New Intelligent Tools to Adapt NL interface to Corporate Environments

Svetlana Chuprina[1][0000−0002−2103−3771] and
Igor Postanogov[1][0000−0002−7791−4789]

Perm State University, Bukireva Str. 15, 614990, Perm, Russia
chuprinas@inbox.ru, ipostanogov@outlook.com

Abstract. This paper is devoted to new aspects of Natural Language Interface to Relational Database (NLIDB) integration into third-party corporate environments related to control data access. Because there is no schema information in the input NL query and the different relational database management system (RDBMS) requires different metadata types and rules to control data access, developers meet a problem addressed to automatic data access control in the case of NL interface implementation to relational databases. In the paper, we suggest a comprehensive approach which takes into account permissions throughout the pipeline of transforming NL query into SQL query with an intermediate SPARQL representation. Our integration solutions based on well-known Ontology Based Data Access (OBDA) approach, which gives us the opportunity to adapt the proposed solutions to the specifics of access control facilities of various RDBMS. Suggested approach has been implemented within intelligent service, named Reply.

Keywords: Natural Language Interface · Natural Language Query to Relational Database · Database Access Control · Ontology Based Data Access · Intelligent Information System

1 Introduction

Nowadays there is a trend for providing natural language interfaces (NL interfaces, NLI) for human-machine communication. According to the Gartner's hype cycle report for emerging technologies [11], in 2017 Virtual Assistants with NL interfaces, such as Siri (iOS), Google Assistant and Yandex.Alisa (Android), Cortana (Windows) were at the peak of the cycle. Now the advances in this area (such as NL understanding) have a major impact on the corporate environment. Because a lot of time of employees is still spent on formulating of queries for retrieving data (see, for example, [6]), it is beneficial for companies to use high-level tools to simplify this process by means of NLIDBs. NLI lowers the entry threshold for newcomers and gives an ability to formulate ad hoc queries.

NLIDBs can be classified as domain-dependent and domain-independent. The domain-dependent usually demonstrates better results but are highly restricted to the domain specifics. In [3] we have presented our domain-independent system

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

`Reply` for transforming traditional information systems (IS) into intelligent ones with NL querying. End users are provided with new NLI to their existing IS by means of Reply without source code modification. It is also possible to start developing a new IS with NLI as a ready-made component to its database. To provide NLI to relational database (RDB) we have used an ontology-based data access approach that helps us automatically transform input NL query into SPARQL query and then into SQL query taking into account the specifics of concrete RDBMS, database schema, and its content.

Towards creating a platform which can be used as a self-service solution, we have taken a number preexisted stand-alone third-party modules and complemented them with our own modules to build a complete solution. Although many NLIDBs have been developed since 60s, there are only a few reports about real-world installation (e.g., [16]). Bearing in mind that "NLIDBs will not be widely used in businesses until their configuration is so easy that any computer professional could be able to perform it" [12], we have developed a number of tools, which automate the configuration throughout the whole pipeline. Thanks to that, our solution favorably distinguishes from others.

While bringing our system into real-world enterprise infrastructure we have encountered the problem that neither our, nor any third-party transformation module used in our pipeline was not taking into account user's access privileges for the data sources. E.g., SPARQL to SQL module expects that used configuration maps elements of the ontology to available elements of the database. Direct module integration led to permission violation exceptions at execution stage accompanied with error messages that are incomprehensible to the end user.

To the best of our knowledge, there is no published approach for automated handling data access policies in ontology-based NLIDBs, which use black box NL to SPARQL and SPARQL to SQL modules that are unaware of policy restrictions.

The main contributions of our research presented in the paper are the following:

1. adaptive ontology-driven approach to enforce practical value of NLIDBs taking into account the issues of restricted data source access;
2. algorithm for handling access policy in NLIDBs based OBDA solutions, where SPARQL is used as an intermediate representation language;
3. demonstration of the proposed approach with an example.

The paper is divided into six sections. Section 2 contains the description of the suggested approach. Section 3 includes a demo example. Section 4 presents a preliminary evaluation of the approach. Section 5 describes the related work. Section 6 contains a conclusion and future perspectives.

2 NLIDB Obstacles and OBDA Solutions

In this paper, we will focus on access policies for 2 types of database objects such as tables and columns. Access privileges for rows (row-level security) are

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

usually implemented with the help of data views. The user cannot violate access privileges for rows and if some data row shouldn't be returned to the user according to the access control requirements, it should have already been filtered from the result. For cell-level privileges it is common to mask the not accessible actual value and inform the user that the value was masked. Before discussing the NLIDB obstacles, let's say a few words about data access control in traditional information system UIs.

For a traditional information systems UI it is very easy to report current access restrictions. If there are some UI controls that lead to the execution of queries that cannot be executed, they can be disabled or hidden. Queries that cannot be executed are queries that access restricted tables (for table-level restrictions) or restricted columns in table references and search conditions clauses (for column-level restrictions). Asterisk signs in select lists are explicitly expanded to the available columns. In the case of whole database search (e.g., by substring) the search should be done only in the available tables and columns.

User interface for an ad hoc query usually has controls for specifying blocks in the resulting SQL query, e.g., there are controls for choosing output fields, conditions, sorting order etc. For the specification stage, all restricted tables and columns should be disabled or hidden. If UI suggests possible values for columns that are affected by row or cell-level security, the suggestions should contain only visible values. For the building stage, where the resulting SQL query is formulated, the building engine should use only those paths and join conditions for connecting required tables, which avoid restricted objects. If this is not possible, advanced ad hoc query UIs suggest which fields or tables should be excluded to make query building successful.

For NLIDB it is usually difficult to inform the user what information is available via NLI. The easiest way to do this is to describe the available information in the documentation. More advanced user interfaces suggest autocompletion tools. Another problem is a lack of adequate reporting if the query fails, which can occur for several reasons. In this paper, we focus on the case when the query has been successfully interpreted and the execution has failed due to permission violation.

This problem can be solved in two main ways:

1. Interpreting the query without considering permissions, executing the query and catching the exception, interpreting the exception and reporting the reason why this exception has happened.
2. Considering permissions on the interpretation stage and trying to avoid permission violation.

The challenge of the first way is that the exception could have happened due to a number of reasons, so it is difficult to take into account all the related issues. That is why we chose the second way.

Considering permissions on the interpretation stage is specific to the NLIDB type. Different NLIDB types may be classified by architecture into four groups [1]:

1. pattern-matching;

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

2. syntax-based systems;
3. semantic grammar systems;
4. intermediate representation languages systems.

For the intermediate representation language system, one could use SPARQL. Therefore, an input NL query is transformed into SPARQL query which is later transformed into SQL query by an OBDA framework. Reply has this type of architecture with SPARQL as the intermediate representation language.

OBDA systems built on top RDBs can be implemented in two different ways: materialized and virtual. According to the materialized approach, all the data from RDBs should be transformed into ontology representation and stored in RDF triplestores. In the virtual approach, the data is stored only in RDBs, and querying the virtual RDF graph using SPARQL leads to execution the corresponding SQL query over the original source thus avoiding the cost of materialization [2].

From the security point of view, another benefit of the virtual approach is that data is not duplicated in RDF triplestore. Although access control can be specified at the RDF level [7], not so many IT-specialists who knows not only how to specify user access rights in the RDBMS, but also how to specify user access rights in RDF triplestore. With that in mind, many companies would reject such OBDA systems because the possibility of data leaks due to incorrect configuration and may overweight the potential benefit of the intelligent system applying. If the virtual approach is used, end user can only access the data which she/he can access using traditional ad hoc query interface.

In OBDA solutions for NLIDBs there are 4 components where permissions could be considered:

1. OBDA framework;
2. NL to SPARQL translator;
3. ontology;
4. mappings between elements of an RDB schema and elements of an ontology.

For this paper, we treat OBDA framework as a black box with ontology, mappings and SPARQL query as input and an SQL query as an output. With this assumption, we show how OBDA solutions for NLIDBs could be enhanced.

The original architecture is presented at Fig. 1. For the sake of simplicity do not depict the NL to SPARQL module errors on the diagram.

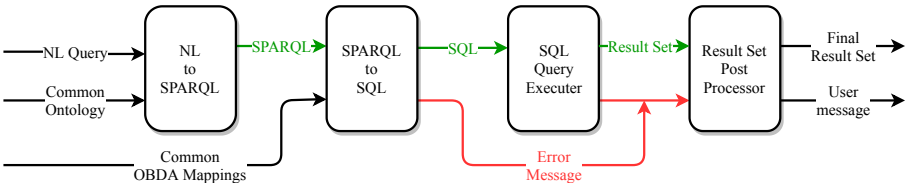

Fig. 1. Original Reply's execution stage.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

OBDA mapping specification consists of 2 blocks: *Source* and *Target* (in the Ontop mappings notation). *Source* is an SQL SELECT query that would be executed by RDB to raise triplets defined in *Target*.

On the assumption that *Target* contains only one template for an RDF triple, this triple has one of the following types:

1. `<instance_uri> a <class_uri>`
2. `<instance1_uri> <object_property_uri> <instance2_uri>`
3. `<instance_uri> <data_property_uri> <value>`

SQL SELECT query has the following structure:

`SELECT <select_list> FROM <from_item> WHERE <condition>`

When considering the permission aspect, *select_list* and *conditions* from the SQL SELECT query could use fields, which could be forbidden. In addition, the *from_item* can refer to the forbidden tables or fields. Although SQL looks different from the NL query point of view, their building blocks are the same. NL query can vary output fields, requested entities types, and conditions. Like in SQL, output fields can be specified implicitly.

The easiest way to prevent execution failure of the translated query is to remove elements of the ontology that are mapped to the forbidden elements of the RDB. In this case, the queries that contain removed elements of ontology could be misinterpreted. Firstly, the closest element in the edited ontology maybe not related to the concept/relation which the user has asked for. Secondly, if the closest concept/relation could not be found, the related part of the query would be completely ignored, and the NLIDB has not enough data to inform the user about the situation and give some recommendations about the query reformulation. Worse, in this case, the user can conclude that the system cannot handle complex queries. Finally, the system would not know why the user was not satisfied with the query result and would not give any recommendations about query reformulation.

In order to tackle this problem, we suggest the following approach. For each user (or for each role if multiple users have the same privileges) automatically a new instance of ontology and a set of mappings are created by means of our personalization module. The individual user's instance of ontology is created by copying the common ontology and annotating the elements for which mappings have been removed and no other mappings left. The individual user's set of mappings is created by removing from the common mapping specification only those ones, *Sources* of which suppose access to restricted tables or fields. This process is presented at Fig. 2.

To provide the best user experience translators from NL to SPARQL should handle permission-based restrictions. Unfortunately, none of the tools that we have analyzed [4, 15, 18] support this. Here we suggest an approach how the existing NL to SPARQL translators can be minimally modified to support permission-based restrictions.

One of the main requirements is that each output data property, that wasn't explicitly specified in the user's query, should be wrapped into `OPTIONAL` block of the related SPARQL query. If there are no mappings for the data property

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

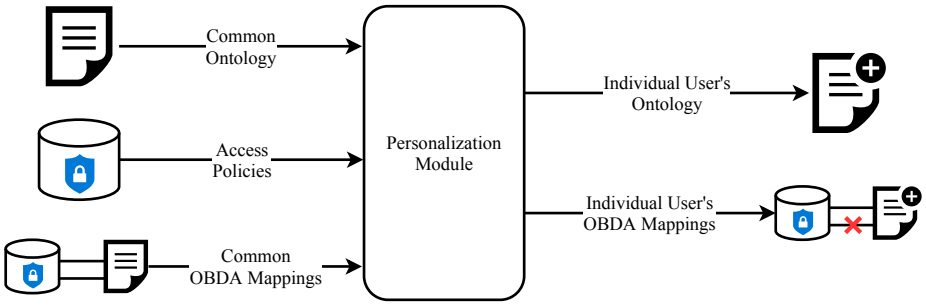

Fig. 2. Creating individual user's instance of ontology and set of mappings.

that is wrapped into `OPTIONAL` block [17], OBDA framework would return the corresponding SQL query where the not found column would have `NULL` values.

To make Reply more adaptable to the user preferences the default values for output attributes of the queried concepts are supported. E.g., if some user types the query "list clients", a predefined subset of fields would be returned per client. Additionally, the system supports application-wide defaults, and the user can also choose the preferred attributes. This approach is implemented by excluding `OPTIONAL` blocks with output data properties that are not in the corresponding preferred list. An alternative connection between the related classes via object properties can be specified by means of `UNION`.

Algorithm 1 describes steps of our approach in case of permission change. Algorithm 2 can be used for modifying intermediate SPARQL query for user's NL query. The complete transformation process is presented at Fig. 3. The new modules are marked by the blue color.

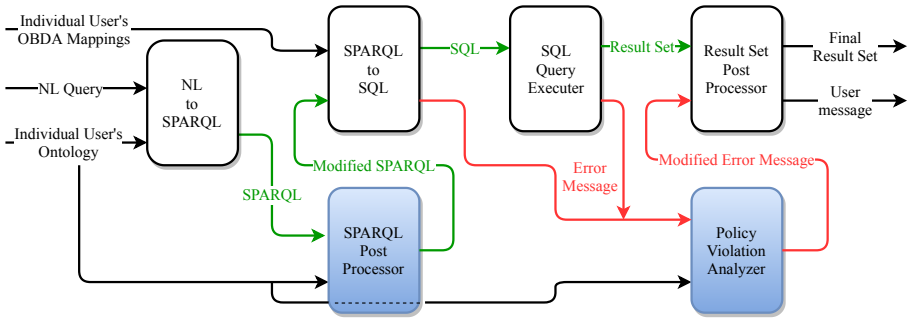

Fig. 3. Modified Reply's execution stage.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

```
foreach user U in tracked users do
    retrieve permissions from the RDB for U;
    if permissions changed then
        create a copy of common mappings B for U;
        foreach table T used in the Sources of mappings do
            if access to T is not allowed then
                remove all mappings, that reference T;
            end if
        end foreach
        foreach column C used in the Sources of mappings do
            if access to C is not allowed then
                remove all mappings, that reference C;
            end if
        end foreach
        foreach ontology element E that had mappings in B do
            if there are no spawning mappings left then
                mark E as unavailable due to the policy restrictions;
            end if
        end foreach
    end if
end foreach
```

Algorithm 1: Generating users' mappings on access policy change.

```
transform NL query Q into SPARQL query SPQ;
init empty list of errors ERRS;
foreach non-optional ontology element E in SPQ do
    if E is marked as unsupported then
        add into ERRS an error that E is unsupported;
    else if E is marked as unavailable due to the policy restrictions
     then
        add into ERRS an error that E is unavailable;
    end if
end foreach
if ERRS is not empty then
    return ERRS;
end if
foreach optional ontology data property P in SPQ do
    if P is not in the preferred set of attributes then
        remove P from SPQ;
    end if
end foreach
transform SPQ into SQL query SQ;
return execute SQ;
```

Algorithm 2: Executing NL query taking into account access policy.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

3 Example

To demonstrate the suggested approach, we present an example of our demo database about apartment agreements. Each house contains a number of apartments. Some of the apartments are rented, and some of them have been sold. For each rented apartment there is a rental agreement, and for each sold apartment there is a service agreement.

A person who has a running rental agreement is called tenant, and a person who owns an apartment (has a running service agreement from the landlord's point of view) — owner. The fragment of this database schema is presented at Fig. 4.

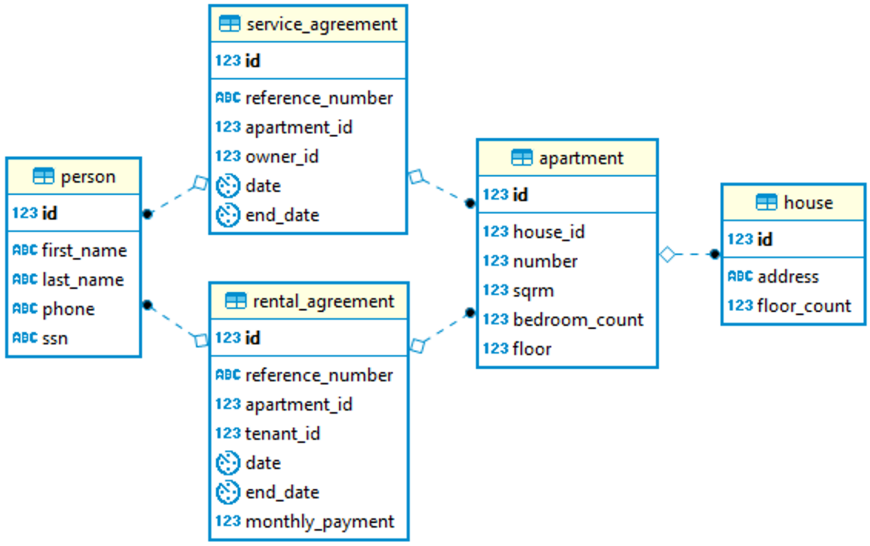

Fig. 4. Fragment of database schema about apartment agreements.

The restrictions on this database are the following: technical support specialist have no access to *service_agreement* table (therefore have no access to the information who are the owners in the list of people), and have no access to a *social security number* of any person. If the database has row-level access policies, the forbidden rows would be hidden by the database management system itself so there is no need in a special treatment on the OBDA layer. If there is a cell-level access policy, restricted sells would be marked as hidden (either by `NULL` or predefined value) and would not need the special treatment.

As it was suggested above, the individual user's set of mappings was created by removing from the source mapping specification only those ones, *Sources* of which suppose access to restricted tables or fields.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

Fig. 5 presents a result of comparison between source mapping rules file and the corresponding individual user's file. The latter lacks mappings related to *service_agreement* table and *SSN* column.

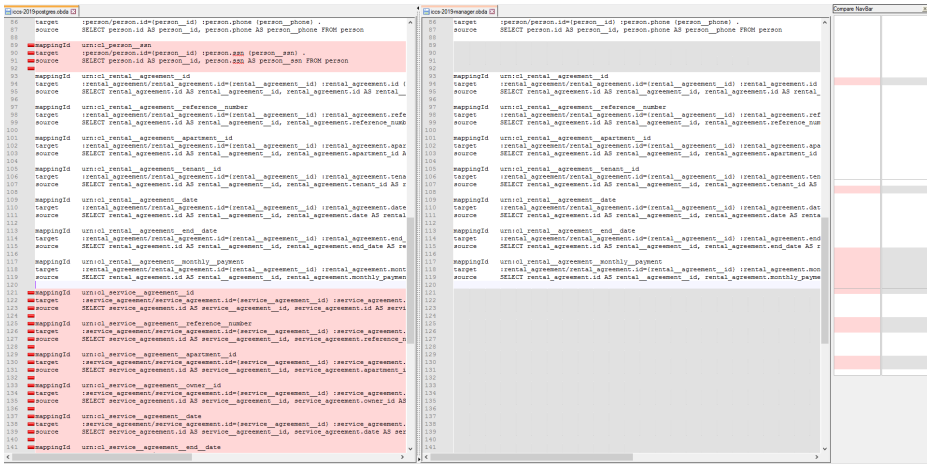

Fig. 5. Comparison between source and individual mapping rule files.

For NL query "Return all information about the tenants" the traditional NL to SPARQL engine would detect that person has an *id*, *first_name*, *last_name*, *phone*, and *SSN* and would formulate a SPARQL query returning all these columns. Running this query would lead to an access denied exception due to the *SSN*'s restriction. To solve this problem one could supply the NL to SPARQL engine information about the restricted fields or post-process the generated query. The easiest solution is to put the expected fields in `OPTIONAL` blocks. In that case, because the mapping for the forbidden fields has been removed, the OBDA framework would just skip them without any exception.

The same solution is adequate for the cases when all the output fields are available and explicitly specified (e.g., NL query "Return first and last name of tenants"). For NL query "Return first name, last name and SSN of tenants" OBDA framework would fail to generate the SQL query because of the lack of mappings for *SSN*. To report the user that the field *SSN* was detected in the query but was omitted because of the policy restrictions, our system parses the generated SPARQL query to find the data properties for which there are no mappings. This approach is error-prone, that's why we suggest for NL to SPARQL query engines to explicitly output all detected classes, objects and data properties as well as the role they play in the query.

All constraints, retrieved from the query should be specified in the SPARQL query as usual. If the user constrains the available field, the query would be successfully executed (e.g., "Return all information about tenants with last name Doe"). If the user constrains the restricted field ("Return all information about

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

a person with SSN 078-05-1120"), the OBDA framework would fail to generate the corresponding SQL query due to the lack of mappings for *SSN* attribute. There is an option to remove the constraint automatically and to answer the query without this constraint, but we believe that most of the time it would be not the result the user expects and it is more adequate to explain to the user why this query has no opportunity to be executed.

Although there is no table named *tenant* in the database, we specify tenants as people who have a running rental agreement. The same approach applies to the owners. The user with the permission policy specified above would not have access to service agreements so she/he not able to get the expected response to the query "Return all information about service agreements".

For NL query "Return all information about people at 1650 El Prado" the system will establish that the only object property available between *person* and *apartment* (which is a part of the *house* at 1650 El Prado) is *having a rental agreement*, and would return only data about *tenants*. If the user explicitly asks about the *owners* ("Return all information about owners at 1650 El Prado") the system demonstrates the intelligent capabilities and returns the suggestion to reformulate the query asking about *people* (superclass) or *tenants* (sibling).

An example of the SPARQL query that would be generated for the NL query "Return all information about people at 1650 El Prado":

```
SELECT DISTINCT ?person_fn ?person_ln ?person_phone ?person_ssn {
  {
    ?agreement :service_agreement_person_id_fkey ?person.
    ?agreement :service_agreement_apartment_id_fkey ?apartment.
  } UNION {
    ?agreement :rental_agreement_tenant_id_fkey ?person.
    ?agreement :rental_agreement_apartment_id_fkey ?apartment.
  }
  OPTIONAL { ?person :person.first_name ?person_fn. }
  OPTIONAL { ?person :person.last_name ?person_ln. }
  OPTIONAL { ?person :person.phone ?person_phone. }
  OPTIONAL { ?person :person.ssn ?person_ssn. }
  ?apartment :apartment_house_id_fkey ?house.
  ?house :house.address ''1650 El Prado''.
}
```

The first block of the `UNION` clause and the SSN's `OPTIONAL` clause would be omitted by an OBDA framework due to the lack of mappings which have been removed according to our approach.

4 Implementation and testing specifics

The suggested approach was implemented in our system Reply which transforms traditional information systems into intelligent ones with a natural language query interface. In the previous version of Reply administrator had an ability to

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

create multiple independent versions of intelligent systems by specifying multiple pairs of ontology and mappings. In the current version, in addition to that, a base pair of ontology and mappings could be specified and a permission tracking option could be enabled. In that case, our system automatically retrieves access policies from RDB metadata and generate multiple configurations per selected users according to the suggested approach. This process can be rerun from the system administrator's GUI, by an API call or automatically by our system. The automatic permission tracking is available for a subset of supported databases.

For example, to automatically track changes in user permissions for `PostgreSQL` we generate a code for creating an *event trigger* that is fired in case of `GRANT` or `REVOKE` DDL commands. The generated code should be executed by a database superuser. In the body of the trigger, we raise a *notification* to a channel that our system *listens* to. PostgreSQL's metadata about user permissions is retrieved from the *information_schema.column_privileges* view.

To parse SQL queries in the *Sources* of the mappings we use `JSqlParser`, and for manipulating generated SPARQL query — `Apache Jena`.

Because we haven't found a NL to SPARQL translator that outputs implicitly requested attributes in a consistent way to our approach, testing the whole pipeline starting with NL query would not provide more information than starting with SPARQL query. As it was said above, our approach is independent not only from the language used by the user but also from the specifics of the query, from the specifics of the implementations of NL to SPARQL and SPARQL to SQL translators.

To test our system, we use the database from the example above (see Fig. 4). The database schema contains one-to-many and many-to-many relations, as well as multiple paths between tables. We have generated SPARQL queries, which use combinations of available and forbidden tables and columns. Our system has successfully reported permission violation and suggested what elements of the NL query should be avoided (even though there was no one). In the real-world example, the reported elements might not be explicitly included in the user's NL query, and so to make the report even clearer, tighter integration with NL to SPARQL translator is needed. We have also tested whether our system correctly reacts to permission updates.

5 Related Work

Modern corporate NLIDBs have a number of additional requirements that are not usually considered by academia:

1. taking into account user's access privileges for the data sources;
2. providing out-of-the-box integration with corporate chat platforms;
3. reusing existing corporate knowledge resources for automation the configuration stage.

At present, there are no platforms, which provide a comprehensive OBDA solutions for NLIDB and at the same time cover all of its pipeline stages. Nevertheless, there are a number of tools, which implement the individual stages

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

of the pipeline to transform NL query into SQL query with the intermediate SPARQL representation [2, 4, 15, 18]. Almost all of them expect that either all instances in ontology or the whole database are available. This is not the case for corporate environment where the users usually have only limited access to data. The papers [10, 12] present sufficiently complete overview of the NLIDB state of the art, including benefits and limitations of different solutions.

Another key factor for successful installation NLIDB into corporate environment is bringing a tight integration with existing NL tools. One of the most widely used types of them is corporate chat platforms that support chat bots. Amongst other things, hat bots can report a status of the running systems, run tests and even deploy code to the production environment. From the earliest versions, Reply had an API that can be called from user's NL query interface to retrieve the query result. While bringing Reply into real-world installations we have developed out of the box wrappers for the most popular corporate chat platforms. Reply can be integrated into third-party web-based information systems as a separate web-page. Due to the fact that we provide HTTP REST API, Reply can also be used by desktop applications.

Previously, to automate Reply configuring we have successfully implemented a number of our [14] and other's [5] state of art techniques. E.g., we automatically convert RDB's schema to ontology representation and enrich it with synonyms, parent and child concepts using our or third-party ontology or linguistic resources [8, 13]. In corporate environment, we can also use corporate knowledge resources and document storages as a source for ontology learning. Nowadays, many of knowledge resources use wiki or wiki-based markup language. Our approach for constructing ontologies from knowledge bases being developed is similar to the approach used for building DBpedia [9].

To improve the comprehensiveness of NLIDB's and make the NL-interface enable users to get really practical results we suggest new intelligent tools to adapt NL-interface to corporate environments on self-service principles.

6 Conclusion

In the paper, we presented the new approach to intelligent handling database access policies within OBDA. It helps to tackle the problems of adaptation of NLIDBs to corporate environments. Our approach does not depend on the methods used to implement the transformation of NL query to SPARQL as well as SPARQL to SQL so it is applicable to a wide range of OBDA solutions for NLIDBs.

According to the algorithm, the system automatically modifies the generated SPARQL query taking into account the access policies of target DBMS and user's preferences. The algorithm is demonstrated on the apartment agreements database example.

In future, we plan to apply developed tools and services not only for querying RDBMS using natural language but also for inserting data into them. We believe that it is possible to use our intelligent tools to improve the human-computer

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

interaction within traditional corporate environments and also in other environments, such as human-centric Internet of Things.

References

1. Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural Language Interfaces to Databases - An Introduction. Natural Language Engineering **1**, 29–81 (1995). https://doi.org/10.1017/S135132490000005X
2. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL Queries over Relational Databases. Semantic Web Journal **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217
3. Chuprina, S., Postanogov, I., Nasraoui, O.: Ontology Based Data Access Methods to Teach Students to Transform Traditional Information Systems and Simplify Decision Making Process. Procedia Computer Science **80**, 1801–1811 (2016). https://doi.org/10.1016/j.procs.2016.05.458
4. Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., Lehmann, J.: AskNow: A Framework for Natural Language Query Formalization in SPARQL. In: Sack, H., Blomqvist, E., d'Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C. (eds.) The Semantic Web. Latest Advances and New Domains. pp. 300–316. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-34129-3_19
5. Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., Mora, J.: BootOX: Practical Mapping of RDBs to OWL 2. In: Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015. pp. 113–132. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_7
6. Kharlamov, E., Solomakhina, N., Özçep, Ö.L., Zheleznyakov, D., Hubauer, T., Lamparter, S., Roshchin, M., Soylu, A., Watson, S.: How Semantic Technologies Can Enhance Data Access at Siemens Energy. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 8796, pp. 601–619 (2014). https://doi.org/10.1007/978-3-319-11964-9
7. Kirrane, S., Mileo, A., Decker, S.: Access control and the Resource Description Framework: A survey. In: Semantic Web (2017). https://doi.org/10.3233/SW-160236
8. Kostareva, T., Chuprina, S., Nam, A.: Using Ontology-Driven Methods to Develop Frameworks for Tackling NLP Problems. In: Supplementary Proceedings of the Fifth International Conference on Analysis of Images, Social Networks and Texts (AIST 2016), Yekaterinburg, Russia, April 6-8, 2016. pp. 102–113 (2016), `http://ceur-ws.org/Vol-1710/paper11.pdf`
9. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al.: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web **6**(2), 167–195 (2015). https://doi.org/10.3233/SW-140134
10. Nihalani, N., Silakari, S., Motwani, M.: Natural language Interface for Database: A Brief review. International Journal of Computer Science Issues (IJCSI) **8**(2), 600–608 (2011)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3

11. Panetta, K.: Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017, https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/, accessed: 2019-01-05
12. Pazos R., R.A., González B., J.J., Aguirre L., M.A., Martínez F., J.A., Fraire H., H.J.: Natural Language Interfaces to Databases: An Analysis of the State of the Art, pp. 463–480. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33021-6_36
13. Postanogov, I., Jastrzab, T.: Ontology Reuse as a Means for Fast Prototyping of New Concepts. In: Kozielski, S., Mrozek, D., Kasprowski, P., Malysiak-Mrozek, B., Kostrzewa, D. (eds.) Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation. pp. 273–287. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-58274-0_23
14. Postanogov, I.: Towards Automating the Creation of OBDA Systems. Procedia Computer Science **150**, 511–517 (2019). https://doi.org/10.1016/j.procs.2019.02.086, proceedings of the 13th International Symposium "Intelligent Systems 2018" (INTELS'18), 22-24 October, 2018, St. Petersburg, Russia
15. Shekarpour, S., Marx, E., Ngonga Ngomo, A.C., Auer, S.: SINA: Semantic Interpretation of User Queries for Question Answering on Interlinked Data. Journal of Web Semantics (2015). https://doi.org/10.1016/j.websem.2014.06.002
16. Waltinger, U., Tecuci, D., Olteanu, M., Mocanu, V., Sullivan, S.: Natural Language Access to Enterprise Data. AI Magazine **35**, 38 (2014). https://doi.org/10.1609/aimag.v35i1.2502
17. Xiao, G., Kontchakov, R., Cogrel, B., Calvanese, D., Botoeva, E.: Efficient Handling of SPARQL OPTIONAL for OBDA. In: Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L.A., Simperl, E. (eds.) The Semantic Web – ISWC 2018. pp. 354–373. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_21
18. Xu, K., Feng, Y., Zhao, D.: Xser@QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing. In: CEUR Workshop Proceedings (2014). https://doi.org/10.1007/978-3-662-45924-9_30

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_3