Paper

Query Optimization in Teradata Warehouse

Agnieszka Gosk

Abstract—The time necessary for data processing is becoming shorter and shorter nowadays. This thesis presents a definition of the active data warehousing (ADW) paradigm. One of the data warehouses which is consistent with this paradigm is teradata warehouse. Therefore, the basic elements of the teradata architecture are described, such as processors parsing engine (PE) and access module processor (AMP). Emphasis was put on the analysis of query optimization methods. There is presented the impact of a primary index on the time of query execution. Furthermore, this paper shows different methods of optimization of data selection, data joins and data aggregation. All these methods can help to minimize the time for data processing. This paper presents experiments which show the usage of different methods of query optimization. At the end some conclusions about different index usage are included.

Keywords—active data warehouse, query optimization, teradata.

1. Introduction

The time of data processing is important nowadays. There are popular solutions which improve this time, for example, OLAP systems, streaming databases. There is also a new solution – active data warehousing (ADW), which is not used as often as the systems mentioned before.

The ADW paradigm is related to data warehouse, which is updated as fast as possible. ADW allows minimization of the time between events and decisions which are made in connection with this event. Therefore, such decisions are much more valuable. The primary objectives of ADW are to decrease the time of decision-making, as well as to enhance the reliability of these decisions [1].

The reliability of decisions can be increased thanks to basing them on current data. Therefore, in data warehouse which is consistent with the ADW paradigm, data should be updated as fast as possible [2]. At the moment of appearance of any event (modification of the data in a source system) the data warehouse should be updated. It is possible by introducing a mechanism of triggers, which after the appearance of an event that is meeting certain conditions, update the data warehouse [3].

A rapid response to a query is possible through optimization of queries. The overview of selected query optimization methods is the objective of this document.

2. Theoretical background

2.1. Teradata Architecture

The teradata architecture is very specific. It is presented in Fig. 1 [4].

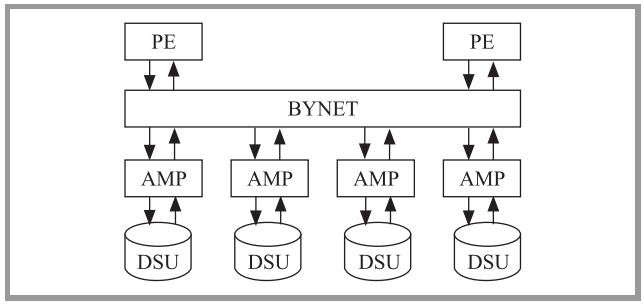

Fig. 1. The architecture of teradata warehouse.

The BYNET is a high-speed network element. It is used to transfer data between parsing engine (PE) and access module processor (AMP).

The PE is a virtual processor. It is responsible for communication between client application and the database (receives a request from the client applications and returns response rows to the requesting client). When PE receives a query, it checks the session parameters (manages session) and divides the query into steps. Then it controls the step execution that is performed by the AMPs. The PE has a few elements, which are described below.

The parser checks if a query, which was sent by a client application, is written correctly. It checks its syntax and whether the user has appropriate rights to all objects, which were used in that query. The optimizer chooses the best method of query execution. For example it can choose a sequence of table joins. The best method of the query execution is presented as a tree and is sent to the generator. The generator converts the tree, which was sent by the optimizer, into steps and sends all the steps to the dispatcher. The dispatcher sends steps of the query to the appropriate AMPs. Then it controls the execution of all the steps and the sequence in which they are executed. Some steps can be executed parallelly, but there are steps that can be executed only after finishing other steps.

The AMP is a virtual processor. It controls a specific disk subsystem – virtual disk. The AMP manages its own disk subsystem and sets the response rows on the basis of its own disk subsystem. It can execute aggregation, sorting,

joins. All data transformations are executed according to the steps, which were sent by the PE.
The disk storage unit (DSU) is a physical disk subsystem – virtual disk. It is managed by one and only one AMP [4].

In teradata warehouse there are several types of indexes:

– primary index,

– secondary index,

– join index,

– hash index.

It is important to build indexes, because they can dramatically improve the time of data processing. The description of each type of indexes is given below.

2.1.1. Primary Index

It is the most important index, because it has to be in each table. When a primary index is created, the database does not build any additional table, which can store values of that index. If a primary index is not created on a table, then the database creates it. On the basis of values of the column set that define the index, the hash value of each row is calculated. This value determines on which AMP the mentioned row is going to be kept. Therefore, when a search for data with a specific index value is performed, the database hashes this value. On the basis of this hash value the database knows on which AMP appropriate data is stored. Only one AMP software is searching for the requested data. So the entire table does not have to be scanned.
There are two types of primary indexes:

– unique primary index (UPI),

– nonunique primary index (NUPI).

In a table with a unique primary index each value of primary index has to be unique [4].

2.1.2. Secondary Index

Secondary indexes are not required. They do not affect the data distribution. Like a primary index, a secondary index can be:

– unique,

– nonunique.

A secondary index can improve the time of query execution, when a table with a defined secondary index value is searched. Additionally, a unique secondary index forces uniqueness of the index values.
When a secondary index is built on a base table, a subtable is created. This subtable stores secondary index values, secondary index hash values and hash values of the primary index of each row. Therefore, when a search for data with a defined secondary index value is preformed, the database hashes this value. The AMP, which was indicated by the secondary index hash value, searches for the appropriate row in the subtable. In this row there is the hash value of the primary index. This last hash value indicates the AMP which stores the requested row from the base table. Finally, the indicated AMP searches for the requested row [4].

2.1.3. Join Index

A join index can be defined in one or more tables. When the join index is built on a base table/tables, a subtable is created. In this subtable there is a copy of some data from the base table/tables or a subset of base table columns. A query can be executed accessing the index (subtable) instead of joining and accessing the base tables. Generally, join indexes can improve the time of data processing [4].

2.1.4. Hash Index

A hash index can be compared with a join index and a secondary index. Like the join index defined on one table, the hash index can redistribute rows from the base table across the AMPs. Like a secondary index each row of the hash index has a pointer to an appropriate row from the base table [4].

Summarizing, it is known that the primary indexes influence data distribution. Therefore, they can improve the time of all operations. The time of data selections can be improved by primary index, secondary index or hash index. The data joins can be executed faster thanks to primary indexes, secondary indexes, hash indexes or join indexes. The join indexes also improve the time of data aggregations.

But it is not known how strong various types of indexes can improve the time of different operations. It is difficult to say how much disk storage various indexes can occupy or how many costs they cause. In the next part of this paper some experiments are presented, which give some answers.

3. Experiments

Experiments were performed on the same server, with the following parameters:

- Dual Core AMD Opteron
  Processor 880
  2,39 GHz
- 2,00 GB RAM

The Microsoft Windows Server 2003 Standard Edition system was installed on the server.
Experiments were executed on the Teradata Warehouse 8.1Demo. In this version of the teradata system there are only 2 AMP processors and 1 PE processor available. In the DEMO version BYNET element does not exist and disk space is limited to 4 GB.

Two tables were prepared and used to carry out tests. A definition of these tables is presented below.

```
CREATE TABLE CLIENTS (
    client_id INTEGER NOT NULL,
    name VARCHAR(20),
    surname VARCHAR(20),
    street VARCHAR(20),
    home_no VARCHAR(8),
    city VARCHAR(20),
    tariff_plan VARCHAR(15),
    status CHAR(1),
    phone_type VARCHAR(18),
    phone_number INTEGER,
    activation_date DATE);
```

```
CREATE TABLE CLIENT_CONNECTIONS(
    client_id INTEGER NOT NULL,
    connection_type VARCHAR(1),
    connection_direction VARCHAR(2),
    count INTEGER,
    volume INTEGER);
```

The CLIENTS table stores basic data about clients, who are active. The CLIENT_CONNECTIONS table stores data about connections, which were performed by active clients. In the experiments described, various primary indexes and added indexes are defined in the tables. These indexes are presented in experiment descriptions.
For each table four sets of data were prepared. In each set there was a different number of clients and a different number of rows. Tables 1 and 2 present information about the CLIENTS and CLIENT_CONNECTIONS tables.

Table 1
The CLIENTS table statistics

Data set	Number of clients	Number of rows	Size[B]
A	500 000	500 000	43 822 080
B	1 000 000	1 000 000	86 585 344
C	1 500 000	1 500 000	129 408 512
D	2 000 000	2 000 000	172 274 688

Table 2
The CLIENT_CONNECTIONS table statistics

Data set	Number of clients	Number of rows	Size[B]
A	500 000	2 141 184	77 952 000
B	1 000 000	4 282 709	155 574 784
C	1 500 000	6 425 485	232 721 920
D	2 000 000	8 567 631	310 469 632

The problem of query optimization is not new. There are many papers concerning this. Many of the optimization methods show how to write a query. Disappointingly, in teradata warehouse these methods are not effective. But there are methods of query optimization, which rely on index usage. These methods can be used in teradata warehouse. In this paper these methods are presented.

3.1. Influence of Data Distribution on Query Execution Time

As it was said, a primary index is the most important index which influences data distribution. In this section experiments which show how data distribution determines the time of the execution of various types of queries, are presented.

Experiment 1

Firstly, there was checked the primary index affected on the time of selection of all rows from the CLIENTS table. In this experiment a following query was executed:

```
SELECT *
FROM CLIENTS;
```

The skew factor shows how equally data is distributed across the AMP processors. The bigger this factor value is, the more unequally data is distributed. During this experiment the primary index of the CLIENTS table was changed, so the skew factor of the CLIENTS table changed. In Fig. 2, there are line graphs which show the query execution time in seconds as a function of data set size for different values of the skew factor.

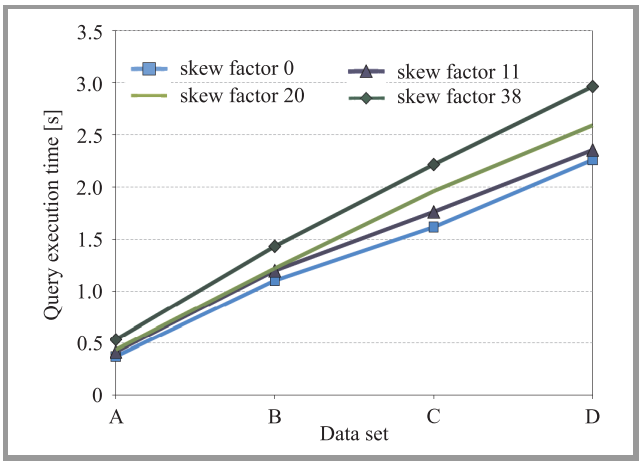

Fig. 2. Influence of the primary index on the time of data selection.

In the figure it is shown that the query execution time increases with growing number of rows. For the same set of data, the query execution time is slower for the CLIENTS table with bigger skew factor values (about 25% for data set D). Therefore, it must be remembered that unequal data

distribution influences the time of data selection significantly.

Experiment 2

This experiment checks how data distribution influences the time of data join. For this test a following query, which chooses number of different connections from CLIENT_CONNECTIONS table for active clients from CLIENTS table, was selected:

```
SELECT C.client_id, CC.connection_type,
    CC.connection_direction, CC.count
FROM CLIENTS C,
    CLIENT_CONNECTIONS CC
    C.client_id = CC.client_id;
```

This query was performed on various sets of data. In the CLIENTS and CLIENT_CONNECTIONS tables there were different primary indexes (they were created on different columns sets). Once the skew factor for the CLIENTS table was 0 and once 38. Results are shown in Fig. 3, there

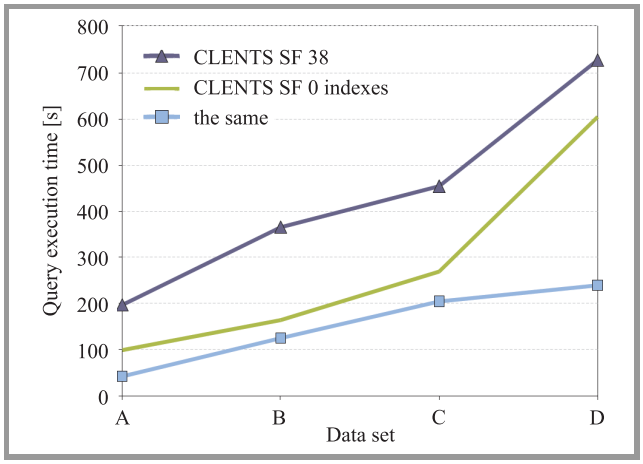

Fig. 3. Influence of the primary index on the time of data joins.

are line graphs which show the query execution time in seconds as a function of data set size for different indexes available in the database. The line graphs are marked according to description:

- The same indexes. In the CLIENTS and CLIENT_CONNECTIONS tables there were the same primary indexes. They were defined on the client_id column.
- CLIENTS SF 0. In the CLIENTS and CLIENT_CONNECTIONS tables there were various primary indexes. The skew factor of the CLIENTS and CLIENT_CONNECTIONS tables was 0.
- CLIENTS SF 38. In the CLIENTS and CLIENT_CONNECTIONS tables there were different primary indexes. The skew factor of the CLIENTS table was 38, and the skew factor of the CLIENT_CONNECTIONS table was 0.

It can be seen, that the time of data join depends on a value of the skew factor. When only one table has a high value of the skew factor the time of query execution is getting worse. In the case when skew factor of CLIENTS table is 38, the time of data processing is even 2 times greater than in the case when skew factor of CLIENTS table is 0. The time of data joins is the best when the tables, which are joined, have the same primary indexes.

3.2. Different Methods of data Selection Optimization

Experiment 3

This experiment examines how indexes can influence the selection of one row from a table.
For the needs of this experiment a following query, which chooses one row from the CLIENTS table, was prepared:

```
SELECT *
FROM CLIENTS
WHERE phone_number = 300001019;
```

During this experiment indexes, which were created in the phone_number column of the CLIENTS table, were changed. Figure 4 presents the experiment results – the query execution time in seconds as a function of data set

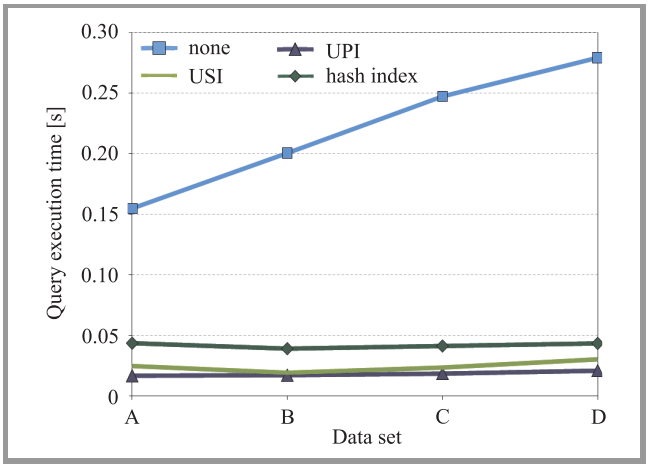

Fig. 4. Influence of various indexes on the time of data selection.

size in a form of different line graphs. These experiments were performed when different indexes were available. The description of graphs is presented below:

- UPI – a unique primary index of the CLIENTS table is defined on the phone_number column,
- NONE – a primary index of the CLIENTS table is defined on the client_id column, on the phone_number column there is no index,
- USI – a unique secondary index of the CLIENTS table is defined on the phone_number column, the CLIENTS.client_id column – is the primary index,

- Hash – a hash index of the CLIENTS table is defined on the phone_number column.

The worst results are received when there is no index on the phone_number column. The best results are received when the primary index is created on the phone_number column. When database uses USI or hash index the results are similar to results when the database uses UPI. In Fig. 4 it can be seen that whatever any index is used: hash, secondary or primary, the time of selection of one row from the table is not dependant on the number of rows in this table.

It must be remembered that USI and hash indexes cause additional costs, they increase the time of table updating and they use additional storage space. In Table 3 there is presented the time of 2 000 000 rows insertion and deletion from the CLIENTS table when additional indexes are available in the database.

Table 3
The time of 2 000 000 rows insertion and deletion from the CLIENTS table (data set D), when additional indexes are available in the database

Operation	NONE	USI	Hash index
Insert [s]	1	9	603
Delete [s]	1	1	238

Secondary and hash indexes on the CLIENTS.phone_number column require additional storage space (for data set D) - 68 MB and 84 MB. Therefore, it is better to create the secondary index, because it influences the time of data insertion and deletion from the CLIENTS table less than the hash index and it occupies less disk space than the hash index.

Experiment 4

The next experiment checks how different indexes influence the selection of many rows from one table.

To carry out this test, a following query, which chooses clients from the CLIENTS table who use PT_1 tariff plan, was prepared:

```
SELECT *
FROM CLIENTS
WHERE tariff_plan = 'PT_1';
```

During this experiment the number of available tariff plans was changed. A different number of clients used plan 'PT_1', therefore, the query which is presented above, returns a different number of rows. The results of this experiment are presented in Fig. 5, there are line graphs which show the query execution time in seconds as a function of number of rows, which are returned, for different indexes available in the database. The line graphs are described as it is presented below:

- NONE – a primary index of the CLIENTS table is defined on the client_id column, on the tariff_ plan column there is no index.
- NUPI – a nonunique primary index of the CLIENTS table is defined on the tariff_plan column.
- NUSI – a unique primary index of the CLIENTS table is defined on the client_id column and a nonunique secondary index of this table is defined on the tariff_plan column

It can be seen that the best results are received when in the CLIENTS.tariff plan column the nonunique primary index is created. The worst results are received when in the same column there is no index.

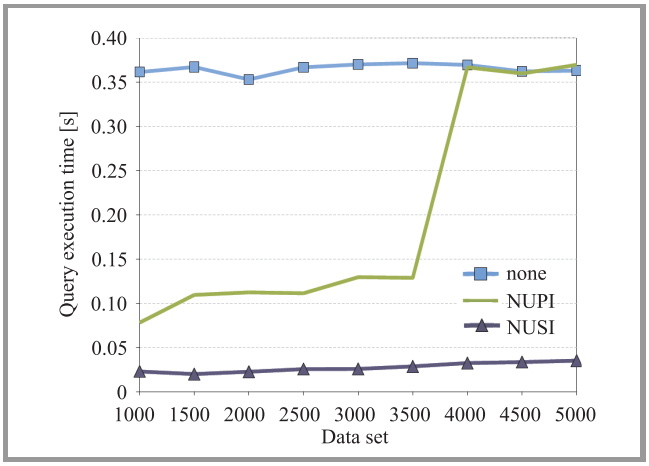

Fig. 5. Influence of different indexes on the time of selection of many rows from the table.

An interesting line graph is received when on the CLIENTS.tariff_plan column a nonunique secondary index is created. When a query returns 3500 records or less the time of query processing is quite good, but when a query returns more than 3500 records, the results are the same as in the case when on the CLIENTS.tariff_plan column there is no index. It is so because when a query returns more than 3500 records it is more efficient to retrieve rows from the base table than from the secondary index. When command EXPLAIN is used to check how a query is executed, two different explanation are returned. One, in the case when a query returns 3500 records or less, the other when a query returns more than 3500 records.

NUSI occupies about 20 MB of disk space and it influences the time of data insertion and deletion from the CLIENTS table slightly. Therefore, when a primary index cannot be created on a column set, which is used for data selection, it can be replaced by a secondary index.

3.3. Different Methods of Optimization of Data Joins

This section presents how different type of indexes can influence the time of data join. It is checked by the experiment which is presented bellow.

Experiment 5

To execute this experiment the following query was prepared, which calculates the number of types of connection directions for voice connections for active clients who have PT_1 tariff plan:

```
SELECT C.client_id,
    COUNT(DISTINCT CC.connection_direction)
FROM CLIENTS C,
    CLIENT_CONNECTIONS CC
GROUP BY C.client_id
WHERE C.client_id = CC.client_id
    AND CC.connection_type = 'V'
    AND C.tariff_plan = 'PT_1';
```

The query which was presented above, was executed using different types of indexes. The results are presented

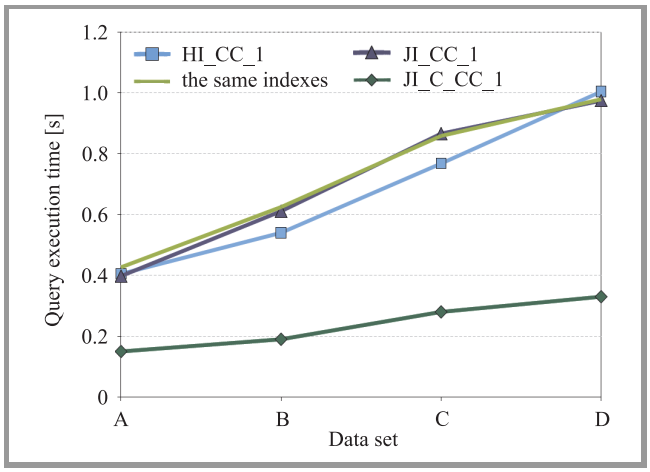

Fig. 6. Influence of different indexes on the time of data join.

in Fig. 6, where the line graphs show the index influence on the time of data join according to description:

- JI_CC_1 – a join index JI_CC_1 is defined on the CLINET_CONNECTIONS table;
- JI_C_CC_1 – a join index JI_C_CC_1 is defined on the CLIENTS and CLIENT_CONNECTIONS tables;
- HI_CC_1 – a hash index HI_CC_1 is defined on the client_id column of the CLIENTS_CONNECTIONS table;
- THE SAME PI – In the CLIENTS and CLIENT_CONNECTIONS tables there are the same primary indexes (on the client_id column).

Definition of join indexes JI_CC_1 and JI_C_CC_1 is presented below.

```
CREATE JOIN INDEX JI_CC_1 AS
SELECT *
FROM CLIENT_CONNECTIONS
PRIMARY INDEX(client_id);
```

```
CREATE JOIN INDEX JI_C_CC_1 AS
SELECT C.client_id, CC.connection_direction
FROM CLIENTS C,
    CLIENT_CONNECTIONS CC
WHERE C.client_id = CC.client_id
    AND CC.connection_type = 'V'
    AND C.tariff_plan = 'PT_1'
PRIMARY INDEX(client_id);
```

When the JI_CC_1 join index, the hash index or the same primary indexes on the CLIENTS and CLIENT_CONNECTIONS tables are used, the time of query processing is similar. In each case data in the JI_CC_1, HI_CC_1 and the CLIENT_CONNECTIONS table is distributed by the client_id column. When the query is executed, data is read directly from the JI_CC_1, HI_CC_1 or the CLIENT_CONNECTIONS table in the different cases and it is joined with the CLIENTS table in each case. In joined tables data is distributed by the same key, therefore, data is joined in AMPs.

When on the CLIENTS and CLIENT_CONNECTIONS tables there are different indexes, the time of query processing is 20 minutes for data set D. It is so because data has to be redistributed across AMPs during join.

The best query execution time is received when join index is defined on the CLIENTS and CLIENT_CONNECTIONS tables (JI_C_CC_1). In this case the two tables do not have to be joined, data can be read directly from JI_C_CC_1. But indexes like JI_C_CC_1 do not have as wide usage as indexes like JI_CC_1.

It must be remembered that the JI_C_CC_1 index influences the time of data insertion and deletion from the CLIENTS and CLIENTS_CONNECTIONS tables. Indexes JI_CC_1 and HI_CC_1 influence the time of data insertion and deletion only from the CLIENTS table. When in the database the JI_C_CC_1 index is defined, the time of 8 567 631 rows insertion into the CLIENT_CONNECTIONS table is about 3 minutes, when in the database index JI_CC_1 or HI_CC_1 is defined this time is about 20 minutes. It is so because the time of data insertion or deletion from the table is dependent on the number of rows from this table, which are inserted into the index, which is defined on this table. Therefore, the more rows in the index, the longer time of data insertion and deletion from the table.

The size of indexes JI_C_CC_1, JI_CC_1 and HI_CC_1 is 308MB, 690KB and 119 MB. It could be critical when database storage is limited.

3.4. Different Methods of Data Aggregation Optimization

This section presents a method of data aggregation optimization. The experiment performed is presented below.

Experiment 6

To show how different indexes influence the time of data aggregation, a query was prepared, which summarizes the number of connections and volume of these connections for clients from the CLIENT_CONNECTIONS table:

```
SELECT client_id,
    SUM(count) as count,
    SUM(volume) as volume
FROM CLIENT_CONNECTIONS CC
GROUP BY client_id, ;
```

The query was executed when various indexes were available in the database. The results are presented in Fig. 7 and particular line graphs show the time of query processing in

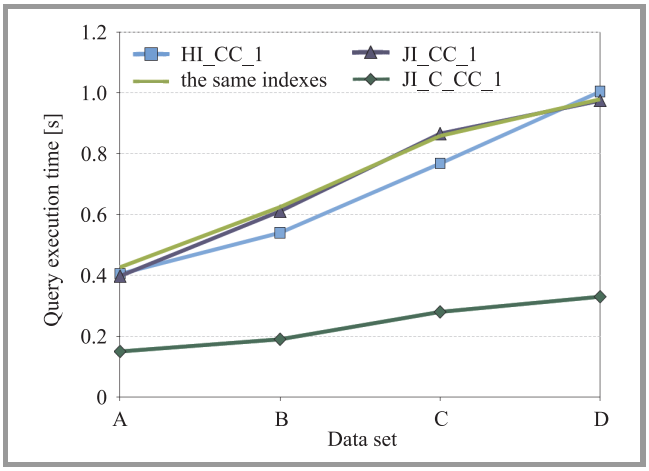

Fig. 7. Influence of different indexes on the time of data aggregation

seconds as a function of data set size, when different indexes are available:

- NONE – in the CLIENT_CONNECTIONS table there is a primary index defined on the column set: client_id, connection_type, connection_direction;
- AI_CC_1 – in the database there is available the AI_CC_1 index;
- AI_CC_2 – in the database there is available the AI_CC_2 index;
- PI – in the CLIENT_CONNECTIONS table there is a primary index defined on the client_id column.

A definition of AI_CC_1 and AI_CC_2 indexes is presented below:

```
CREATE JOIN INDEX AI_CC_1 AS
SELECT client_id, connection_type,
    SUM(count), SUM(volume)
FROM CLIENT_CONNECTIONS
GROUP BY client_id, connection_type;
```

```
CREATE JOIN INDEX AI_CC_2 AS
SELECT client_id, SUM(count), SUM(volume)
FROM CLIENT_CONNECTIONS
GROUP BY client_id;
```

When in the database there are not available any join indexes, the time of query processing is not good. It can be improved when a join index is created or when the primary index is changed. When the primary index is created on the same column as the column which groups data in the query, which is presented above, the time of query execution is quite short. The reason for this behavior is because data aggregation is performed in AMPs.
When in the database the AI_CC_2 index is available, the time of query processing is promising. It is because data is read directly from the index, and it does not need to be aggregated. While in the database there is available the AI_CC_1 index, the time of query processing is worse. Data is read directly from the index, but it has to be additionally aggregated. However the AI_CC_2 index has wider usage than the AI_CC_1 index. The AI_CC_1 can be additionally used to query which groups data by client_id and connection_type. The AI_CC_1 index occupies more disk storage than the AI_CC_2 index.

4. Conclusions

As it was said the primary index is the most important index. It has influence on the time processing of all operations. The time of data join is shorter when joined tables have the same primary indexes. The time of data aggregation is shorter when a table has the same primary index as the grouping column in a query. However, as the experiments have shown, when the primary index is chosen, the most important is the skew factor value of the table. The smaller it is, the better the processing time of all operations is. Furthermore, the primary index should be adapted to executed joins and aggregation.
When there is need to optimize time of data selection on the basis of column set, we can choose the primary index on this column set, a secondary index or a hash index. However, the secondary index and the hash index increase the time of insertion and deletion of data from a table. Most often the hash index has a bigger maintenance cost than the secondary index.

During the data join, there can be used: a join index, a hash index or the primary index can be changed. Different indexes cause different additional costs. Most frequently, the hash index causes lower costs than the join index. Data aggregation can be optimized by the join index or the primary index. The join index which has aggregated data can help to avoid aggregation during query execution.
The above-mentioned conclusions were drawn on the basis of experiments, which were presented in the previous sections. However, in different conditions costs caused by indexes can change. Therefore, it should be remembered that when indexes are chosen, it is most important to calculate additional costs caused by this indexes. It must be known what is more important the processing time improvement or the database space size. Then the best indexes can be chosen.

References

[1] S. Brobst and J. Rarey, "The five stages of an Active Data Warehouse evolution", *Teradata Magazine Online*, 2001 [Online]. Available: http://www.ncr.com/online_periodicals/brobst.pdf

[2] M. Gonzales, "Getting Active", *DB2 Mag.*, iss. 1, Q1, 2005 [Online]. Available: http://www.dbmag.intelligententerprise.com/story/showArticle.jhtml?articleID=59300861

[3] E. Kanana and M. Farhi, "Enhancing data preparation processes using triggers for active datawarehousing", in *Proc. Int. Conf. Data Mining*, Las Vegas, USA, 2006, pp. 153–160.

[4] Teradata Documentation, Database Design, pp. 335–561, Introduction to Teradata. NCR Corporation, 2005.

Agnieszka Gosk received the M.Sc. degree in computer science from the Warsaw University of Technology (WUT), Poland, in 2009. She had been employed by the National Institute of Telecommunication in Warsaw till 2009. She is currently working in the area of data warehousing for a telecommunication operator. Her scientific interests include: data mining, modeling and decision support.
e-mail: agnieszkagosk@gmail.com