ASSESSMENT OF IMPACT OF SELECTED FACTORS ON THE EFFECTIVENESS OF COST-BASED OPTIMIZER IN DATABASE SYSTEMS

Renata Pacholewicz
Department of Marine Electronics
Gdynia Maritime University
Poland

ABSTRACT

The paper presents the results of experiments, which aim was to evaluate the influence of selected factors on the cost and time of the query in database systems. The cost and execution time are criteria of assessment of query plans by the cost-based optimizer of a database system. Essential for constructing queries and for the efficiency of cost-based optimizer are: indexing, aliasing, asymmetry of the data and the availability of statistics referring to the tables. Experiments confirmed that the appropriate way to implement a query, based on the use of indexation or aliasing, and on modification of query parameters with the systematic generation of statistical information about the data can significantly reduce the time to obtain query results, offload the processor and reduce the number of disk reads. The experiment was conducted using the real data in the database ORACLE

Keywords: database, query efficiency, cost-based optimizer.

1. INTRODUCTION

The amount of data stored in databases is increasing day after day and even hour after hour. Growth of the collected data makes that searching for a detailed information becomes more challenging, and absorbs more and more time. The growth is also the reason that producers of database systems offer some new solutions. The latter are often associated with improving methods of data access and with reduction of the query execution time. The aim of the research presented in the paper is to picture some selected ways of optimizing data access and effectiveness thereof when compared with the access time or the cost of access without using them.

In practice, a database query is formulated on the basis of the declarative notation in which the user indicates the search criteria. This declarative way of expressing the expectations of the user does not specify how to find the data. The description of how to find the data in the database corresponds to the database management

system, which generates a query plan called microcode. Query plans can be different, as different can be query to the database formulated by the user. Furthermore, for the complexity of a query directed to a database there may be a variety of alternative query plans. The choice of the query execution plan is done by a module of the database management system called query optimizer.

One of the standard ways of formulating query plans relies on the rule-based optimizer, however this is a subroutine which is superseded by cost-based optimizers as more elastic. Rule-based optimizer is a set of rules which contains guideline information for database engine, on the basis of them the latter is able to deliver the needed data in the shortest time. An optimizer is not a product but a property of a given database engine. Every producer of a database software, for example: IBM (DB2 bases), Microsoft (MSSQL) or OpenSource group (PostgreSQL), develops its own database engine, hence it supports its own algorithms which aim is to analyze the data access time to make it the shortest. In the case of basis PostgreSQL (it contains its own cost optimizer) the main factor to choose the query plan is the estimation of I/O disk operations cost. Its makers claim that this is the main factor and that is why PostgreSQL chooses the plan with the least number of such operations. Notice that in this case thanks to the code is open it is possible to analyze the used optimization codes (when a base is commercial then codes are classified).

As it has been mentioned an alternative approach is to use the cost-based optimizer. The way of operation of the cost-based optimizer relies on generating all the theoretically possible query plans on the basis of such information as: table size, number of rows, the width of the key. On the ground of this a plan is created which is stored in the table PLAN_TABLE. Then the engine of the database evaluates the set of plans of the query by analytical assigning the time of execution to them. The analysis performed by the cost-based optimizer relies on the statistics from the tables and on the indexes referring to the number of records stored in them, unique keys, etc. These statistical information becomes available for the optimizer after execution of the command ANALYZE upon the given table or they are collected by DBMS_STAT package. If the table is not analyzed and there is no information about the statistics, in order to choose the best path for data access, cost-based optimizer uses the rules of logic. Formulated SQL query will be subjected to optimization of cost only in the case that the statistical analysis has been performed for at least one of the tables listed in this command. Then the optimizer seeks for the best path on the basis of information from the data dictionary [1]. Eventually, this query plan is used for which the estimated execution time is the shortest.

In practice, the evaluation of query plans does not use the real time of query execution but the cost of a query. For the products of Oracle the cost of the query equals to the approximate time of one operation O in the operating system. In database products of Oracle—from version 9i—cost-based optimizer is a tool by default.

The aim of the paper is to evaluate the influence of selected factors such as indexing, aliasing, asymmetry of data and availability of statistics referring to tables on the effectiveness of a query (the selection of the best query execution plan) implemented by the cost-based optimizer module in the Oracle database system. Below, at first the tested database environment and the plan of experiments, then the results of experiments are discussed. In the conclusions some general remarks are included.

2. ENVIRONMENTAL TEST DATABASE

Measurements of the time and cost of a query was performed in the environment presented in Fig. 1.

For estimation of the query execution plan cost the cost-based optimizer is to assess numbers of records (so called cardinality) which will be processed in different phases of the realization of the query. Precision of these assessments is important for decisions referring to, for example, the sequence and the algorithm of joining the tables. Estimation of cardinality is done on the basis of the available statistics of tables and columns thereof. The optimizer is not always infallible, especially when there are some compound predicates in the query or the available statistics are imprecise or out of date [2].

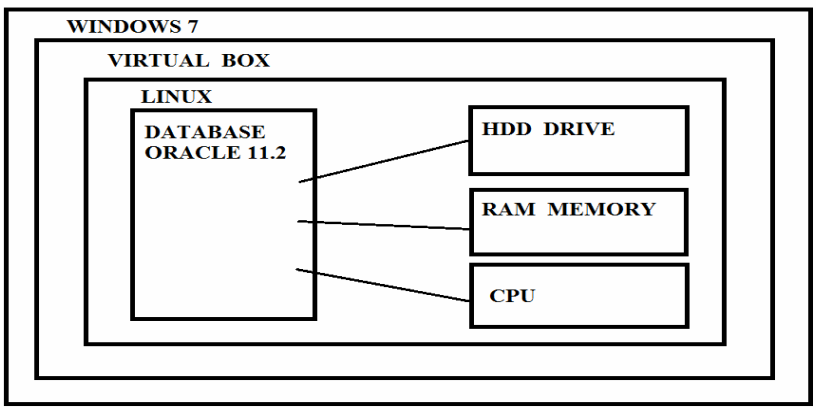

Fig. 1. Database environment schema used in the experiment

In experiments OracleVM VirtualBox virtualization platform (the only one fully supported by Oracle) was used. It is based on implementation of the architecture of 64-bits Intel X86 and Windows 7 (64-bits also). Database environment schema used in the experiment is shown in Figure 1.

Operating system ORACLE Enterprise Linux Server 5 in the 64-bits version (recommended by ORACLE) was chosen as the Guest system. Oracle database

version 11.2.0.2.0 was installed in it. Resources for the virtual machine were allocated as follows:

- the amount of RAM: 900 MB,
- i7 processor units: 1,
- system disk: hda1 11GB,
- database disk: hdb1 12GB.

The database files, i.e. the so-called binaries (database software), data and system files (redolog file, control file, undo, users, temp, system, sysaux) were placed on the disk sdb1. Type charset was set as follows:

- nls_language AMERICAN,
- nls_territory AMERICA,
- nls_date_format DD-MON-RR,
- nls_characterset AL32UTF8,
- nls_nchar_set AL16UTF16.

It was important for the proper functioning of the database to reserve in the memory an appropriate size of SGA (System Global Area, see fig. 2), which is a shared area of memory that stores data and the control information for the database instance. System software allocates memory area every time you start the database and releases it at closing. Users who log into the database, use the memory space for sharing information. In order to maintain optimal system performance database memory allocation practice is the relatively large size of the SGA to be able to store in RAM as much information as possible and thereby minimize the number of operations I / O between the database instance and hard disks [3].

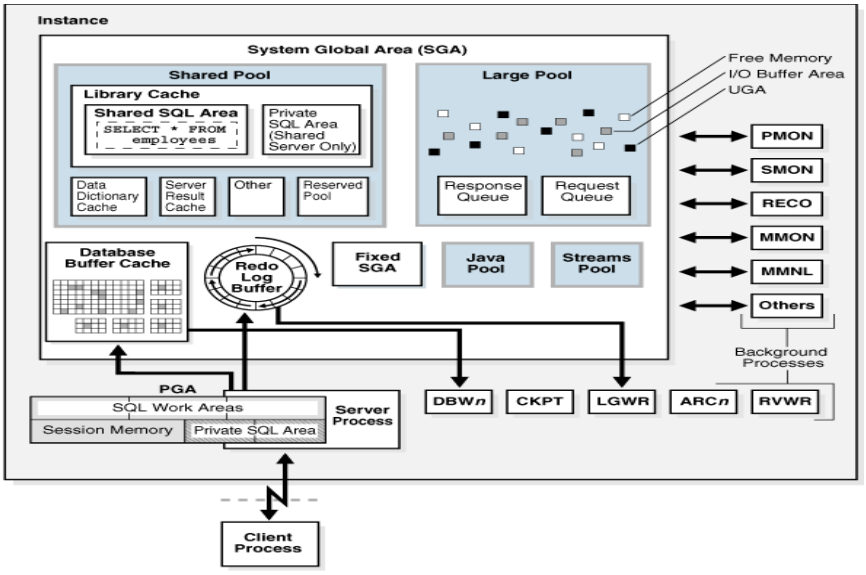

Fig. 2. The schema of distribution of SGA memory in RAM allocated to the database [4]

Table 1 shows the parameters that are associated with the allocation of appropriate amounts of RAM for each area of operation. The value zero means that the memory allocation for the indicated parameters is done automatically—the database decides on it itself based on its load. The database administrator specifies only the upper limit of the memory that the database can book into the operating system – Memory Target parameter.

Table 2 shows the selected parameters of cost-based optimizer, which are used by the database management system and are taken into account when choosing the query execution plan.

Table 1. The values of the basic parameters of the base memory

Parameter	Type	Value
SGA	Big Integer	457179136 kB
Shared Pool	Big Integer	0
Large Pool	Big Integer	0
PGA	Big Integer	0
SGA_Target	Big Integer	0
Memory max target	Big Integer	457179136 kB
Memory target	Big Integer	457179136 kB

Table 2. Optimizer parameters

Parameter	Type	Volue
Optimizer_dynamic_sampling	Integer	2
Optimizer_index_caching	Integer	0
Optimizer_index_cost_adj	Integer	100
Optimizer_mode	String	ALL_ROWS
Query_rewrite_enabled	String	True
Skip_unusable_indexes	Boolean	True
Statistics_level	String	All

The experiment has been performed in interprocess communication, using SQLPLUS program that was run on the same machine as the database. This allowed that the experiment results were independent from the network bandwidth and from the load the client's computer. Further, the software SQLPLUS was used as the database administrator's console. Experiments were conducted with active AUTOTRACE database function, it allowed to display execution plans with SQLPLUS.

3. EXPERIMENTS AND THE RESULTS DISCUSSION

The aim of the experiment was to evaluate the effect of indexation, aliasing, data asymmetry and unsuitable data analysis for the cost and time of the query, and thus for the efficiency of the cost-based optimizer in database system ORACLE.

3.1. Impact of indexation

The first experiment was to verify the impact of indexation or lack thereof on the cost of the query [5]. It was assumed that the assessment of the cost of making an SQL query and its execution time will be carried out on operations such as SELECT and for five different sizes of databases. In the first series of experiments 5 queries to the tables were performed on which the index had been removed with the structure of the tables maintained (the number and width of columns were the same). The difference between individual measurements was the assumed number of records stored in the table. In a second series of experiments the same tables as in the first series were used, but an index was set up in them, its structure is presented in Figure 3. Moreover, each database query was restricted to the same size of the sample, it was 1000 records.

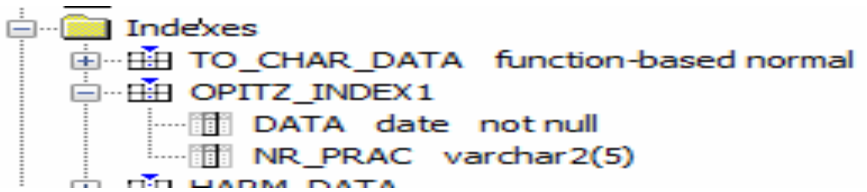

Fig. 3. The structure of the index used

The results of the experiment which aim was to evaluate the impact of indexation and its lack on the cost and the time of a query, depending on the size of the database are shown in Figures 4.

The experiment showed that the set-up of the index has a significant impact on reducing the cost of a query, and hence also for the duration of its execution. In the analyzed case it was based on two columns from the test table, including the column of the primary key. Especially significant benefit was observed for the table that contained more than 8,000,000 records. The experiment confirms that the index set-up has eliminated the full scan of the test table and thus greatly reduced the number of physical reads from the disk.

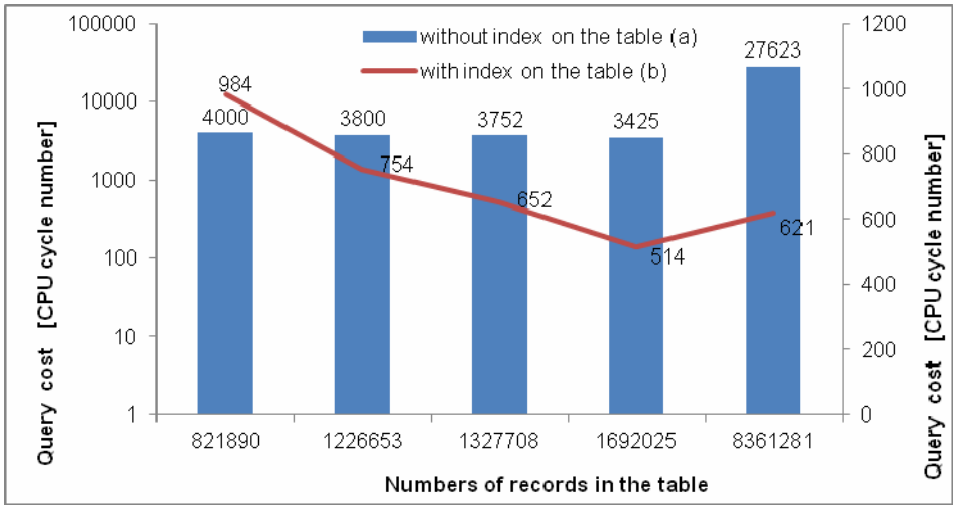

Fig. 4. The cost of a query for different sizes of data in the database without index on the table (**a)** and with index on the table (**b)**

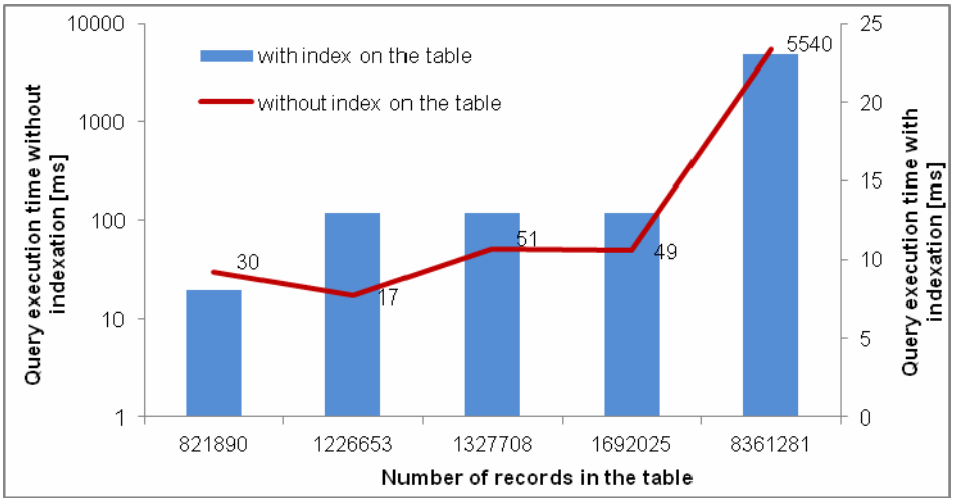

Fig. 5. Estimated time [ms] of the query execution for different sizes of data in the database

3.2. Impact of aliasing

The aim of the second experiment was to examine the effect of aliasing on the time and cost of a query. The time and cost of the query was tested in two cases with an alias put on the table and with the alias removed. The test case without an alias is not hypothetical because in practice faced with the need to obtain the information

from more than one table, the names of which have a dozen characters, and with the complexity of the query you often involuntarily omit the alias that you had defined. In fact, aliasing is not only a facility but it eliminates unnecessary searching the database and returning erroneous results, especially when the names of the columns used in the query are the same for different tables. In addition, the influence of the change of the parameter of optimizer OPTIMIZER_ INDEX_COST_ADJ on the time and cost of a query was verified. Cases with alias and without it were examined.

```
SELECT nvl(round(nvl(Sum(CZAS_NPRZ)+CASE WHEN data_umowy>='2000/01/01' THEN abs35 ELSE
0 END,0)/60,1),0) wynik FROM
(
select a.NR_PRAC, a.KOMORKA, a.DATA, a.CZAS_NORM_PRZEPR, a.CZAS_DYZUR, a.CZAS_NPRZ,
a.CZAS_NPRZ_W, a.ABSENCJA, to_char(data_baza,'YYYY/MM/DD'), abs35, data_umowy, abs33
from t_karta_cz a, s_slow_z3 s, t_ew_pracownik b, tab_grupy_przebieg p where
a.data between '2000/07/01' and '2013/09/30'
and a.absencja in ('CD','CC','CS','CG','CH','CW','CX','W','WZ','WN','OD','O','P','S','UZ','PU')
and a.nr_prac=b.nr_prac
and a.nr_prac=p.nr_prac
and p.nazwa_op='PM_POZNAN_KURZAJIRENA'
and a.komorka=s.kod
AND b.data_baza<='2000/09/30'
AND p.data_od=(SELECT Max(data_od) FROM tab_grupy_przebieg z, t_karta_cz a WHERE
a.nr_prac=z.nr_prac
AND z.data_od<=a.data
and z.nr_mpk in(select nr_mpk from lista_mpk_wz where nazwa_op='PM_POZNAN_KURZAJIRENA')
and z.nazwa_op='PM_POZNAN_KURZAJIRENA'
and z.char_zatr NOT IN ('23','24','31','41','42','43','44','45','46','47','51','91','92','93')
)
)
WHERE Nvl(abs33,'N')='N' OR Nvl(abs33,'N')='T'
group by CASE WHEN DATA_UMOWY>='2000/01/01' THEN abs35 ELSE 0 END
```

Fig. 6. Query SELECT formulated for the evaluation of the benefits of aliasing tables

Aliases that have been removed are marked with grey in Figure 6. Collected results are presented in Figure 7. The middle column represents the point of reference for the presented results, because they present the cost and time of a query that was stored properly and optimizer parameters had not been modified. Omission or removal of aliases resulted in a twofold increase in the cost — column on the left (9832 processor cycles)—and up to 30-fold increase in the time required to execute the query—also on the left (1580ms)—compared to baseline or cost (4389 CPU cycles) and time (53ms).

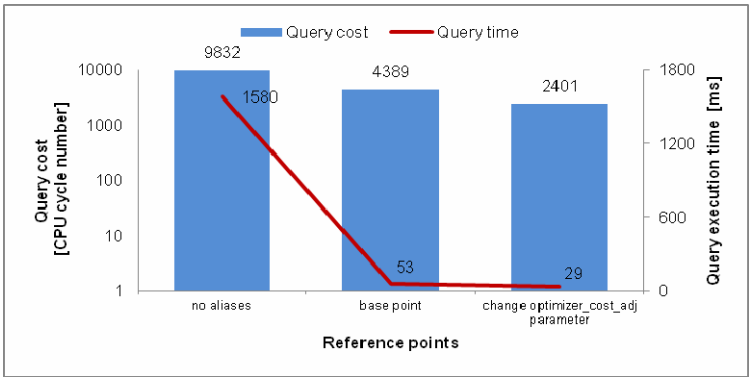

Fig. 7. The impact of aliasing and its absence in the time and cost of a query

The right column in Figure 7 illustrates the results obtained after changing the parameters of the optimizer. Based on this it can be seen the change of the parameter (optimizer_cost_adj) reduces execution costs by 50% compared to the representative sample (middle column). This is because as a result of the change the optimizer is forced to a more efficient use and checking the indexes defined for the tables. This is confirmed also by the print of the query plan (see Figure 8a and 8b). For example the scan of index Z3_PK_KOD_KOMOD changed from FAST FULL SCAN to INDEX RANGE SCAN, whereby the number of rows to be searched decreased from 1277 to 1 and thereby the number of active bytes—from 8939 to 7.

a)

```
          TABLE ACCESS BY INDEX ROWID| T_KARTA_CZ          |     2 |    56
            INDEX RANGE SCAN         | OPITZ_INDEX1        |    29 |
         TABLE ACCESS BY INDEX ROWID | T_EW_PRACA          |     1 |    41
           INDEX UNIQUE SCAN         | T_EP_PK             |     1 |
        INDEX RANGE SCAN             | Z3_PK_KOD_KOMOD     |     1 |
      TABLE ACCESS BY INDEX ROWID    | S_SLOW_Z3           |     1 |    16
     TABLE ACCESS FULL               | S_CENTRA            |    20 |    40
    INDEX UNIQUE SCAN                | T_EO_PK             |     1 |
   TABLE ACCESS BY INDEX ROWID       | T_EW_OSOBA          |     1 |    15
  INDEX FAST FULL SCAN               | Z3_PK_KOD_KOMOD     |  1277 |  8939
 INDEX FAST FULL SCAN                | Z3_PK_KOD_KOMOD     |  1277 |  8939
INDEX FAST FULL SCAN                 | Z3_PK_KOD_KOMOD     |  1277 |  8939
INDEX UNIQUE SCAN                    | PK_S_JEDN           |     1 |     5
```

b)

```
          TABLE ACCESS BY INDEX ROWID| T_KARTA_CZ          |     2 |    56
            INDEX RANGE SCAN         | OPITZ_INDEX1        |    29 |
         TABLE ACCESS BY INDEX ROWID | T_EW_PRACA          |     1 |    41
           INDEX UNIQUE SCAN         | T_EP_PK             |     1 |
        TABLE ACCESS BY INDEX ROWID  | T_EW_OSOBA          |     1 |    15
          INDEX UNIQUE SCAN          | T_EO_PK             |     1 |
       INDEX RANGE SCAN              | Z3_PK_KOD_KOMOD     |     1 |     7
      TABLE ACCESS BY INDEX ROWID    | S_SLOW_Z3           |     1 |    16
       INDEX RANGE SCAN              | Z3_PK_KOD_KOMOD     |     1 |
     INDEX UNIQUE SCAN               | PK_S_JEDN           |     1 |     5
    INDEX RANGE SCAN                 | Z3_PK_KOD_KOMOD     |     1 |     7
   INDEX RANGE SCAN                  | Z3_PK_KOD_KOMOD     |     1 |     7
TABLE ACCESS FULL                    | S_CENTRA            |    20 |    40
```

Fig. 8. Query execution plans: before changing the parameters of the optimizer (a), after the change (b)

3.3. Data asymmetry in the table

In the third experiment the influence of the asymmetry of the data on the cost of downloaded data have been evaluated. The problem of asymmetry refers to the situation, when an index is defined for a column, but its type is inappropriate for the data stored in the latter. If the data stored in the given column get the assumed values, for example 0 or 1, a unique index is to be defined. Otherwise, the optimizer will not be able to determine the correct distribution (the number of records having the value 0 or 1 in the column) and it will assume a uniform distribution of each of the values that occur in it. Such an assumption makes that during the selection of records whole the table is viewed row by row [6].

The study referred to a CUSTOM table consisting of four columns; the scheme, layout, and width thereof is shown in Figure 9.

```
SQL> desc custom
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 CUST_ID                                            NUMBER
 LAST_NAME                                          VARCHAR2(30)
 FIRST_NAME                                         VARCHAR2(30)
 STATE                                              VARCHAR2(1)
```

Fig. 9. Schema of the examined table

Information stored in column STATE is used to describe the status of the order and it may get only two values (O—order status Open and C—Close—the order closed). Additional information about the array CUSTOM includes:

- the number of records stored in the table during the experiment: 1M,
- the number of records on the status C: 999 900,
- the number of records on the status O: always.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1470893142

-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |  1000K|    52M|  2468   (1)| 00:00:30 |
|   1 |  TABLE ACCESS FULL| CUSTOM |  1000K|    52M|  2468   (1)| 00:00:30 |
-------------------------------------------------------------------------------
```

Fig. 10. Execution plan for the query selecting all the records with the status O (Open) from the table CUSTOM

In Figure 10 it can be seen that the lack of current statistics forced the optimizer to assume as a default that the data in the table are evenly distributed 50/50. Under this assumption, database ORACLE carries out a full table scan every time, i.e. it

performs TABLE ACCESS FULL, checking the status of orders for each of the millions of records stored in the freezer. The cost of the query in this case was very high and equaled 2468; the amount of data downloaded for analysis was 52MB.

To eliminate the problem of a full table scan a unique index was set up on the column STATE; statistics for the table and all indexed columns (indexes) were collected.

Fig. 11. Execution plan for a query selecting all the records with the status O (Open) from the table CUSTOM, after collecting statistics

The calculation of the current statistics for the table and its indexes caused that the cost of execution fell to 4 CPU cycles, see Figure 11. When compared to the results of previous experiments, the increase in speed of execution of queries is eminent.

3.4. The impact of statistics and the number of records in a table

In the fourth experiment the impact of inappropriate data contained in the database for the cost of queries was examined. The situation of inappropriate data occurs frequently when an analysis of tables and indexes is carried out, and in the given moment they do not contain typical amount of data. For effective choosing the best execution plan a cost-based optimizer must have the accurate information about the volume of the data in the examined objects, or in other words it needs the access to current statistical information about them. To sum up, the most common situations when inadequate analysis of the data may occur are:

- no current statistics,
- analysis of the table at the time when it is empty and not at the moment when it contains hundreds, thousands or millions of rows.

As a test object table CUSTOM3 was used. It was built on the basis of pre-test table CUSTOM with some additional assumptions such as:

- number of rows: 1 million,
- lack of calculated statistics for the table,

- no indexes,
- number of rows retrieved in a sample of 100 - all with the status O (Open),
- disabled dynamic sampling (automatic statistics),

Figure 12 shows the execution plan for the baseline experiment.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1873066319

--------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |  7414 |   354K|  2468   (1)| 00:00:30 |
|*  1 |  TABLE ACCESS FULL| CUSTOM3 |  7414 |   354K|  2468   (1)| 00:00:30 |
--------------------------------------------------------------------------------

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

   1 - filter("STATE"='O')
```

Fig. 12. The execution plan for table CUSTOM3, no current statistics, there are no indexes

As it is shown in Figure 13 the index set-up without calculating statistics resulted in the decrease of the cost of the query to 42. Figure 14 shows the cost of a query after collecting statistics, but only for the same table CUSTOM3. Execution analysis only for this table resulted in the return to the baseline, i.e. to the full overview of the table row by row.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1742187172

--------------------------------------------------------------------------------
| Id  | Operation        | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |            |  7414 |   354K|    42   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| DR_CUSTOM3 |  7414 |   354K|    42   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   1 - access("STATE"='O')
```

Fig. 13. The execution plan for CUSTOM3, lack of statistics, the index on the columns STATE, CUST_ID, LAST_NAME, FIRST_NAME

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1873066319

--------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |   500K|    26M|  2470   (1)| 00:00:30 |
|*  1 |  TABLE ACCESS FULL| CUSTOM3 |   500K|    26M|  2470   (1)| 00:00:30 |
--------------------------------------------------------------------------------

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

   1 - filter("STATE"='0')
```

Fig. 14. The execution plan for CUSTOM3, statistics collected only for the table, the index on the columns STATE, CUST_ID, LAST_NAME, FIRST_NAME

Gathering statistics both for the table and its index, see Figure 15, allows for the reduction of execution costs to 3 processor cycles only, and to eliminate the problem of inadequate data analysis.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1742187172

--------------------------------------------------------------------------------
| Id  | Operation        | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |            |   100 |  5500 |     3   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| DR_CUSTOM3 |   100 |  5500 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------

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

   1 - access("STATE"='0')
```

Fig. 15. The execution plan for CUSTOM3, statistics collected for the table and the index, an index defined for the columns STATE, CUST_ID, LAST_NAME, FIRST_NAME

4. CONCLUSION

The results of experiments presented in this paper show how many factors can affect the cost of a query and its time. As the experiment results show the following factors are important to reduce both the cost and the time of a query: indexing, lack of indexes on the table, improperly defined indexes, outdated statistics on tables and indexes. Errors in the query formulation are adverse, too. They significantly decrease the efficiency of cost-based optimizer, as it is shown in the second experiment.

The experiments confirm the relevance of the query analysis, and thus the analysis of the query execution plan. Analysis of query plans allows on:

- verification of the assumed indexing,
- verification of the cost of execution,
- the assessment of the volume of the downloaded bytes,
- trace the individual steps that the database must go through to send the requested data to the user.

Another way to increase efficiency is to optimize query execution plans. On the other hand a purely technical solution would be to improve hardware, it is, to provide some additional equipment, including processors, disks and RAM exchange for more efficient ones.

REFERENCES

[1] Gurry M., *Oracle SQL Tunning Pocket Reference*, Wydawnictwo Helion, Gliwice 2002.

[2] Zakrzewicz M., http://explainit.pl/blog/?paged=2 [online], [09.09.2014].

[3] Cyran M., Lane P., *Oracle Database Concepts, 10g Release 1 (10.1)*[online], Oracle Press, December 2003, http://docs.oracle.com/cd/B12037_01/server.101/b10743.pdf [16.06.2014].

[4] Ashdown L., Kyte T., *Oracle Database Concepts*, 11g Release 2 (11.2)[online], Oracle Press, September 2011, http://docs.oracle.com/cd/E29505_01/server.1111/e25789/process.htm [16.06.2014].

[5] http://pl.wikipedia.org/wiki/Indeks_%28bazy_danych%29 [online],[16.06.2014].

[6] Muryjas P., Skublewska-Paszkowska M., Gutek D., *Współczesne technologie informatyczne. Eksploatacja baz danych.*, Politechnika Lubelska [online], Lublin 2011, http://www.bc.pollub.pl/dlibra/docmetadata?id=675&from=&dirids=1&ver_id=&lp=1&QI= [16.06.2014].