Parametrization and Tuning

Knowledge objectives

1. Explain the contents of the logic, virtual and physical spaces
2. Explain the correspondences between different levels in design, ANSI/SPARC architecture and DBMS objects' spaces
3. Explain the usefulness of extensions
4. Explain the usefulness of tablespaces
5. Draw the relationships between tablespaces, segments, files and extensions in Oracle
6. Name three user roles and explain how their work impacts database tuning
7. Name nine elements we should analyze regarding a query execution and say whether they are in the query plan or not
8. Exemplify system parameters
9. Name five table parameters
10. Explain the consequences of the choice for the fillfactor

Understanding objectives

1. Given an access plan, explain how the query would be executed and which algorithms it will use
2. Given an SQL sentence (giving rise, at most, to a process tree with one selection and one join nodes), find all the structures that may be used to improve its performance

Application objectives

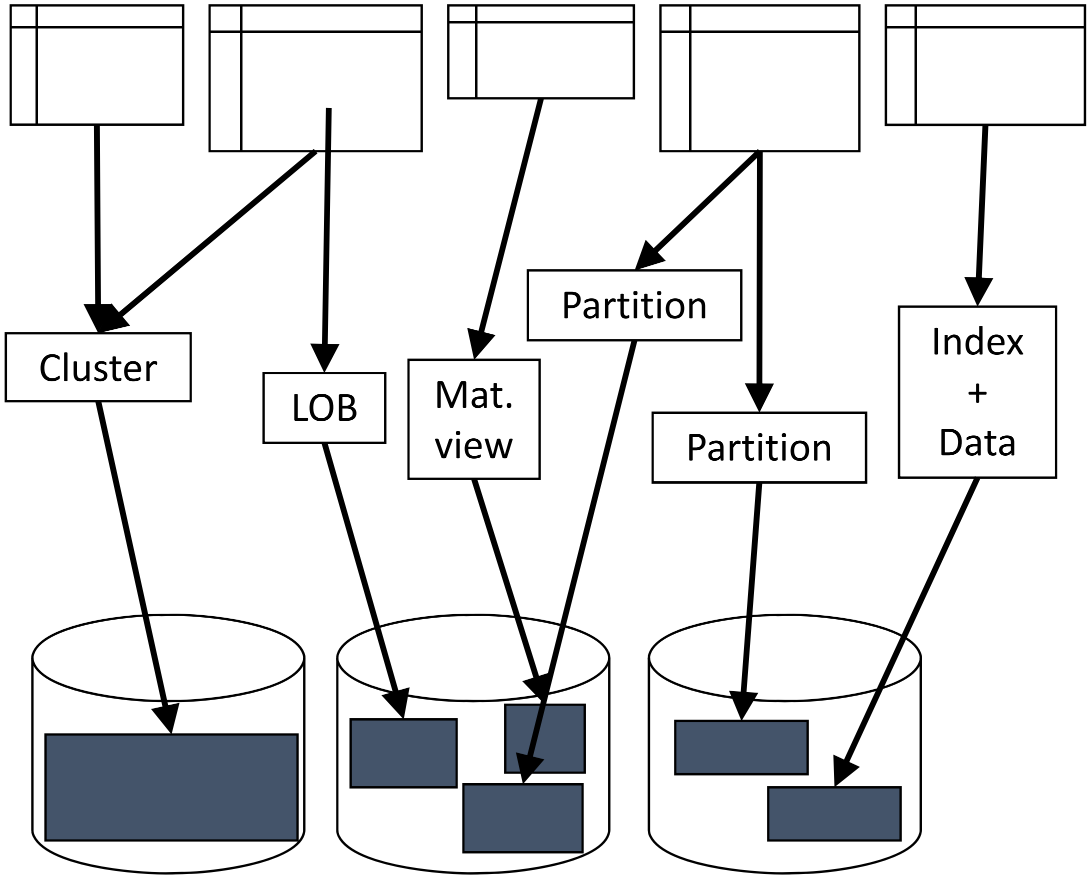
1. Given a workload, a set of tables including tuples and a constraint in terms of space, define the best structures for these tables that fit in the space and optimize the cost provided by Oracle's query optimizer

Tablespaces

Terminology

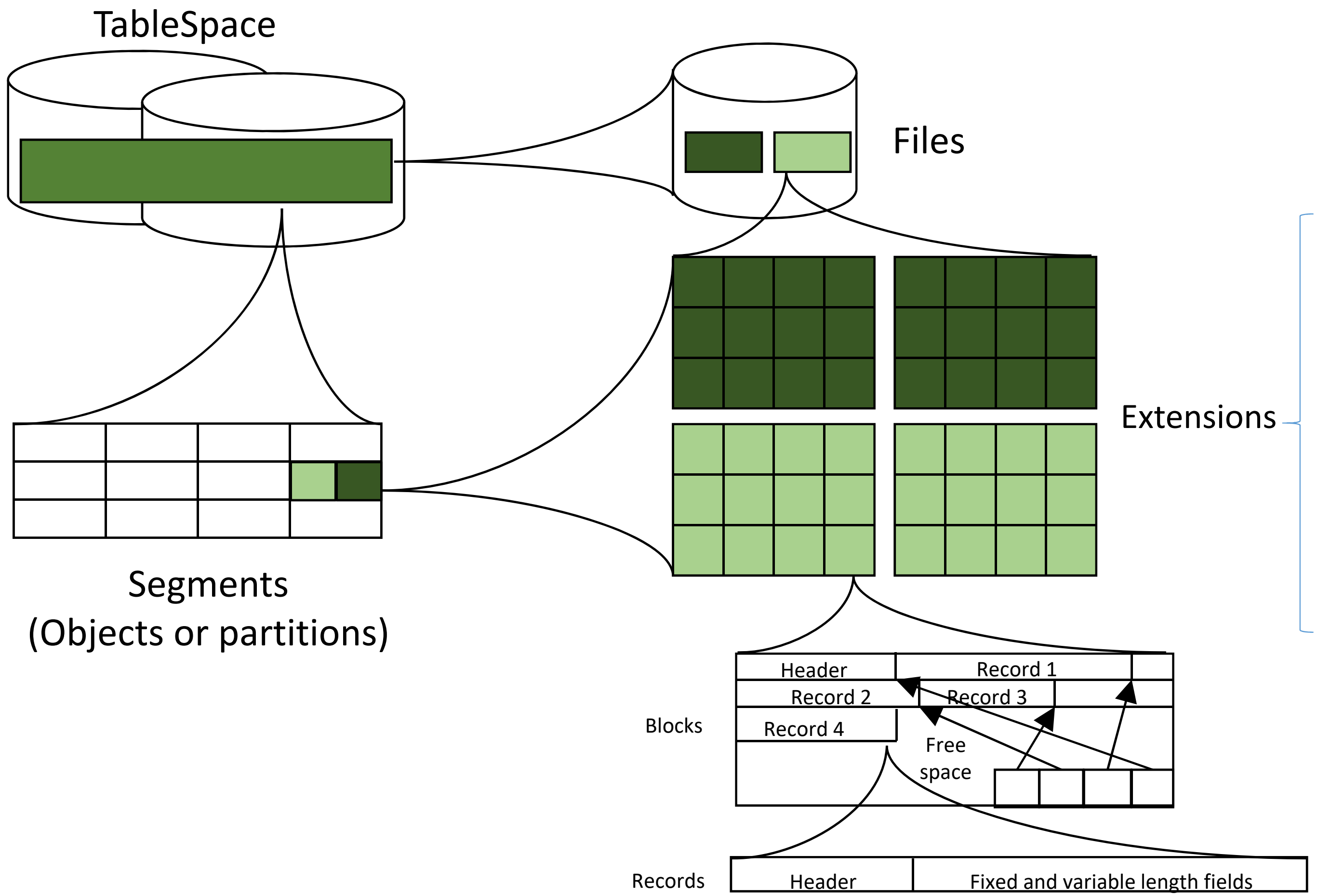
Design steps	ANSI/SPARC	Three spaces
Conceptual (classes)		
	External (views)	
Logic (relations)		
	Conceptual (tables)	Logic (tables)
		Virtual (tables+)
Physic (tables+)		
	Physic (files)	Physic (files)

Three spaces



- Logic space
  - Tables (relations)
    - Rows (tuples)
    - Columns (attributes)
- Virtual space
  - Pages
    - Records
      - Fields
  - Partitions
  - Views
    - Non-materialized
    - Materialized
  - Indexes
  - Clusters
  - Tablespaces
- Physical space
  - Files
    - Extensions
      - Blocks

Logical - Physical space relationship in Oracle



- Assigned to exactly one:
  - File
  - Segment
- Automatic acquisition
- Size
  - Much bigger than OS I/O buffer
  - Multiple of OS I/O buffer
- <u>Space physically consecutive</u>

Tablespaces

- Can be associated to several files (potentially in different storage devices)
  - Provides a theoretically unlimited DB size
  - Separate different access patterns to improve performance
- Fix a set of physical characteristics of database objects
  - Temporality
  - Logging
  - Block size
  - I/O cost
  - Extent management (in Oracle)
  - Segment management (in Oracle)

https://www.postgresql.org/docs/current/manage-ag-tablespaces.html

Number of tablespaces needed

- Catalog
- Atomic data and primary indexes
- Materialized views
- Secondary indexes
- Persistent stored modules
- Temporal
  - Used in the intermediate nodes of the process trees
- Rollback segment
  - If filled up, the transaction cannot modify anything else
  - Can be explicitly assigned to a transaction
- Audit

Parameters

Kinds of system parameters

- Allow to configure the behaviour of the system regarding:
  - File locations (e.g., data_directory, config_file, …)
  - Connection (e.g., max_connections, port, ssl, …)
  - Resource consumption (e.g., shared_buffers, work_mem, max_worker_processes, …)
  - Write Ahead Log (e.g., wal_buffers, wal_recycle, …)
  - Query planning (e.g., seq_page_cost, random_page_cost, cpu_tuple_cost, …)
  - Error reporting (e.g., log_filename, log_connections, …)
  - Run-time statistics (e.g., track_activities, track_oi_timing, …)
  - Automatic vacuuming (e.g., autovacuum, autovacuum_vacuum_threshold, …)
  - Lock management (e.g., deadlock_timeout, max_locks_per_transaction, …)
  - …

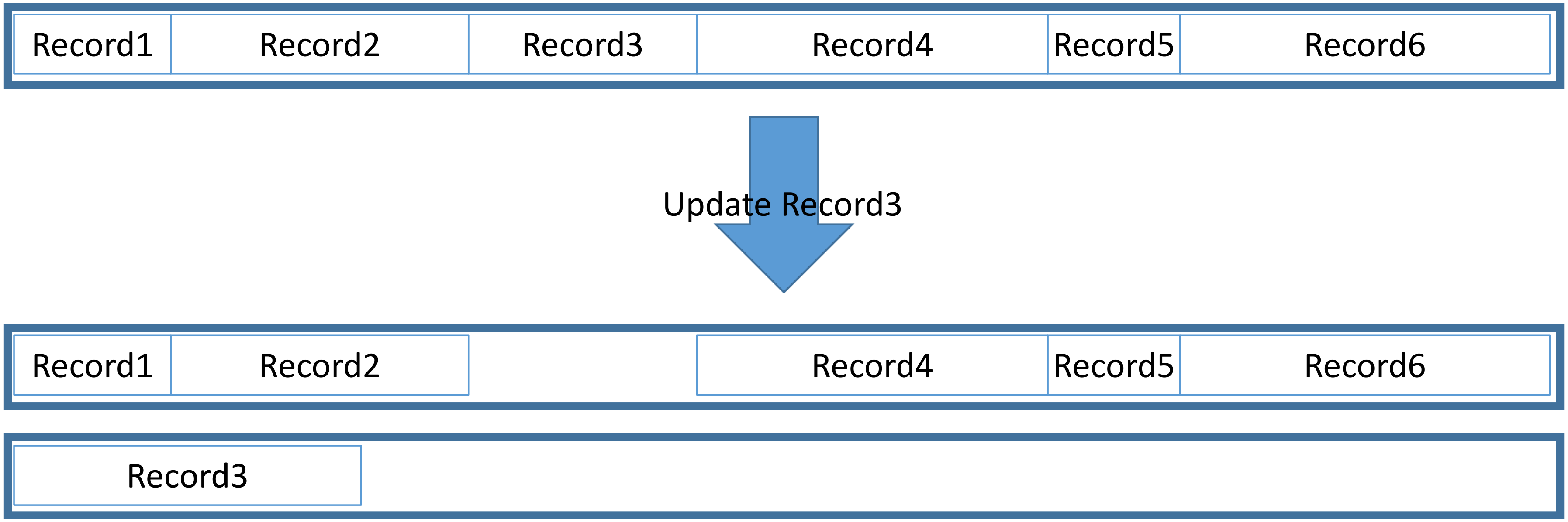
https://www.postgresql.org/docs/current/runtime-config.html

Table storage parameters

- parallel_workers
- autovacuum_enabled
- vacuum_truncate
- toast_tuple_target
- fillfactor

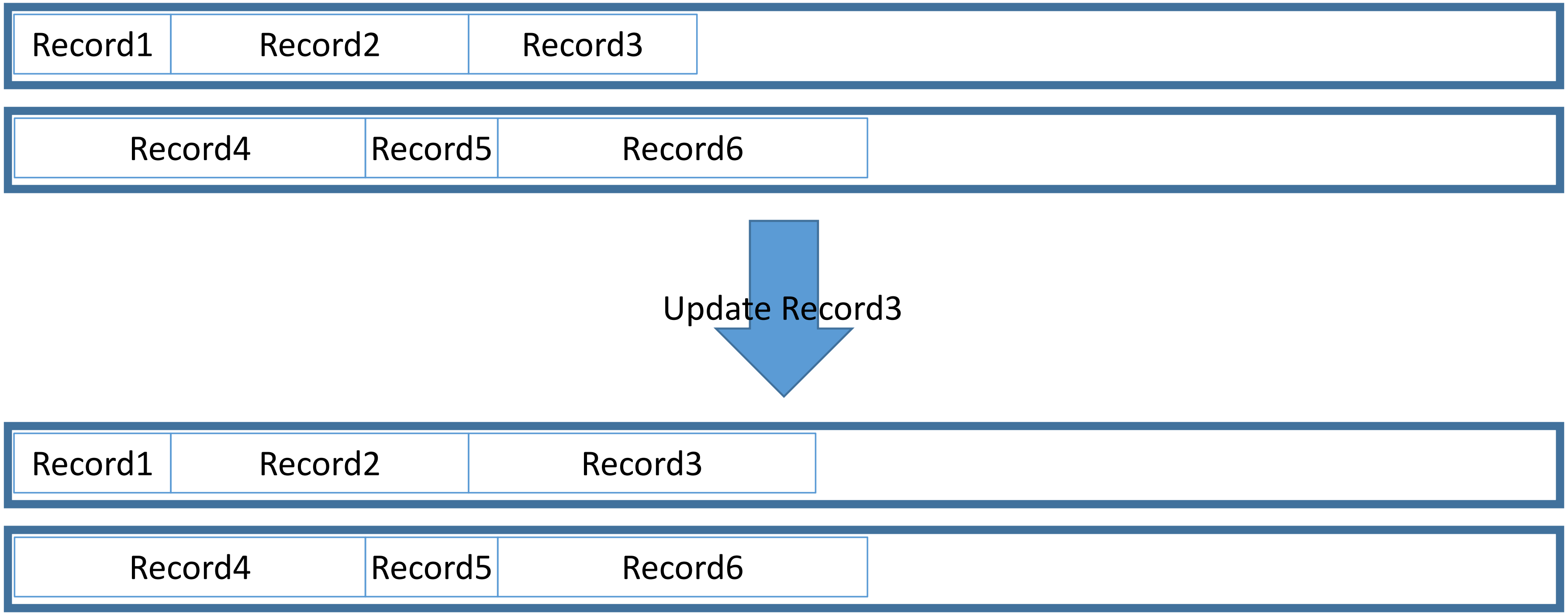
https://www.postgresql.org/docs/current/sql-createtable.html

Fillfactor 100%



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Fillfactor 60%



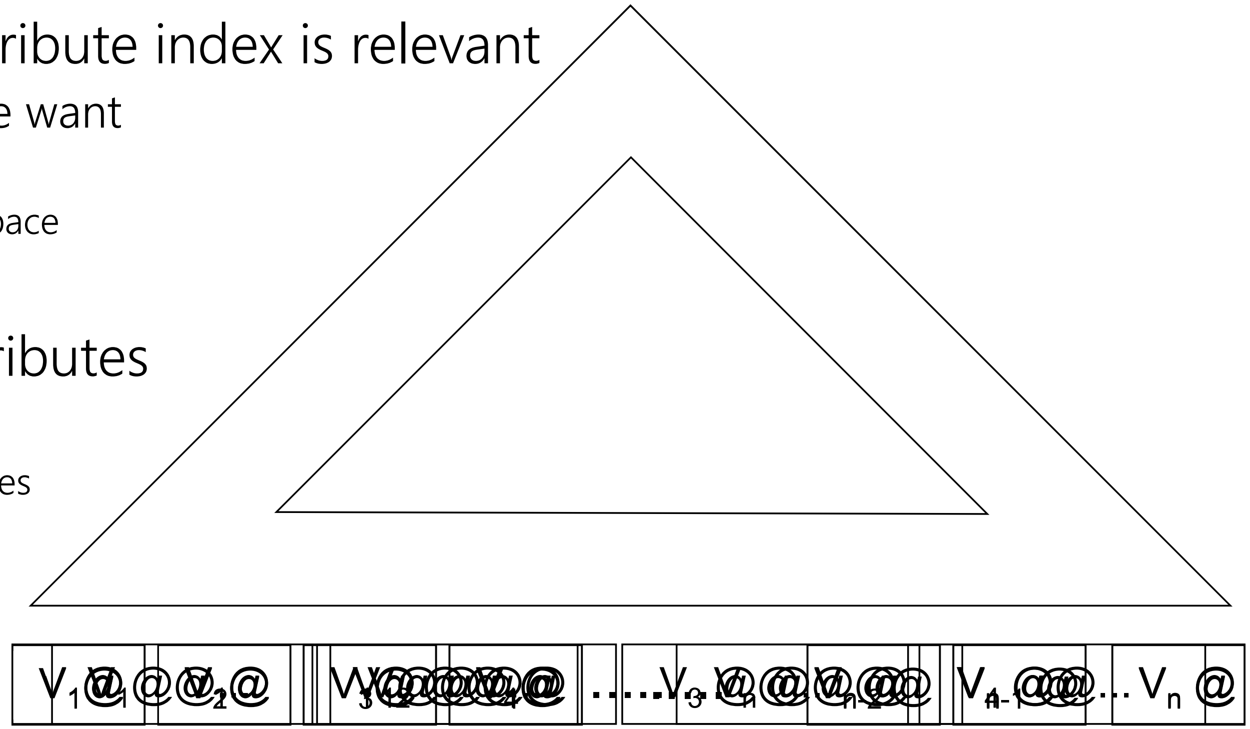
Bitmap indexes

Limitations of B-tree index

- Not useful for grouping, aggregations, or many joins

- Order of attributes in the multi-attribute index is relevant
  - We can define as many indexes as we want
    - Only one Clustered index
    - For big tables, they may use too much space

- Works better for very selective attributes
  - Few repetitions per value
    - This is not true in multidimensional queries



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

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

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

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

- Probability of some tuple in a block fulfilling P
  $1-(1-SF)^R$

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

Bitmap vs B-tree

- 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
- Easy indexing of 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

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

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

Tuning

Tuning

- Definition: It is the activity of making a DB application run faster
- People involved:
  - Administrator
    - Defines system parameters
      - DBMS
      - OS
      - Hw
  - Designer
    - Defines DDL sentences
  - Application programmer
    - Defines DML sentences
- Tools involved:
  - Catalog
    - Statistics
  - Query plans

Performance improvement given a workload

- Input
  - Available space
  - Workload
    - List of queries (with frequencies)
    - List of modifications (with frequencies)
  - Performance objective
    - Total
    - Per query
- Output
  - Set of structures
    - B-tree
    - Hash
    - Clustered index
    - Clustered structure
    - Bitmap
  - Normalization/Denormalization of tables
  - Partitioning of every table
  - Set of materialized views

What matters in the query execution

- In the access plan:
  - Access path for each table
  - Algorithms used for each operation
  - Operation order
  - Usage of the temporal area
    - Intermediate results
    - Sorting
    - Hashing
  - I/O vs CPU cost
- Not in the access plan:
  - Logic vs Physical disk accesses (i.e., cache hits)
  - Number of locks
  - Number of deadlocks/timeouts
  - Time in the locking queues

Combinatorial explosion of indexes

Finding the best set of indexes is computationally complex, because we should take into account:

a) Different kinds of indexes
   - For a database with *t* tables, *a* attributes each, and considering 5 kinds of structures, we can define $5 \cdot t \cdot a$
b) Multi-attribute indexes
   - For a table with *n* attributes we can define $n!/(n-c)!$ different indexes of *c* attributes
c) Modifications (not only queries)
d) Incompatibilities between structures
e) Constrains
   1) Space
   2) Maintenance time

Heuristics to choose indexes (I)

1. A non-clustered index will never worsen a query
   - A non-clustered index may be just ignored in a query
   - An index could improve or worsen a modification
2. The smaller a table, the more useless its indexes
   - Proportionally, they will use too much space
   - They may generate even more accesses
   - Sequential disc access will make the difference
3. An index should improve, at least, one statement
   - If it improves more than one, much better
   - Do not forget modifications
4. Look at the predicate
   - Equality suggests Hash, and does not discard B+ nor Bitmap
   - A range suggests B+ (or Bitmap), and discards Hash
   - Many repetitions suggest Bitmap, and discard B+ and Hash

Heuristics to choose indexes(II)

5. Consider multi-attribute indexes (attribute order matters)
    - The attributes must belong to the same table
    - They may allow to answer a query by themselves (no table access)
    - Many mono-attribute bitmap indexes will be more flexible
6. Consider Clusters
    - A table can have, at most, one
    - Range (or repetitions) queries are clear candidates
    - If the associated B+ is enough, the cluster is useless
7. Choose between Hash and B+
    - Better Hash if used in a join algorithm (Row Nested Loops)
    - Better Hash for HUGE tables
    - Hash is useless for range conditions
    - Better B+ than Hash if we have distribution problems
        - E.g., too many repetitions
8. Choose between B+ and Bitmap
    - Better Bitmap in terms of performance
        - Specially with many repetitions
    - Better B+ in terms of space if the index has not many repetitions
    - Better B+ in scenarios with many concurrent modifications

Candidate indexes to be created

Critical query

SELECT name, age, salary
FROM people
WHERE department ='CS' AND age>40;

Useful

B+ over department and age

Useless

Hash over age

Algorithm to choose among candidates

Greedy algorithm :

Do

1) Consider those candidate indexes that fit in the available space and update time
2) Sort indexes based on the performance improvement they induce
3) Materialize first index in the list, if it improves performance

While performance improved and there is available space and update time

- Modify the set of indexes as user needs evolve

Example of index selection

Example of index selection (I)

- D = 1 sec; C = 0 sec
- Table information:
  - $B_{Authors}$=5,000
  - $R_{Authors}$=4
  - B_{Books}=10,000
  - R_{Books}=10
- Attribute information:
  - Ndist(theme)=100
  - Ndist(author)=20,000
  - Ndist(name)=20,000
- Available structures:
  - B+ (order 75)
  - Clustered
  - Hash (with 0 sec of execution time for hash function)
  - Clustered structure
- Available join algorithms:
  - Hash Join
  - Sort-Match
  - Clustered Structure scan
- Memory pages
  - Hash Join: 102
  - Sort: 101
- Query frequencies:
  - Q1 (60%): SELECT * FROM books WHERE theme=X;
  - Q2 (30%): SELECT * FROM authors WHERE name=Y;
  - Q3 (10%): SELECT * FROM books b, authors a WHERE b.author = a.name;
- Available disk space: 22,000 blocks

Books (title, author, theme, …)

Authors (name, …)

Example of index selection (II)

Costs without indexes:

- Time: 11,250 (10,000·60%+ 2,500·30%+ 45,000·10%) sec/query
- Space: 15,000 blocks

Q1 (60%): SELECT * FROM books WHERE theme=X;
Q2 (30%): SELECT * FROM authors WHERE name=Y;
Q3 (10%): SELECT * FROM books b, authors a WHERE b.author = a.name;

		Overspace	Q1 (60%)	Q2 (30%)	Q3 (10%)			
					HJ	SM	Scan	Avg
Books	B+ (theme)	1011	1012	2500	45000	75000		5857
	Clustered (theme)	6011	153	2500	50000	80000		5842
	Clustered (author)	6011	15000	2500	50000	40000		13750
Authors	B+ (name)	203	10000	3	45000	75000		10501
	Clustered (name)	2703	10000	3	47500	57500		10751
	Hash(name)	168	10000	2	45000	75000		10501
~~Both~~	~~Clustered Structure~~	~~7500~~	~~22500~~	~~11250~~			~~22500~~	~~19125~~

Example of index selection (III)

Costs if there is a Clustered index for books.theme:

- Time: 5,842 sec/query
- Space: 21,011 blocks

Q1 (60%): SELECT * FROM books WHERE theme=X;
Q2 (30%): SELECT * FROM authors WHERE name=Y;
Q3 (10%): SELECT * FROM books b, authors a WHERE b.author = a.name;

		Overspace	Q1 (60%)	Q2 (30%)	Q3 (10%)		
					HJ	SM	Avg
Authors	B+(name)	203	153	3	50000	80000	5093
	~~Cluster(name)~~	~~2703~~	~~153~~	~~3~~	~~52500~~	~~62500~~	~~5343~~
	Hash(name)	168	153	2	50000	80000	5092

Costs if there is a Clustered index for books.theme and a Hash for authors.name:

- Time: 5,092 sec/query
- Space: 21,179 blocks

Closing

Summary

- Three spaces
- Files, extensions, blocks, records and fields
- Parameters
  - Fillfactor
- Bitmap indexes
- Tuning
  - Workload-based index selection

Bibliography

- J. Sistac et al. *Tècniques avaçades de bases de dades*. EDIUOC, 2000
- D. Shasha and P. Bonnet. *Database Tuning*. Elsevier, 2003
- R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007
- M. Golfarelli and S. Rizzi. *Data Warehouse Design*. McGrau-Hill, 2009