Optimization: Bitmap Indexes

Knowledge objectives

1. Justify the usefulness of a multi-attribute index
2. Explain what a bitmap index is
3. Justify the usefulness of a bitmap index
4. Explain how to find the record given a bitmap position based on the Hakan factor

Understanding Objectives

1. Decide when a multi-attribute index can be used in the evaluation of a predicate
2. Give the approximate cost of a selection on a table with a bitmap index (without coding)
3. Encode a bitmap index using run-length
4. Evaluate a simple predicate (one logical operation) using run-length encoded bitmap
5. Encode a bitmap index using bitmap-sliced technique
6. Evaluate a simple predicate (one logical operation) using bitmap-sliced technique

MULTI-ATTRIBUTE INDEXES

Usefulness of multi-attribute trees

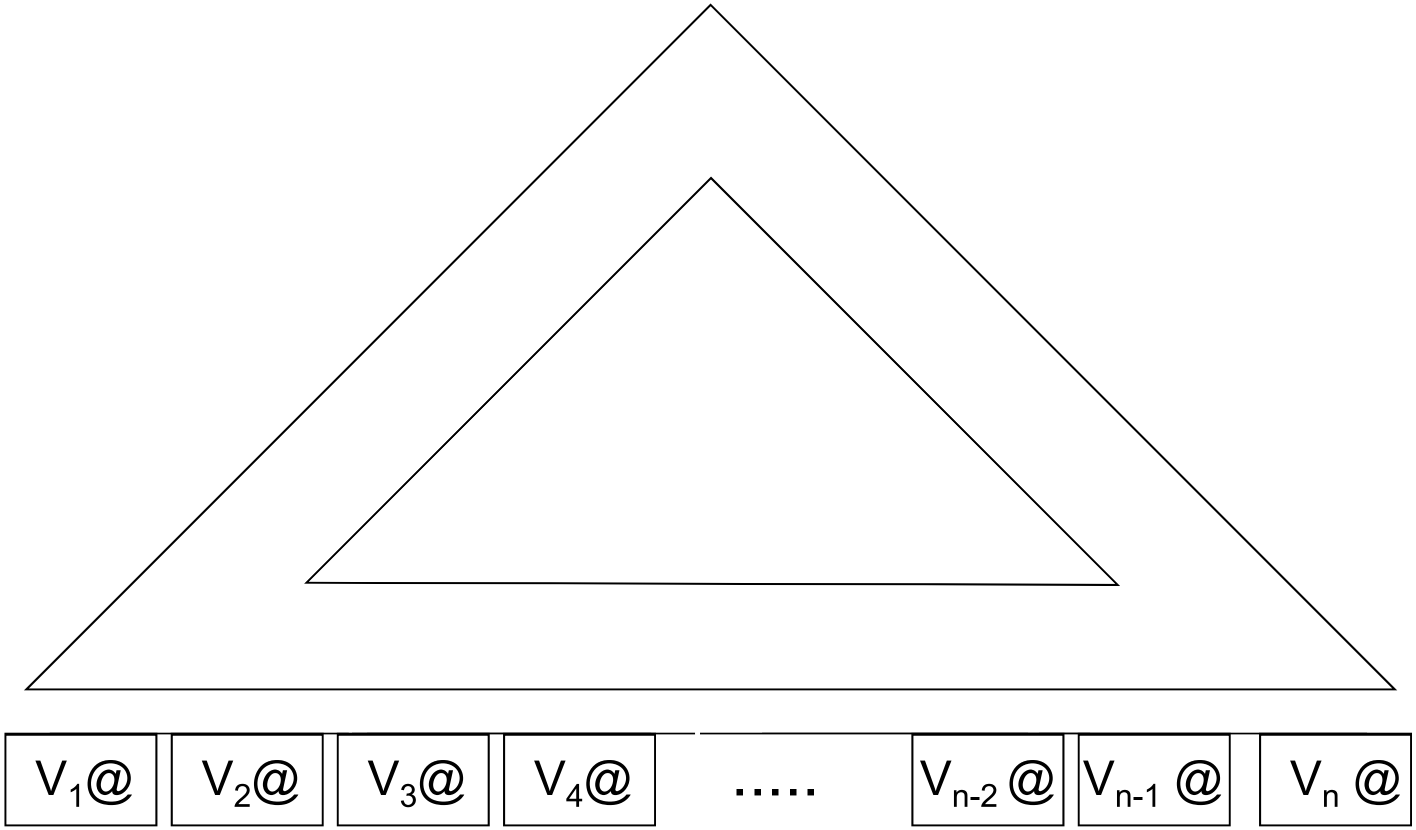
- Need more space
  - For each entry, keeps attributes A_1, .., A_k
  - May result in more levels, worsening access time
- Modifications are more frequent
  - Every time one of the attributes in the index is modified
- It is much more efficient than intersecting RID lists (to evaluate conjunctions)
- 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}

BITMAP INDEX

B-tree (I)

- Specially useful for simple queries
  - Without grouping, aggregations, or many joins
- Works better for very selective attributes (few repetitions per value)
  - Attributes in multidimensional queries are usually not very selective
- Order of attributes in the index is relevant
  - We can define as many indexes as we want
    - We can define only one Clustered index
    - For big tables, they may use too much space

B-tree (II)



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	OR	0	=	0	AND	0	=	0
1		0		1		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	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 several tuples (given one value): SF=1/ndist
      - $\lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-1)/ndist)^R)) \cdot D$
    - Search for several tuples (given several values): SF=v/ndist
      - $v \cdot \lceil |T|/bits \rceil \cdot D + (B \cdot (1-((ndist-v)/ndist)^R)) \cdot D$
    - Search for one tuple: SF=1/|T|
      - $\lceil |T|/bits \rceil \cdot D + D$

Comparison

- Better than B-tree 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 COMPRESSION

Bitmap compression variants

- Bitmap-encoded index
  - Create a look-up table coding values into integers
    - Facilitates coding hierarchies
      - 0 to 3 for Catalan provinces (mask 00xx)
      - 4 to 7 for Galician provinces (mask 01xx)
      - 8 to 15 for Andalucian provinces (mask 1xxx)
- Bit-sliced index
  - Code each value into binary
    - Each bit goes to the corresponding bit-vector
      - We only have $\lceil \log_2 ndist \rceil$ bit-vectors
  - ANDing either the bit or the negated for each column you can check the value:
    - A IN (1,3) $\Rightarrow$ (v_0=1 AND v_1=0) OR (v_0=1 AND v_1=1)
      $\Rightarrow$ v_0=1

Bitmap-encoded-sliced: example

Barcelona	Tarragona	Lleida	Girona
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

Encoding

Look-up table

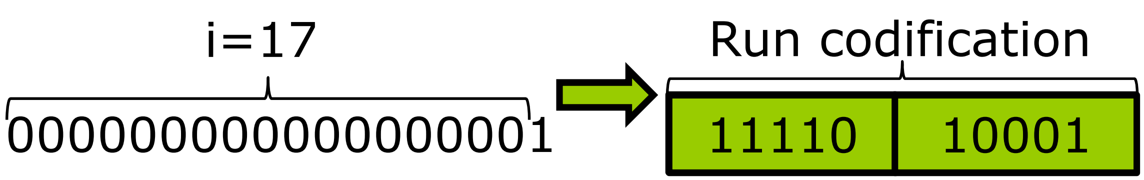
Barcelona	0
Tarragona	1
Lleida	2
Girona	3

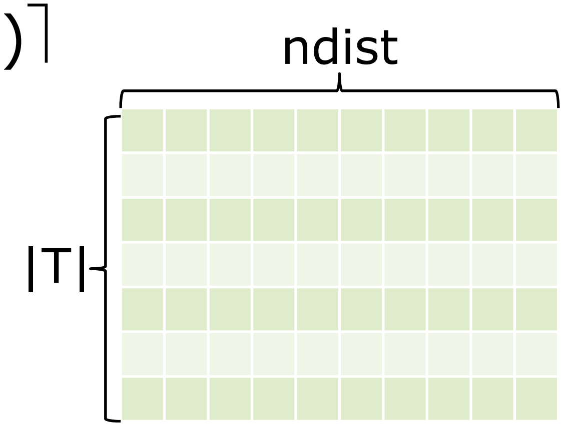
Slicing

2^1	2^0
0	0
0	0
1	1
1	0
0	1
0	0
1	1
0	1
0	0
0	0

Run-length encoding

- A "run" is a set of *i* zeros followed by "1"
  - We codify it by a suitable binary encoding of *i*
- Codification of a run
  a) Content: Binary coding (e.g., 17=$\overbrace{10001}^{n=5}$)
  b) Length: n-1 ones and a zero (e.g., 5=11110)
- Size gain
  - Size for coding *i*: $2\cdot\lceil \log_2 i \rceil$
  - Average value of *i*: $|T|\cdot(ndist-1)/|T| = ndist-1$
  - Size of a bitmap: $|T|\cdot 2\cdot\lceil \log_2(ndist-1) \rceil$





Run-length encoding example

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

Run-length →

Catalunya	León	Madrid	Andalucía
1	3	2	2
0	4	3	2
1	2		2
0	2		3
2			
3			
2			
2			
1			
0			

Binary →

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

Operating run-length encoded bit-vectors

- We need to decode
  - Not necessarily the whole vector at once
    - One run at a time

A OR B $\Rightarrow$ Being $l_A(i)$ *and* $l_B(j)$ the length of the runs plus one, we know that the $\Sigma_1^k l_X(i)$ record must be in the output, for X being both A and B

$\{a|\exists x, a=\Sigma_1^x l_A(i)\} \cup \{a|\exists x, a=\Sigma_1^x l_B(i)\}$

A AND B $\Rightarrow$ Being $l_A(i)$ and $l_B(j)$ the length of the runs plus one in both indexes, we must generate in the output:

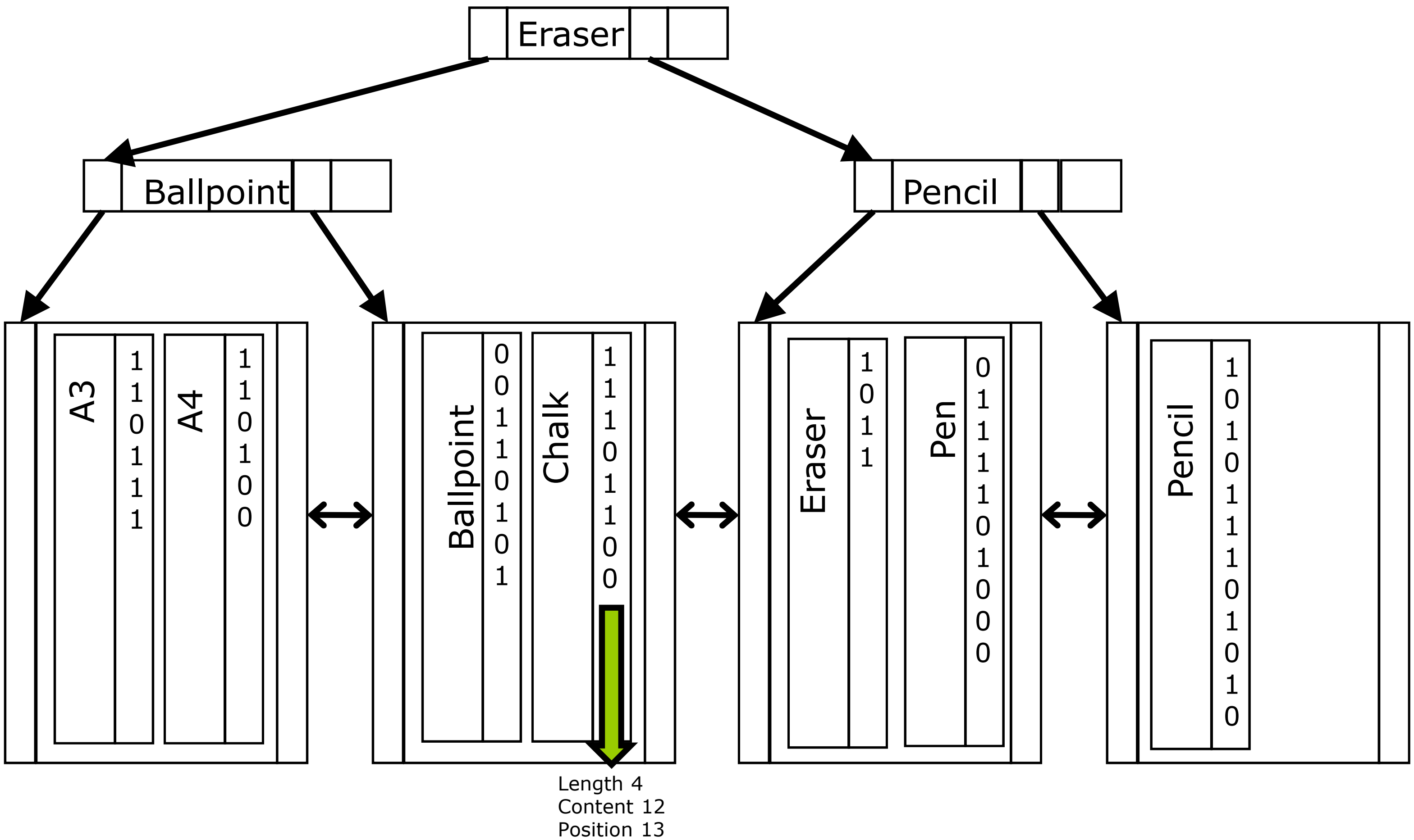
$\{a|\exists x\exists y, a=\Sigma_1^x l_A(i)=\Sigma_1^y l_B(j)\}$

Bitmap indexes in Oracle: Declaration

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

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

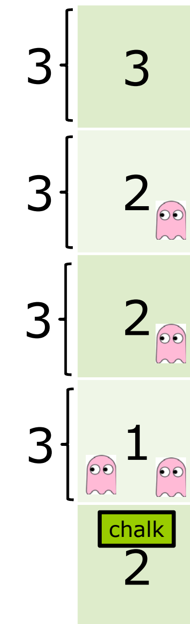
Bitmap-index in Oracle: Storage



Finding records of variable length

a) Maintain a secondary structure
- Having the position as key
- Pointing to the disk address of the record

b) Block-wise implementation
- Fix the number of bits used per block
- Use Hakan factor
  - Describes the expected maximum number of rows that a block can hold (e.g., 3)
  - Derived from column definition (and statistics)
    - Affected by
      - Number of columns
      - Column data types and lengths
      - NOT NULL constraints



Setting Hakan factor in Oracle

ALTER TABLE <name> [MINIMIZE | NOMINIMIZE] RECORDS_PER_BLOCK;

- The table cannot be empty
- The table cannot have any bitmap index

CLOSING

Summary

- Multi-attribute index
- Bitmap-index
  - Cost
  - Encoding
    - Bitmap-encoded
    - Bit-sliced
    - Run-length encoding

Bibliography

- P. Valduriez. *Join Indices*. ACM TODS, 12 (2), June 1987. Pages 218-246
- M. Golfarelli and S. Rizzi. *Data Warehouse Design*. McGrau-Hill, 2009
- H. Garcia-Molina, J. D. Ullman, and J. Widom. Database Systems. Prentice Hall, 2009