Physical space and workload optimization

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

Understanding Objectives

1. Given an access plan generated by Oracle, 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

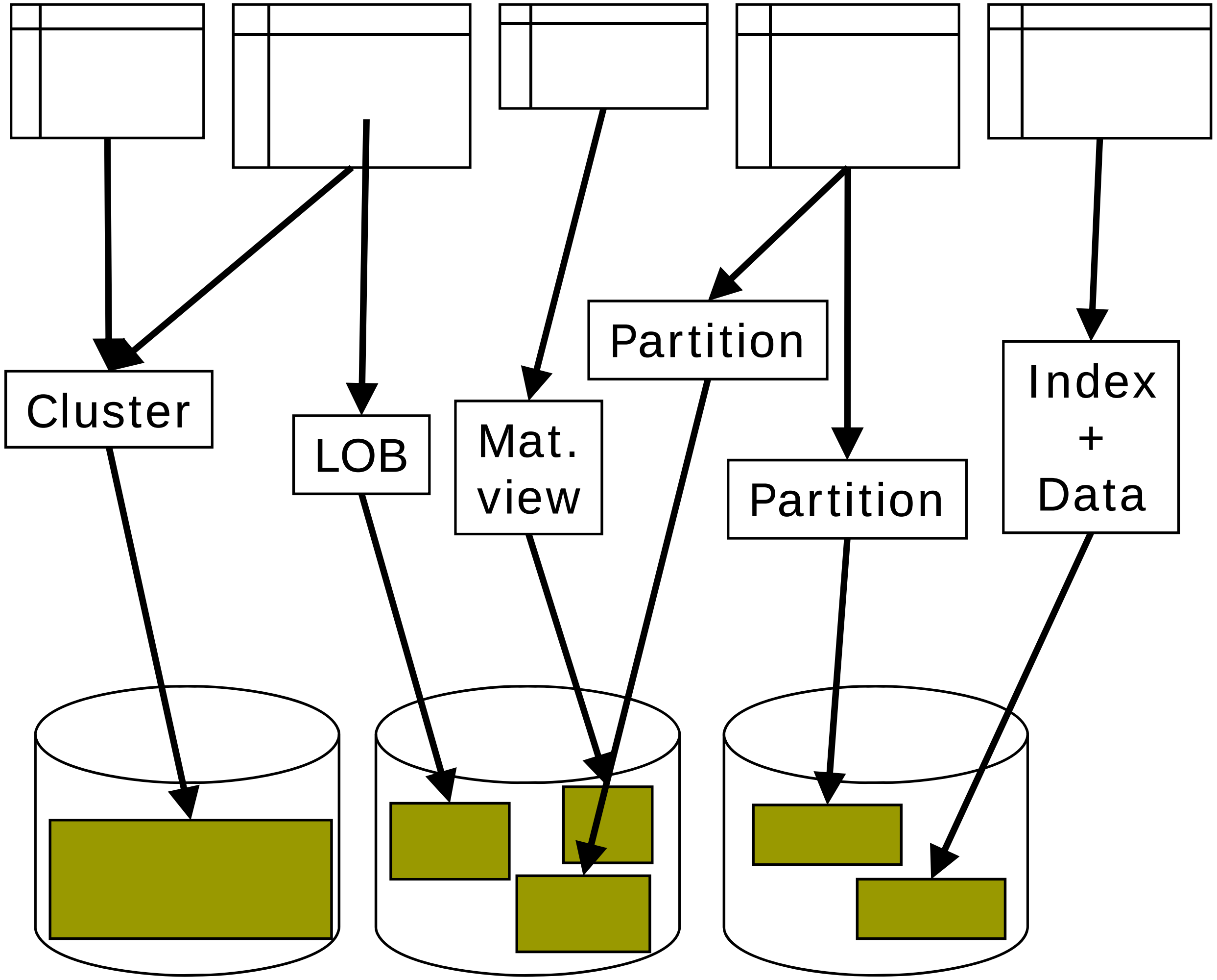
Application Objectives

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

Terminology

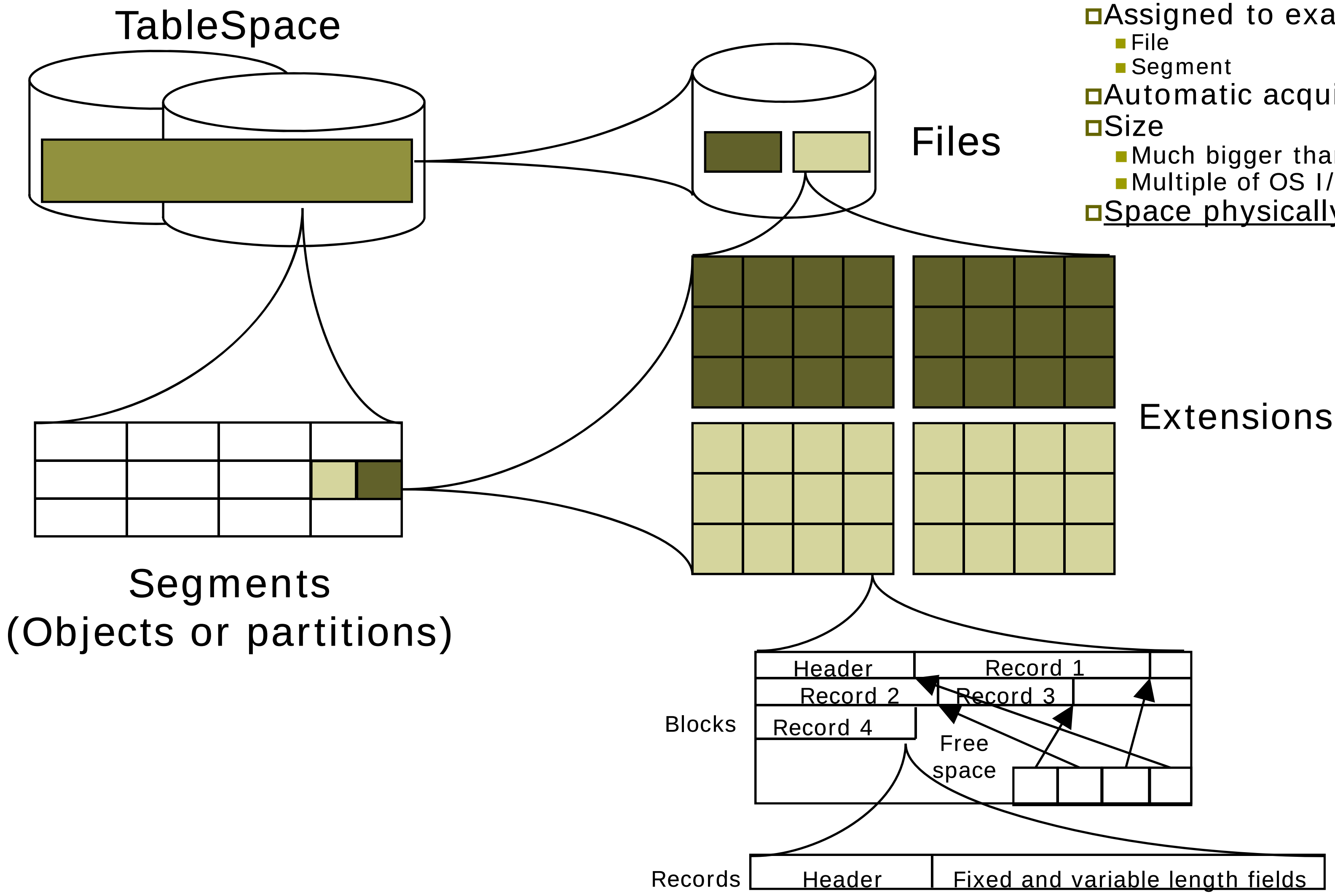
Design steps	ANSI/SPARC	Three spaces
Conceptual (classes)		
	External (views)	
Logic (relations)		
	Conceptual (tables)	Logic (tables)
Physic (tables+)		Virtual (tables+)
	Physic (files)	Physic (files)

Three spaces



- Logic space
  - Tables (relations)
    - Rows (tuples)
    - Cols (attributes)
- Virtual space
  - Pages
    - Records
      - Fields
  - Partitions
  - Views
    - Not materialized
    - Materialized
  - Indexes
  - Clusters
  - Tablespaces
- Physical space
  - Files
    - Extensions
      - Blocks

Logical - Physical space relationship



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

Tablespaces

- Can contain several files (potentially in different storage devices)
  - Provides a theoretically unlimited DB size
- Fix a set of physical characteristics of database objects
  - Temporality
  - Logging
  - Block size
  - Extent management
  - Segment management

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

Tuning

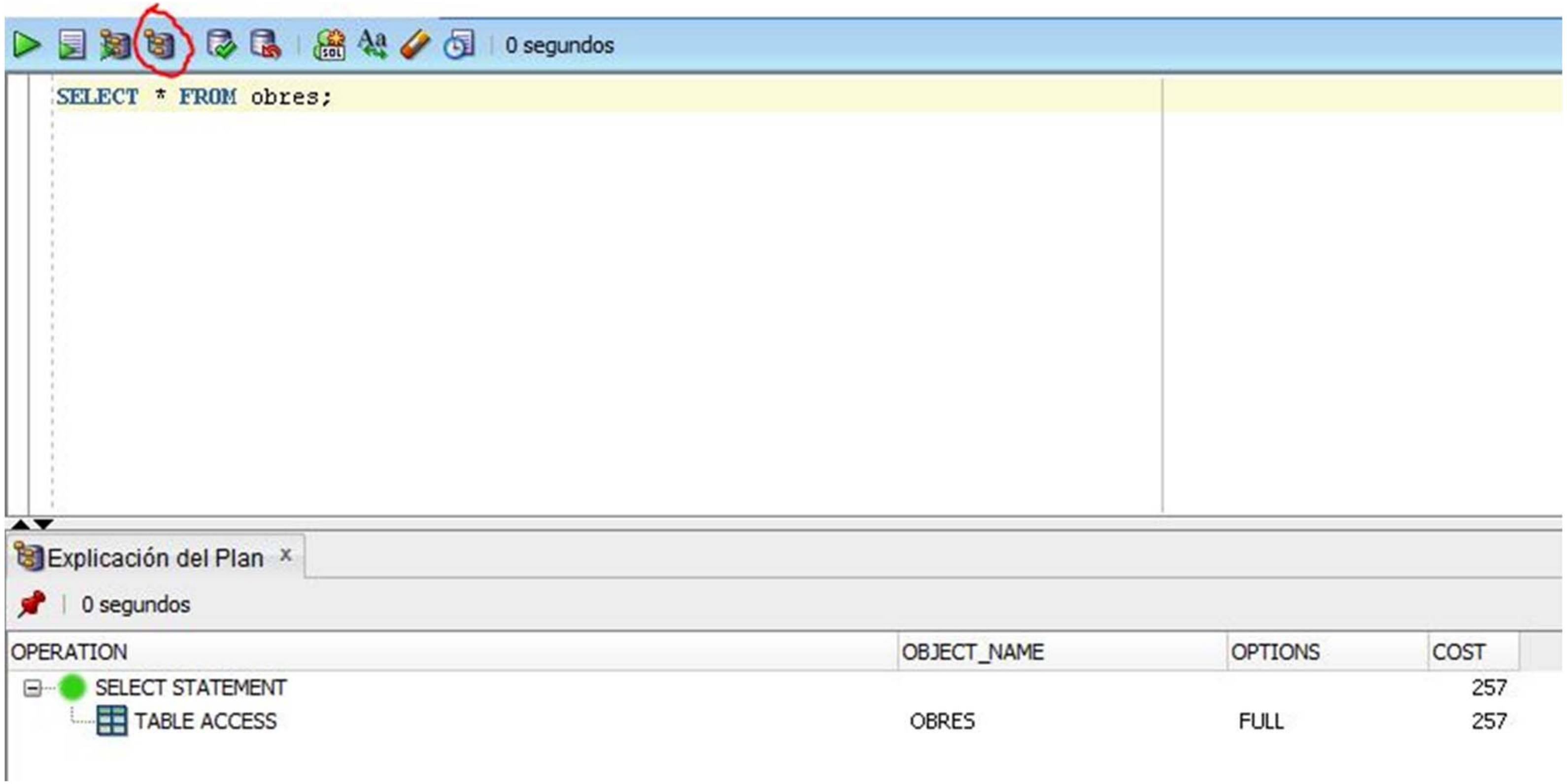
- Definition: It is the activity of making a DB application run faster
- People involved:
  - Administrator
    - Defines system parameters
      - DBMS
      - OS
      - Hw
  - Designer
    - Defines DDL sentences
  - Application programmer
    - Defines DML sentences
- Tools involved:
  - Catalog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

Example of query plan in Oracle 10g

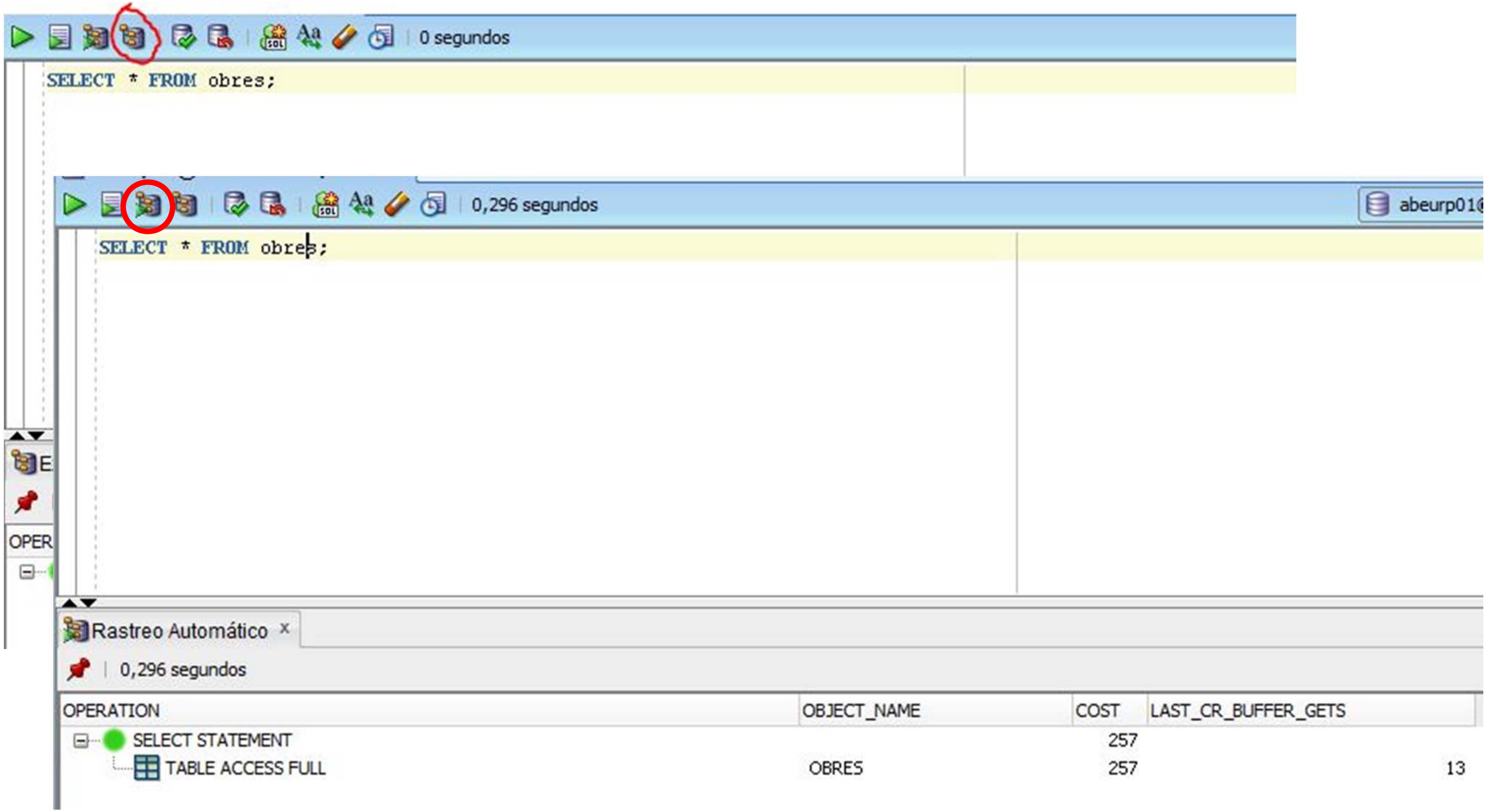
```
CREATE TABLE departments (          id INTEGER PRIMARY KEY);
CREATE TABLE employees (            id INTEGER PRIMARY KEY,
                                    dpt INTEGER REFERENCES departments,
                                    name CHAR(256));
EXPLAIN PLAN SET STATEMENT_ID='my_st' INTO my_table FOR
    SELECT *
    FROM employees e, departments d
    WHERE e.dpt=d.id AND d.id>1;

SELECT plan_table_output FROM table(dbms_xplan.display('my_table', 'my_st', 'typical'));
```

```
---------------------------------------------------------------------------------
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              | 20000 |  5156K|   690   (1)| 00:00:09 |
|   1 |  NESTED LOOPS        |              | 20000 |  5156K|   690   (1)| 00:00:09 |
|*  2 |   INDEX RANGE SCAN   | SYS_C006766  |     2 |     4 |     1   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL  | EMPLOYEES    | 10000 |  2558K|   345   (1)| 00:00:05 |
---------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("D"."ID">1)
   3 - filter("E"."DPT">1 AND "E"."DPT"="D"."ID")
```

Example of plan in SQLDeveloper



Example of plan in SQLDeveloper



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
  - Number of locks
  - Number of deadlocks/timeouts
  - Time in the locking queues

Performance improvement given workload

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

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 $4 \cdot t \cdot a$
  b) Multiattribute indexes
     - For a table with *n* attributes we can define $n!/(n-c)!$ different indexes of up to *c* attributes
  c) Modifications (not only queries)
  d) Incompatibilities between structures
  e) Constrains
     a) Space
     b) Maintenance time

Rules to improve query performance (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

Rules to improve query performance (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
     - Eg: 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

Usefulness of an index

Critical query

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

Useful

B+ over department and age

Useless

Hash over age

Solution to the indexes explosion

- Greedy algorithm:
  1. Do
     a. Consider those candidate indexes that fit in the available space (and maintenance time)
     b. Sort indexes based on the performance improvement they induce
     c. Create first index in the list if it improves performance

     While performance has been improved and there is space

- Modify the set of indexes as user needs evolve

Example of index selection (I)

Books (title, author, theme, ...)

Authors (name, ...)

- 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

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 (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 (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

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

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

Summary

- Three spaces
- Files, extensions, blocks, records and fields
- Tuning
  - Index selection based on workload

Bibliography

- Jaume 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