
  - Num>’3’ AND Let>‘b’ NO
  - Num>’3’
  - Let=‘e’
  - Let>’b’
  - Num=‘3’ OR Let=‘a’

Multi-attribute tree



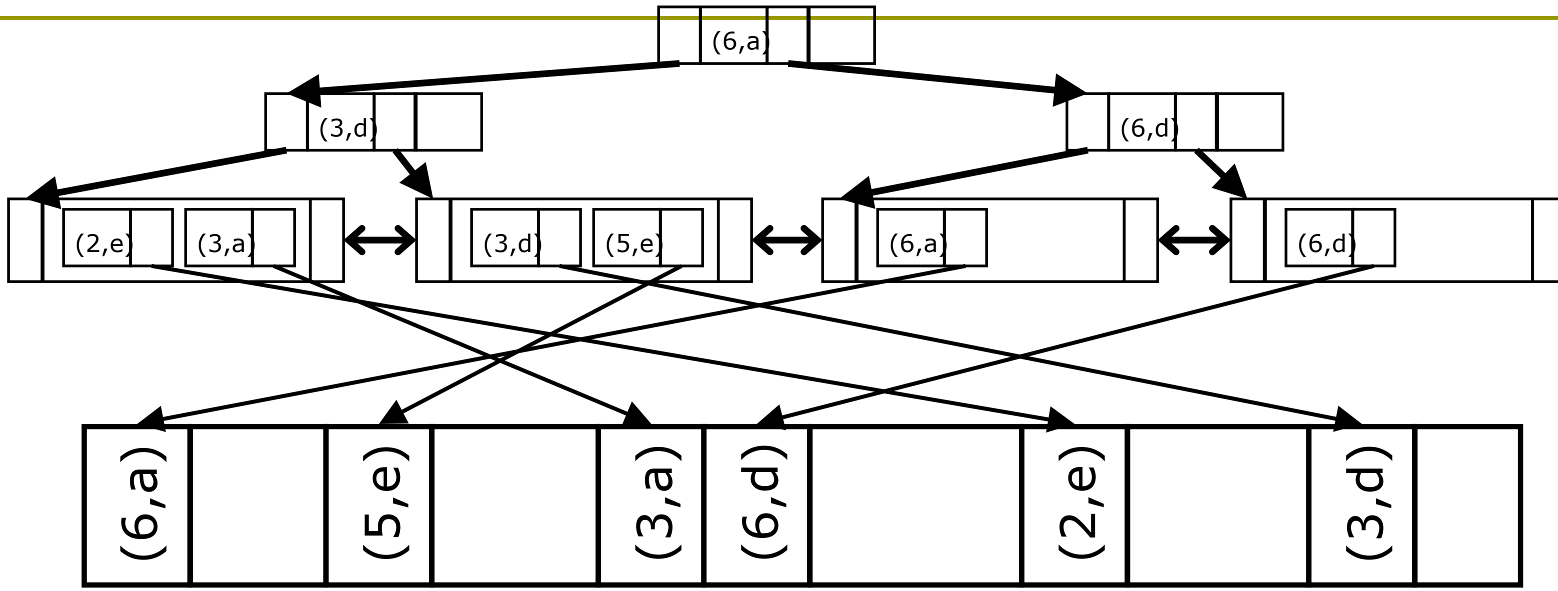
- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b' YES
  - Num='3' YES
  - Num>'3' AND Let='a' NO
  - Num>'3' AND Let>'b' NO
  - Num>'3' YES
  - Let='e'
  - Let>'b'
  - Num='3' OR Let='a'

Multi-attribute tree



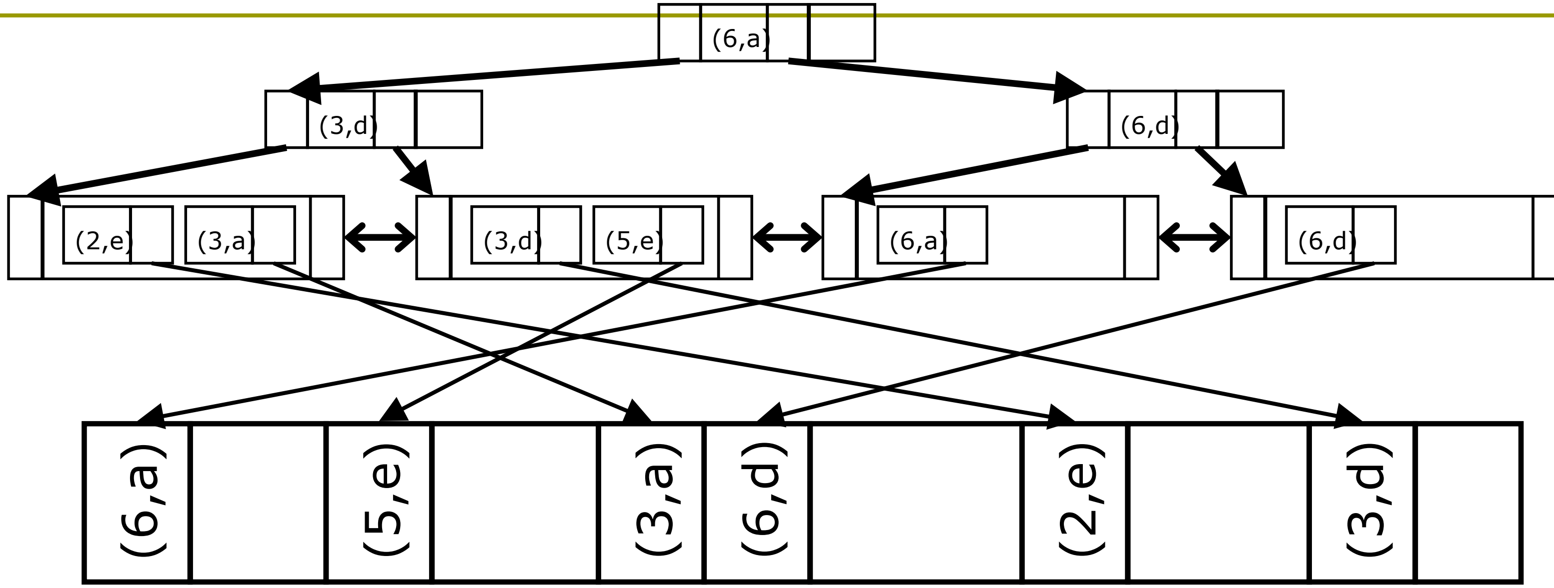
- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b' YES
  - Num='3' YES
  - Num>'3' AND Let='a' NO
  - Num>'3' AND Let>'b' NO
  - Num>'3' YES
  - Let='e' NO
  - Let>'b'
  - Num='3' OR Let='a'

Multi-attribute tree



- Queries:
  - Num=‘3’ AND Let=‘d’ YES
  - Num=‘3’ AND Let>‘b’ YES
  - Num=‘3’ YES
  - Num>’3’ AND Let=‘a’ NO
  - Num>’3’ AND Let>‘b’ NO
  - Num>’3’ YES
  - Let=‘e’ NO
  - Let>’b’ NO
  - Num=‘3’ OR Let=‘a’

Multi-attribute tree



- Queries:
  - Num='3' AND Let='d' YES
  - Num='3' AND Let>'b' YES
  - Num='3' YES
  - Num>'3' AND Let='a' NO
  - Num>'3' AND Let>'b' NO
  - Num>'3' YES
  - Let='e' NO
  - Let>'b' NO
  - Num='3' OR Let='a' NO

Index-only Query Answering (IOQA)

- Index-only query answering (IOQA) happens when we answer a query using indexing structures and **without** accessing any table file
- It can be used in three main cases: projections, aggregations and joins.

Examples follow:

- Projection

  SELECT age
  FROM people

  SELECT DISTINCT age
  FROM people

  We can create an index (either B+, hash or bitmaps) and answer the query from the index, without accessing the people table

- Aggregates

  SELECT MIN(age)
  FROM people

  SELECT AVG(age)
  FROM people
  WHERE department = 3

  For the first one, we can follow the same strategy as above. For the second one, we can create a multi-attribute index on people(department, age) – in this order!

Index-only Query Answering (IOQA)

Example cont'd:

- Joins

```
SELECT p.name
FROM people p, departaments d
WHERE p.id=d.boss;
```

We can create an index on department (boss) and another on people (id, name) to answer this query by means of IOQA

Summary

- Bitmap-index
- Multi-attribute index usage
- Index-only query answering

Bibliography

- D. Shasha and P. Bonnet. *Database Tunning*. Elsevier, 2003
- C. T. Yu and W. Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann, 1998
- P. Valduriez. *Join Indices*. ACM TODS, 12 (2), June 1987. Pages 218-246
- R. Ramakrishnan and J. Gehrke. *Database Management Systems*. 3rd edition. McGraw-Hill, 2003
- M. Golfarelli and S. Rizzi. *Data Warehouse Design*. McGrau-Hill, 2009