Performance Analysis of the Database of Telecommunications

Michal Kvet,
Linda Majerčiaková
University of Žilina
Žilina, Slovakia
majerciakova@stud.uniza.sk

Antoine Gibhardt
CESI Ecole d'Ingénieurs
Bordeaux, France

Luka Miljković
University of Belgrade
Belgrade, Serbia

Wassim Bouhtout
ECE Paris-Lyon
Paris, France

In this article, we will explain and describe the different steps of implementing a database, the generation of the data, and the operations we did on them. Furthermore, this project has for objective to highlight the different performance fluctuations and what they are linked to.

I. Introduction

Over the years, the amount of data has been growing exponentially. Every new system and device produce data that must be processed and analysed. Moreover, the amount of data is not the only problem. The scalability of the system is also an issue. When there is that much fluctuation in the amount of data processed, the conception of the Databases architecture has to be adapted and optimized. These changes raise new problems and new questions about the concept of databases and how to interact with them [1], [2], [3].

In this article, we will mainly discuss the processes we've been through to create a functional and exploitable database. The main purpose is to create an environment in which we can do request and exploit the data in different ways. That way, we will be able to do a performance analysis bases on a multi factors modification.

To begin with, we created a Data Model that highlights the relations between the different tables and how they are connected. This Data model also allows us to create the SQL script used to generate the tables, their indexes, and relations. Then, we generated random data with a Java script. This way, we can start interacting with the database and its data. It was crucial to generate a fair amount of data so that we could simulate a real-life situation. In the end, we did different operations while changing the process or the variables. This allowed us to see the performance differences and to understand how to use a database more efficiently nowadays.

Performance of the processing is crucial, whereas the database system layer forms the main background managing and manipulating data. Multiple structures can form data layer. Section 2 deals with the main performance aspects by referencing existing streams. Section 3 deals with the database analysis. The used data model is introduced in section 4. Section 5 highlight the individual operations done on the data level. The main contribution of the paper is in section 6, proposing performance analysis and results.

II. Related techniques

Data in the database can be formed by various techniques. In general, relational database systems are still very often used. They are based on the entities and relationships in the logical sphere, represented by the database tables physically. Each table is delimited by the unique identification of the object – the primary key, which must be unique, as well as minimal. Data are inserted in the bulk operations, supervised by the transactions ensuring atomicity, consistency, isolation, and durability. Thus, online transaction processing (OLTP) is the core element of relational database technology [1]. To limit anomalies and reflect storage demands by reducing duplicate tuples, data structure normalization is performed, secured by the referential integrity dividing the data attributes into particular tables [2]. Thus, one of the elements to ensure performance is just the structure itself. The opposite system of the relational systems is formed by the analytical processing layer (OLAP), by which priority is given to data retrieval. Normalization is not so strict, focusing on the pre-calculated values stored in the fact table as a dominant table, interconnected by multiple dimension tables [4], [5].

In any system, one side of the corridor is related to the data retrieval process to ensure consistency, as well as complexity, reliability, and timely availability. Index structures form the separate access layer, mostly formed by the B+tree indexes, whereas they do not regrade with the data amount extension. Moreover, they are rather wide than high, so the traverse path is extended in a minimal manner with a significant increase in data. Other indexing techniques are bitmaps (mostly used in OLAP) or hash indexes if the relevant hash function can be found and identified [2], [5].

Although indexes provide satisfactory solutions for data retrieval, it is impossible to specify all suitable sets in the system. The main limitation is related to the data manipulation operations changing the content of the tables. Namely, changes must be applied in the indexes during the transaction. Therefore, the transaction itself cannot be ended before all changes are incorporated into the index set, which can be really demanding. Suppose, that the B+tree index is always balanced [3] . In the following part, emphasis is made on the data retrieval methods. The sequential table scanning method takes each block of the table from the database storage and shifts it to the memory for evaluation. It brings the limitation related to the empty blocks, fragmentation, etc [1], [2], [3], [7].

ISSN 2305-7254

Vice versa, index usage identifies relevant data block sets in the first phase by getting address lists of them. Then, only those that hold a particular value to be processed in the result set are loaded and evaluated. As a result, the data block amount is strongly limited by increasing the overall performance [9]. Moreover, storage demands and sources are significantly lowered, offering wide techniques of parallelism.
Related to the performance, even if the index is not suitable for the query, relevant blocks can be identified by focusing on the data fragmentation and empty blocks if the Update operation stream is high. The Master index can be referenced to focus on the blocks instead of the rows themselves [9].
Finally, when dealing with performance, there are two aspects related to physical storage. The first perspective is associated with the block size [9]. By default, each block can hold 8KB of data. However, by changing the storage granularity, either for the tables or indexes, a significant performance increase can be reached. The second perspective is associated with partitioning, by splitting the table set into multiple parts, operated by the local or global indexes [8], [9], [10].
Non-relational databases are also forming the relevant part of the information processing. However, the integrity is not so strict in that case, and transaction support is limited to strongly focus on the data amount to be handled. It reduces the time processing demands for the individual operations at the cost of possible inaccuracies, which, however, are not significantly related to the enormous amount of time-varying data to be handled, processed, and evaluated.

III. Database analysis

Before creating the data model itself, it is necessary to find out what data needs to be stored and the relationships between them.

By analyzing the existing systems, we found that the most important attributes are the called and calling phone number, call start time and call type. These attributes are stored in tables for the call, phone number, and customer. It was also necessary to solve the relationship between the called and calling number, which we solved by creating a table. This allowed the caller to call several numbers at once, which can be called a group call [6].

In the analyzed databases, the company and country of customer's phone number were stored as attributes. We decided that they would be separate tables [6].

The analyzed databases also lacked a roaming solution. We solved this problem and included the tables for roaming and the roaming country in our database [6].

IV. Data model

A. Country

This table represents countries and is defined by the attributes:

- id_country (Primary Key): The id of the country.
- country_name: The name of the country.
- prefix: country telephone codes.
- currency: Currency of the country.

B. Company

This table corresponds to the different operators of each country in the country table and is defined by the attributes:

- id_company (Primary Key): The id of the company.
- id_country (Foreign Key): The country id corresponds to that of the country table.
- company_name: The name of the company.

C. Client

This table corresponds to the client of the operators in the operator table and is defined by the attributes:

- id_client (Primary Key): The id of the client.
- first_name: Client's first name.
- last_name: Client's last name.
- birth_date: Client's birthday.

D. Phone Number

This table corresponds to the telephone number of the customers and is defined by the attributes:

- id_number (Primary Key): The id of the phone number.
- id_client (Foreign Key): The id client corresponds to that of the client table.
- id_company (Foreign Key): The id company corresponds to that of the company table.
- number_of_phone: The number phone of the client.
- pin: The pin code of the phone number.

E. Call Type

This table corresponds to the different types of possible communications such as: calls, sms, viber. And is defined by the attributes:

- id_type (Foreign Key): The id of the communication type.
- type_name: The name of the type.
- fee: The fee which is associated with the type.

F. Roaming Zone

This table corresponds to the roaming zones belonging to each country and is defined by the attributes:

- id_zone (Primary Key): The id of the zone.
- id_country_number (Foreign Key): The id of the country in the country table to which the zone belongs.
- name_zone: The name of the zone.
- sms_fee: The fee associated with each character of a text message for the roaming zone.
- call_fee: The fee associated with minutes of a call for the roaming zone.

G. Zone country

This table corresponds to the membership of each country to the zones of the other countries and is defined by the attributes:

- id_country (Composite Primary Key): The id of the country belonging to the roaming zone.

- id_zone (Composite Primary Key): Id of the roaming zone.

H. Call

This table corresponds to the calls, messages and viber calls of each client and is defined by the attributes:

- id_call (Primary Key): The id of the call.
- id_type (Foreign Key): The id of the type of communication (call,sms,viber).
- id_number_from (Foreign Key): The caller's number id which corresponds to the id in the number phone table.
- id_zone (Foreign Key): The roaming zone's id which corresponds to the id in the roaming zone table.
- start_time: The date and time of the sending of the call/sms.
- end_time: The date and time of the end of the call (it is null for a sms).
- sms_text: The text of the sms (it is null for a call or viber call).
- total_fee: Total fees of the communication.

I. Call To

This table allows us to indicate the recipients of the calls or sms and is defined by the attributes:

- id_number (Composite Primary Key): id number of the recipient which corresponds to the id in the number phone table.
- id_call (Composite Primary Key): id of the call which corresponds to the id in the call table.

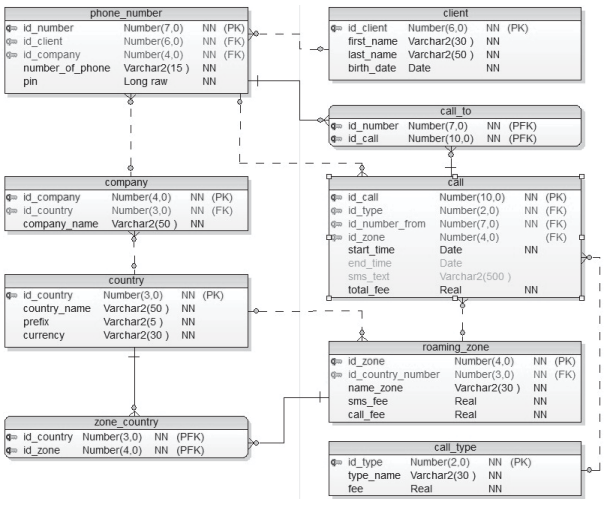

Fig. 1. Data model

V. DATA INSERTION

The data generation was done from a java code that was in charge of generating a text file containing all the sql insertion queries.

To do this, we have created classes corresponding to each class, then in the main class, we have created for each table arraysList of the same type as the table containing all entries that will be inserted into the table [9] [10].

Once the arrayslist of each table have been filled, we have made a script that allows to record each entry in the sql table.

A. Generation of countries

For the generation of the country, I took in the internet a csv file which contains more than one hundred countries with both the name of the country and the currency associated to the country. For the prefix of each country, we derive it by putting "+" with a number which is 100 plus an incrementation for each country.

For each entry, the id of the country is incremented.

B. Generation of companies

As for the country table, I took on internet a csv file which contains a list of operator companies in which each operator is associated with a country

For each entry, the id of the company is incremented.

C. Generation of clients

As previously, I took in internet 2 csv files, one containing a list of first name and another one which contain a list of last names. Then for each client I took randomly a first name from the list of first name and last name from the list of last names.

For the birth date, we generate a random date between 1930 and 2022

For each entry the id of the client is incremented.

D. Generation of phone numbers

For each client we took randomly a company from the arraylist of companies, and we generate randomly a number with the prefix of the county of the company, in order to generate a number phone.

For each entry the id of the phone number is incremented.

E. Generation of Call Type

We generate 3 communication types which are: call, sms, viber.

For each entry the id of the call type is incremented.

F. Generation of Roaming Zone and Zone Country

For each country we created 3 roaming zone in the arraylist of roamingzones, and then for each other countries we put them randomly in one of the 3 roaming zone created in the arraylist of zone country.

For each entry of roaming zone, the call fee and sms fee is choose randomly and the id of the roaming zone are incremented.

G. Generation of Call and CallTo

We created 100 000 call and for each one we took randomly a number phone, a call type, and a roaming zone which belong to the country of the number phone, then we compute the total fee thanks to the fee in roaming zone table the longer of the call or message and also the fee in call_type

table. And then for each of these calls we generate between one and 3 recipients in the CallTo table.

For each entry the id of the call is incremented.

VI. PERFORMANCE EVALUATION

After creating the data model and creating and filling the database, we started working with the database. We created partitions, indexes, and examined their impact on execution. We also examined the effect of different queries.

A. Partitions

Horizontal partitioning is convenient for database optimization when some table contains a large number of records. In that case, records can be divided into several groups called partitions according to a certain criterion that we specify when creating a table. This allows the partition to be accessed directly so that queries are executed faster and more efficiently [7] [8].

In our model, the call table has the most records (100.000), so we decided to perform horizontal partitioning over it and test whether performance is improving. The criterion for determining the partition is the duration of the call. However, as this attribute does not exist in the call table, a virtual column call_duration is created which is calculated by subtracting the start_time and end_time attributes and multiplying the obtained result by 1440. The obtained result represents the number of call minutes. For text messages, the call duration is set to 0, ie the start_time and end_time are the same, so that they can be included in the partitions. One of the disadvantages of horizontal partitioning is that it cannot include null values in the scope of the criteria.

Finally, we created a range partitioning and divided the data into 4 partitions, where the partitions for calls approximately the same size:

1. text messages (33403 records)

2. short calls - values for call duration less than 25 minutes (22902 records)

3. Medium length calls - values for call duration between 25 and 50 minutes (23239 records)

4. long calls - values for call duration over 50 minutes (20434 records)

It's proven that horizontal partitioning is good for query execution efficiency when we know exactly from which partition, we want to get certain data. The following pictures show the difference in costs and query execution time between queries that use where statement and queries in which we know which partition we are accessing.

```
select id_number_from
 from call
  where call_duration > 0 and call_duration < 25;
```

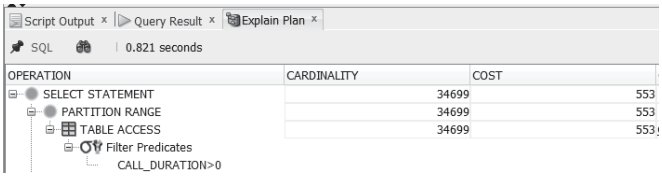

Fig. 2. Result of simple query without use partition

```
select id_number_from
 from call PARTITION(small_calls);
```

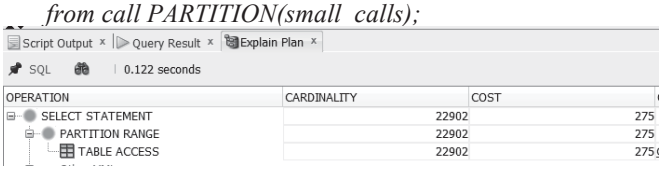

Fig. 3. Result of simple query with use partition explicitly

Also for some complex queries:

```
select count(*)
 from call c
   join phone_number pn
        on(c.id_number_from = pn.id_number)
   join company co on(pn.id_company = co.id_company)
   join country ct on(co.id_country = ct.id_country)
    where ct.id_country = 1 and c.call_duration > 50;
```

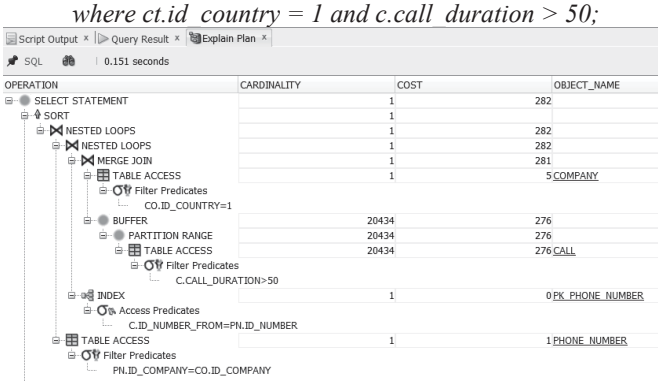

Fig. 4. Result of complex query without use partition

```
select count (*)
  from call partition(big_calls) c
       join phone_number pn
            on (c.id_number_from = pn.id_number)
       join company co
            on (pn.id_company = co.id_company)
      join country ct on (co.id_country = ct.id_country)
    where ct.id_country = 1;
```

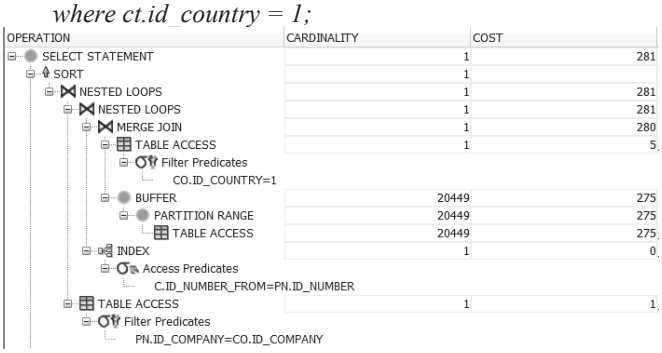

Fig. 5. Result of a simple query with use partition explicitly

However, horizontal partitioning has not performed well in queries when we do not know which partition we are accessing. For example, if we executed the same complex query over a table that has partitions, the execution cost would be higher than if the query was executed over a table that has no partitions, but in some cases, execution time would be reduced.

```
select count(*)
  from call
    where EXTRACT(year FROM start_time) = 2021;
```

SQL | 0.058 seconds

OPERATION	CARDINALITY	COST
SELECT STATEMENT	1	176
SORT	1	
TABLE ACCESS	25265	176
Filter Predicates		
EXTRACT(YEAR FROM INTERNAL_FUNCTION(START_TIME))=2021		

Fig. 6. Result of executing query when table is not partitioned

SQL | 0.114 seconds

OPERATION	CARDINALITY	COST
SELECT STATEMENT	1	1099
SORT	1	
PARTITION RANGE	1000	1099
TABLE ACCESS	1000	1099
Filter Predicates		
EXTRACT(YEAR FROM INTERNAL_FUNCTION(START_TIME))=2021		

Fig. 7. Result of executing query when table is partitioned

B. Indexes

We then examined the effect of indexes on the database. We created several selects to examine the impact of indexes on performance. The main factors for comparison are cost and time.

First, we created a query that order all phone numbers in the United States by the number of calls from that number. There was created 3 indexes for select:

```
select country_name, number_of_phone,
       count (id_call) as num_of_calls
  from call ph_call
    join phone_number ph_num
       on (ph_call.id_number_from = ph_num.id_number)
   join company using (id_company)
   join country co_from using (id_country)
  where co_from.country_name = 'United States'
  group by id_country, country_name,
           id_number, number_of_phone, currency
    order by num_of_calls desc;
```

For table ‘country’ was created index:

```
create index ind_country1
  on country (country_name, id_country;
```

For table ‘phone_number’ was created index:

```
create index ind_phone_num1
  on phone_number
       (id_number, id_company, number_of_phone);
```

Last index was created for table ‘call’:

```
create index ind_call1 on call (id_call, id_number_from).
```

After the creation of the indexes, the system used these indexes and the cost decreased from 1101 to 102, the total execution time of the order also decreased.

SQL HotSpot | 3,063 seconds

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
SORT		ORDER BY
HASH		GROUP BY
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
MERGE JOIN		CARTESIAN
TABLE ACCESS	COUNTRY	FULL
Filter Predicates		
CO_FROM.COUNTRY_NAME='United States'		
BUFFER		SORT
PARTITION RANGE		ALL
TABLE ACCESS	CALL	FULL
TABLE ACCESS	PHONE_NUMBER	BY INDEX ROWID
INDEX	PK_PHONE_NUMBER	UNIQUE SCAN
Access Predicates		
PH_CALL.ID_NUMBER_FROM=PH_NUM.ID_NUMBER		
INDEX	PK_COMPANY	UNIQUE SCAN
Access Predicates		
PH_NUM.ID_COMPANY=COMPANY.ID_COMPANY		
TABLE ACCESS	COMPANY	BY INDEX ROWID
Filter Predicates		
COMPANY.ID_COUNTRY=CO_FROM.ID_COUNTRY		

Fig. 2. Explain Plan without the use of indexes

SQL HotSpot | 2,975 seconds

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
SORT		ORDER BY
HASH		GROUP BY
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
MERGE JOIN		CARTESIAN
INDEX	IND_COUNTRY1	RANGE SCAN
Access Predicates		
CO_FROM.COUNTRY_NAME='United States'		
BUFFER		SORT
INDEX	IND_CALL1	FAST FULL SCAN
INDEX	IND_PHONE_NUM1	RANGE SCAN
Access Predicates		
PH_CALL.ID_NUMBER_FROM=PH_NUM.ID_NUMBER		
INDEX	PK_COMPANY	UNIQUE SCAN
Access Predicates		
PH_NUM.ID_COMPANY=COMPANY.ID_COMPANY		
TABLE ACCESS	COMPANY	BY INDEX ROWID
Filter Predicates		
COMPANY.ID_COUNTRY=CO_FROM.ID_COUNTRY		

Fig. 3. Explain Plan using indexes

Then, we used 4 indexes for select, which write the number of sent SMS messages for each country:

```
select country_name, count (id_call) as call_count
  from call cll
   join phone_number pn
     on (pn.id_number = cll.id_number_from)
   join company using (id_company)
   join country using (id_country)
   join call_type using (id_type)
  where type_name = 'Sms'
   group by id_country, country_name;
```

There was used index ‘ind_country1’ for table ‘country’. For table ‘call_type’ was created index:

create index ind_type1 on call_type (type_name, id_type);

For table ‘phone_number’ was created index:

create index ind_phone_num2
on phone_number (id_number, id_company).

Last index was created for table ‘call’:

create index ind_call2
on call (id_call, id_number_from, id_type).

After using of the indexes, the performance time decreased many times and the cost decreased from 1101 to 111.

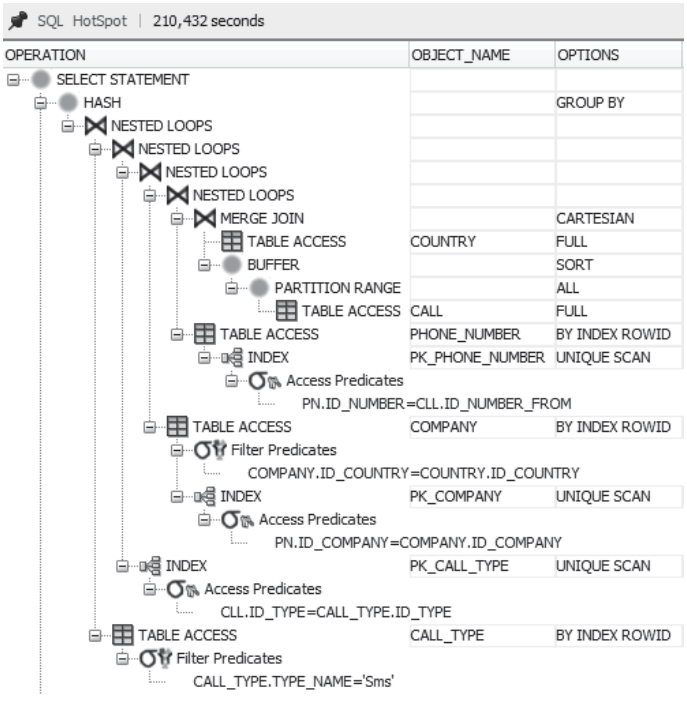

Fig. 4. Explain Plan without the use of indexes

Lastly, an index was created for select, which finds all clients whose last name starts with 'FRI':

select first_name, last_name
from client
where substr (last_name, 1, 3) = 'FRI';

There was created index for table ‘client’:

create index ind_client1
on client (substr (last_name, 1, 3), first_name, last_name);

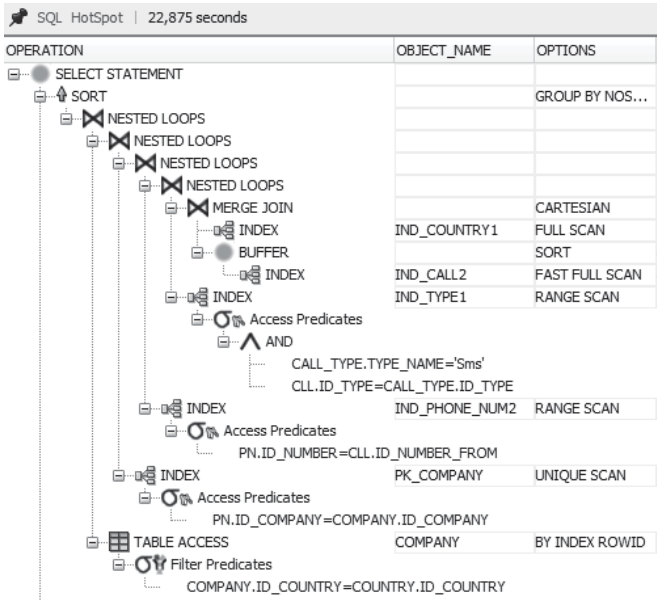

Fig. 5. Explain Plan using indexes

After using the index, the cost was reduced from 15 to 2. Instead of a full table scan, index range scan was used, and the query execution time was also reduced.

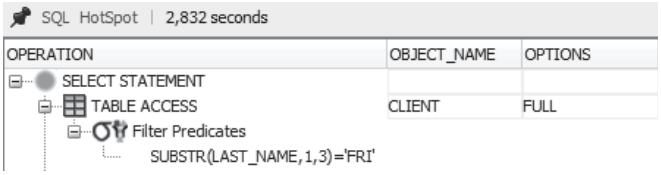

Fig. 6. Explain Plan without the use of indexes

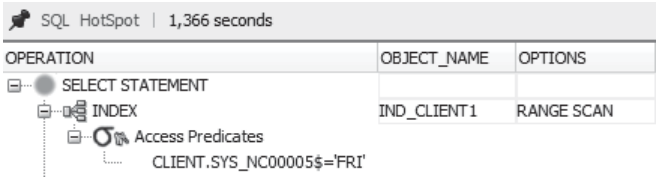

Fig. 7. Explain Plan using indexes

C. *IN and EXISTS clauses*

In the next step, we compared the performance of the IN and EXISTS clauses. The clauses were compared for smaller and larger sub-query results.

We first compared clauses where the result of the subquery is relatively small. It is a select that find all companies located in France. We made EXISTS clause as:

```
select *
 from company co
  where exists
    (select *
     from country cou
       where country_name = 'France'
         and co.id_country = cou.id_country);
```

Then we made IN clause:

```
select * from company
  where id_country in
   (select id_country
    from country
     where country_name = 'France');
```

In terms of time, the IN clause was faster than EXIST clause.

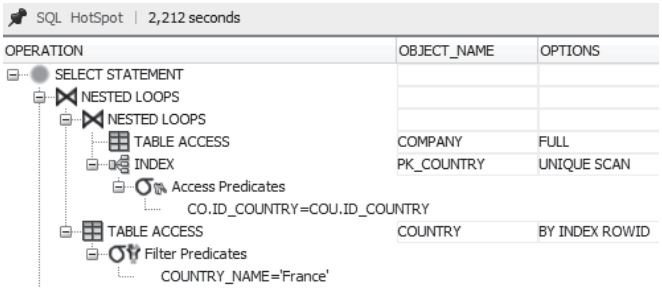

Fig. 8. Explain Plan of EXISTS clause

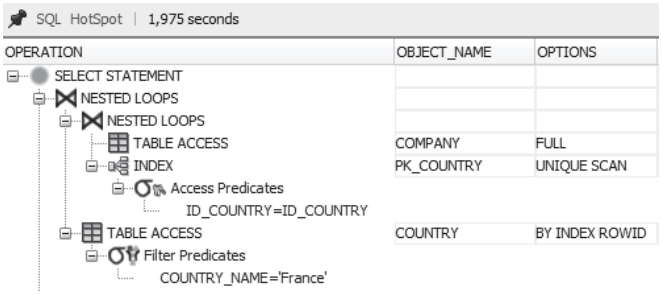

Fig. 9. Explain Plan of IN clause

We then compared the clauses if the result of the subquery was relatively large. Select query finds all calls that belong to the United States. EXISTS clause is:

```
select *
 from call ll
  where exists  (select *
                 from phone_number pn
                  join company using (id_company)
                  join country using (id_country)
                   where country_name = 'United States'
                   and ll.id_number_from = pn.id_number);
```

IN clause look like:

```
select *
 from call
   where id_number_from in
     (select id_number
       from phone_number
         join company using (id_company)
         join country using (id_country)
           where country_name = 'United States');
```

In this case, there was a change and the EXISTS clause was faster in terms of time.

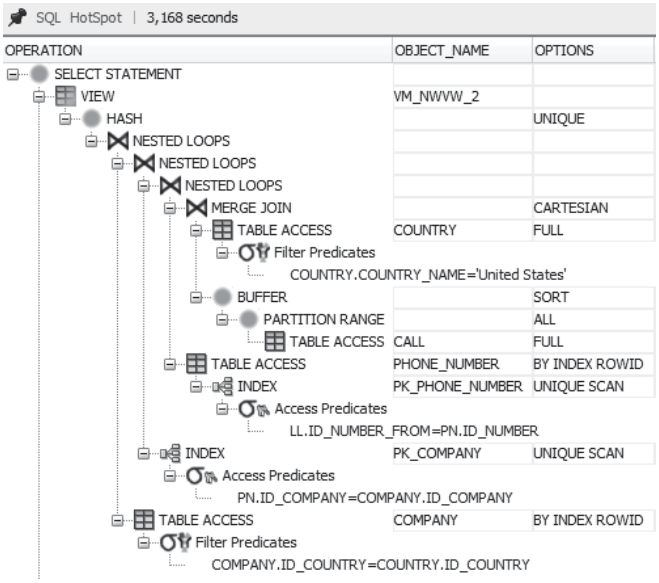

Fig. 10. Explain Plan of EXISTS clause

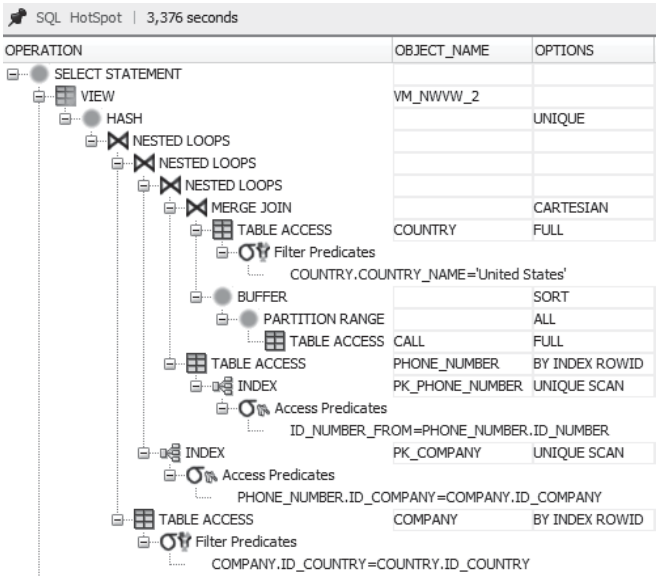

Fig. 11. Explain Plan of IN clause

VII. Conclusion

In this article we wanted to describe creation and then examination of database for telecommunications.

We were able to create a database using a data model. To create the architecture of a database, using a visual tool to create it help understanding what relations are important to improve the systems efficiency.

Moreover, we generate random data to import it into the database. This allowed us to interact with the database and more importantly, with its data.

Finally, we could do several experiments by interrogating the database in different ways. We could do a performance analysis that showed us what are the more efficient ways of using a database and how to optimize the processes.

Our experiments show that if we know in advance which parts of the table we will access frequently, it is good to use partitions. Horizontal partitions greatly improve performance. However, if we need access to the parts where the partitioning is broken, the performance is much worse.

We have also found that using indexes is very beneficial. Not only for simple queries but also for more complex ones in which multiple tables are accessed.

In the question of IN and EXISTS clauses, we concluded that it mainly depends on the sub-query. If the sub-query is smaller, it is preferable to use the IN clause, and if it is larger, it is better to use the EXISTS clause.

So, we can say that when creating and using a database, it also depends on how we use it. Then we can use the right partitions, indexes, and clauses to be able to work with the database efficiently.

|In the future, emphasis will be done on the data distribution in the multiple domain cloud storage.

Acknowledgment

This publication was realized with support of the Operational Programme Integrated Infrastructure in frame of the project: Intelligent systems for UAV real-time operation and data processing, code ITMS2014+: 313011V422 and co-financed by the European Regional Development Found.

References

[1] Bryla, B.: Oracle Database 12c The Complete Reference, Oracle Press, 2013, ISBN – 978-0071801751

[2] Burleson, D. K.: Oracle High-Performance SQL Tuning, Oracle Press, 2001, ISBN - 9780072190588

[3] Dudáš A., Škrinárová J, Vesel E.: Optimization design for parallel coloring of a set of graphs in the High-Performance Computing. In: Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics. pp 93-99. ISBN 978-1-7281-3178-8.

[4] Dudáš A., Škrinárová J.: Edge coloring of set of graphs with the use of data decomposition and clustering . In: IPSI Transactions on internet research: multi-, inter-, and trans-disciplinary issues in computer science and engineering. special issue, Selected topics in computer science. Vol. 16, no. 2 (2020), pp. 67-74. ISSN 1820-4503.

[5] Delplanque, J., Etien, A., Anquetil, N., Auverlot, O.: Relational database schema evolution: An industrial case study, IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Spain, 2018, pp. 635-644

[6] Mandal, S., Maji, G.: Integrating Telecom CDR and Customer Data from Different Operational databases and Data warehouses into a Central Data Warehouse for Business Analysis, India, 2016

[7] Eisa, I., Salem, R., Abdelkader, H.: A fragmentation algorithm for storage management in cloud database environment, Proceedings of ICCES 2017 12th International Conference on Computer Engineering and Systems, Egypt, 2018

[8] Kvet, M. (2019). Complexity and Scenario Robust Service System Design. In Information and Digital Technologies 2019: conference proceedings, Žilina, 2019, ISBN 978-1-7281-1400-2, pp. 271-274.

[9] Kvet, M.: Managing, locating and evaluating undefined values in relational databases. 2020

[10] Steingartner, W., Eged, J., Radakovic, D., Novitzka, V.: Some innovations of teaching the course on Data structures and algorithms. In 15th International Scientific Conference on Informatics, 2019.