

Searching for an Optimal Data Platform for Relevant Information Search in Inorganic Chemistry and Materials Science

Victor A. Dudarev^{1, 2}[0000-0001-7243-9096] and Sergey S. Babikov¹[0000-0002-3856-2231]

¹ National Research University Higher School of Economics, Moscow, 109028, Russia

² A.A. Baikov Institute of Metallurgy and Materials Science of RAS (IMET RAS), Moscow, 119334, Russia

vdudarev@hse.ru

Abstract. Choosing the most suitable database management system is one of the most critical challenges in developing any information system operating on big data. When selecting, as a rule, the overall system speed is considered the main criterion regarding certain data structures due to the subject area specifics. In the current article, using the example of searching for relevant information on inorganic compounds, an attempt is made to analyze the possibility of using relational and graph database management systems (DBMSs) to build a data storage subsystem. Graph-based database implementations, powered by SQL Graph and Neo4j, are considered and compared with a relational version based on SQL Server. Typical query execution speed comparative analysis is carried out when searching for relevant information in the field of inorganic chemistry and materials science. It's shown that, due to the graph nature of relevance definition, graph DBMS outperforms relational.

Keywords: DB, DBMS, Neo4j, SQL Server, SQL Graph, inorganic chemistry, relevant information search.

1 Introduction

This work is devoted to studying ways to solve the relevant chemical objects search problem in the metabase – a database (after this referred to as DB) containing information on the contents of integrated information systems in the field of inorganic chemistry and materials science. On three different DBMSs, a data structure is developed and filled with data to search for relevant chemical objects; and the typical search queries execution speed is analyzed. The study is essential since the query execution time for relevant information search in the current system version is unsatisfactory (the average request time to the Metabase, serving the imet-db.ru, is up to 1.5-2 seconds in the worst cases). Moreover, although currently, it's possible to find chemical objects with a relevant chemical system from integrated sources, it's necessary to improve system capabilities so to extend the relevant chemical entities search to the chemicals substances and modifications level.

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Relational DBMSs are a popular solution for creating storage subsystems and, in fact, are classic. With their help, a massive number of information systems have been built, the strengths of which include the ability to work with a reasonably large amount of data, the rigor of data representation in the form of a relational relationship, ensuring referential integrity, and support for transaction processing. Thus, when properly designed, many domain models can easily fit into a set of interrelated relationships with a clear, easily perceived structure. However, when trying to create M:N (many-to-many) type relationships between stored entities, one must introduce additional tables to define such relationships. This approach, in its turn, clutters the data schema additionally; moreover, it entails the need to use additional relational join to build up the desired query result. This, in turn, not only complicates the writing of SQL queries but also contributes to a decrease in the efficiency of their execution.

Thus, if the objects stored in the database are associated with the same type of objects by M:N relations, relational systems experience rather significant difficulties in processing such queries. In work [1], it is shown that the relevant information concept in an integrated IS on the properties of inorganic substances is associated with a variety of connections between chemical objects of the same type, which are represented by a graph. Consequently, the search for relevant information is formulated through the search in the graph. Obviously, with this formulation, a relational database containing chemical objects list becomes less effective.

For such search queries, a graph database should be the most suitable medium for storing information and providing quick access to it. In this sense, graph databases have the advantage in relevant information search - the objects and relationships stored in them can natively represent the relevance graph [2], simplifying access to it and, consequently, increasing query efficiency compared to relational databases. Due to the amount of data on chemical objects and their presentation methods, the problem arises to develop optimal, from the DBMS features point of view, data structures for representing information in the subject domain and studying their effectiveness in various situations.

To search for optimal storing and processing data ways, it was decided to implement pilot versions of search subsystems for relevant information based on graph DBMS: Neo4j, as the most popular graph DBMS [3], and SQL Graph [4], as an extension of SQL Server, on which the Metabase is implemented. It is interesting to compare the operation speed of two graph databases based on different platforms with the speed of a relational Metabase running SQL Server [5] (from now on SQL) on the problem of finding relevant chemical objects. This paper discusses SQL, Neo4j, and SQL Graph database structures and evaluates queries using various metrics.

2 Problem Domain Objects and Relations

In current research we discuss the Metabase part only which contains chemical objects described elsewhere within the integrated information systems. For simplicity we do not consider data structures containing information about which information sources and which chemical object properties are described.

2.1 Chemical Objects Hierarchy

According to [1], the Metabase describes chemical objects of three types:

- System – a set of chemical elements representing the qualitative substance composition.
- Compound – is determined not only by chemical elements set but also by each element quantitative content in the composition of substance, solution, or mixture (described by the chemical formula).
- Modification – is determined not only by a chemical formula but also by crystal structure type designation.

Chemical objects hierarchy, illustrating in detail the above definitions, is shown below (see Fig. 1).

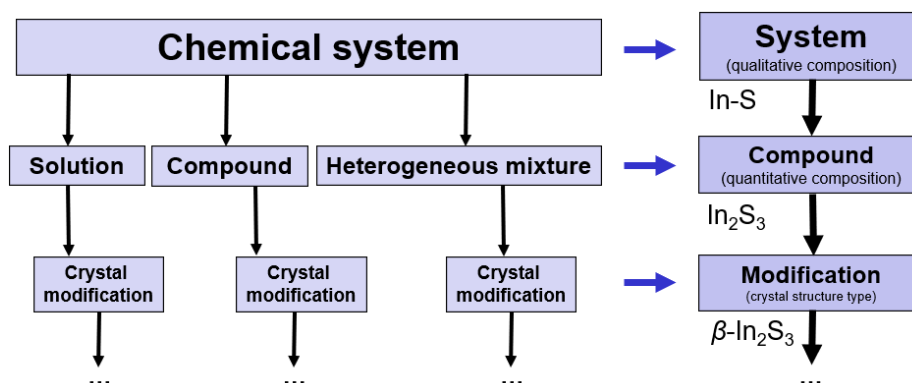


Fig. 1. Chemical objects hierarchy [1].

2.2 Relationships Between Chemical Objects

Given three levels in the chemical objects hierarchy, three relationship types would be required to characterize relationships between entities (see Fig. 2):

- SUPERSYSTEM – it is a relationship between child and a parent chemical systems. A system is designated as a child or relevant if and only if the system contains all the chemical elements contained in the parent system and the number of elements in the system is precisely one element more than in the parent (i.e. parent system's chemical elements are a subset of a child).
- COMPOUND – it is a relationship between a chemical system and a compound belonging to the same system. A compound is associated with this relationship with the system if and only if the set, representing the qualitative substance composition, is equivalent to the chemical elements set of the system.

- **MODIFICATION** – it is a relationship between a substance and its crystal modifications. A crystal modification is a child node for a substance if it has the same qualitative and quantitative composition as the parent substance.

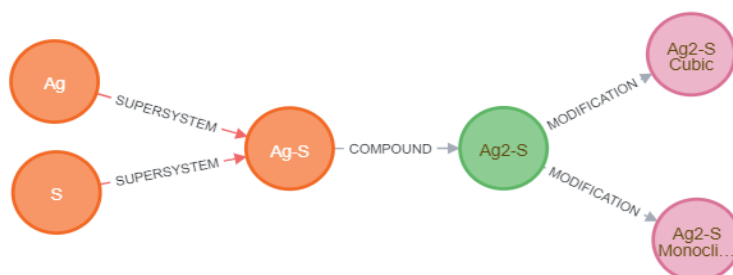


Fig. 2. Relationships between nodes in metabase structure.

According to the graph theory terminology, the connections between chemical objects are called edges. The types of relationships between chemical entities are the names of the edges. Further we will analyze data structures and queries that search for relevant objects only at the system level since the complexity of their implementation is no different from similar queries at substances or modifications level.

3 Relevant Information Search Implementation in SQL Server

3.1 Structure

Since a normalized relational data structure with a list of chemical elements in a particular relationship will make SQL queries extremely ineffective, it was decided to use denormalized relations. So, for example, the system is specified as a string consisting of chemical elements designations, placed lexicographically in ascending order and separated by the "-" symbol. Moreover, a data attribute is added that specifies the chemical system length, i.e. the power of chemical elements set (`ElementNumber` – the number of elements). However, it is worth to note that this value can be calculated from the list of elements. Thus, for the sake of speed, strictly speaking, even the first normal form (1NF) is not satisfied for the DB structure. However, as a performance gain, we get a quick search through the system (thanks to the indices) and the elements it contains. Note that even with a normalized version of the database, a view similar in functionality to the existing table can be easily obtained using materialized views (a view with a clustered index).

Three attributes define a main relation structure, i.e. `Meta_Systems` table (see Fig. 3).

- `Elements` – list of the system chemical elements (primary key, `varchar(32)` type);

- ElemNumber – the elements count in the chemical system (int type);
- IsInHierarchy – an auxiliary column used to incrementally populate the relevant objects list (it is used in the relational version of metabase only; int type).

	Elements	ElemNumber	IsInHierarchy
1	-Ag-Ba-N-	3	0
2	-Ag-Cr-Cu-	3	0
3	-Ag-Cu-Si-	3	0
4	-Ag-O-Pt-	3	0

Fig. 3. Chemical systems table fragment (SQL Server).

Thus, to obtain chemical systems, it is enough to query the Meta_Systems table:

```
SELECT * FROM Meta_Systems;
```

A similar structure is possessed by tables with chemical substances and modifications with the addition of information on the quantitative composition and crystal modification.

3.2 Search for Relevant Chemical Systems

Some scalar and table functions have been added to the database to facilitate search for relevant chemical objects.

Below is a query using the GetSystemsFromRelationalMetabase table function to get all the relevant systems for hydrogen:

```
SELECT [Elements], [SuperElements] as 'Child systems'
FROM dbo.GetSystemsFromRelationalMetabase('-H-');
```

The query result (see Fig. 4) is presented as two column relation indicating the parent and corresponding child systems.

	Elements	Child systems
1	-H-	-Ac-H-
2	-H-	-Ag-H-
3	-H-	-Al-H-
4	-H-	-Am-H-
5	-H-	-Ar-H-

Fig. 4. Search result for relevant chemical objects for hydrogen system "H" in SQL Server.

Consider the GetSystemsFromRelationalMetabase function code:

```

SELECT s.[Elements], s.ElemNumber, superS.[Elements] AS
SuperElements, superS.ElemNumber AS SuperElemNumber
FROM Metabase.dbo.Meta_Systems AS s
CROSS APPLY Metabase.dbo.GetSuperSystem([Elements], El-
emNumber) AS superS WHERE s.[Elements] = @supersystem;

```

First, the function finds the system passed as a parameter in the Meta_Systems table. The GetSuperSystem function is called for the corresponding @supersystem parameter value only, and its execution result is appended to the system record by the CROSS APPLY operator. So for the parent system passed as a @supersystem parameter value one can get a list of child chemical systems .

Consider the code for GetSuperSystem function, which is used in the GetSystemsFromRelationalMetabase function:

```

SELECT [Elements], ElemNumber from dbo.Meta_Systems WHERE
ElemNumber=@ElemNumber+1 AND @ElemNumber=(select
count(value) as cnt from (
select value from dbo.STRING_SPLIT(@Elements, '-') WHERE
value<>'')
INTERSECT
select value from dbo.STRING_SPLIT([Elements], '-') WHERE
value<>'') as tab);

```

The function returns children for the system passed as a parameter. The WHERE clause specifies the number of chemical elements in child systems (it should be one more than the number of chemical elements in the parent system). Further, the subquery inside this function takes the [Elements] column from the main query and, using the INTERSECT operator, removes those systems that do not contain the parent system. Since this subquery is correlated, the [Elements] value correlates with the outer query, leaving only child systems as a result.

3.3 Search for Relevant Chemical Systems on the Current System

The current version of imet-db.ru Web site uses a different query form to search for relevant chemical systems. Since the above search query (section 3.2) takes an unacceptably long time (up to 11 seconds in the worst case), it was decided to create a special cache table of correspondences between parent chemical systems and children. The structure of the table is quite simple (see Fig. 5) and the query for obtaining the relevant systems for the hydrogen ('H') could be the following:

```

DECLARE @supersystem VARCHAR(256) = 'H'
DECLARE @cnt INT = Metabase.dbo.GetElementsCountFrom-
String(@supersystem)
SET @supersystem = '-' + @supersystem + '-'
SELECT @supersystem 'ParentSystem', H.[Elements]
'ChildSystems'

```

```

FROM Metabase.dbo.Meta_SystemsHierarchy AS H
INNER JOIN Metabase.dbo.Meta_Systems AS S
ON H.[Elements]=S.[Elements]
WHERE H.ParentElements=@supersystem AND S.ElementNumber=@cnt+1
UNION
SELECT @supersystem, ParentElements
FROM Metabase.dbo.Meta_SystemsHierarchy as H
INNER JOIN Metabase.dbo.Meta_Systems AS S
ON H.ParentElements=S.[Elements]
WHERE H.[Elements]=@supersystem AND S.ElementNumber=@cnt-1

```

	Elements	ParentElements
1	-Ac-Br-	-Ac-
2	-Ac-Br-O-	-Ac-
3	-Ac-C-H-O-	-Ac-
4	-Ac-Cl-	-Ac-
5	-Ac-Cl-O-	-Ac-

Fig. 5. Parent-children correspondence table structure in SQL Server.

The @supersystem variable represents the parent hydrogen system ('H'), the @cnt variable represents the number of elements in the @supersystem. When assigning a value to @cnt, the GetElementsCountFromString function calculates the number of elements in the system passed as a parameter by splitting the system formula string by the '-' symbol and counting the number of rows received. Further, in the SELECT query, those systems are selected whose parent formula is @supersystem and the number of elements in which is greater by one. Then, using the UNION operator, the child systems found in the same way are joined, but the number of chemical elements is less by one. In the further development of the graph database, the functionality associated with the UNION operator was decided to remove and leave only child systems with a large number of elements of the parent system per one. Using such an implementation of the relevant objects search, the query duration was reduced to appropriate values (hundreds of milliseconds).

This query type is not considered in the current study for comparison of search speed since the database uses pre-prepared data (pre-calculated relevancy lookup table), making the further search query execution time comparison with graph databases incorrect. So we are going to compare the execution speed of query from section 3.2.

4 Relevant Information Search Implementation in SQL Graph

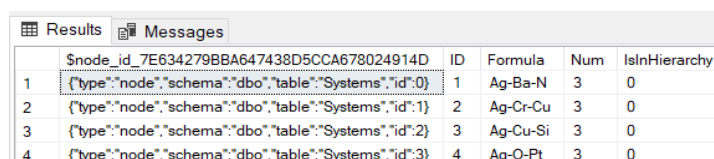
4.1 Structure

The metabase implementation in SQL Graph DBMS consists of three tables with chemical objects of all types and three tables with the corresponding relationships described in section 2.

Chemical Objects

Each object type table contains the information about chemical objects of a particular type and a column with unique object ID in the table. The query to get information about systems and the execution result can be seen on a figure (see Fig. 6):

```
SELECT * FROM Systems;
```



	\$node_id_7E634279BBA647438D5CCA678024914D	ID	Formula	Num	IsInHierarchy
1	{'type':'node','schema':'dbo','table':'Systems','id':0}	1	Ag-Ba-N	3	0
2	{'type':'node','schema':'dbo','table':'Systems','id':1}	2	Ag-Cr-Cu	3	0
3	{'type':'node','schema':'dbo','table':'Systems','id':2}	3	Ag-Cu-Si	3	0
4	{'type':'node','schema':'dbo','table':'Systems','id':3}	4	Ag-O-Pt	3	0

Fig. 6. A part of query result for getting all systems in SQL Graph.

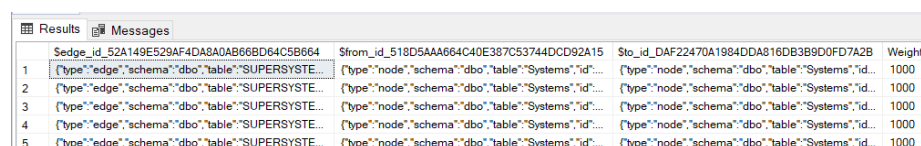
Relationships

Relationship tables have a different structure. They always contain at least three columns:

- \$edge_id that denotes the ID of the edge that connects two chemical entities.
- \$from_id containing the IDs of the first chemical objects.
- \$to_id containing the ID of the second chemical objects.

The table also contains several additional attributes to store developer-defined edge properties. The query code for getting all the edges in the SUPERSYSTEM relationship table and the result of its execution could be seen at figure (see Fig. 7):

```
SELECT * FROM SUPERSYSTEM;
```



	\$edge_id_52A149E529AF4DA8A0AB66BD64C5B664	\$from_id_518D5AA664C40E387C53744DCD92A15	\$to_id_DAF22470A1984DDA816DB3B9D0FD7A2B	Weight
1	{'type':'edge','schema':'dbo','table':'SUPERSYSTEM','id':0}	{'type':'node','schema':'dbo','table':'Systems','id':0}	{'type':'node','schema':'dbo','table':'Systems','id':0}	1000
2	{'type':'edge','schema':'dbo','table':'SUPERSYSTEM','id':1}	{'type':'node','schema':'dbo','table':'Systems','id':1}	{'type':'node','schema':'dbo','table':'Systems','id':1}	1000
3	{'type':'edge','schema':'dbo','table':'SUPERSYSTEM','id':2}	{'type':'node','schema':'dbo','table':'Systems','id':2}	{'type':'node','schema':'dbo','table':'Systems','id':2}	1000
4	{'type':'edge','schema':'dbo','table':'SUPERSYSTEM','id':3}	{'type':'node','schema':'dbo','table':'Systems','id':3}	{'type':'node','schema':'dbo','table':'Systems','id':3}	1000
5	{'type':'edge','schema':'dbo','table':'SUPERSYSTEM','id':4}	{'type':'node','schema':'dbo','table':'Systems','id':4}	{'type':'node','schema':'dbo','table':'Systems','id':4}	1000

Fig. 7. Query result for taking all SUPERSYSTEM edges (SQL Graph).

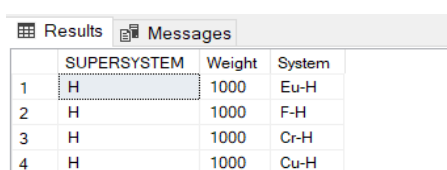
In addition to the required columns, the Weight column has been added, which is the only optional property for the edge, which at the current development stage contains a

constant. But in the future it will store a measure reflecting the strength of the bond between chemical objects (edge weight) [1].

4.2 Search for Relevant Chemical Systems

So let's consider the query code for obtaining relevant systems for the hydrogen system ('H') and the corresponding execution result (see Fig. 8):

```
SELECT * FROM GetSystems('H');
```



	SUPERSYSTEM	Weight	System
1	H	1000	Eu-H
2	H	1000	F-H
3	H	1000	Cr-H
4	H	1000	Cu-H

Fig. 8. A part of the search result for related chemical system objects for hydrogen system "H" in SQL Graph.

Call to the `GetSystems` function returns a table with three columns: the first column is the parent system ("H" in our example), the second column – the Weight (constant value in the current version), and the third column – the child systems. The `GetSystems` function code is the following (using `MATCH` predicate for SQL Graph):

```
SELECT s1.Formula 'SUPERSYSTEM', e.Weight , s2.Formula  
'System'  
FROM Systems AS s1, SUPERSYSTEM AS e, Systems AS s2  
WHERE MATCH(s1-(e)->s2)  
AND s1.Formula = @supersystem
```

The function is a simple `SELECT` query that takes a parent system as a `@supersystem` parameter. The `MATCH` predicate for the SQL Graph is used in `WHERE` clause to find the relevant chemical objects for the parent system with `SUPERSYSTEM` relationships.

5 Revevant Information Search Implementation in Neo4j

5.1 Structure

The conceptual metabase structure in Neo4j is quite similar to that in SQL Graph. But since the data on edges in Neo4j, together with the chemical objects data, can be stored at the individual records level, the search should be performed even more efficiently. The developer does not need to create tables for graph structure as in SQL Graph. It is enough to define all types of edges and chemical objects. To manipulate data and run queries in Neo4j DBMS, Cypher language has been developed [6].

The graph's vertices represent chemical objects in Neo4j DB. The edges reflect the relationship between the vertices. The stored vertex and edge types are described in section 2.

Chemical Objects

All vertices in Neo4j are depicted by circles, the circle color changes depending on the vertex type. Systems and substances have four properties:

- `Identity` – vertex ID;
- `Labels` – vertex type;
- `Formula` – vertex formula;
- `Num` – the number of chemical elements in the vertex formula.

A query in Cypher [7] is shown below. It returns (see Fig. 9) a vertex for the Ba-Ga-Si chemical system and illustrates the above mentioned properties:

```
MATCH (s:System) RETURN s LIMIT 1;
```



Fig. 9. Cypher query result to get the properties for Ba-Ga-Si chemical system in Neo4j.

Substances contain only the first three properties from the systems and do not contain the `Num` property. Modifications have the same properties as systems, but the `Num` property is replaced with the `Modification` property (see Fig. 10). The query code for receiving modification data in Cypher could look like:

```
MATCH (m:Modification) RETURN m LIMIT 1;
```



```
{
  "identity": 172591,
  "labels": [
    "Modification"
  ],
  "properties": {
    "formula": "H6-I6-O20-Pt",
    "modification": "Hexagonal"
  }
}
```

Fig. 10. Query result to get the H₆I₆O₂₀Pt hexagonal modification properties in Neo4j.

Relationships

An edge in Neo4j has five properties:

- Identity – edge ID;
- Start – ID of the first chemical object;
- End – ID of the second chemical object;
- Type – edge type;
- Weight – edge weight.

The edges differ from each other only in the Type and ID properties. A query example to illustrate (see Fig. 11) the properties of an edge in Cypher could look like:

```
MATCH ()-[e:SUPERSYSTEM]->() RETURN e LIMIT 1
```



```
{
  "identity": 973456,
  "start": 102902,
  "end": 124936,
  "type": "SUPERSYSTEM",
  "properties": {
    "weight": 1000
  }
}
```

Fig. 11. Query result to get the edge properties in Neo4j.

5.2 Search for Relevant Chemical Systems

In Cypher query code to search adjacent systems for the hydrogen system ('H') could be formulated in the following way:

```
MATCH (s1:System {formula: 'H'})-[]->(s2:System) RETURN
s1.formula as supersystem, collect(s2.formula) as system
LIMIT 1
```

In this query, we should specify the parent system *s1* formula and use the MATCH operator to find the *s2* children. The query returns a table (see Fig. 12), in the first column of which is the parent system name ('H'), and in the second – an array of all child chemical systems.

Fig. 12. Query result to search the relevant chemical system objects for 'H' system in Neo4j.

6 Query Performance Comparison

6.1 Query Type Definition

To compare queries execution speed in different DBMS correctly, it is necessary to use the same test data, i.e. database contents, and formalize the data structure they should return. So the query answer structure should be identical for all DBMS.

The query should search for child systems and return a list/table with two attributes and one row (not counting the headers). The first attribute should designate the parent chemical system, the second attribute – the number of child systems found that are considered to be relevant for the current system. The Table 1 shows the returned structure and data using hydrogen ('H') as an example of parent system.

Table 1. Query result for finding the relevant chemical objects for the hydrogen system (the same for all databases).

SuperSystem	Child systems count
H	85

Considering that the query performance is highly dependent on the analyzed data volume and distribution, several measurements were carried out on data sets corresponding to chemical systems with different numbers of elements: 1, 2, 3, and 4. For each of the four groups, 10 chemical systems were determined (the same for all databases). Further, for each set, measurements were carried out in every database implementations, after that the results were averaged over each group. Thus, queries were tested on 40 chemical systems (10 for each group) for every database implementation.

6.2 Measuring Time

To measure the query execution time in the SQL DBMS and SQL Graph, SQL Profiler was used – a tool for monitoring query execution statistics. To calculate the query execution time in the Neo4j DBMS, the query execution timer was built into the client software was used due to the absence of a regular query profiler in Neo4j.

The results of query execution time measurement (on a 4-core computational node) are shown at Fig. 13.

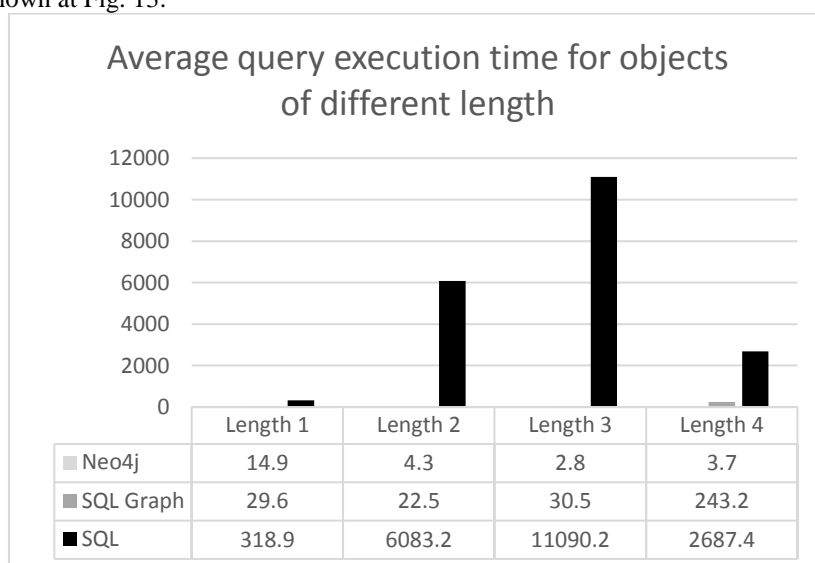


Fig. 13. DBMS comparison chart by average query execution time in inorganic chemistry relevant information search.

Here you can see a sharp decrease in the queries execution time in SQL for a length equal to 4. This is due to the fact that the number of chemical systems containing 5 different elements, for which systems of length 4 are parents, is much less than the number of systems containing 4 different elements, which are children for systems of length 3. Keeping in mind the fact that no optimization was made to relational SQL database to perform correct comparison with other DBMS types to answer the query, direct table scan is used in execution plan as shown by SQL Server Management Studio. Thus it's quite understandable that table scan for 3810 rows is faster than the table scan for 23929 rows (linear dependency). A graph illustrating the number of objects with their corresponding length is shown at Fig. 14. So our attempt to explain relatively small execution time for chemical systems with length of 4 is quite fair although we should be careful with our guesses since we do not have full information about the internal query optimization mechanisms in SQL Server (SQL Server is not an open-source product).

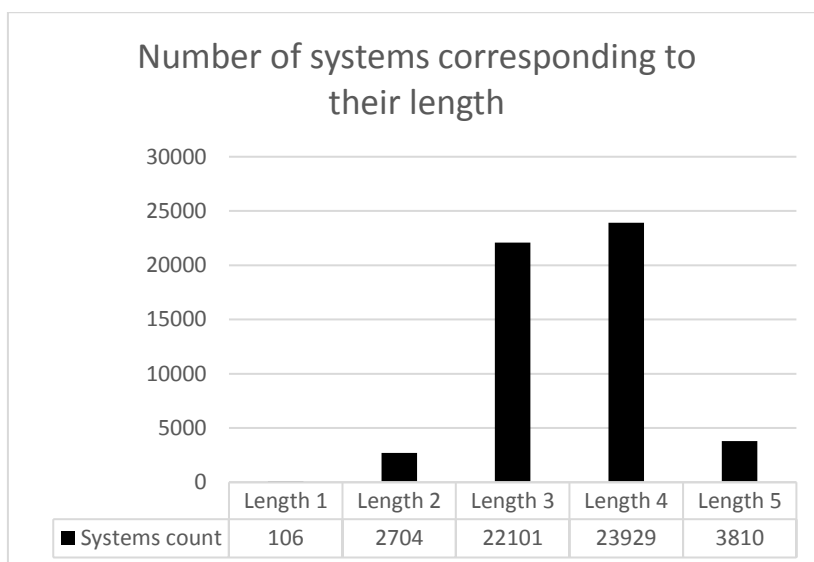


Fig. 14. Systems comparison by number of objects of their corresponding length

7 Conclusion

Given the graph nature of relevance definition in the field of inorganic chemistry and materials science, the leadership of graph versions of the database seems to us quite natural and beyond doubt. However, the results obtained require some comments, given the relatively significant differences in the query execution time.

The relational models inefficiency for representing inherently graph information about relevant chemical systems inevitably puts SQL Server-based implementation at a losing position. However, the scale of these differences is impressive: the relational implementation indicators are dozens of times worse than those in graph databases in the most uncomplicated cases (with one element in the parent set and, therefore, two elements in the children) and about three orders of magnitude worse than the best indicators of graph databases on objects with four elements in systems.

The lag of SQL Graph from Neo4j is quite noticeable, especially when the number of elements in chemical system increases, the gap reaches significant values. At the same time, it should be noted that the graph extension for SQL Server, in general, copes well with its tasks. Considering that this extension appeared only in 2017 and is now being developed by Microsoft, there is no doubt about increasing the functionality and processing speed, and deep integration with SQL Server, including transactional level, makes the data subsystem under the control of this DBMS quite flexible.

At the moment Neo4j is the best in terms of speed and functionality from all the DBMS when working with graph databases. Considering that Neo4j was being developed since 2003 [8], the technology can be regarded as mature enough for industrial

deployment. Additionally, it is worth noting, that the specialized query language Cypher used for Neo4j and its data format also contributes to the query execution speed and an opportunity to use additional optimization techniques.

Complete information and measurement results obtained in the current study are available in appendix [9]. The authors are very grateful to N.N. Kiselyova who inspired the current research and made significant contribution to the relevant search idea regarding to inorganic chemistry. This work was supported in part by the Russian Foundation for Basic Research, project no. 18-07-00080. The study was carried out as part of the state assignment (project no. № 075-00328-21-00).

References

1. Dudarev V.A., Kiselyova N.N., Temkin I.O. On Information Search Measures and Metrics Within Integration of Information Systems on Inorganic Substances Properties. In: Elizarov A., Novikov B., Stupnikov S. (eds) Data Analytics and Management in Data Intensive Domains. DAMDID/RCDL 2019. Communications in Computer and Information Science, 2020, vol. 1223, p. 47-58. Springer, Cham. https://doi.org/10.1007/978-3-030-51913-1_4
2. Robinson I., Webber J., Eifrem E. Graph Databases - Second Edition. O'Reilly, 2015. ISBN 978-1-491-93089-2
3. DB-Engines Ranking of Graph DBMS, <https://db-engines.com/en/ranking/graph+dbms>, last accessed 2021/04/25.
4. Graph processing with SQL Server and Azure SQL Database, <https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview?view=sql-server-ver15>, last accessed 2021/04/25.
5. Strate J. Expert Performance Indexing in SQL Server 2019: Toward Faster Results and Lower Maintenance Paperback. Apress, 2019. ISBN 978-1484254639.
6. Neo4j Desktop, <https://neo4j.com/docs/operations-manual/current/installation/neo4j-desktop/>, last accessed 2021/04/25.
7. Baton J., Bruggen R Learning Neo4j 3.x - Second Edition. Packt Publishing Ltd, 2017. ISBN 9781786466143.
8. Sonal Raj. Neo4j High Performance. Packt Publishing Ltd, 2015. ISBN 978-1-78355-516-1.
9. Supplementary material, <https://drive.google.com/drive/folders/1PZiM-KPjhInfldudrrO9pzNsScuGknjo?usp=sharing>, last accessed 2021/04/25.