| **Group:** | **89** | **Lab User** | **286** |
| --- | --- | --- | --- |
| **Student:** | **Alberto Pascau Sáez** | **NIA:** | **100495775** |
| **Student:** | **Silvia Bayo Martínez** | **NIA:** | **100495811** |
| **Student:** | **Carmen Serrano Pérez** | **NIA:** | **100495711** |

# Introduction

It will consist of an introductory paragraph, analyzing the problem to be solved.

Expose initial state: physic design, and prototypic workload (frequent processes) composition and cost. State the goal and summarize the steps you are about to follow for attaining it.

# Analysis

*Describe the setting: the current (initial) physical design and the prototypical workload (frequent processes). Add a screenshot with the (initial) average cost of the workload.*

*For each sentence in the workload (each query) find out, explain, and analyze the execution plan and basic statistics (focusing on consistent gets and timing). Point out the weaknesses and strengths of the initial physical design according to the needs of your specific problem (the workload). Incorporate screenshots of performance as you deem appropriate.*

*Propose improvements to the physical design based on the analysis of each instruction run individually, and comment on the expected benefits and the drawbacks that it could bring about on the global system (if any).*

## Output of the RUN\_TEST PROCEDURE

The run test procedure was run with 10 as the input for the number of iterations. The result obtained was:

begin

2 pKG\_COSTES.RUN\_TEST(10);

3 end;

4 /

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

Iteration 6

Iteration 7

Iteration 8

Iteration 9

Iteration 10

RESULTS AT 23/04/2025 17:41:45

TIME CONSUMPTION (run): 1069.1 milliseconds.

CONSISTENT GETS (workload):60493 acc

CONSISTENT GETS (weighted average):6049.3 acc

PL/SQL procedure successfully completed.



From this output we can analyse two important pieces of information.

1. The time consumption for the tests was of: **1069.1 milliseconds**
2. The consistent gets (weighted average) had: **6049.3 acc**

After developing all optimizations the tests will be run again to check if the development made any change (we hope so) and analyse the quantity of that change.

## QUERY 1: Analisis

Given the first query:

select \* from editions where pub\_place='…'; -- values like 'Madrid', 'Segovia', or 'Barataria', to name a few examples



We’ll perform an initial analysis by setting the autotrace and executing queries for Madrid and Segovia. This is because Madrid is a popular value (many rows will be returned) while Segovia isn’t. And this difference will help us find a balanced approach to the optimization of the query.

SQL> select \* from editions where pub\_place='Madrid';

82450 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 1741989577

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 207 | 44712 | 2054 (1)| 00:00:01 |

|\* 1 | TABLE ACCESS FULL| EDITIONS | 207 | 44712 | 2054 (1)| 00:00:01 |

------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("PUB\_PLACE"='Madrid')

Statistics

----------------------------------------------------------

41 recursive calls

86 db block gets

12927 consistent gets

958 physical reads

17436 redo size

17922970 bytes sent via SQL\*Net to client

60833 bytes received via SQL\*Net from client

5498 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

82450 rows processed

SQL> select \* from editions where pub\_place='Segovia';

71 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 1741989577

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 207 | 44712 | 2054 (1)| 00:00:01 |

|\* 1 | TABLE ACCESS FULL| EDITIONS | 207 | 44712 | 2054 (1)| 00:00:01 |

------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("PUB\_PLACE"='Segovia')

Statistics

----------------------------------------------------------

1 recursive calls

0 db block gets

7566 consistent gets

4494 physical reads

0 redo size

16790 bytes sent via SQL\*Net to client

418 bytes received via SQL\*Net from client

6 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

71 rows processed



We can see that the autotrace shows us a lot of data, from this data, the important output we focus on is the number of reads and consistent gets for each query:

* **MADRID:** 12927 consistent gets ; 958 physical reads
* **SEGOVIA:** 7566 consistent gets ; 4494 physical reads

Interesting also to notice the difference between the number of rows returned for Madrid and for Segovia. Madrid has, as hypothesized, many more than segovia.

We can also use the following query to see the data of all tables to see the editions data, this analysis will also be useful for other queries as we focus only on table editions.

SQL> select table\_name, avg\_row\_len, num\_rows, blocks from user\_tables;

TABLE\_NAME AVG\_ROW\_LEN NUM\_ROWS BLOCKS

-------------------------------------------------------------------------------------------------------------------------------- ----------- ---------- ----------

ASSIGN\_BUS

ASSIGN\_DRV

BIBUS 0 0 5

BIBUSERO 0 0 5

BIBUSERO\_STATE 0 0 5

BIBUSES

BIBUS\_STATE 0 0 5

BOOKS

BOOK\_EDITIONS 0 0 0

BOOK\_ENTRIES 0 0 5

BOOK\_LOANS 0 0 0

TABLE\_NAME AVG\_ROW\_LEN NUM\_ROWS BLOCKS

-------------------------------------------------------------------------------------------------------------------------------- ----------- ---------- ----------

BOOK\_RESERVATIONS 0 0 5

COPIES

COURSES 0 0 5

DRIVERS

EDITIONS 221 240632 7552

ENROLLMENTS 0 0 5

ID\_ROUTES 6 150 5

LOANS

MORE\_AUTHORS

MUNICIPALITIES

MUNICIPALITY 0 0 0

TABLE\_NAME AVG\_ROW\_LEN NUM\_ROWS BLOCKS

-------------------------------------------------------------------------------------------------------------------------------- ----------- ---------- ----------

M\_LIBRARY 0 0 0

POSTS

PROFESSORS 0 0 5

ROUTES

SERVICES

SROUTES 0 0 5

STOPS

STUDENTS 0 0 5

USERS

31 rows selected.



De esta salida nos importan los únicos valores que no son cero, aquellos de la tabla ediciones pues es la única que se ha analizado.

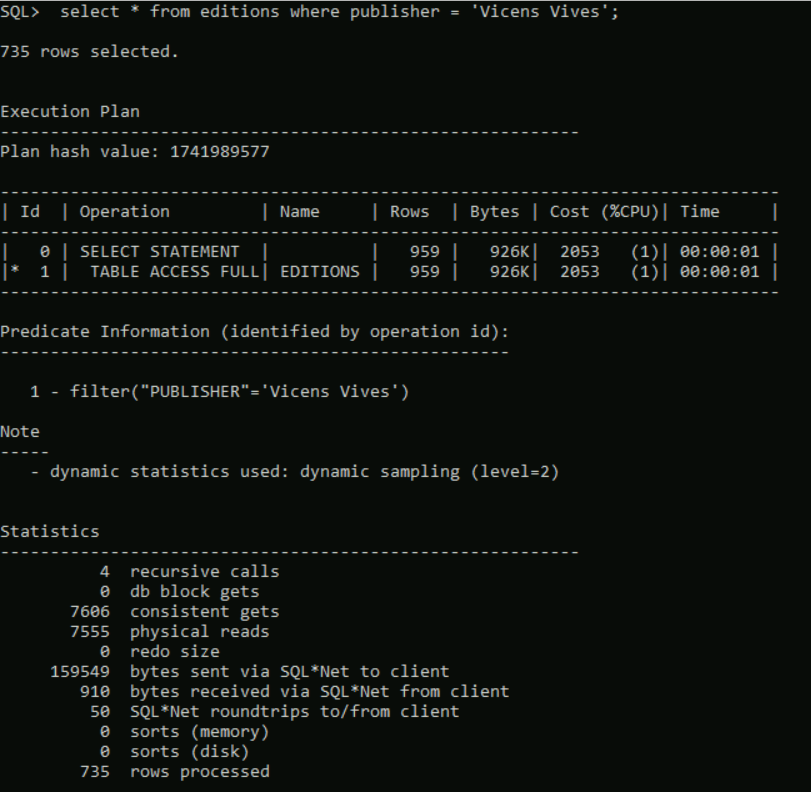
Podemos observar que la tabla contiene un número de entradas igual a **240632** y un número de bloques igual a **7552.** Estos valores los usaremos para futuras comparaciones tras realizar las optimizaciones.

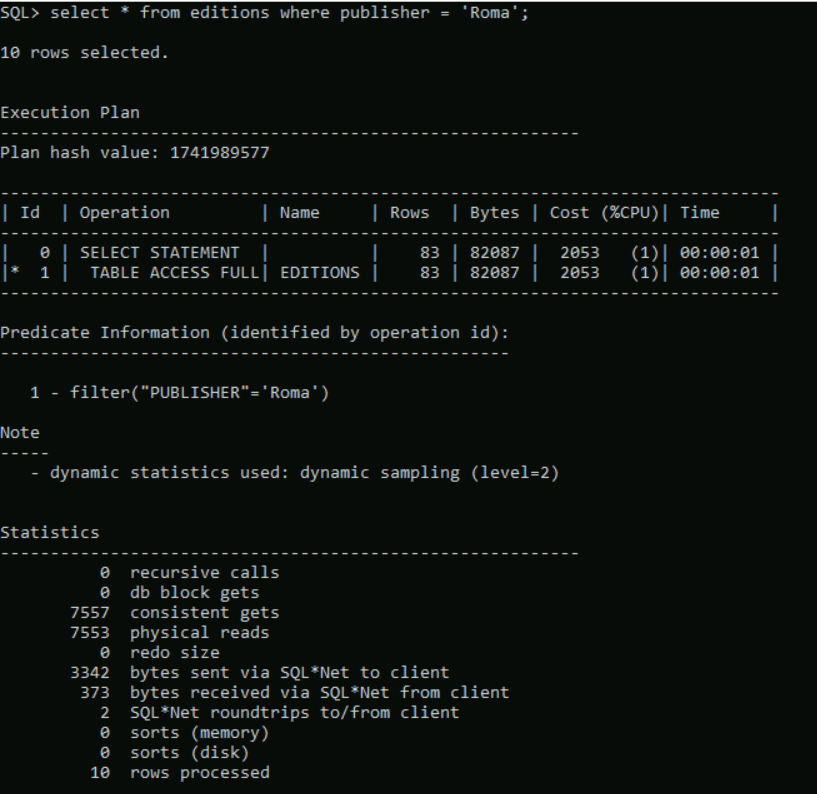
## QUERY 2: Analisis

Given the second query:

select \* from editions where publisher = '...';

We performed an initial analysis by setting autotrace and executing queries with two representative values: 'Vicens Vives' (a frequent publisher) and 'Roma' (a less frequent one)





From the execution of the second query using two different values, we observed the following:

* For 'Vicens Vives' (a frequent publisher, 735 rows returned), the system required 7606 consistent gets and 7555 physical reads
* For 'Roma' (an infrequent publisher, 10 rows returned), the system still required 7557 consistent gets 7553 physical reads

Despite the significant difference in the number of rows returned (735 vs. 10), both queries triggered a full table scan, with nearly identical access costs. This demonstrates that the current physical design does not take advantage of the column's selectivity.

## QUERY 3: Analisis

## QUERY 4:Analisis

Given the fourth query:

select \* from editions;

We’ll perform the analisis with the autotrace. Besides, we’ll also use the output of last part of the analysis performed for the first query

The output after performing the query with the autotrace on is:

SQL> select \* from editions;

240632 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 1741989577

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 240K| 49M| 2055 (1)| 00:00:01 |

| 1 | TABLE ACCESS FULL| EDITIONS | 240K| 49M| 2055 (1)| 00:00:01 |

------------------------------------------------------------------------------

Statistics

----------------------------------------------------------

42 recursive calls

87 db block gets

23119 consistent gets

3071 physical reads

17668 redo size

57147402 bytes sent via SQL\*Net to client

176814 bytes received via SQL\*Net from client

16044 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

240632 rows processed



As in previous queries we focus on the consistent gets and physical reads, there is a total of **23119 consistent gets** and **3071 physical reads**

# Physical Design

*Following the analysis done in the previous section, settle a complete physical design of the whole DB (at least one, yet you can propose several alternatives). Improvements may include changes in base organizations (hashing, clustering), auxiliary organizations (indexes), redundancies (immediately refreshed materialized views), DB block (bucket) parameters, etc.*

*Notice that some improvements aimed at improving a given part of the workload could worsen other operations. Justify every design decision.*

## QUERY 1: Physical design

In order to optimize the query we’ll see three methods used.

1. Creation of indexes.

We’ll create a secondary index on editions. After this we’ll perform our queries with the index hint so that the index search is forced and we have no doubts that we are using the index we just created.

create index idx\_ed1 on editions(pub\_place);

-- Check the query is being executed with the index

select \* from editions where pub\_place='Madrid';

-- If not, force the query to execute with the index

select /\*+ index(editions) \*/ \* from editions

where pub\_place='Madrid';

-- For trying with segovia

select /\*+ index(editions) \*/ \* from editions

where pub\_place='Segovia';

-- For madrid, it is better to do a fullscan

select /\*+ full(editions) \*/ \* from editions

where pub\_place='Madrid';



Now we need to analyze what this index did by running again the autotrace on both queries. After doing so we find the following results:

1. For madrid with index:

SQL> select /\*+ index(editions) \*/ \* from editions where pub\_place='Madrid';

82450 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 4255503933

------------------------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 192 | 41280 | 78 (0)| 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID BATCHED| EDITIONS | 192 | 41280 | 78 (0)| 00:00:01 |

|\* 2 | INDEX RANGE SCAN | IDX\_ED1 | 192 | | 3 (0)| 00:00:01 |

------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - access("PUB\_PLACE"='Madrid')

Statistics

----------------------------------------------------------

1 recursive calls

0 db block gets

18261 consistent gets

7721 physical reads

0 redo size

19595058 bytes sent via SQL\*Net to client

60859 bytes received via SQL\*Net from client

5498 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

82450 rows processed



1. For segovia with index:

SQL> select /\*+ index(editions) \*/ \* from editions where pub\_place='Segovia';

71 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 4255503933

------------------------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 192 | 41280 | 78 (0)| 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID BATCHED| EDITIONS | 192 | 41280 | 78 (0)| 00:00:01 |

|\* 2 | INDEX RANGE SCAN | IDX\_ED1 | 192 | | 3 (0)| 00:00:01 |

------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - access("PUB\_PLACE"='Segovia')

Statistics

----------------------------------------------------------

1 recursive calls

0 db block gets

80 consistent gets

2 physical reads

0 redo size

18569 bytes sent via SQL\*Net to client

444 bytes received via SQL\*Net from client

6 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

71 rows processed



1. For madrid with a full scan:

select /\*+ full(editions) \*/ \* from editions where pub\_place='Madrid';

82450 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 1741989577

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 192 | 41280 | 2054 (1)| 00:00:01 |

|\* 1 | TABLE ACCESS FULL| EDITIONS | 192 | 41280 | 2054 (1)| 00:00:01 |

------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("PUB\_PLACE"='Madrid')

Statistics

----------------------------------------------------------

1 recursive calls

0 db block gets

12889 consistent gets

7552 physical reads

0 redo size

17923258 bytes sent via SQL\*Net to client

60854 bytes received via SQL\*Net from client

5498 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

82450 rows processed

SQL>



* We see that for Madrid, with the inde, the number of consistent gets is higher than the initial one, this is not good.   
  **Value of consistent gets before: 12927  
  Value of consistent gets after: 18261**
* However it works for low-coincidence values, it is more efficient, see with segovia where the number of entries decreases  
  **Value of consistent gets before: 7566  
  Value of consistent gets after:80**

After looking at the results we can also think about performing a full scan with Madrid (as shown in the code above), with that we see that the full scan is more optimal than the secondary index search.

1. Because the index is not really making that much of a difference for madrid we’ll implement a cluster on the *pub\_place* property of the editions table.

drop cluster places;

create cluster places(pub\_place varchar2(50));

create table editions(..) cluster places(pub\_place);

create index idx\_places on cluster places;



The editions table of the NEW\_creation.sql script is then modified as:

CREATE TABLE Editions(

ISBN VARCHAR2(20),

TITLE VARCHAR2(200) NOT NULL,

AUTHOR VARCHAR2(100) NOT NULL,

LANGUAGE VARCHAR2(50) default('Spanish') NOT NULL,

ALT\_LANGUAGES VARCHAR2(50),

EDITION VARCHAR2(50),

PUBLISHER VARCHAR2(100),

EXTENSION VARCHAR2(50),

SERIES VARCHAR2(50),

COPYRIGHT VARCHAR2(20),

PUB\_PLACE VARCHAR2(50),

DIMENSIONS VARCHAR2(50),

PHY\_FEATURES VARCHAR2(200),

MATERIALS VARCHAR2(200),

NOTES VARCHAR2(500),

NATIONAL\_LIB\_ID VARCHAR2(20) NOT NULL,

URL VARCHAR2(200),

CONSTRAINT pk\_editions PRIMARY KEY(isbn),

CONSTRAINT uk\_editions UNIQUE (national\_lib\_id),

CONSTRAINT fk\_editions\_books FOREIGN KEY(title,author) REFERENCES books(title,author)

) cluster places(pub\_place);



See that the cluster is added in the last line.

1. Finally, we’ll analyze the performance after using a hashkeys table to see if it is better or worse than when using the index, for that we first modify the creation of the cluster to use the hashkeys instead of the index:

drop cluster places;

create cluster places(pub\_place varchar2(50))

single table Hashkeys 251;

create table editions(..) cluster places(pub\_place);



After performing this change the analysed results are:

1. For madrid with cluster hashkeys

SQL> select \* from editions where pub\_place='Madrid';

82450 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 3284490970

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 188 | 40420 | 30 (0)| 00:00:01 |

|\* 1 | TABLE ACCESS HASH| EDITIONS | 188 | 40420 | 30 (0)| 00:00:01 |

------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - access("PUB\_PLACE"='Madrid')

Statistics

----------------------------------------------------------

2 recursive calls

0 db block gets

8090 consistent gets

1792 physical reads

0 redo size

17922933 bytes sent via SQL\*Net to client

60833 bytes received via SQL\*Net from client

5498 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

82450 rows processed



1. For Segovia with cluster hashkeys:

SQL> select \* from editions where pub\_place='Segovia'

2 ;

71 rows selected.

Execution Plan

----------------------------------------------------------

Plan hash value: 3284490970

------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 188 | 40420 | 30 (0)| 00:00:01 |

|\* 1 | TABLE ACCESS HASH| EDITIONS | 188 | 40420 | 30 (0)| 00:00:01 |

------------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - access("PUB\_PLACE"='Segovia')

Statistics

----------------------------------------------------------

2 recursive calls

0 db block gets

14 consistent gets

1 physical reads

0 redo size

16917 bytes sent via SQL\*Net to client

419 bytes received via SQL\*Net from client

6 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

71 rows processed



We can see how both consistent get values have really improved, Madrid has gone from 12 thousand to 8 thousand while segovia is even better than with indexes at 14 vs 80, and not to mention compared to the original data of thousands. This is the one that will be pointed to at the evaluation.

# Evaluation

You have measured the performance of the initial physical design and stated it. After implementing your improved physical design and measured the new performance, compare both and analyze the results obtained (comment divergences with expected results). Add screenshots for backing your evaluation.

# Concluding Remarks

Firstly, make conclusions on the work and the results obtained. Reflex on (defend or criticize) the achieved result (if you think it is good, explain why).

After stating your results, comment on your achievement through this *labwork*, and all assignments in general: required effort, knowledge gain, progress, etc. You can also propose improvements for further editions (focus, size of the problem, requested items, deadlines, supporting materials, etc.). Finally, you can add comments on the whole course (lacks in the syllabus, issues you would like to study more deeply, non-useful issues, etc.).