SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 756
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 99–117 P.

Mapping between Relational Databases and OWL Ontologies: an Example

Guntars Bumans
Department of Computing, University of Latvia
Raina bulv. 19, Riga, LV-1586, Latvia
guntars.bumans@gmail.com

This paper shows how relational databases can be used to define a bridging mechanism between relational database and OWL ontology. We demonstrate on a simple yet completely elaborated example how mapping information stored in relational tables can be processed using SQL to generate RDF triples for OWL class and property instances. This technology provides the means to use relational database as a powerful tool to transform relational data to Semantic Web layer.

Keywords: relational databases, RDF triples, mappings.

1 Introduction

There are several studies or tools allowing mapping relational databases (RDBs) to RDF schema or OWL ontologies. Some of the most notable approaches of this kind are R2O [1], D2RQ [2], Virtuoso RDF Views [3, 4] and DartGrid [5]. There is W3C RDB2RDF Incubator Group [6] related to standardization of RDB to RDF mappings, the group has published its survey of mapping RDBs to RDF [7].

R2O [1] approach defines declarative and extensible language (in xml) to describe mapping between given RDB and an OWL ontology or RDFS schema so that tools can process this mapping and generate triples that correspond to source RDB data. D2RQ [2] technology is another bridging technology where one can use SQL to describe the mapping information. This language is closer to SQL level and is not as declarative as R2O. Both D2RQ [2] and Virtuoso RDF Views [3, 4] allow retrieving instance data from RDB on-the-fly during the execution of SPARQL queries over the RDF data store.

The aim of this paper is to demonstrate a very simple standard SQL-based RDB to RDF/OWL mapping approach that is based on defining correspondence between the tables of the database and the classes of the ontology, as well as between table fields/links in the database and datatype/object properties in the ontology (with possible addition of filters and linked tables in the mapping definition), and later automatically generating SQL statements that generate the RDF triples that correspond to the source database data.

Our work setting for RDB to RDF/OWL translation involves the assumption that both the database and the ontology (or RDF schema) are given. The translation is not meant to be on-the-fly because huge amount of data can be involved. This corresponds to the practical database semantic re-engineering task, as advocated in [8, 9, 10, 11] in the setting of Latvian medical research databases.

It should be possible to translate the mappings specified here into other RDB-to-RDF/OWL mapping formalisms (e.g., R2O [1], D2RQ [2], Virtuoso RDF Views [3]), thus obtaining alternative implementations of these mappings.

There is at least a conceptual possibility to create the mapping between the source RDB schema and target OWL ontology by means of model transformations described in some transformation language (e.g. MOF QVT [12], ATL [13], or MOLA [14]); however, typically these translations are not supported on data in RDB or RDF formats and require the use of an intermediate format (a so-called model repository, such as EMF [15]) which may not be feasible for large data sets.

Our approach is to go for direct translation of RDB-stored data into (conceptual) OWL ontology format that can be executed on the level of DBMS.

In Section 2 of this paper we describe the mapping method and provide the table structure for storing mapping data. Section 3 introduces the demonstration example. Section 4 describes and demonstrates the instance generation process for OWL classes, OWL datatype properties and OWL object properties. Section 5 concludes the paper.

2 A Mapping Schema

We propose a bridging mechanism between relational databases and OWL ontologies. We assume that the ontology and the database have been developed separately. Most often the database is of legacy type but the ontology reflects the semantic concerns regarding the data contents. Our approach is to make a mapping between these structures and store the mapping in meta-level relational schema (we are working towards mapping specification language that is suitable for the end user, which however is beyond the scope of this paper). This approach allows us to use relational database engine to process mapping information and generate SQL sentences that, when executed, will create RDF/OWL-formatted data (RDF triples) describing instances of OWL classes and OWL datatype and OWL object properties that correspond to the source RDB data.

In the simplest case, an OWL class corresponds to a RDB table, an OWL datatype property corresponds to a table field, and an OWL object property corresponds to a foreign key. In real life examples the mappings are not so straightforward.

For example, an OWL class *Person* could be a domain for OWL datatype property *personAddress*. But the corresponding database table *persons* could have a foreign key reference to some other table having address information. To complicate things even more, one property of type *xsd:string* can correspond to a combination of columns spread over many tables in the database (e.g., country, city, street information stored in separate tables). Other possible causes of direct mapping impossibility are subclass relation in the ontology, the use of many to many relations, the non-existence of “natural” foreign keys in RDB. Often databases are normalized and their structure is optimized out of performance concerns thus hiding true conceptual meaning. To deal with all this complexity, we introduce mapping schema (Fig. 1). We will call it *mapping DB*. Source database (legacy type) will be denoted by *source DB* in this paper.

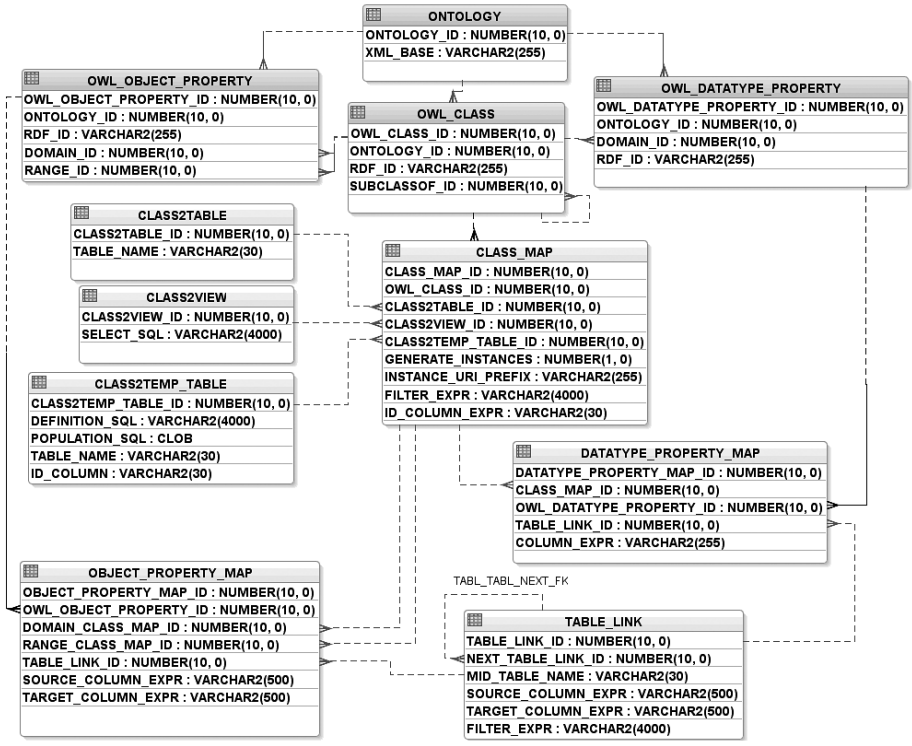

Fig. 1. A mapping schema between OWL ontology and relational database

The OWL ontology basic information (classes and properties) is stored in tables *ontology*, *owl_class*, *owl_object_property* and *owl_datatype_property*. For each OWL class the table *class_map* defines the connections to RDB objects that can be a table (a real, existing table in a source database), a view (a view can be defined inline using column *class2view.select_sql*), or a temporary table (*class2temp_table*). In typical cases only real existing source database tables are referenced: however, the view and temporary table techniques are available to handle certain advanced mapping situations. For example, the temporary table mechanism allows to create a new table in triple generation process (*definition_sql* column) and fill it with data (*populate_sql).* This mechanism can be used both for complex data-specific mappings (e.g. splitting a comma-separated list into independent element data values), and for providing an auto-generated identity column as a resource for OWL class instance URI generation. To simplify the further description, we will assume that mapping is to real database tables *(class2table)*. This way we do not lose the functionality that is required in the forthcoming example that does not use either inline defined views or temporary tables. From the perspective of transformation process, mapping specified in *class2table* can be also directed to named view in the database. Both tables and named views have columns and no other table specific information is used.

Each row in the *class_map* table contains a further filter possibility on the referenced RDB object (the *filter_expr* column), as well as the specification for

target OWL class instance URI generation on the basis of the data contained in the referenced RDB object. The URI of an OWL class instance is defined based on a record in the *class_map* table, by concatenating the *ontology.xml_base* value with *instance_uri_prefix* column value (typically holding the name of the referenced RDB object), followed by the contents of *id_column_expr* (column or column expression) evaluated in the referenced RDB object in *source DB*. The *generate_instances* column specifies whether the OWL class instances indeed have to be generated by this *class_map* specification. If the instances are not to be generated, the *class map* can still be used as a reference point in further *object_property_map* and *datatype_property_map* definitions.

Table *object_property_map* holds specifications for instance generations for OWL object properties. Each record in this table is based on a *class_map* record for both domain and range of the object property, as it describes or references the rules how the triples connecting the domain and range instances by this object property are formed. In the simplest case, a column expression is specified for both DB objects (tables) corresponding to the domain and range class maps (in *source_column_expr* and *target_column_expr*, respectively), and the values of these expressions are required to coincide for a triple to be created. If DB objects (tables) for domain and range class maps cannot be joined directly based on column expressions but they can be joined through some intermediate table joins then these middle joins are specified by rows in *table_link* table (more on this in Sub-section 4.3).

The table *datatype_property_map* holds specifications for instance generation for OWL datatype properties. Each record in this table is based on a *class_map* record for domain of the datatype property. The record maps domain to DB table (or view) and specifies how to generate subject part of generated triples: filtering, URI creation is done the same way as for OWL instance generation. Column *column_expr* specifies how to generate object part of triples (value for range). In the simplest case it is evaluated in the table specified for domain class map. If the value is to be taken from some other table then *table_link* row can be used to specify join to that table. Example: *Person* table is domain table but range for the property is *address* that is stored in *Address* table to which *Person* has foreign key.

A *mapping DB* can hold information of more than one OWL ontology to database mapping. For each such mapping there should be a separate row in *ontology* table and foreign key to it from *owl_class*, *owl_datatype_property* and *owl_object_property* table.

3 A Mapping Example

To better explain our proposed approach, we will use a simple example taken from [8] in Fig. 2. and 3. Below are a sample database schema and a corresponding ontology (OWL class *Thing* is omitted for simplicity).

For example, the classes *Student* and *Course* in this sample ontology have corresponding tables *student* and *course* in the sample database. To get instance data for OWL object property *takes* the table link path is needed: tables *student* and *registration* joined on *student_id* and *registration* and *course* joined on *course_id*.

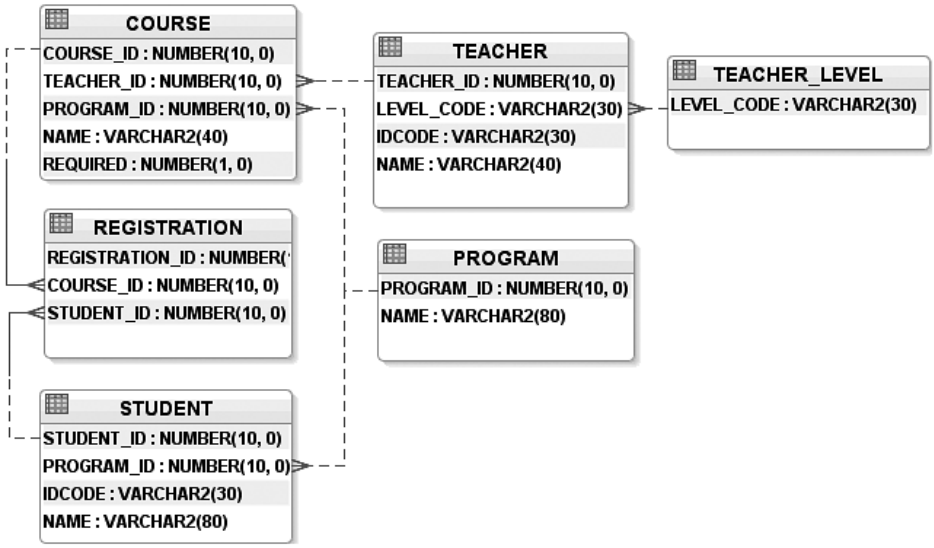

Fig. 2. A sample relational database schema

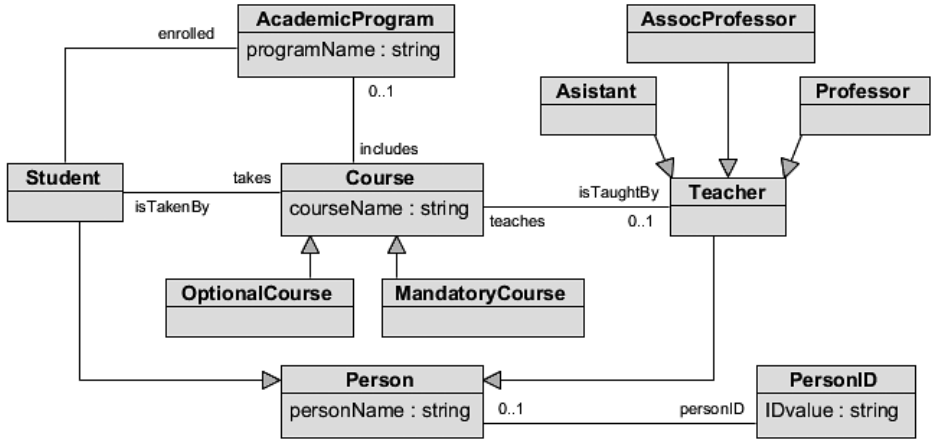

Fig. 3. Sample ontology

Class *personID* instances are populated from s*tudent* and *teacher* tables (*idcode* column).

Classes *Asistant*, *AssocProfessor* and *Professor* all get instance data from one table *teacher* but each has a different filtering. It means that string data 'level_code=…' must be written in *class_map.filter_expr* in corresponding rows.

In the following tables we show the actual data tables of our sample database. This specific data set will be used as an example.

Table 1

Table *program* data

program_id	name
1	Computer Science
2	Computer Engeneering

Table 2

Table *teacher_level* data

level_code
Assistant
Associate Professor
Professor

Table 3

Table *course* data

course_id	name	program_id	teacher_id	Required
1	Programming Basics	2	3	0
2	Semantic Web	1	1	1
3	Computer Networks	2	2	1
4	Quantum Computations	1	2	0

Table 4

Table *student* data

student_id	name	idcode	program_id
1	Dave	123456789	1
2	Eve	987654321	2
3	Charlie	555555555	1
4	Ivan	345453432	2

Table 5

Table *teacher* data

teacher_id	name	idcode	level_code
1	Alice	999999999	Professor
2	Bob	777777777	Professor
3	Charlie	555555555	Assistant

Table 6

Table *registration* data

registration_id	student_id	course_id
1	1	2
2	1	4
3	2	1
4	2	3
5	3	2

Mapping data between these two models inserted in our *mapping DB* is shown in Sub-sections 4.1–4.3 in appropriate places when describing instance generation methods. For our mapping example we have *owl_ontology.ontology_id=1*.

The basic information from OWL ontology (classes, datatype and object properties) is encoded in tables *ontology, owl_class, owl_datatype_property* and *owl_object_property* in an obvious way.

4 OWL Instance Generation

In this section, we describe the instance generation process. it is done by SQL select statements executed in *mapping DB* to generate another SQL select statement that in turn generates RDF triples when executed in *source DB.* In special cases when *table_link* is to be used in generation process more than once procedural language (java + jdbc) can be applied. However, that is not the case in the example discussed in this paper.

4.1 OWL Class Instance Generation

Link between tables *owl_class* and *class_map* is used to generate RDF triples for OWL class instances. Unique URI for these instances are formed concatenating fields *ontology.xl_base, owl_class.rdf_id* and value derived from evaluating the expression described in *id_column_expr* in *source DB.* To generate triples for OWL class instances that are based on real tables (having foreign key from *class_map* to *class2table*) we need to create SQL select statement based on tables *ontology, owl_class, class_map* and *class2table.* If source data comes from database view or temporary table then query needs to be modified (*class2view/class2temp_table* instead of *class2table*).

OWL class mappings are listed in the next table. In the example only the mappings to database tables are used. Data from tables *class_map* and referenced tables *owl_class, class2table* is listed below.

Table 7

OWL class mappings to database tables

class_map_id	OWL class (rdf_id)	table_name	filter_expr	id_column_expr	instance_uri_prefix	generate_instances
1	Teacher	teacher		teacher_id	Teacher	0
2	Assistant	teacher	level_code= 'Assistant'	teacher_id	Teacher	1
3	AssocProfessor	teacher	level_code= 'Associate Professor'	teacher_id	Teacher	1
4	Professor	teacher	level_code= 'Professor'	teacher_id	Teacher	1
5	Student	student		student_id	Student	1
6	Course	course		course_id	Course	0
7	MandatoryCourse	course	required=1	course_id	Course	1
8	OptionalCourse	course	required=0	course_id	Course	1
9	PersonID	teacher		idcode	PersonID	1
10	PersonID	student		idcode	PersonID	1
11	AcademicProgram	program		program_id	Program	1

Most of the class mappings are used for the real OWL class instance generation. There are, however, a few class mappings that are not used in the class instance generation, but which will be further referenced in datatype property mappings.

With an SQL statement it is possible to generate another SQL statement which, when executed in *sample DB*, would generate instance RDF triples. Executing script *OWL_instance_gen.sql* (see Appendix for code) against our sample data, we obtain row set with generated SQL statements, one of which is:

```
  SELECT '<http://lumii.lv/ex#Course'
 || course.course_id || '>' as subject,
 '<type>' as predicate,
 '<lumii#MandatoryCourse>' as object
FROM course
WHERE required=1
```

Executing all generated statements in our sample *source DB* we obtain the following triples, with duplicates removed. The duplicates in the example come from the fact that one *teacher* table row and one *student* table row have the same *idcode* value (the same person being student and teacher at the same time). In Table 8 we use the prefix "lumii" to denote "http://lumii.lv/ex", and the predicate notation "type" to stand for http://www.w3.org/1999/02/22-rdf-syntax-ns#type.

Table 8

Generated OWL class instance RDF triples

Subject	Predicate	Object
< lumii #Course1>	<type>	<lumii#OptionalCourse>
< lumii #Course2>	<type>	<lumii#MandatoryCourse>
< lumii #Course3>	<type>	<lumii#MandatoryCourse>
<lumii#Course4>	<type>	<lumii#OptionalCourse>
<lumii#PersonID123456789>	<type>	<lumii#PersonID>
<lumii#PersonID345453432>	<type>	<lumii#PersonID>
<lumii#PersonID555555555>	<type>	<lumii#PersonID>
<lumii#PersonID777777777>	<type>	<lumii#PersonID>
<lumii#PersonID987654321>	<type>	<lumii#PersonID>
<lumii#PersonID999999999>	<type>	<lumii#PersonID>
<lumii#Program1>	<type>	<lumii#AcademicProgram>
<lumii#Program2>	<type>	<lumii#AcademicProgram>
<lumii#Student1>	<type>	<lumii#Student>
<lumii#Student2>	<type>	<lumii#Student>
<lumii#Student3>	<type>	<lumii#Student>
<lumii#Student4>	<type>	<lumii#Student>
<lumii#Teacher1>	<type>	<lumii#Professor>
<lumii#Teacher2>	<type>	<lumii#Professor>
<lumii#Teacher3>	<type>	<lumii#Assistant>

4.2 OWL Datatype Property Instance Generation

Table *datatype_property_map* allows to specify for each *owl_datatype_propery* several possible value generation mappings, each based on some *class_map*. This linking allows to obtain domain instance URIs for an OWL datatype property. The property range values are obtained from table columns or expressions thereof. In the simplest case when range is to be mapped to column in the same table specified through *datatype_property_map*→*class_map*, we use column *column_expr.* If the property range is mapped to table column (or column expression) in a linked table, we specify the link expression in table *table_link* (the *source_column_expr, target_column_expr* and *mid_table_name* columns encode the linking (table join) conditions). There is no *table_link* usage for OWL datatype property generation in the example.

Table 9 represents data in table *datatype_property_map* and referenced tables *class_map* and *owl_datatype_property* in case when no table link is used (no *table_link* table usage). One can compare the first column in Table 7 and Table 9 below. For example, property *personName* is linked to *class_map_id*=1 and *class_map_id*=5 that correspond to class maps for OWL classes *Teacher* and *Student*. Instances are not directly generated for *Teacher* class *(generate_instances=0)*. Class instances are generated for subclasses *Professor, AsocProfessor* and *Asistant* classes. As *instance_uri_prefix, table_name* and *id_column_expr* have the same value in the class map for superclass (*Teacher* in this case), it enables correct generation of the subject part of triples for OWL datatype properties. There is no need to make class map for each subclass. As to correctness of the mapping, the class map to super class should have the same filtering as union of all subclasses. In the case of *Teacher* it has no filter (*filter_expr* is empty for *class_map_id*=1) but filters for sub-class maps *(class_map_id:2,3,4)* are *level_code='Assistant', level_code='Associate Professor'* and *level_code='Professor'*. All these together produce all *teacher* rows and *Teacher* class map with no filtering corresponding to the same row set.

Table 9

OWL datatype property class mappings to database table column expressions (data from tables *datatype_property_map* and referenced *class_map, class2table* and *owl_datatype_property*)

class_map_id	OWL_datatype_property	table_name	column_expr	filter_expr
6	courseName	Course	name	
11	programName	Program	name	
1	personName	Teacher	name	
5	personName	Student	name	
9	IDValue	Teacher	idcode	
10	IDValue	Student	idcode	

Executing script *generate_sql4datatype_props.sql* (see Appendix for the code) against our sample data, we obtain row set with generated SQL statements, one of which is:

```
  SELECT
 '<lumii#optionalCourse'
 || course.course_id || '>' as subject,
 '<lumii#courseName>' as predicate,
 name as object
FROM course
WHERE required=0
```

Executing all generated statements in our sample *source DB*, we obtain the following triples, duplicates removed (note the abbreviations, as in Table 8).

Table 10

Generated OWL datatype property instance RDF triples

Subject	Predicate	Object
<lumii#Course1>	<lumii#courseName>	Programming Basics
<lumii#Course2>	<lumii#courseName>	Semantic Web
<lumii#Course3>	<lumii#courseName>	Computer Networks
<lumii#Course4>	<lumii#courseName>	Quantum Computations
<lumii#PersonID123456789>	<lumii#IDValue>	123456789
<lumii#PersonID345453432>	<lumii#IDValue>	345453432
<lumii#PersonID555555555>	<lumii#IDValue>	555555555
<lumii#PersonID777777777>	<lumii#IDValue>	777777777
<lumii#PersonID987654321>	<lumii#IDValue>	987654321
<lumii#PersonID999999999>	<lumii#IDValue>	999999999
<lumii#Student1>	<lumii#personName>	Dave
<lumii#Student2>	<lumii#personName>	Eve
<lumii#Student3>	<lumii#personName>	Charlie
<lumii#Student4>	<lumii#personName>	Ivan
<lumii#Teacher1>	<lumii#personName>	Alice
<lumii#Teacher2>	<lumii#personName>	Bob
<lumii#Teacher3>	<lumii#personName>	Charlie
<lumii#Program1>	<lumii#programName>	Computer Science
<lumii#Program2>	<lumii#programName>	Computer Engeneering

4.3 OWL Object Property Instance Generation

Rows in *object_property_map* specify how to generate instances for OWL object properties. The references to *class_map* through foreign keys *domain_class_map* and *range_class_map* determine *source DB* tables for subject and object of generated triples for property instances. The *class_map* rows in column *filter_expr* determine filtering on these tables. These tables are joined by column expressions specified in *source_column_expr* and *target_column_expr.* They are joined directly or by using one or more intermediate table joining steps. The latter require usage of one or more rows in *table_link.*

First we shall discuss direct joining of domain table to range table without *table_link.* An OWL datatype property can have several mappings – several rows in *object_property_map.* In this case the triple generation will process all of them. In the process of

triple generation for OWL object properties (also for OWL datatype properties) the *class_map* column *generate_instances* is not used. This field is only for OWL class instance generation. URI for subject and object part of triple are determined by concatenation of *ontology.xml_base, class_map.instance_uri_prefix* and evaluation of *id_column_expr* in *source DB.* URI of predicate part of generated triples are determined by concatenation of *ontology.xml_base* and *owl_object_property.rdf_id*. Table 11 represents data from *object_property_map*, and referenced *owl_object_property*, as well as two *class_map* rows for subject and object and corresponding *class2table* rows. See Table 7 for more details on referenced *class_map* rows.

Table 11

Owl object property mappings to database tables pairs for domain and range

class_ map_id (domain)	class_ map_id (range)	object_ property	table_ name (domain)	table_ name (range)	source_ col_expr	target_ col_expr
11	6	includes	program	course	program_id	program_id
5	10	personID	student	student	student_id	student_id
1	9	personID	teacher	teacher	teacher_id	teacher_id
5	11	enrolled	student	program	program_id	program_id
1	6	teaches	teacher	course	teacher_id	teacher_id

Reading the data, we can see that OWL object properties generally map to table pairs corresponding to domain and range class pair. *PersonID* object property is an exception because it has *Person* class as domain and *PersonID* class as range and both these classes have mappings to 2 tables: *student* and *teacher.* For this property two mappings exist (*object_property_map* rows), one of which maps *student* table for domain to *student* table for range. The mapping is based on *student_id* column (*source_column_expr*, *target_column_expr*). The second row maps *teacher* table to *teacher* table based on *teacher_id* column in a similar way.

To generate RDF triples for OWL object property instances the data represented in Table 11 above can be used. A framework of SQL for main information retrieval for generation process is as follows.

```
  SELECT
 <domain_table>_1.<domain_class_map_id→class_map.id_column_expr>,
 <range_table>_2.<range_class_map_id→class_map.id_column_expr>
FROM <domain_table> AS <domain_table>_1
 INNER JOIN <range_table> AS <range_table>_2
 ON <domain_table>_1.<domain_column_expr>
 = <range_table>_2.<range_column_expr>
```

The suffixes _1 and _2 are added here to prevent name collision. For example, in the case of mapping for *PersonID* property (for *student*) query joins *student* table to itself because *object_property_map* table specifies two tables via *domain_class_map* and *range_class_map* although the tables are the same.

```
  SELECT student_1.student_id, student_1.program_id
FROM student AS student_1
INNER JOIN student AS student_2
ON student_1.student_id = student_2.student_id
```

For *enrolled* property the query is

```
  SELECT student_1.student_id, program_2.program_id
FROM student AS student_1
INNER JOIN program AS program_2
ON student_1.program_id = program_2.program_id
```

An SQL script for OWL object property instance generation can be defined in a similar way as it was done for OWL class and OWL datatype property instance generation.

Executing script *generate_sql4object_props.sql* (see Appendix for code) against our sample data, we produced row set with generated SQL statements, one of which was:

```
  SELECT
 '<lumii#Program' || program_1.program_id || '>' as subject,
 '<lumii#includes>' as predicate,
 '<lumii#Course' || course_2.course_id || '>' as object
FROM program program_1 INNER JOIN course course_2
 ON program_1.program_id = course_2.program_id
WHERE 1=1 AND 1=1
```

Executing all generated statements in our sample *source DB*, we produced the following triples (note the abbreviations, as in Table 8).

Table 12

Generated OWL object property instance RDF triples

Subject	Predicate	Object
<lumii#Student1>	<lumii#enrolled>	<lumii#Program1>
<lumii#Student2>	<lumii#enrolled>	<lumii#Program2>
<lumii#Student3>	<lumii#enrolled>	<lumii#Program1>
<lumii#Student4>	<lumii#enrolled>	<lumii#Program2>
<lumii#Program1>	<lumii#includes>	<lumii#Course4>
<lumii#Program1>	<lumii#includes>	<lumii#Course2>
<lumii#Program2>	<lumii#includes>	<lumii#Course1>
<lumii#Program2>	<lumii#includes>	<lumii#Course3>
<lumii#Student1>	<lumii#personID>	<lumii#PersonID123456789>
<lumii#Student2>	<lumii#personID>	<lumii#PersonID987654321>
<lumii#Student3>	<lumii#personID>	<lumii#PersonID555555555>
<lumii#Student4>	<lumii#personID>	<lumii#PersonID345453432>
<lumii#Teacher1>	<lumii#personID>	<lumii#PersonID999999999>
<lumii#Teacher2>	<lumii#personID>	<lumii#PersonID777777777>
<lumii#Teacher3>	<lumii#personID>	<lumii#PersonID555555555>
<lumii#Teacher1>	<lumii#teaches>	<lumii#Course2>
<lumii#Teacher2>	<lumii#teaches>	<lumii#Course3>
<lumii#Teacher2>	<lumii#teaches>	<lumii#Course4>
<lumii#Teacher3>	<lumii#teaches>	<lumii#Course1>

Now we shall discuss the table link usage. It is required for instance generation of OWL object property *takes* which is between *Student* and *Course* OWL classes and requires to join tables *student* and *course* through *registration*. Table *object_property_*

map links to *class_map* two rows for subject and object through *domain_class_map_id* and *range_class_map_id* foreign keys. That produces pair of two relations (tables). To join these tables *source_column_expr* and *target_column_expr* are used. If these tables, cannot be joined directly, then *table_link* table is to be used. It stores information about middle steps in table traversing. To support joining table *t1* with *t2* through middle table, the *table_link* columns has these meanings:

mid_table_name- table name in the middle step,

source_column_expr- joins <*mid_table_name*> table to *t1* by this column expr.,

target_column_expr- joins <*mid_table_name*> table to *t2* by this column expr.,

filter_expr- additional filter expression on table <*mid_table_name*>,

next_table_link_id- foreign key to the same table to implement more intermediate steps if needed (t1→mid_table_1→mid_table_2 → … →mid_table_n→t2).

Table 13 and Table 14 represent OWL object property mapping data for properties that need table links (*object_property_map.table_link* is not null). Data comes from tables *owl_object_property, object_property_map* as well as their referenced table rows. the corresponding *table_link* data follows. *Filter_expr* is not used in the example.

Table 13

Owl object property mappings to database tables pairs for domain and range when table link is used

class_ map_id (domain)	class_ map_id (range)	object_ property	table_ name (domain)	table_ name (range)	source_ column_ expr	target_ column_ expr
5	6	takes	student	Course	student_id	course_id

Table 14

***Table_link* table data**

mid_table_name	source_column_expr	target_column_expr	next_table_link_id
registration	student_id	course_id	

The join condition is:

```
  <domain_table>.<source_column_expr>=
<mid_table_name>.<table_link.source_column_expr>
AND
<mid_table_name>.<table_link.target_column_expr>=
<range_table>.<target_column_expr>
```

In this case the exact condition is:

```
  student.student_id=registration.student_id
AND
registration.course_id=course.course_id
```

Executing script *generate_sql4object_props_table_links.sql* (see Appendix for the code) against our sample data, we obtain row set with generated SQL statements, one of which is:

```
  SELECT
'<lumii#Student' || student_1.student_id || '>' as subject,
'<lumii#takes>' as predicate,
'<lumii#Course' || course_2.course_id || '>' as object
 FROM student student_1
INNER JOIN registration registration_3
 ON student_1.student_id = registration_3.student_id
INNER JOIN course course_2
 ON registration_3.course_id = course_2.course_id
WHERE 1=1 AND 1=1 AND 1=1 AND 1=1
```

Executing it in sample *source DB* we get the following triples.

Table 15

Generated OWL object property instance RDF triples when *table_link* table used

Subject	Predicate	Object
<lumii#Student1>	<lumii#takes>	<lumii#Course2>
<lumii#Student2>	<lumii#takes>	<lumii#Course4>
<lumii#Student3>	<lumii#takes>	<lumii#Course1>
<lumii#Student4>	<lumii#takes>	<lumii#Course3>
<lumii#Student5>	<lumii#takes>	<lumii#Course2>

4.4 The result of RDF Triple Generation

When all generated SQLs were executed in our example database, we produced the following triple set, essentially being data export from original relational database to RDF format for target OWL ontology. Following the data is a union of data in Table 8, 10, 12 and 15 with shorthands “lumii” and “type” expanded.

```
<http://lumii.lv/ex#Course1>  <http://lumii.lv/ex#courseName>  Programming Basics
<http://lumii.lv/ex#Course2>  <http://lumii.lv/ex#courseName>  Semantic Web
<http://lumii.lv/ex#Course3>  <http://lumii.lv/ex#courseName>  Computer Networks
<http://lumii.lv/ex#Course4>  <http://lumii.lv/ex#courseName>  Quantum Computations
<http://lumii.lv/ex#Student1> <http://lumii.lv/ex#enrolled> <http://lumii.lv/ex#Program1>
<http://lumii.lv/ex#Student2> <http://lumii.lv/ex#enrolled> <http://lumii.lv/ex#Program2>
<http://lumii.lv/ex#Student3> <http://lumii.lv/ex#enrolled> <http://lumii.lv/ex#Program1>
<http://lumii.lv/ex#Student4> <http://lumii.lv/ex#enrolled> <http://lumii.lv/ex#Program2>
<http://lumii.lv/ex#PersonID123456789><http://lumii.lv/ex#IDValue> 123456789
<http://lumii.lv/ex#PersonID345453432><http://lumii.lv/ex#IDValue> 345453432
<http://lumii.lv/ex#PersonID555555555><http://lumii.lv/ex#IDValue> 555555555
<http://lumii.lv/ex#PersonID777777777><http://lumii.lv/ex#IDValue> 777777777
<http://lumii.lv/ex#PersonID987654321><http://lumii.lv/ex#IDValue> 987654321
<http://lumii.lv/ex#PersonID999999999><http://lumii.lv/ex#IDValue> 999999999
<http://lumii.lv/ex#Program1> <http://lumii.lv/ex#includes> <http://lumii.lv/ex#Course2>
<http://lumii.lv/ex#Program1> <http://lumii.lv/ex#includes> <http://lumii.lv/ex#Course4>
<http://lumii.lv/ex#Program2> <http://lumii.lv/ex#includes> <http://lumii.lv/ex#Course3>
<http://lumii.lv/ex#Program2> <http://lumii.lv/ex#includes> <http://lumii.lv/ex#Course1>
<http://lumii.lv/ex#Student1> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID123456789>
<http://lumii.lv/ex#Student2> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID987654321>
<http://lumii.lv/ex#Student3> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID555555555>
<http://lumii.lv/ex#Student4> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID345453432>
<http://lumii.lv/ex#Teacher1> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID999999999>
<http://lumii.lv/ex#Teacher2> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID777777777>
<http://lumii.lv/ex#Teacher3> <http://lumii.lv/ex#personID>  <http://lumii.lv/ex#PersonID555555555>
<http://lumii.lv/ex#Student1> <http://lumii.lv/ex#personName> Dave
<http://lumii.lv/ex#Student2> <http://lumii.lv/ex#personName> Eve
```

<http://lumii.lv/ex#Student3>	<http://lumii.lv/ex#personName>	Charlie
<http://lumii.lv/ex#Student4>	<http://lumii.lv/ex#personName>	Ivan
<http://lumii.lv/ex#Teacher1>	<http://lumii.lv/ex#personName>	Alice
<http://lumii.lv/ex#Teacher2>	<http://lumii.lv/ex#personName>	Bob
<http://lumii.lv/ex#Teacher3>	<http://lumii.lv/ex#personName>	Charlie
<http://lumii.lv/ex#Program1>	<http://lumii.lv/ex#programName>	Computer Science
<http://lumii.lv/ex#Program2>	<http://lumii.lv/ex#programName>	Computer Engeneering
<http://lumii.lv/ex#Student1>	<http://lumii.lv/ex#takes>	<http://lumii.lv/ex#Course4>
<http://lumii.lv/ex#Student1>	<http://lumii.lv/ex#takes>	<http://lumii.lv/ex#Course2>
<http://lumii.lv/ex#Student2>	<http://lumii.lv/ex#takes>	<http://lumii.lv/ex#Course3>
<http://lumii.lv/ex#Student2>	<http://lumii.lv/ex#takes>	<http://lumii.lv/ex#Course1>
<http://lumii.lv/ex#Student3>	<http://lumii.lv/ex#takes>	<http://lumii.lv/ex#Course2>
<http://lumii.lv/ex#Teacher1>	<http://lumii.lv/ex#teaches>	<http://lumii.lv/ex#Course2>
<http://lumii.lv/ex#Teacher2>	<http://lumii.lv/ex#teaches>	<http://lumii.lv/ex#Course3>
<http://lumii.lv/ex#Teacher2>	<http://lumii.lv/ex#teaches>	<http://lumii.lv/ex#Course4>
<http://lumii.lv/ex#Teacher3>	<http://lumii.lv/ex#teaches>	<http://lumii.lv/ex#Course1>
<http://lumii.lv/ex#Course1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#OptionalCourse>
<http://lumii.lv/ex#Course2>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#MandatoryCourse>
<http://lumii.lv/ex#Course3>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#MandatoryCourse>
<http://lumii.lv/ex#Course4>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#OptionalCourse>
<http://lumii.lv/ex#PersonID123456789>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#PersonID345453432>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#PersonID555555555>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#PersonID777777777>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#PersonID987654321>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#PersonID999999999>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#PersonID>
<http://lumii.lv/ex#Program1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#AcademicProgram>
<http://lumii.lv/ex#Program2>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#AcademicProgram>
<http://lumii.lv/ex#Student1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Student>
<http://lumii.lv/ex#Student2>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Student>
<http://lumii.lv/ex#Student3>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Student>
<http://lumii.lv/ex#Student4>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Student>
<http://lumii.lv/ex#Teacher1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Professor>
<http://lumii.lv/ex#Teacher2>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Professor>
<http://lumii.lv/ex#Teacher3>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://lumii.lv/ex#Assistant>

We note that the RDF triple instances obtained here are the same, as obtained in [8] by a manual translation SQL definition, or by D2RQ [2] mapping definition approach. In our case the manual user work needed for the triple creation consists of filling in the appropriate data in the tables *class_map*, *object_property_map* and *datatype_property_map*, as well as *table_link* (see the data in Tables 7, 9, 11, 13, 14). If compared to D2RQ solution of the same instance generation problem, as provided in [8], we note that we have provided a more compact representation of input data by the user since a D2RQ mapping cannot be made aware of the subclass relation in the target ontology. In the example containing a 6-fold specification of instance generation for object property 'teaches' (3 subclasses of 'Teacher' times 2 subclasses of 'Course') a tremendous increase of data volume occurs in case of ontologies with large subclass hierarchies (the instance generation for object property 'teaches' is defined here as a single row in Table 10). If compared to D2RQ [2] approach our method requires no custom SQL writing for mapping definitions, except to define custom views in *class2view* which has not been necessary in our example of *source DB.*

5 Conclusions

In this paper we have demonstrated an example of how relational database itself can be used to create mapping between a source relational database (legacy type) and

target OWL ontology and to generate RDF triples for instance data. The work is still in progress, which means new use cases are studied and the mapping schema is being continuously improved. Next step in our research is to study possibilities for SPARQL to SQL translation in correspondence to the defined mapping.

We plan to apply the current functionality to transform relational data to RDF format in real life medical database [9, 10]. Although our RDB to OWL mapping specification format and implementation can be used together with different end-user mapping specification languages, we are working to define a language that would allow defining the correspondence between target ontology and its corresponding RDB schema elements in a user friendly way.

I would like to thank Karlis Cerans at the Institue of Mathematics and Computer Science, the University of Latvia, for his support and assistance.

References

1. J. Barrasa, A. Gómez-Pérez. Upgrading relational legacy data to the semantic web. In: *Proc. of the 15th International World Wide Web Conference (WWW 2006)*, Edinburgh, United Kingdom, 23–26 May 2006, pp. 1069–1070.
2. D2RQ Platform. Available: http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/.
3. C. Blakeley. RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping). OpenLink Software, 2007.
4. OpenLink Virtuoso Platform. Automated Generation of RDF Views over Relational Data Sources. Available: http://docs.openlinksw.com/virtuoso/rdfrdfviewgnr.html.
5. W. Hu, Y. Qu. Discovering Simple Mappings between Relational Database Schemas and Ontologies. In: *Proc. of the 6th International Semantic Web Conference (ISWC 2007), 2nd Asian Semantic Web Conference (ASWC 2007)*, Busan, Korea, 11–15 November 2007, LNCS, 4825, pp. 225–238.
6. http://www.w3.org/2005/Incubator/rdb2rdf/.
7. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.
8. G. Barzdins, J. Barzdins, K. Cerans. From Databases to Ontologies. Semantic Web Engineering in the Knowledge Society. In: J. Cardoso, M. Lytras (eds.), *IGI Global*, 2008, pp. 242–266. ISBN: 978-1-60566-112-4
9. G. Barzdins, S. Rikacovs, M. Veilande, M. Zviedris. Ontological Re-engineering of Medical Databases. *Proceedings of the Latvian Academy of Sciences*, Section B, Vol. 63, No. 4/5 (663/664), 2009, pp. 20–30.
10. G. Barzdins, E. Liepins, M. Veilande, M. Zviedris. Semantic Latvia Approach in the Medical Domain. In: H. M. Haav, A. Kalja, *Proceedings of the 8th International Baltic Conference on Databases and Information Systems*. Tallinn University of Technology Press, 2008, pp. 89–102.
11. J. Barzdins, G. Barzdins, R. Balodis, K. Cerans et al. Towards Semantic Latvia. In: *Proccedings of the 7th International Baltic Conference on Databases and Information Systems*, 2006, pp. 203–218.
12. Object Management Group MOF QVT Final Adopted Specification. Available: http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf.
13. ATLAS Model Transformation Language. Available: http://www.eclipse.org/m2m/atl/.
14. MOLA resources. Available: http://mola.mii.lu.lv/.
15. Eclipse Modeling Framework Project (EMF). Available: http://www.eclipse.org/modeling/emf/.

Appendix

We provide listings of SQL scripts which, when executed in *mapping DB*, generate SQL scripts which, in turn, when executed in *source DB*, generate RDF triples for instances of OWL classes, OWL datatype properties, and OWL object properties.

They are not easily readable as two SQL levels are mixed. They show that mere SQL statement can do the task. They generate SQL statements using string concatenation function || as it is in Oracle DB. They also use Oracle DB functions NVL (null value replacement), NVL2 (return value depends on parameter being null or not null). It is easy to rewrite these SQLs for another DB if needed.

SQL script ***OWL_instance_gen.sql*** that generates SQL statement for RDF triple generation for OWL class instances

```
  SELECT 'SELECT '
 || '''<' || o.xml_base || cm.instance_uri_prefix || ''''
 || ' || '     || c2t.table_name
 || '.' || cm.id_column_expr || ' || ''>'' as subject'
 || ',''' || '< http://www.w3.org/1999/02/22-rdf-syntax-ns#type >''
 as predicate'
 || ',''' || '<' ||o.xml_base || c.rdf_id || '>'' as object'
 || ' FROM ' || c2t.table_name
 || NVL2(cm.filter_expr,' WHERE ', '') || cm.filter_expr as sql4rdf
FROM ontology o,       owl_class c, class_map cm, class2table c2t
WHERE o.ontology_id = c.ontology_id AND
 c.owl_class_id = cm.owl_class_id AND
 cm.class2table_id = c2t.class2table_id AND
 o.ontology_id=1 AND   cm.generate_instances=1
```

SQL script ***generate_sql4datatype_props.sql*** that generates SQL statements for RDF triple generation for OWL datatype property instances

```
  SELECT 'SELECT '
 || '''<' || o.xml_base || cm.instance_uri_prefix || ''''
 || ' || ' || c2t.table_name
 || '.' || cm.id_column_expr || ' || ''>'' as subject'
 || ',''' || '<' || o.xml_base || dp.rdf_id || '>'' as predicate'
 || ',' || dpm.column_expr || ' as object'
 || ' FROM ' || c2t.table_name
 || NVL2(cm.filter_expr,' WHERE ', '') || cm.filter_expr
FROM
 owl_datatype_property dp, datatype_property_map dpm,
 class_map cm, class2table c2t, ontology o
WHERE dp.owl_datatype_property_id=dpm.owl_datatype_property_id AND
 dpm.class_map_id = cm.class_map_id AND
 cm.class2table_id = c2t.class2table_id AND
 dp.ontology_id = o.ontology_id AND o.ontology_id=1
```

SQL script ***generate_sql4object_props.sql*** that generates SQL statements for RDF triple generation for OWL object property instances without intermediate table link usage

```
  SELECT
 'SELECT '
 || '''<' || o.xml_base || cm_domain.instance_uri_prefix || ''''
 || ' || ' || c2t_domain.table_name || '_1'
 || '.' || cm_domain.id_column_expr || ' || ''>'' as subject'
 || ',''' || '<'|| o.xml_base || op.rdf_id || '>'' as predicate'
 || ','
```

```
   || '''<' || o.xml_base || cm_range.instance_uri_prefix || ''''
 || ' || ' || c2t_range.table_name || '_2'
 || '.' || cm_range.id_column_expr || ' || ''>'' as object'

   || ' FROM '
 || c2t_domain.table_name || ' ' || c2t_domain.table_name || '_1'
 || ' INNER JOIN '
 || c2t_range.table_name || ' ' || c2t_range.table_name || '_2'
 || ' ON ' || c2t_domain.table_name
 || '_1.' || opm.source_column_expr
 || ' = ' || c2t_range.table_name || '_2.' || opm.target_column_expr
 || ' WHERE ' || NVL(cm_domain.filter_expr ,' 1=1 ')
 || 'AND ' || NVL(cm_range.filter_expr , ' 1=1 ')
 AS generated_SQL
FROM
 owl_object_property op,
 ontology o,
 object_property_map opm,
 class_map cm_domain,
 class_map cm_range,
 class2table c2t_domain,
 class2table c2t_range
WHERE
 op.ontology_id=o.ontology_id AND
 op.owl_object_property_id=opm.owl_object_property_id AND
 opm.domain_class_map_id =cm_domain.class_map_id AND
 opm.range_class_map_id =cm_range.class_map_id AND
 cm_domain.class2table_id=c2t_domain.class2table_id AND
 cm_range.class2table_id=c2t_range.class2table_id AND
 opm.table_link_id IS NULL AND op.ontology_id=1
ORDER BY 1
```

SQL script ***generate_sql4object_props_table_links.sql*** that generates SQL statements for RDF triple generation for OWL object property instances with one intermediate table link usage

```
  SELECT
 'SELECT '
 || '''<' || o.xml_base
 || cm_domain.instance_uri_prefix || ''''
 || ' || ' || c2t_domain.table_name || '_1'
 || '.' || cm_domain.id_column_expr || ' || ''>'' as subject'

 || ','''' || '<'|| o.xml_base || op.rdf_id || '>'' as predicate'

 || ',' || '''<' || o.xml_base
 || cm_range.instance_uri_prefix || ''''
 || ' || ' || c2t_range.table_name || '_2'
 || '.' || cm_range.id_column_expr || ' || ''>'' as object'

 || ' FROM '
 || c2t_domain.table_name || ' ' || c2t_domain.table_name || '_1'
 || ' INNER JOIN '
 || tl.mid_table_name || ' ' || tl.mid_table_name || '_3'
 || ' ON ' || c2t_domain.table_name || '_1. '
```

```
 || opm.source_column_expr
 || ' = ' || tl.mid_table_name || '_3.' || tl.source_column_expr
 || ' INNER JOIN '
 || c2t_range.table_name || ' ' || c2t_range.table_name || '_2'
 || ' ON ' || tl.mid_table_name || '_3.' || tl.target_column_expr
 || ' = ' || c2t_range.table_name || '_2.' || opm.target_column_expr
 || ' WHERE ' || NVL(cm_domain.filter_expr ,' 1=1 ')
 || 'AND ' || NVL(cm_range.filter_expr , ' 1=1 ')
 || 'AND ' || NVL(tl.filter_expr , ' 1=1 ')
 || 'AND ' || NVL(tl.filter_expr , ' 1=1 ')
 AS generated_SQL
FROM
 owl_object_property op,      ontology o,
 object_property_map opm,     class_map cm_domain,
 class_map cm_range,                        class2table c2t_domain,
 class2table c2t_range,                     table_link tl
WHERE
 op.ontology_id=o.ontology_id AND
 op.owl_object_property_id=opm.owl_object_property_id AND
 opm.domain_class_map_id =cm_domain.class_map_id AND
 opm.range_class_map_id =cm_range.class_map_id AND
 cm_domain.class2table_id=c2t_domain.class2table_id AND
 cm_range.class2table_id=c2t_range.class2table_id AND
 opm.table_link_id=tl.table_link_id AND
 opm.table_link_id IS NOT NULL AND op.ontology_id=1
```