Relational Database Performance Comparation

Dmitriy S. Gudilin
Bauman Moscow State Technical University
Moscow, *Russian Federation*
dima.gudilin@mail.ru

Aleksei E. Zvonarev
Bauman Moscow State Technical University
Moscow, *Russian Federation*
defygee@yandex.ru

Boris S. Goryachkin
Bauman Moscow State Technical University
Moscow, *Russian Federation*
bsgor@mail.ru

Dmitry A. Lychagin
Bauman Moscow State Technical University
Moscow, *Russian Federation*
lychagin.dmitry@gmail.com

2023 5th International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE) | 979-8-3503-9952-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/REEPE57272.2023.10086872

***Abstract*—This paper discusses the performance of different relational databases using query analysis and optimization under different conditions. The three most popular relational databases are PostgreSQL, MySQL and MS SQL. The basic query execution time for different numbers of operators, columns, and rows was taken as the main criterion. As a result of experiments, it was determined that all databases with small amounts of stored information in the basic configuration operate in accordance with the declared performance. However, with small amounts of data, PostgreSQL is the most preferred option for use, since it is the fastest to process information. It has also been proven that MS SQL has the largest rate increment when increasing the number of columns from all databases under investigation.**

Keywords—databases, PostgreSQL, MySQL, MS SQL, performance, DBMS.

I. Introduction

In today's world, databases are used everywhere. Any site, application or large structure requires a data store from which information will be taken for work. Each user action can be recorded in the event log to further improve internal processes or user experience. At the same time, the data takes up a large amount of space, so the speed of reading them is very important. Moreover, even when leaving for cloud storage and trends towards the transition to non-relational, object-oriented and cluster databases, relational databases still do not lose popularity and are used in the most modern solutions [1].

The most popular relational databases at the moment are three databases: PostgreSQL, MySQL, MS SQL. They are easy to install, do not require special support and provide advanced functionality for building tables for any needs. However, in addition to basic technical characteristics, such as general technical purpose, basic structure, the user does not always have a complete idea for what purposes which database can be used. There are reviews of databases on the Internet, but there is no exact comparison of the operating time of systems anywhere.

That is why it is necessary to study databases in great detail for their more efficient use. The purpose of current work is to study the internal features of databases based on the analysis of the time parameters of query execution under various conditions.

II. Related Work

For each database, one of the most important criteria is performance [2]. In many articles, a comparison of the execution time of queries in relational and NoSQL databases was made [3]. There is also a more detailed study of the execution time of requests for reading NoSQL DBMS using the example of MongoDB [4][5]. This article discusses the comparison of relational DBMS.

III. Description and Calculation of Parameters Affecting Database Performance

In this paragraph a closer look at the structure and parameters of query execution in all three of the above databases is taken.

In PostgreSQL, executing a query consists of the following steps:

1) The parser creates a tree structure that represents the SQL query and can be interpreted by subsequent subsystems as row text.

2) After the parser creates the parsing tree, the analyzer performs semantic analysis to create a query tree, with the Query [6] structure defined in parsenodes.h serving as the root. This structure includes info data for the query, such as the command type (INSERT, SELECT, etc.), and each branch contains information for a specific part of the query, forming a list or tree [7].

3) Scribe - applies the rule system and, if needed, converts the query tree in accordance with the rules, that are stored in the pg_rules system directory.

4) The Scheduler receives the modified query tree from the Scribe module and creates a plan tree (request tree) that can be processed efficiently by the execution module.

PostgreSQL's Scheduler [8] uses cost-based default optimization, which means that it doesn't apply optimization based on rules or hints. Instead, it estimates costs for different operations and uses these costs to determine the most efficient plan for executing the query.

Costs are dimensionless and relative, and are calculated by functions located in the costsize.c file. For example, the cost of scanning data sequentially or using an index is estimated by the cost_seqscan() and cost_index() functions, respectively.

PostgreSQL uses 3 types of cost: initial, execution, and total. The initial cost is incurred before the first tuple is received, such as the cost of reading index pages to access 1 tuple in the required table. The execution cost is the cost of retrieving all tuples, and the total cost is the sum of the start-up cost and the execution cost.

For a standard SELECT query, the cost of sequential scanning is calculated by the cost_seqscan() function [9].

In sequential scanning, the startup cost is 0 and the execution cost is given by the following expression:

$$T = ((t_{tup} + t_{op}N_{op})N_{tup} + S_pN_p)k_{comp}\ (S_s + t_{st}\ N_{st}) \quad (1)$$

where T is the total execution time of the request [10],

t_{tup} is the cost of scanning the tuple, by default 0.01. This cost is taken from the costize.c setup file;

t_{op} - the cost of one operator, by default 0.0025. This cost is taken from the costize.c setup file;

N_{op} - the number of operators, that is, the number of columns with conditions;

N_{tup} - number of tuples;

S_p - number of pages;

N_p - the cost of scanning the page, by default 1;

t_{st} - empirical cost factor of one column, for this configuration PostgreSQL 0.0186;

S_s - initial cost of searching by columns, for this PostgreSQL configuration 0.9814;

N_{st} - number of columns in the table;

kcomp - computer performance factor, for the device on which the experiment was carried out - 0.0003. The factor can vary depending on system performance [11]. For example, a computer with a Pentium processor has a coefficient order of 0.01, although for modern server solutions, this coefficient can reach 10^{-6}.

The number of the pages on which the database is located cannot be calculated manually, but data can be obtained using a PostgreSQL query [12]. The query below displays information about the number of pages on which a specific table is located [13]:

SELECT relpages, reltuples FROM pg_class WHERE relname = 'data'

Any operation in MySQL goes through the following steps:

- Start;
- Checking permissions;
- Opening tables;
- Initialization;
- Optimization;
- Preparation;
- Execution of the request;
- End of request execution;
- End of the process;
- Closing tables;
- Cleaning.

SHOW PROFILE query is used to evaluate these parameters for a particular query.

For the operations considered in this experiment, general execution time of all stages Sstart, except "Query execution," will remain unchanged and will be 32 conditional units.

The total query execution time can be found using expression:

$$T = (S_{start}+N_{op}t_{op}+t_{tup}N_{tup})k_{comp}\ (S_s+t_{st}N_{st}) \quad (2)$$

where T is the total execution time of the query,

t_{tup} - the cost of scanning tuple, for the selected MySQL configuration 0.002;

t_{op} - the cost of one operator for the selected MySQL 50 configuration;

N_{op} - the number of operators, that is, the number of columns with conditions;

N_{tup} - number of tuples;

S_{start} - initial cost of execution;

t_{st} is the empirical cost factor of one column for the selected configuration of MySQL 0.0241;

S_s - initial cost of searching by columns, for the selected MySQL configuration 0.9759;

N_{st} - number of columns in the table;

k_{comp} - computer performance factor, for the device on which the experiment was carried out - 0.0003. The coefficient can vary from 10-6 to 10-2.

Since **Microsoft SQL** is a closed-source system [14], theoretical data on the estimate of the query execution speed is not contained in the open information space. However, it is possible to obtain program execution plan data during the execution process.

The SET SHOWPLAN_ALL command [15] returns data as a rowset that forms a hierarchical tree representing the sequence of steps that the SQL Server Query Handler performs during the execution of each statement. Each statement reflected in the output has one line with the statement text followed by several lines with detailed descriptions of the XStep [16]. Table Ishows the main gained hyperparameters [17]:

TABLE I. DEPENDENCE OF HYPERPARAMETER VALUES ON THE NUMBER OF MS SQL ROWS

Number of rows	***100***	***1000***	***10000***	***100000***	***377657***
EstimateIO	0	0	0	0	13.808
EstimateCPU	0	0	0.001	0.010	0.411
AvgRowSize	1336	1336	1336	1336	1336
EstimatedTotalSubtreeCost	0.007	0.040	0.380	3.779	14.223

IV. EXPERIMENTAL DATABASE PERFORMANCE STUDIES

Due to the fact that the query execution speed vary depending on the performance of the information processing devices, it is necessary to clarify the initial data at which the experiments will be carried out.

Technical devices:

Processor: 12 core processor with a clock speed of 4.2 GHz;

- RAM: DDR4 16 GB at 3000 MHz;

- Disk: Empty SSD 500 GB drive at 350 Mbps;
- Software: Windows 11 operating system;
- Proprietary software from database distributors (PgAdmin for PostgreSQL, MySQL workbench for MySQL, SQL server for Microsoft SQL) in the basic configuration.

As the basis of the experiment, a file with the datacet of cycling trips was taken, containing 377,657 lines.

The practical value is obtained as a result of the arithmetic mean rate of query execution based on three measurements. Practical time provides the DBMS by displaying information in the user console. For comparison, take the following parameters: the number of rows, the number of operators and the number of columns. Consider the tables of the dependence of the query execution time on the database and the selected parameters obtained experimentally [18].

Table II shows the comparison of the query execution time depending on the number of rows [19]:

TABLE II. RUNTIME DEPENDENCIES ON THE NUMBER OF ROWS

Number of rows	***100***	***1000***	***10000***	***100000***	***377657***
PostgreSQL, ms	0.01	0.2	4	25	120
MySQL, ms	18	23	38	124	234
MS SQL, ms	6	12	67	130	1034

Table III shows a comparison of query execution times depending on the number of operators:

TABLE III. RUNTIME DEPENDENCIES ON THE NUMBER OF OPERATORS

Number of operators	***1***	***2***	***3***	***4***	***5***
PostgreSQL, ms	100	135	151	160	189
MySQL, ms	251	266	281	296	311
MS SQL, ms	1034	1070	1104	1123	1145

Table IV shows the runtime comparisons from the number of columns in different DBMS:

TABLE IV. RUNTIME DEPENDENCIES ON THE NUMBER OF COLUMNS

Number of columns	***1***	***5***	***10***	***15***	***20***
PostgreSQL, ms	100	135	151	160	189
MySQL, ms	251	266	281	296	311
MS SQL, ms	1034	1070	1104	1123	1145

V. EVALUATION OF THE OBTAINED RESULTS

The data obtained experimentally are correct and make it possible to estimate the calculations' error of the query execution time in the databases under consideration and thereby confirm or refute the expressions (1) and (3), as well as determine the effect of the selected parameters on the performance (efficiency) of the databases. Tables V, VI and VII show the execution times of the analytically calculated queries.

TABLE V. RUNTIME DEPENDENCIES ON THE NUMBER OF ROWS

Number of rows	***100***	***1000***	***10000***	***100000***	***377657***
PostgreSQL, ms	0.03	0.3	3	30	114
MySQL, ms	24	25	30	84	251

TABLE VI. RUNTIME DEPENDENCIES ON THE NUMBER OF OPERATORS

Number of operators	***1***	***2***	***3***	***4***	***5***
PostgreSQL, ms	114	130	146	163	178
MySQL, ms	234	260	289	324	355

TABLE VII. RUNTIME DEPENDENCIES ON THE NUMBER OF COLUMNS

Number of columns	***1***	***5***	***10***	***15***	***20***
PostgreSQL, ms	120	123.7	132.5	144.2	148
MySQL, ms	251	275.2	305.4	335.7	365.9

Comparative analysis of the query execution time depending on the number of lines in different databases is presented in Fig. 1, Fig. 2 and Fig. 3 as a histogram.

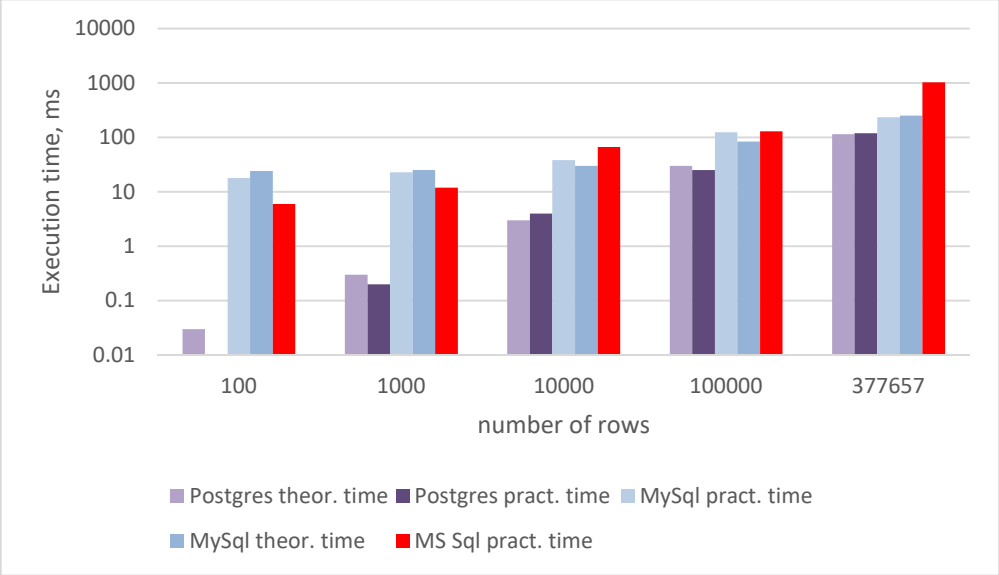

Fig. 1. Dependencies of query execution time on the number of rows.

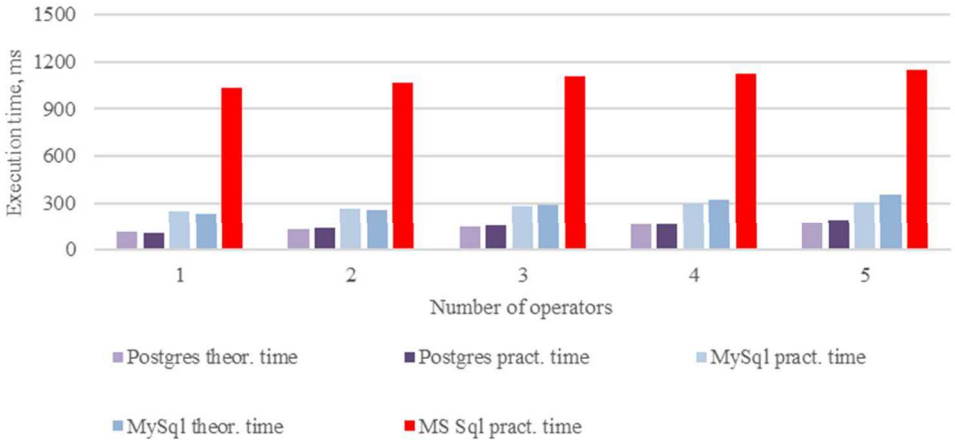

Fig. 2. Dependencies of query execution time on the number of operators.

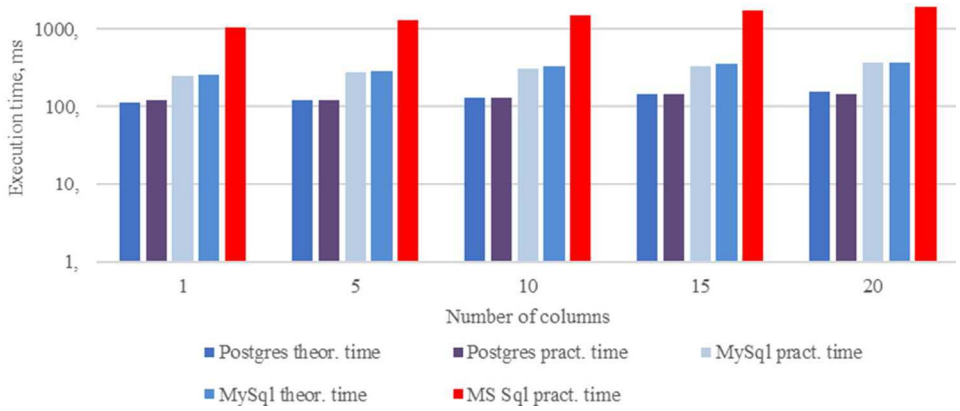

Fig. 3. Dependencies of query execution time on the number of columns.

This figure shows that when executing queries, regardless of the number of rows, the theoretical results are close to practical. Thus, the formulas can be considered valid (adequate) and used in other data, while the general tendency of increasing the cost of queries depending on the number of rows will be repeated. It is also clear from the histogram that the use of PostgreSQL DBMS will be the most optimal for analyzing this dataset, since PostgreSQL gives the fastest query results even with a large amount of information.

Fig. 2 graphically shows the results obtained from the analysis of the time of execution of requests from the number of operators.

According to this graph, it can be seen that with a slight increase in the number of operators, PostgreSQL remains the fastest when executing queries on a dataset > 300,000 rows.

Fig. 3 is a graphical illustration of the analysis of query times from the number of columns [20].

Fig. 3 shows that all databases except MS SQL have a small increment in execution rate when the number of columns increases. This means that all databases except MS SQL have a good information output subsystem with a small number of columns relative to the number of rows. However, MS SQL is also suitable for storing data with a small number of columns.

VI. Conclusion

In this work, the performance of three popular databases was analyzed: PostgreSQL, MySQL, MS SQL. Three key parameters were compared: number of rows, number of operators, and number of columns. As a result of the comparison, it was revealed that for small amounts of data (up to 100,000 rows, up to 30 columns and upto 30 operators) all databases work with the declared performance. The total query execution time under any conditions does not exceed 1.2 seconds, which corresponds to the perception time of one element, so this time is not criticalwhen analyzing data.

References

[1] A. E. Zvonarev, M. V. Vinogradova, D. S. Gudilin, and D. A. Lychagin, “Process analytics usage in the banking sector,” Artificial Intelligence in Automated Data Management and Processing Systems, pp.47-52, 2022.

[2] D. A. Lychagin, M. V. Vinogradova, D. S. Gudilin, and A. E. Zvonarev, “Organization of data storage in the conditions of import substitution,” Artificial Intelligence in Automated Data Management and Processing Systems, pp.364-370, 2022.

[3] E. A. Eliseeva, B. S. Goryachkin, M. V. Vinogradova, and M. V. Chernenkiy, “Estimating the execution time of search queries in NoSQL and object-relational databases,” International Scientific Journal "Dynamics of Complex Systems - XXI Century," Radiotekhnika Publishing House - Moscow, no. 2, pp. 44-51, 2022.

[4] B. S. Goryachkin and D. A. Lychagin, “Performance analysis of MongoDB DBMS when executing read requests,” Information-analytical and Intellectual Systems for Production and Social Sphere, pp. 13-19, 2022.

[5] E. A. Eliseeva, B. S. Goryachkin, and M. V. Vinogradova, “Investigation of DBMS performance when working with cluster databases based on ergonomic analysis,” Scientific and Educational

Journal for Students and Teachers, STUDNET: Publishing house LLC "Electronic Science" - Moscow, vol. 5, no. 4. pp. 2889-2910, 2022.

[6] E. O. Kiktenko, A. S. Zelenetsky, and A. K. Fedorov, “Practical quantum multiparty signatures using quantum-key-distribution networks,” Amer Physical Soc, vol. 105, 2022.

[7] Y.A. Grigoriev, “Estimation of execution time of SQL-queries to databases,” Mechanical Engineering and Computer Technologies, no. 01, 2012.

[8] M. Stonebraker, L. A. Rowe and M. Hirohama, "The implementation of POSTGRES," in IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 1, pp. 125-142, March 1990.

[9] I. S. Vershinin and A. R. Mustafina, "Performance Analysis of PostgreSQL, MySQL, Microsoft SQL Server Systems Based on TPC-H Tests," 2021 International Russian Automation Conference (RusAutoCon), 2021, pp. 683-687

[10] L. G. Wiseso, M. Imrona and A. Alamsyah, "Performance Analysis of Neo4j, MongoDB, and PostgreSQL on 2019 National Election Big Data Management Database," 2020 6th International Conference on Science in Information Technology (ICSITech), 2020, pp. 91-96

[11] M. -G. Jung, S. -A. Youn, J. Bae and Y. -L. Choi, "A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment," 2015 8th International Conference on Database Theory and Application (DTA), 2015, pp. 14-17

[12] I. I. Butenko, I. N. Telnova, and V. V. Garazha, “Prospective scientific research trend identification methods (based on the analysis of gas fuel-related publications),” Automatic Documentation and Mathematical Linguistics, vol. 56, no. 1, pp. 11–25, 2022, doi: 10.3103/S0005105522010034.

[13] M. Sharma, V. D. Sharma and M. M. Bundele, "Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j," 2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE), 2018, pp. 1-5

[14] C. Mariani, M. Mercês, M. Holanda, E. D. O. Ribeiro and M. C. Victorino, "Hive and MySQL - Query performance comparison for a multidimensional model," 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), 2022, pp. 1-6

[15] A. Wickramasekara, M. P. P. Liyanage and U. Kumarasinghe, "A comparative study between the capabilities of MySQL and ClickHouse in low-performance Linux environment," 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer), 2020, pp. 276-277

[16] K. I. Satoto, R. R. Isnanto, R. Kridalukmana and K. T. Martono, "Optimizing MySQL database system on information systems research, publications and community service," 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), 2016, pp. 1-5

[17] C. A. Galindo-Legaria et al., "Optimizing Star Join Queries for Data Warehousing in Microsoft SQL Server," 2008 IEEE 24th International Conference on Data Engineering, 2008, pp. 1190-1199

[18] Q. Peng, C. Jian-hui and C. Qing, "Study of XML & SQL Server and Application in Distributed Integration," 2007 8th International Conference on Electronic Measurement and Instruments, 2007, pp. 4-369-4-372

[19] A. R. Masalimova, M. A. Khvatova, L. S. Chikileva, E. P. Zvyagintseva, V. V. Stepanova, and M. V. Melnik, “Distance learning in higher education during COVID-19,” Frontiers in Education, vol. 7, 2022, doi: 10.3389/feduc.2022.822958.

[20] Andrey Briko, Vladislava Kapravchuk, Alexander Kobelev, Ahmad Hammoud, Steffen Leonhardt, Chuong Ngo, Yury Gulyaev, and Sergey Shchukin, “A way of bionic control based on EI, EMG, and FMG signals,” Sensors, vol. 22, no. 1, p. 152, 2021, doi: 10.3390/s22010152.