UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Project on

Database Optimization

Submitted by

Andre Nogueira
Mohana Fathollahi

November 15, 2020

Table of Contents

1. Optimizations of the database given a single query

The goal of this part is to build data base structures to optimize performance of the data base, reducing the running time cost. Tree questions with the highest marks was selected in this part to describe the access path and the best indexing method.

1.1 Question 1

Do the physical design so that the execution of the following query is optimal:

SELECT sum (pressupost)
from obres
where id = 500;

In this query selects just one tuple, once id is composed by unique values, so the access path would be search one tuple. Based on Table 1-1, we have 1000 rows so search in these rows without indexing is time consuming.

Table 1-1: General information

NUM_ROWS	AVG_ROW_LEN
1,000	885

After running Btree, cluster index and hash, we found that hash has a lower cost when compared with other methods. That is because in Oracle, records are stored in the buckets of the hash instead of the addresses and when the index is small enough Oracle keeps the hash function in memory, all of these reasons makes hash cost the one with the lowest cost. So, in front of small number of repetition a hash index is the best option.
Finally the result of running hash is presented in Table 1-2 with cost of zero.

Table 1-2: Execution plan-hash

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	0	1	7
SORT (AGGREGATE)			0	1	7
TABLE ACCESS (HASH)	OBRES	ANALYZED	0	1	7

1.2 Question 2

Do the physical design so that the execution of the following query is optimal:

SELECT sum(pressupost)
from obres
where id > 5 and id < 800;

In this question, we have a range in the predicate and therefore, the access path would be search several tuples. The number of tuples that exist in this range is 796 that approximately cover 80 percent of table. In this case, a table scan is likely to be much faster because it requires fewer seeks. Result of running are presented in Table 1-3.

Table 1-3: execution plan-cluster index

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼SELECT STATEMENT		ALL_ROWS	113	1	7
▼SORT (AGGREGATE)			113	1	7
TABLE ACCESS (FULL)	OBRES	ANALYZED	113	796	5,572

1.3 Question 3

Do the physical design so that the execution of the following query is optimal:

SELECT sum(pressupost)
from obres
where id >= 5 AND id <= 10;

In this question the predicate has a range but not as large as the one from question number 2, in this range we have just 6 rows, which is approximately 0.6 percent of the data, so the access path is search several tuples.
In this question table scan is not useful because the range of data is small. After running Btree (result presented in Table 1-4) and cluster index (result presented in

Table 1-5), we can see that the cost of using the cluster index is lower than using Btree. Because when using cluster index we will have ordered data that works faster for this query.
Additionally using hash index is not useful because indexes are not small and cost of using this method will be much more than Btree and cluster indexes.

Table 1-4: Execution plan-Btree

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼SELECT STATEMENT		ALL_ROWS	9	1	7
▼SORT (AGGREGATE)			9	1	7
▼TABLE ACCESS (BY INDEX ROWID BATCHED)	OBRES	ANALYZED	9	7	49
INDEX (RANGE SCAN)	BTRE	ANALYZED	2	7	0

Table 1-5: Execution plan_Cluster index

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼SELECT STATEMENT		ALL_ROWS	3	1	7
▼SORT (AGGREGATE)			3	1	7
INDEX (RANGE SCAN)	SYS_IOT_TOP_1387699	ANALYZED	3	7	49

2. Optimizations of the database given a single query

The goal of this part is to build data base structures to optimize performance of the data base given a workload composed by queries and their frequencies, reducing the running time cost. One question with the highest mark was selected in this part to describe the access path and the best indexing method.

2.1 Question 1

Given the tables and data from the attached file (where you will also find the sentences to look at the queries' cost), do the physical design of the database so that the execution of the following commands is optimal (the frequency of execution of each command is indicated in parentheses):

- (25%) SELECT * FROM empleats WHERE nom=TO_CHAR(LPAD('MMMMMMMMMM',200,'*'));
- (03%) SELECT nom FROM empleats WHERE sou>1000 AND edat<20;
- (25%) SELECT * FROM empleats e, departaments d, seus s WHERE e.dpt=d.id AND d.seu=s.id;
- (47%) SELECT * FROM departaments WHERE seu=4;

Take into account that you can only use 1740 disk blocks overall.

The following figure presents the tables created.

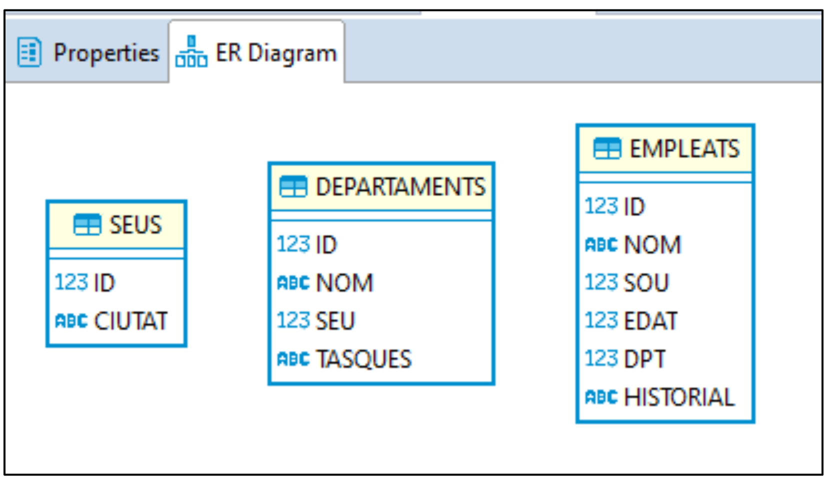


Figure 1 - Tables WL_1

The following table presents statistical and size information about each table.

Table 2-1: Tables Statistics data

	TABLE_NAME	BLOCKS	NUM_ROWS	AVG_ROW_LEN
1	SEUS	2	10	44
2	EMPLEATS	1,182	13,000	718
3	DEPARTAMENTS	434	1,300	2,209

The total number of blocks occupied, without indexes is the following:

Table 2-2 - Tables Statistics data – Total

	BLOCKS
1	1,680

The limit of blocks defined is 1740, letting 60 blocks for creating additional structures.

2.1.1 Preliminary analysis: Choosing Indexes

Considering the workload presented each query was evaluated and indexes tested, as following.

- (25%) SELECT * FROM empleats WHERE nom=TO_CHAR(LPAD('MMMMMMMMMM',200,'*'));

There is no row in the data that fulfill this predicate. Therefore, In the preliminary analysis it is assumed that the DBMS will perform a table Scan. For a table Scan, the best choice is not using any index.

- (03%) SELECT nom FROM empleats WHERE sou>1000 AND edat<20;

This query is searching for several tuples with ranges. A cluster index is not possible, once ‘edat’ and ‘sou’ are not unique values. Hash function is not very good for ranges. Therefore, it was tested only 2 structures, Btrees and Bitmaps in both rows. The indexes in ‘sou’ had no significant effect while the Bitmap index in empleats(edat) presented an improvement of 4% in the cost of this Query, with additional 12 blocks. Although the workload indicates that this query is not very frequent, the additional size in blocks is small and fits in the limitation.

- (25%) SELECT * FROM empleats e, departaments d, seus s WHERE e.dpt=d.id AND d.seu=s.id;

This query involves joining tables respecting the equality conditions. Several possibilities were tested with good results for Hash indexes in departaments(seu), however this structure exceeds the block limit defined. None of the other options resulted in significant cost reductions for this query.

- (47%) SELECT * FROM departaments WHERE seu=4;

This query is a selection of several tuples with an equality. A Hash function could be a good option, however its size exceeds the block limit. A Btree and a Bitmap index were tested, with the Bitmap departaments(seu) presenting better results.

2.1.2 Indexes Created

Based on several tests and in the preliminary analysis presented the following indexes were selected to optimize the workload:

```sql
CREATE BITMAP INDEX index1 ON empleats (edat) PCTFREE 0;
CREATE BITMAP INDEX index2 ON departaments (seu) PCTFREE 0;
```

A Bitmap index in the row 'edat' of the table 'emplats' and another Bitmap index in the row 'seu' of the table 'departments'.

2.1.3 Access plan of each query on the Workload

- (25%) SELECT * FROM empleats WHERE nom=TO_CHAR(LPAD('MMMMMMMMMM',200,'*'));

The query is a table scan. The Access Plan for this query, without any index is the following.

Table 2-3 – Execution Plan – Query 1

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	323	1	718
TABLE ACCESS (FULL)	EMPLEATS	ANALYZED	323	1	718

As expected it is used a full table access to execute the select statement.

- (03%) SELECT nom FROM empleats WHERE sou>1000 AND edat<20;

This query is searching for several tuples, that fulfill the predicates. The following access plan is chosen by the DBMS optimizer.

Table 2-4 – Execution Plan – Query 2

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	90	289	60,401
TABLE ACCESS (BY INDEX ROWID BATCHED)	EMPLEATS	ANALYZED	90	289	60,401
BITMAP CONVERSION (TO ROWIDS)			0	0	0
BITMAP INDEX (RANGE SCAN)	INDEX1		0	0	0

The first step of the access plan is to convert the Bitmap index (in emplats(edat)) in a ROWID index, with cost 0. The select statement is therefore performed in a 'TABLE ACCESS BY INDEX ROWID BATCHED'. According to Oracle support center this execution plan "It is generally used for range (> or <) queries. For this operation, Oracle selects few ROWIDs from the index and then try to access the rows in blocks. This significantly reduces the number of times Oracle must access the blocks thereby improving performance." Resulting in a significantly reduction in the cost of the query.

For comparison, the cost of the query without indexes is presented in the following table.

Table 2-5 – Execution Plan No index – Query 2

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	323	289	60,401
TABLE ACCESS (FULL)	EMPLEATS	ANALYZED	323	289	60,401

- (25%) SELECT * FROM empleats e, departaments d, seus s WHERE e.dpt=d.id AND d.seu=s.id;

The select statement in this query involves two joins.

Table 2-6 – Execution Plan – Query 3

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	1,031	11,267	33,474,257
HASH JOIN			1,031	11,267	33,474,257
TABLE ACCESS (FULL)	SEUS	ANALYZED	2	10	440
HASH JOIN			1,029	11,267	32,978,509
TABLE ACCESS (FULL)	DEPARTAMENTS	ANALYZED	119	1,300	2,871,700
TABLE ACCESS (FULL)	EMPLEATS	ANALYZED	323	13,000	9,334,000

HASH joins are used by the DBMS to perform the query. According to Oracle help center "Hash joins are the usual choice of the Oracle optimizer when the memory is set up to accommodate them. In a HASH join, Oracle accesses one table (usually the smaller of the joined results) and builds a hash table on the join key in memory. It then scans the other table in the join (usually the larger one) and probes the hash table for matches to it." Therefore, the optimizer in this case, does not use any of the indexes created.

- (47%) SELECT * FROM departaments WHERE seu=4;

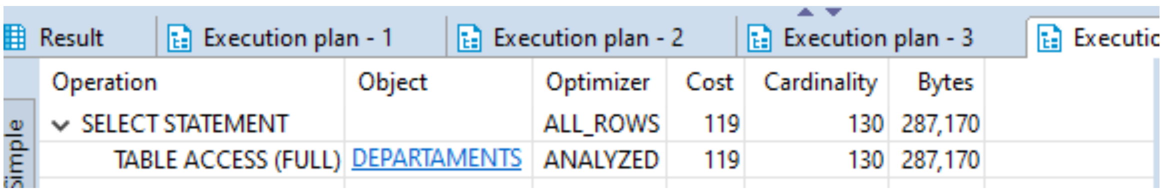
This query is searching for several tuples in one table. The access plan is the following.

Table 2-7 – Execution Plan – Query 4

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	85	130	287,170
TABLE ACCESS (BY INDEX ROWID BATCHED)	DEPARTAMENTS	ANALYZED	85	130	287,170
BITMAP CONVERSION (TO ROWIDS)			0	0	0
BITMAP INDEX (SINGLE VALUE)	INDEX2		0	0	0

The same process of the second query is used by the Optimizer. The first step of the access plan is to convert the Bitmap index (in departments(seu)) in a ROWID index, with cost 0. The select statement is therefore performed in a ‘TABLE ACCESS BY INDEX ROWID BATCHED’. According to Oracle support center this execution plan “It is generally used for range (> or <) queries. For this operation, Oracle selects few ROWIDs from the index and then try to access the rows in blocks. This significantly reduces the number of times Oracle must access the blocks thereby improving performance.” Resulting in a significantly reduction in the cost of the query. For comparison, the cost of the query without indexes is presented in the following table.

Table 2-8 – Execution Plan No index – Query 4



Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	119	130	287,170
TABLE ACCESS (FULL)	DEPARTAMENTS	ANALYZED	119	130	287,170