BIULETYN INSTYTUTU SYSTEMÓW INFORMATYCZNYCH **7** 61−68 (2011)

The optimization of SQL queries by means of drawing up query diagrams

K. WITAN
kwitan@wat.edu.pl

Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw, Poland

A poorly performing database application not only costs users time, but also has an impact on other applications running on the same computer or the same network. The best method to manage with this problem is performing SQL tuning. There are two basic issues to focus on during tuning: how to find and interpret the execution plan of an SQL statement and how to change a SQL query to get a specific alternate execution plan. But most important is how to find the optimal execution plan for the query to use. This article describes a timesaving method developed for finding the optimum execution plan – rapidly and systematically – regardless of the complexity of the SQL query or the database platform being used. It explains how to understand and control SQL execution plan and how to diagram SQL queries to deduce the best execution plan for a query.

Keywords: query optimization, SQL language, relational database.

1. The purpose of optimization

As the development of SQL-based databases progresses, bigger stress is being put on the output of queries created in a given language. The interest in this particular issue is a natural consequence of longstanding researches, which concern the output of databases and obtained results as well as observations resulting from the programmatic experience of numerous specialists within the scope of a given field. It has been an obvious fact that it is no longer sufficient to just write a programmatic command that would return expected results. Also important is a correct construction resulting not only from the syntax established for a query, but also from other various factors (stemming from the data structure and data handling rules) that may influence functional efficiency. Owing to the conclusions gathered during the aforementioned researches, the database producers began to implement mechanisms into their databases that provide the output measurement as well as tools that enable to determine execution plans, correcting the structure of commands written by programmers [2], [3], [6]. However, the offered tools are not always reliable solutions, hence they cannot be regarded as providing sufficient support for the SQL language code authors. For this reason, many programmers created their own informal methods that are aimed at the verification of the actual output as well as the methods of creating optimal plans for query execution. Nonetheless, transforming the SQL optimization into a science requires to elaborate a set of uniform rules and notations. Among a number of unofficial methodologies, the one worth mentioning is the one proposed by Dan Tow and presented in the book, titled ”SQL Tuning”. This methodology of drawing up query diagrams is based upon the The article elaboration of graphs that contain information relevant in terms of output estimation [1]. The methodology also shapes the base for subsequent actions leading to the determination of a proper query form. Next chapters of the article present an algorithm (in the form of steps) elaborated by onself – based on conceptions of methodology mentioned above – which enables optimization of the query built within one's capacity for publication needs.

2. Graphical representation

The graphical part of the methodology presented in the form of directed graph contains standard elements, that is nodes, which represent tables falling under the query. The tables are interconnected by joins ended with arrows that determine the direction of superiority relation resulting from connections between tables. In addition, each node has assigned numerical values, which depict the filtering indexes, and also the join indexes calculated for each table, thus taken into consideration.

2.1. Nodes

For the purpose of maintaining clear graphical form, the whole names of tables are replaced by aliases of given tables. Aliases may be selected randomly, it is important, however, that those name abbreviations are unique and prevent the ambiguity phenomenon.

2.2. Joins

The arrow-ended side of join indicates the superior table in the relation of tables presented as nodes at the graph. Thereby, there are two unique values at the given end within two connected nodes representing database tables, which are marked out by the query.

According to the convention agreed in the methodology, diagrams should always be constructed so that it is possible to place a downward arrow. In other words, the main assumption of this rule is that superior tables are located at the lower level than linked subordinate tables[1].

2.3. Underlined numerical values

The underlined numerical values are placed next to each node and they reflect the filtering index, that is the number of table rows that meet the filtering condition stated in the query [1], [2], [5], [6], [7]. It is calculated upon the data collected from the database, according to the following relation:

Number of rows meeting the filtering condition/ number of all rows

Sometimes, no filtering indicators are specified in the query to given tables or to certain tables from all of them. In such a scenario, the filtering index takes the value of 1.0, because 100 percent of rows meet the criteria of the (non-existing) filter.

2.4. Non-underlined numerical values

Non-underlined numerical values are located next to each node and they determine the join index, corresponding to the average number of rows found in a table represented by the given node in relation to the adequate rows included in the table represented by the node connected with the previously mentioned node[2] [1], [2], [4], [5], [6], [7]. The join index takes the value calculated upon the following relation:

The number of rows meeting the join condition in a given table/The number of all rows in a given table.

3. Data source

The primary data source, which are necessary to calculate indexes are the queries directed to a database. According to these assumptions, the information, which are needed for each table and which should be collected by means of a developing tool, concern: the number of all rows, the number of rows meeting the filtering conditions and rows, which are coherent with a join condition.

4. Elaboration of the query diagrams

The process of diagram preparation may be divided into following stages:

1. A randomly selected alias for the table included within the FROM clause should be located in the middle of the area destined for the graphical representation of the query. From this point onwards, the table will be regarded as an object of interest.
2. Then it is necessary to establish conditions of joins with other tables determined for the main key of the table selected in the previous point. This enables to indicate the tables subordinate to tables of interest [5]. According to previously presented assumptions, the tables identified as subordinate are located above the tables, which are superior to them while the connection between tables is ended with a downward arrow at the side of the superior table.
3. The following action is the establishing of joins leading from the foreign key of the table of interest to the main keys of the remaining tables within the query. Thus, all found tables should be located at the lower level than the table of interest, as they are in a superior relation to this table. Such a join should be marked by an arrow

[1] Superior tables are also defined as detail tables.

[2] Join indexes are divided into superior table join indexes and detail table join indexes, depending on the join side, where the value-calculated node is located. Superior table join indexes usually take values no bigger than 1, whereas the values for detail tables must be nonnegative and never greater than 1.

directed from the table of interest to the second table located at the lower level.

4. Once all the above steps are completed, the next table, included in the query, should be selected and made a new object of interest, for which the whole above-described procedure should be repeated (steps described in points 2-3). This procedure should be continued until all aliases of tables listed in the FROM clause are put into the diagram and until all joins used by the query are taken into consideration.
5. The subsequent stage after all nodes and joins are put into a diagram is the definition of values for filtering indexes and join indexes, which are determined upon real values stored in a database.

5. Elaboration of effective diagram-based plans for executing queries

Once the query is already illustrated by the graph including all relevant information, it is possible to move to the subsequent stage, which, according to the described methodology, is the determination of the optimal plan of query execution.

The effective execution plan should include the following features [1]:

- the cost of executing the plan is proportional to the number of rows returned as a result of the query execution
- it is not necessary to modify the plan as the table volume increases
- the query execution according to the plan does not require extensive memory space, characteristic for hash functions and sorting functions
- the plan is independent from data distribution and it may be utilized for different instances of the database, which function under the supervision of the same application.

There is a number of rules that must be strictly followed as the plan is elaborated. During the initial stage it is necessary to remember that the access to the first table should be based upon the application of the selective index. In turn, the reference to the following table must be executed by utilization of a nested loop based upon the join key. The key indicates the table previously read by the database. The final factor, which renders the plan effective is the sequence of executed joins. Firstly, the joins with nodes located at the lower level than the analyzed node should be executed through primary keys. Then, links based upon no primary keys should be established with nodes at the upper levels.

Upon the above specified rules, the following sequence of operations may be distinguished, which enable to elaborate the optimal plan for query execution:

1. Choose the table with the lowest filtering index.
2. For the table indicated in the previous point make a join with the related table at the lower level by utilizing a nested loop. The decisive factor for the table selection is the lowest filtering index among the tables at the given level, related with the focus table. The graph shifting should be continued to the lowest available level. The aim of this rule is to obtain conditions, in which full, single-valued indexes of primary keys are utilized.
3. If downward shifting is not possible, it is allowed to utilize the nested loops against the tables located at the levels higher than the analyzed table. In such case, the joins are based upon the full indexes of foreign keys.

6. Execution of complex SQL query

Found below is the example of SQL query, subject to optimization in accordance with methodology of drawing up query diagrams.

```
SELECT C.Phone_Number, C.Honorific,
C.First_Name, C.Adress_Id,A.Adress_Id,
C.Last_Name, C.Suffix, A.Street_Addr_Line1,
A.Street_Addr_Line2, A.State_Abbreviation,
A.City_Name, A.Zip_Code, OD.Item_Count,
OD.Deffered_Ship_Date, P.Prod_Description,
OT.Text, S.Shipment_Date, O.Number
FROM Orders O, Order_Details OD, Products
P, Customers C, Shipment S, Addresses A,
Code_Translations OT
WHERE
UPPER (C.Last_Name) LIKE ‘Kowal%’
AND OD.Order_Id = O.Order_Id
AND O. Customer_Id = C.Customer_Id AND
OD.Product_Id = P.Product_Id (+)
AND OD.Shipment_Id = S.Shipment_Id (+)
AND S.Address_Id = A.Address_Id (+)
AND O.Status_Code = OT.Code
AND O.Order_Date > SYSDATE– 366
ORDER BY C.Customer_Id, O.Order_Id
DESC, S.Shipment_Id, OD.Order_Detail_Id;
```

In order to streamline the analysis process, it is possible to create the list of tables and their primary keys as well as to elaborate the descriptions of relations, which indicate the character of tables within particular links.

Tab. 1. The list reflecting the character of tables in specific relations

Relation	The field constituting the base of relation (primary key of the superior table)	The character of the table in relation	
		Superior	Subordinate
OD.Order_Id = O.Order_Id	Order_Id	Orders O	Order_Details OD
O. Customer_Id = .Customer_Id	Customer_Id	Customers C	Orders O
OD.Product_Id = P.Product_Id (+)	Product_Id	Products P	Order_Details OD
OD.Shipment_ Id = S.Shipment_Id (+)	Shipment_Id	Shipment S	Order_Details OD
S.Address_Id = A.Address_Id (+)	Address_Id	Addresses A	Shipment S
O.Status_Code = OT.Code	Code	Code_Trans lations OT	Orders O

The following stages constitute the procedure for drawing the diagram:

1. Selection of the table of interest – O (Fig. 1).

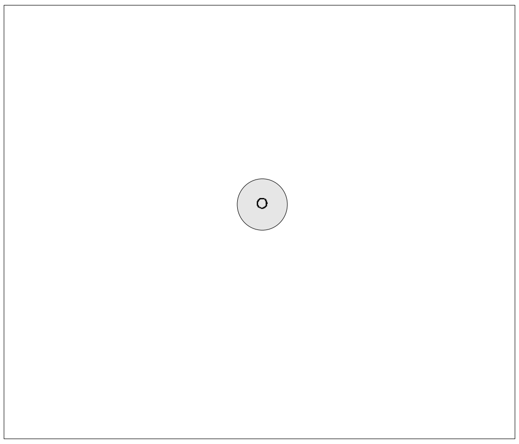

Fig. 1. Table of interest O

2. Indication of the tables subordinate to the table of interest, that is based upon its primary key (Fig. 2).

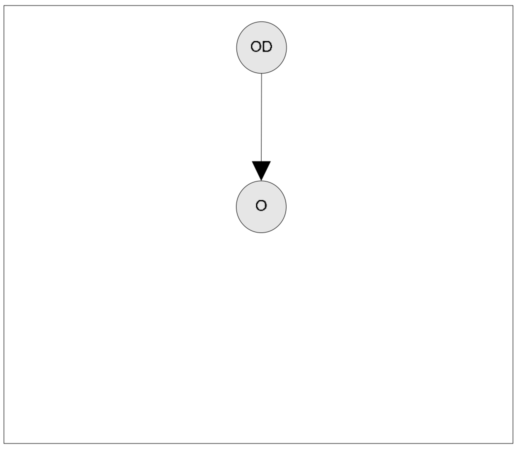

Fig. 2. OD Table subordinate to the O Table

3. Determination of tables superior to tables of interest (Fig. 3).

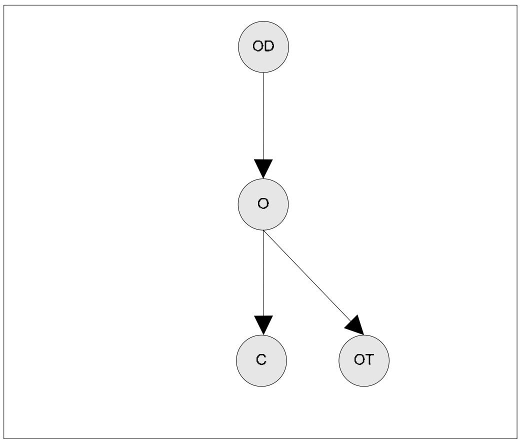

Fig. 3. OD table of interest subordinate to the O Table

4. Determination of a new table of interest – OD
5. Indication of tables superior to the table of interest OD (Fig. 4).

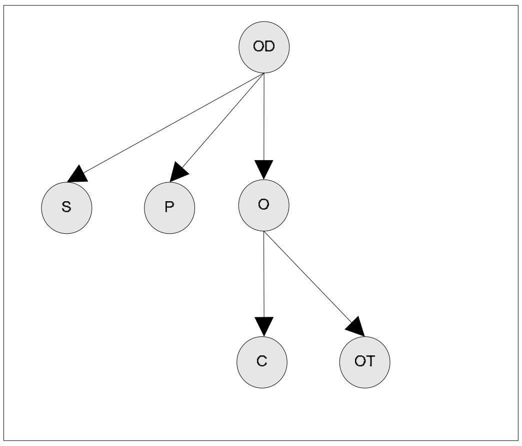

Fig. 4. S, P, O tables superior to the OD Table

6. Verification of the existence of tables subordinate to OD – no tables meeting the requirement.
7. Determination of the subsequent table of interest S
8. Determination of tables superior to S (Fig. 5).

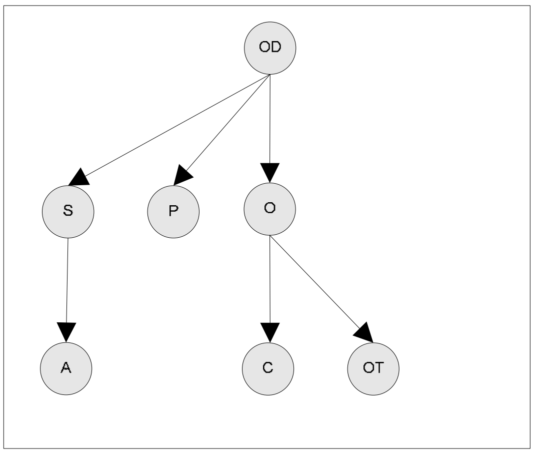

Fig. 5. A table superior to the S Table

9. Verification if all the links between tables included in the query are presented by the diagram – all tables are presented at the diagram.
10. Verification if there are any relations that were omitted in the graphical presentation of the query – no omitted relations found.
11. Determination of the data demand based upon the query. Collection of information from the database, which are necessary to calculate the filtering and join indexes:

Z1: SELECT COUNT(*) W1 FROM Customers WHERE UPPER (C.Last_Name) LIKE 'Kowal%';
W1: 5

Z2: SELECT COUNT(*) W2 FROM Customers;
W2: 5000000

Z3: SELECT COUNT(*) W3 FROM Orders WHERE O.Order_Date > SYSDATE– 366;
W3: 1200000

Z4: SELECT COUNT(*) W4 FROM Orders;
W4: 4000000

Z5: SELECT COUNT(*) W5 FROM Orders O, Customers C WHERE O.Customers_Id=C.Customers_Id;
W5: 4000000

Z6: SELECT COUNT(*) W6 FROM Orders_Details;
W6: 12000000

Z7: SELECT COUNT(*) W7 FROM Orders O, Orders_Details OD WHERE OD.Order_Id=O.Order_Id;
W7: 12000000
Z8: SELECT COUNT(*) W8 FROM Code_Translations;
W8: 4

Z9: SELECT COUNT(*) W5 FROM Orders O, Code_Translations OT WHERE O.Status_Code=OT.Code;
W9: 4000000

12. Calculation of the filtering indexes for tables Fig. 6 – Filtering index of tables):

Customers C – **W1/W2** = 5/50000 = **0,00001**

Orders O – **W3/W4** = 1200000/4000000 = **0,3**

No selectivity conditions have been determined for the remaining tables, thus their filtering index equals 1.

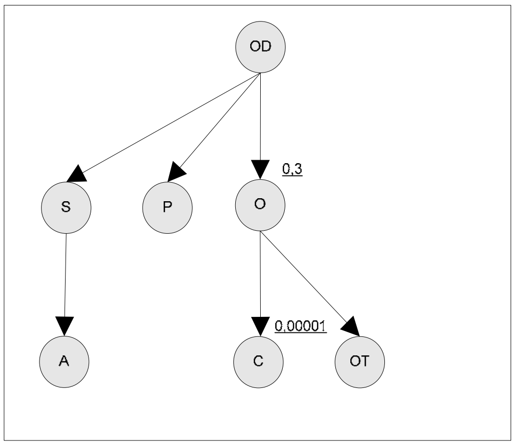

Fig. 6. Filtering index of tables

13. Calculation of join indexes for tables (Fig. 7 / Fig. 8):

(OD=O)/O – **W7/W4** = 12000000/4000000 = **3**

(OD=O)/OD – **W7/W6** = **1**

(O=C)/C – **W5/W2** = 4000000/5000000 = **0,8**

(O=C)/O – **W5/W4** = 4000000/4000000 = **1**

(O=OT)/OT – **W9/W8** = 4000000/4 = **1000000** w przybliżeniu **1M**

(O=OT)/O – **W9/W4** = 4000000/4000000 = **1**

The joins, which concern tables A, P, S and force artificial completion of missing values are omitted in our considerations.

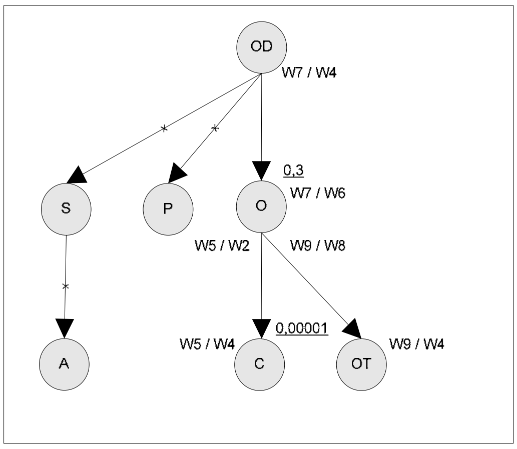

Fig. 7. Join index of tables

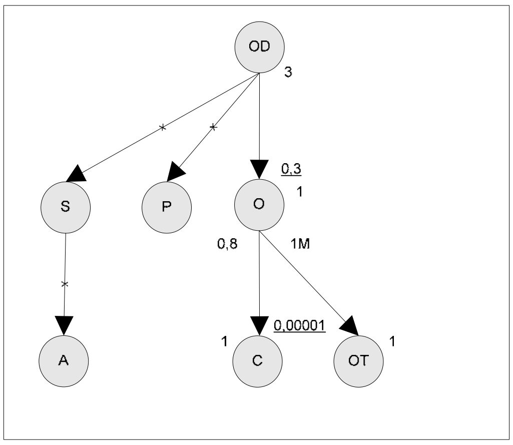

Fig. 8. A list of filtering and join indexes for tables

14. Once the graphical reflection of the query is completed, it is necessary to determine the node having the lowest filtering index value, in accordance with the accepted rules. In this case, it is node C, since its calculated value equals 0.00001.
15. It is not possible to move from C node to the lower level, it is necessary to change the direction of the analysis, that is transition to the node at the higher level, which is related with C. This formed node is referred to as O (first join).
16. It is possible to shift from the O node to the lower level, that is to the OT node (second join). This node is not related with any other nodes at the lower levels, which means that it is necessary to return to the higher level to the O node.
17. In view of the O node, all lower level nodes, which are available for it through links have already been analyzed. Thus, the only possible way is the transition to the upper level to the OD node connected with the O node (third join).
18. In this way, four tables have been joined by three joins: C with O, O with OT, OT with OD.
19. The remaining joins are executed in the context of tables located below the OD node and they are not filtered within the query. Thus, the sequence of joining those tables is random. A similar rule applies to the A table. The joins with a given table may be executed at the random stage of the plan, once the joins with table S are made.
20. Therefore, the optimal sequence of executing the joins [4] is determined by the path leading through the following nodes: C, O, OT, OD, then passing in the random order through S, P and finishing at the A node connected with the S node. Based upon this assumption it is possible to elaborate several equally optimal combinations. One of such combinations is a path of the following sequence: C, O, OT, OD, P, S, A (Fig.9).

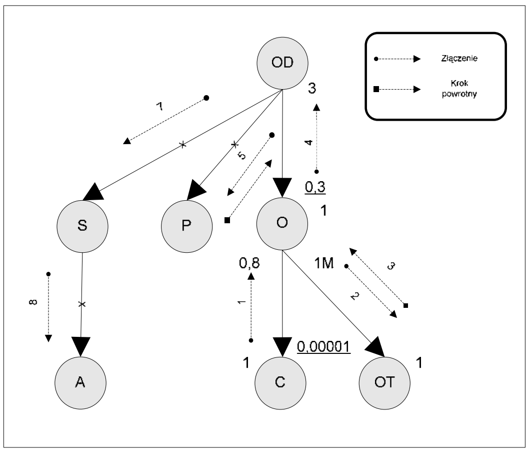

Fig. 9. The path determining the optimal sequence of executing joins

21. For such a procedure it is important to elaborate a plan of efficient execution, where the key aspect is the determined sequence of joins. At this base it is possible to specify the method of application of nested loops, which operate on indexes. In case of an initial table C, the identified index is included within the filtering condition. In turn, the indexes of the remaining tables are the join key

indexes. A detailed plan of the query execution takes the following shape:

- the initial step is the reference to table Customers C using the Last Name index. The query must be modified so that it enables the full access to the index and ensures its full application
- the following step is to make a nested loop against the Orders O table, based upon the index of the Customer_Id foreign key
- the following join of the nested loop is executed with respect to Code_Translations OT and based upon the index of its primary key – Code
- the joining of the nested loops with the Order_Details table is executed upon the base of the Order_Id foreign key index
- the following step is the implementation of the nested loop join against the table Products P. In a given case the index of the Product_Id primary key is used
- during the next stage, an outer join of nested loops is executed, this time with the Shipments table and by using the index of the Shipment_Id primary key
- the last outer join of nested loops applies to the Addresses table and it refers to the index of the index of the Address_Id primary key
- the procedure ends with the sorting of query results.

Each attempt to execute the query, which permits the omission of specified indexes or avoidance of nested loops will not be an optimal execution plan.

7. Benefits of optimization

The improvement of queries is one of the safest methods of modifying applications. Moreover, unlike other methods of optimization, which could generate extra costs (e.g. adding indexes increases the necessary disk space), the method of drawing up query diagrams bears only minor risk of complications that might disturb the operation of a database. Thereby, the benefits of increased efficiency of SQL code optimized by way of the presented method are considerable and the risk of potential complications is little, which makes it an efficient tool that may be utilized in numerous SQL-based database applications.

8. Bibliography

[1] D. Tow, *SQL Tuning*, O’Reilly Media, Inc., United States of America, 2003.

[2] B. Schwartz, P. Zaitsev, V. Tkachenko, J.D. Zawodny, *High Performance MySQL*, O’Reilly Media, Inc., United States of America, June 2008.

[3] W. Dudek, *Bazy danych SQL. Teoria i praktyka*, HELION, 11/2006.

[4] K. Loney, *Oracle Database 11g. Complete Reference*, The McGraw-Hill Companies, Inc, 2009.

[5] J. Price, *Oracle Database 11g SQL*, The McGraw-Hill Companies, Inc, 6/2008.

[6] E. Whalen, M. Schroeter, *Oracle. Optymalizacja wydajności*, HELION, 2003.

[7] R. Vieira, *Beginning SQL Server 2005 Programming*, Wiley Publishing, Inc, Canada, 2006.

Optymalizacja zapytań SQL metodą sporządzania diagramów zapytań

K. WITAN

Niewłaściwie skonstruowane aplikacje bazodanowe nie tylko wymagają poświęcenia nadmiernej ilości czasu na ich obsługę, lecz mają także wpływ na inne aplikacje funkcjonujące na tym samym komputerze, lub w tej samej sieci. Najlepszą metodą pozwalającą na rozwiązanie powyższego problemu, jest przeprowadzenie optymalizacji zapytań. Istnieją dwie podstawowe kwestie na których należy się skoncentrować w trakcie optymalizacji: jak znaleźć i zinterpretować plan wykonania dla zapytania SQL, oraz jak zmodyfikować zapytanie SQL, aby uzyskać określony alternatywny plan wykonania. Jednak najistotniejsze jest jak znaleźć optymalny plan wykonania dla konkretnego zapytania. Bieżący artykuł opisuje metodę, charakteryzującą się niską czasochłonnością, opracowaną w celu wyznaczenia optymalnego planu wykonania – szybko i systematycznie – niezależnie od poziomu złożoności zapytania SQL, lub rodzaju użytej platformy bazodanowej. Wyjaśnia on jak zrozumieć i kontrolować plan wykonania SQL, a także jak opracować diagram zapytania, umożliwiający wybór najlepszego planu wykonania dla zapytania.

Słowa kluczowe: optymalizacja zapytań, język SQL, relacyjne bazy danych.