Wojciech MUELLER, Przemysław IDZIASZEK, Sebastian KUJAWA, Mateusz ŁUKOMSKI, Przemysław NOWAK
Poznań University of Life Sciences, Institute of Biosystems Engineering, Poznań, Poland
e-mail: muellerw@up.poznan.pl

Received: 2018-04-05 ; Accepted: 2018-11-22

MAPPING OF RELATIONAL STRUCTURES IN GRAPH DATABASE NEO4J

Summary

Extension of functionality of most applications including the ones supporting agriculture, as a general rule requires an in-depth knowledge of relational structures creating databases, which can be sometimes difficult to achieve. It can result from the lack of complete technical documentation as well as relatively huge complexity of relational structures. The given publication is a continuation of the author's actions, aimed at creating a moderately universal application allowing to reproduce the existing relational structures created with the use of different relational database management systems (RDBMS), namely SQL Server, MySQL or Oracle into graph form on the level of Neo4j graph database. This form makes it possible to thoroughly recognize complex relational structures with the use of queries prepared in Cypher language in native client, which is made available from the level of the created application. During the construction process of the presented tool, technologies such as ADO.NET, graph database Neo4j together with available programming interface as well proper tables containing metadata were utilized.
***Key words**: mapping, relational structure, graph structure, database, neo4j*

MAPOWANIE STRUKTUR RELACYJNYCH W BAZIE GRAFOWEJ NEO4J

Streszczenie

Rozbudowa funkcjonalności większości aplikacji, w tym również wspomagających rolnictwo z reguły wymaga pełnej znajomości struktur relacyjnych tworzących bazy danych, co czasami może być trudne do osiągnięcia. Powodem może być brak pełnej dokumentacji technicznej oraz względnie duża złożoność struktur relacyjnych. Prezentowana publikacja, to kontynuacja działań autorów, zmierzająca do wytworzenia w miarę uniwersalnej aplikacji, pozwalającej na odwzorowanie istniejących struktur relacyjnych, powstałych przy wykorzystaniu różnych systemów bazodanowych SQL Server, MySQL oraz Oracle, do postaci grafowej na poziomie Neo4j. Ta postać umożliwia wygodne, dogłębne rozpoznawanie złożonej struktury relacyjnej za pomocą pytań konstruowanych w języku Cypher w natywnym programie klienckim udostępnianym z poziomu prezentowanej aplikacji. W procesie budowy prezentowanego narzędzia wykorzystano technologie ADO.NET, bazę grafową Neo4j wraz z dostępnym interfejsem programistycznym oraz odpowiednie tabele zawierające metadane.
***Słowa kluczowe**: mapowanie, struktura relacyjna, struktura grafowa, baza danych, neo4j*

1. Introduction and available technologies

Extension of functionality of applications supporting broadly defined agriculture, independent of the way of realization, as a general rule involves recognition and modification of relational structure of database, with which the given application works in tandem. The recognition of those structures without possessing documentations and with a large number of defined tables and relations between them, is often not an easy case. The existing tool, dedicated to specific systems of databases do make it possible to search relations between tables in a quick way. Making an endeavor to solve this problem was previously a subject matter of authors inquiries, which was expressed in publication [7]. Mentioned publication presents the mechanism of reproducing relational structures created on the level SQL Server together with defined relations into the graph form on the level of Neo4j [5]. This in turn made it possible to make use of Cypher language paying special attention to searching relations between tables, which are represented on the level of graph database with the use of directed edges [1]. The limitation of the proposed solution was narrowing to single SQL Server environment, resulting from the use of defined classes within the namespace *Microsoft.SqlServer.Management.Smo* [2]. The objects created on the above basis made it possible for us to recognize database structure managed by the given SQL Server, while delivering necessary metadata at the same time. They were related with database selected by user.

A new, alternative approach, proposed by us, having more universal character, utilizes the technology called ADO.NET by Microsoft, which contains in itself numerous techniques of access to databases controlled by different database management systems (DBMS) [3]. Depending on the offer by specific producer of DBMS, in total we can make use of ODBC driver, OLEDB technology or native classes for the given database environment during the process. In addition to the above, the three mentioned solutions were presented in chronological order. From the point of view of efficiency, the last solution is preferable to the two remaining ones on condition that it is available. In general within each of the signaled techniques we can realize both connection-oriented and non-connection-oriented model. *Connection* is the key object in both models, enabling communication with DBMS, created openly by programmer or generated without programmer [8]. It offers a number of methods, but the function from the point of view of receiving information about metadata is function called *GetSchema* [12].

Information about relational structures that has been obtained as well as openly defined relations in the form of foreign keys will constitute the basis in the process of creating graphs on the level Neo4j [6]. Querying the existing base mapping only relational structure can be solely realized in client application offered by the producer of Neo4j or through additional functionality, which is yet to be implemented in the information system being under construction [9]. The process of reproducing relational structures in the form of graphs is possible through making programming interfaces available to .NET environment [14]. One of the earlier programming interfaces based on protocol HTTP (Hyper Transfer Protocol) is *Neo4j.Client*. It offers an approach type REST (Representational State Transfer), which means transformation of requests included in protocol HTTP, for operations such as creating, reading, updating and deleting data (CRUD) within their source [27]. At present, a more modern API (Application Programing Interface) called *Neo4.jDriver* is also available, which by contrast is based on network protocol Bolt [11]. It is characterized by generating less network traffic, which from the point of view of the aim of this paper, which is extension of application directed to expanding database environments being subject to metadata penetration, is of secondary importance [26]. In the application, which is under construction, we will solely deal with transfer of metadata, which as a general rule is less numerous that data themselves.

2. Application RELATIONS-Graph-v2

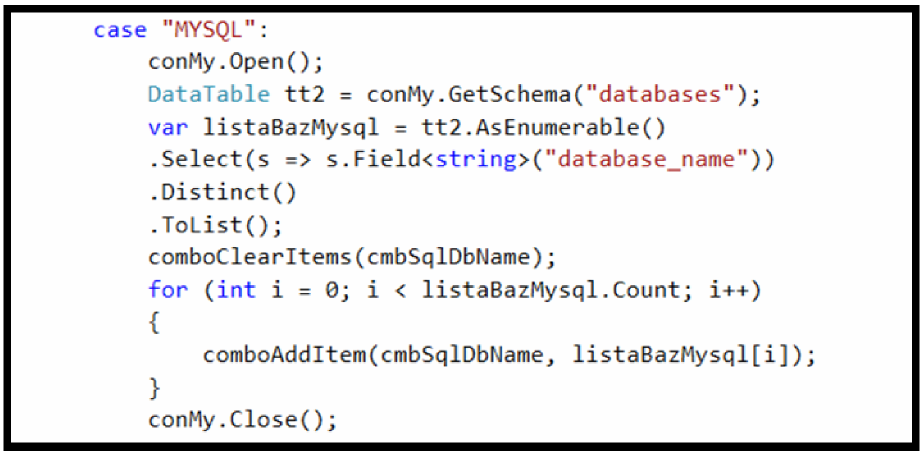
The process of designing application was proceeded in UML notation and it also entailed making another assumptions [4]. Apart from the previous selection of programming interface, ensuring access to data in the form of ADO.NET technology, the relational set DBMS was narrowed down to three [18]. The three database environments, namely MS SQL, Oracle and MySQL were considered to be relatively representative. It seemed pretty obvious that the main obstacle on the way to ensuring rather universal character of this application results from a diversified way of gathering metadata providing information about relational structure, and not the way of mapping them in the graph database [21]. As it has already been mentioned, the object called *Connection*, equipped, among other things, in the method called *GetSchema* is the necessary and key object enabling the operation of gathering data [16]. The use of this method of object combined with proper argumentation allows to gather different informations about all databases, including the system ones, which are managed by database system, as well as in the tables creating a given database, which can be seen in the following code - Fig. 1.

The data taken over are transferred to the object *DataTable*, which after being mapped into collection implementing interface *IEnumerable* constitutes source enabling to question them with the use of LINQ technology. This way of proceedings with the use of ADO.NET was successfully used with reference to MS SQL Server and MySQL [20]. Striving after ensuring universal character of this application induced authors to make and endeavor to utilize the method of object *Connection* on a wider scale through inputting other arguments [24]. Utilizing the argument called *ForeignKeys* in the form of chain, as well as using additional argument in the form of limit table did not give the expected results [23]. In the package of return information there is direct lack of referential informations, which are necessary in the process of creating edges in Neo4j database. Another deficiency is that the given collection and, as a matter of fact, its name are not accepted as the argument of *GetSchema* method on the level of older techniques such as ODBC and OLEDB. Some additional complications occurred while trying to use this method with reference to collection called *Columns* [13]. The returned results did not meet the author's expectations.

Alternative approach, characterized by much bigger capabilities involves the use of INFORMATION_SCHEMA included in the standard SQL-92, behind which there is a set of tables containing metadata [19]. In case of both DBMS MySQL and MS SQL Server we can say about conformity with the aforementioned standard in this issue [25]. By utilizing the aforesaid tables we can partially resign from the method *GetSchema*. In order to gather necessary information about metadata, we need to reach for the following tables INFORMATION_SCHEMA.TABLES, INFORMATION_SCHEMA.COLUMNS, INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE and INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS.

Fragment of code seen in Fig. 2., is an example of query returning all set of information characterized by defined relations.

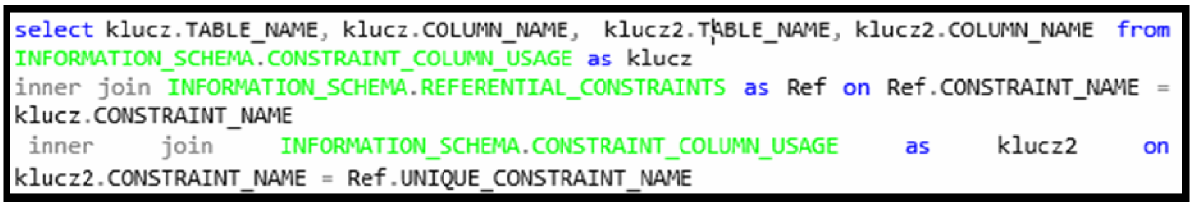
In case of our actions we need to use a wider array of classes creating ADO.NET. The necessary SQL queries can be sent to both the *Command* object and *DataAdapter* alternatively, which in consequence will determine the methods to be used later [22]. A slightly different scenario of conduct was utilized in case of Oracle database system. In this particular case, the basis for gathering all information about relational structure were its queries addressed to views containing metadata, which is illustrated in Fig. 3.



```
case "MYSQL":
    conMy.Open();
    DataTable tt2 = conMy.GetSchema("databases");
    var listaBazMysql = tt2.AsEnumerable()
    .Select(s => s.Field<string>("database_name"))
    .Distinct()
    .ToList();
    comboClearItems(cmbSqlDbName);
    for (int i = 0; i < listaBazMysql.Count; i++)
    {
        comboAddItem(cmbSqlDbName, listaBazMysql[i]);
    }
    conMy.Close();
```

Source: own study / Źródło: opracowanie własne

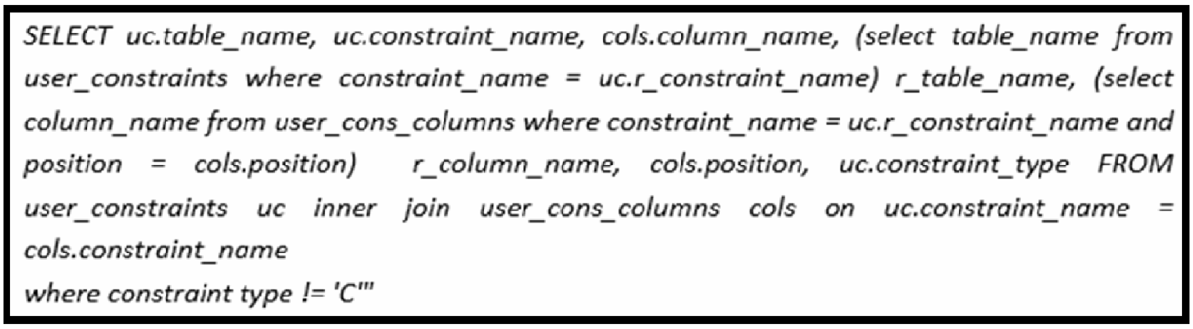
Fig. 1. Gathering information about databases managed by a selected instance MySQL
Rys. 1. Zdobywanie informacji o bazach danych zarządzanych przez wybraną instancję MySQL



```
select klucz.TABLE_NAME, klucz.COLUMN_NAME,  klucz2.TABLE_NAME, klucz2.COLUMN_NAME  from
INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE as klucz
inner join INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS as Ref on Ref.CONSTRAINT_NAME =
klucz.CONSTRAINT_NAME
 inner    join    INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE    as    klucz2    on
klucz2.CONSTRAINT_NAME = Ref.UNIQUE_CONSTRAINT_NAME
```

Source: own study / Źródło: opracowanie własne

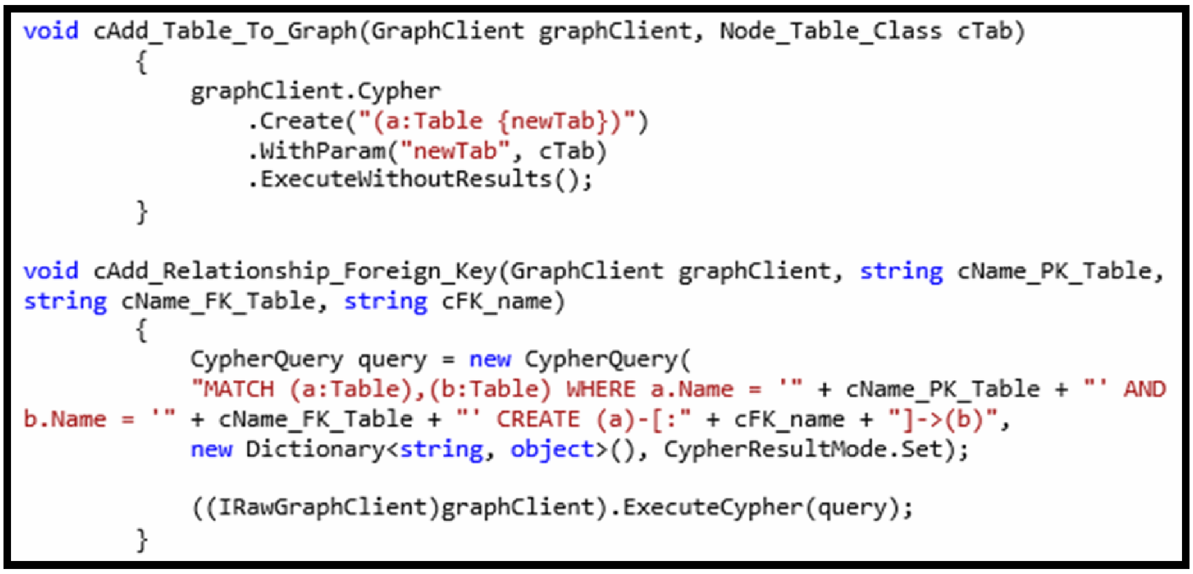
Fig. 2. SQL query returning information characterized by defined relations between tables
Rys. 2. Zapytanie SQL zwracające informacje o zdefiniowanych powiązaniach pomiędzy tabelami



```
SELECT uc.table_name, uc.constraint_name, cols.column_name, (select table_name from
user_constraints where constraint_name = uc.r_constraint_name) r_table_name, (select
column_name from user_cons_columns where constraint_name = uc.r_constraint_name and
position = cols.position) r_column_name, cols.position, uc.constraint_type FROM
user_constraints uc inner join user_cons_columns cols on uc.constraint_name =
cols.constraint_name
where constraint type != 'C'"
```

Source: own study / Źródło: opracowanie własne

Fig. 3. SQL query addressed to DBMS Oracle, including information about related tables
Rys. 3. Zapytanie SQL skierowane do SZBD Oracle, zwracającego informacje o powiązanych tabelach



```
void cAdd_Table_To_Graph(GraphClient graphClient, Node_Table_Class cTab)
        {
            graphClient.Cypher
                .Create("(a:Table {newTab})")
                .WithParam("newTab", cTab)
                .ExecuteWithoutResults();
        }

void cAdd_Relationship_Foreign_Key(GraphClient graphClient, string cName_PK_Table,
string cName_FK_Table, string cFK_name)
        {
            CypherQuery query = new CypherQuery(
            "MATCH (a:Table),(b:Table) WHERE a.Name = '" + cName_PK_Table + "' AND
b.Name = '" + cName_FK_Table + "' CREATE (a)-[:" + cFK_name + "]->(b)",
            new Dictionary<string, object>(), CypherResultMode.Set);

            ((IRawGraphClient)graphClient).ExecuteCypher(query);
        }
```

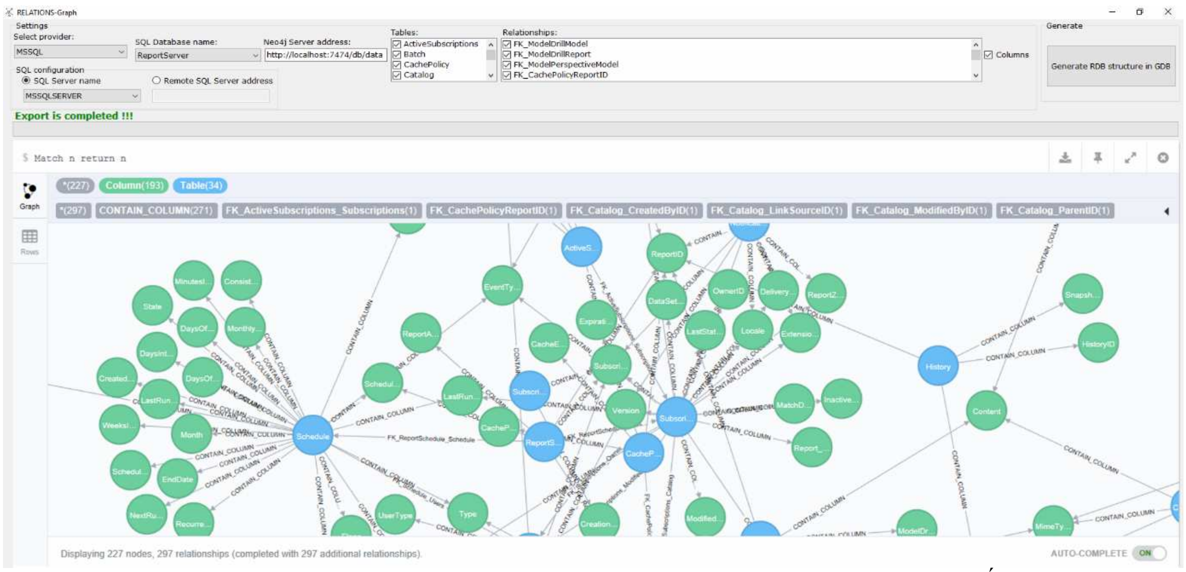
Source: own study / Źródło: opracowanie własne

Fig. 4. Creating nodes and edges in Neo4j with the use of language C# and programming interface GraphClient
Rys. 4. Tworzenie węzłów i krawędzi w Neo4j przy użyciu języka C# i interfejsu programistycznego GraphClient

Another actions of the application, now independent of relational system chosen by the user, entailed reproduction of information about database structures in the form of graphs on the level Neo4j [17]. This entry data embedded in objects *DataTables* were suitably converted into the form of nodes and edges with the use of two separate methods utilizing partly different objects, which the following fragment of code illustrates - Fig. 4.

The above actions of code included in the methods are coerced by the user's choices made in hierarchical way on the level of form, which is presented in Fig. 5.

At first, the user determines data supplier, on that basis, the user gains information about instances of database server installed locally. Then, the user makes a proper selection or alternatively writes connection chain in case of servers localized remotely. This in turn enables the choice of base, and then tables and recognized connections between them, which will be reproduced in graph structures. Location of the graph data is determined in neighboring text field [27]. The above operations constitute the basis for initiating methods creating the graph structures required by us. We create them by pressing the button „Generate RDB structure in GDB". The above form, through indicator light *WebBrowser* makes available for us program client called Neo4j, allowing querying graph database. The questions entered in Cypher language, allow us to search the database, which came into existence, from the perspective of gathering information about reconstructed relational structure [15]. The results are presented in a comfortable way in the form of graph.


RELATIONS-Graph

Settings
Select provider: MSSQL
SQL Database name: ReportServer
Neo4j Server address: http://localhost:7474/db/data
SQL configuration
(●) SQL Server name: MSSQLSERVER
( ) Remote SQL Server address
Tables: ☑ ActiveSubscriptions ☑ Batch ☑ CachePolicy ☑ Catalog
Relationships: ☑ FK_ModelDrillModel ☑ FK_ModelDrillReport ☑ FK_ModelPerspectiveModel ☑ FK_CachePolicyReportID
☑ Columns
Export is completed !!!

```
$ Match n return n
```

*(227) Column(193) Table(34)
*(297) CONTAIN_COLUMN(271) FK_ActiveSubscriptions_Subscriptions(1) FK_CachePolicyReportID(1) FK_Catalog_CreatedByID(1) FK_Catalog_LinkSourceID(1) FK_Catalog_ModifiedByID(1) FK_Catalog_ParentID(1)

Displaying 227 nodes, 297 relationships (completed with 297 additional relationships).

Source: own study / Źródło: opracowanie własne

Fig. 5. The application interface with results of querying the graph base representing a given relational structure
Rys. 5. Interfejs aplikacji z wynikami odpytywania bazy grafowej, reprezentującej wybraną strukturę relacyjną

3. Conclusions

The presented application constitutes continuation of previous actions taken by authors, resulting from limitations of the existing tools used to recognize and search, not allowing at the same time clear visualization of complex relational structures we are interested in. Currently offered solution has more universal character, since it does not only refers to single RDBMS, which is MS SQL Server. Currently designed and created application enables reconstruction of relational structures created on the level SQL Server, MySQL or Oracle in the graph form with the use of Neo4j. It was possible to achieve through the use of ADO.NET technology combined with the use of tables including metadata created and modified by DBMS. Access programming interface for the graph database Neo4 was another component which was utilized. The mapped relational structure in the form of graph can be subject to queries formulated in the language Cypher in order to get to know it in more details.

Increasing the level of universality of application is inseparably connected with its further extension and is largely dependent on the fact if another RDBMS included, implement standard SQL-92 from the perspective of INFORMATION_SCHEMA.

4. References

[1] Celko J., Morgan Kaufman „Joe Celko's Complete Guide to NoSQL" 2013. ISBN: 978-0-12-407220-6.
[2] Dewson R., Apress „Beginning SQL Server for Developers, 4th Edition" 2014. ISBN: 978-1-484202-81-4.
[3] Ellis G., Packt Publishing „Getting Started with SQL Server 2014 Administration" 2014. ISBN: 978-1-782-17241-3.
[4] Fowler A., Wiley „NoSQL For Dummies" 2015. ISBN: 978-1-118-90574-6.
[5] Goel A., Packt Publishing „Neo4j Cookbook" 2015. ISBN: 978-1-78328-725-3.
[6] Gupta S., Packt Publishing „Neo4j Essentials" 2015. ISBN: 978-1-78355-517-8.
[7] Idziaszek P., Mueller W., Rudowicz-Nawrocka J., Gruszczyński M., Kujawa S., Górna K., Balcerzak K.: Visualisation of Relational Database Structure by Graph Database. CMST, 2016, Vol. 22 (4), 217-224.
[8] Johnson E., Jones J., Addison-Wesley „A Developer's Guide to Data Modeling for SQL Server" 2008. ISBN: 978-0-321-49764-2.
[9] Jordan G., Apress „Practical Neo4j" 2015. ISBN: 978-1-484200-23-0.
[10] Kreigel A., Wrox „Discovering SQL" 2011. ISBN: 978-1-4571-0657-6.
[11] Lal M., Packt Publishing „Neo4j Graph Data Modeling" 2015. ISBN: 978-1-78439-730-2.
[12] Masood-Al.-Faroog B.A., Packt Publishing „SQL Server 2014 Development Essentials" 2014. ISBN: 978-1-78217-255-0.
[13] Naboulsi Z., Ford S., Microsoft Press „Coding Faster: Getting More Productive with Microsoft Visual Studio" 2011. ISBN: 978-0-73564-992-7.
[14] Oppel A., McGraw-Hill „Data Modeling" 2009. ISBN: 978-0-07-162398-8.
[15] Panazarino O., Packt Publishing „Learning Cypher" 2014. ISBN: 978-1-78328-775-8.
[16] Powell J., Chandos Publishing „A Librarian's Guide to Graphs, Data and the Semantic Web" 2015. ISBN: 978-1-78063-434-0.
[17] Raj S., Packt Publishing „Neo4j High Performance" 2015. ISBN: 978-1-78355-516-1.
[18] Redmon E., Wilson J.R., O'Reilly Media „Seven Databases in Seven Weeks" 2012. ISBN: 978-1-934356-92-0.
[19] Robinson I., Webber J., Eifrem E., O'Reilly Media „Graph Databases" 2013. ISBN:978-1-449-35626-2.
[20] Robinson I., Webber J., Eifrem E., O'Reilly Media „Graph Databases Second Edition" 2015. ISBN:978-1-491-93200-1.
[21] Sadalage P.J., Fowler M., Pearson Education „NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence" 2013. ISBN:978-0321826626.
[22] Schmalz M., O'Reilly Media „C# Database Basics" 2012. ISBN: 978-1-4493-0998-5.
[23] Sideris Courseware Corp. „Data Modeling: Logical Database Design" 2011. ISBN: 978-1-936930-19-7.
[24] Tiwari S., Wrox „Professional NoSQL" 2011. ISBN: 978-1-4571-0685-9.
[25] Vaish G., Packt Publishing „Getting Started with NoSQL" 2013. ISBN: 978-1-84969-498-8.
[26] Van Bruggen R., Packt Publishing „Learning Neo4j" 2014. ISBN: 978-1-84951-716-4.
[27] Vucotic A., Watt N., Abedrabbo T., Fox D., Partner J., Manning „Neo4j in Action" 2014. ISBN: 978-1-61729-076-3.