HOW TO SOLVE SPATIAL PROBLEMS USING LINKED DATA WITH ORACE: THE CASE OF PLANNING A SHOPPING CENTER IN DELFT.

A thesis submitted to the Delft University of Technology in partial fulfillment
of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Evangelos Theocharous

April 2019

ISBN 999–99–9999–999–9

The work in this thesis was made in the:

Department of OTB - Research for the built environment
Faculty of Architecture & the Built Environment
Delft University of Technology

Supervisors:	Prof.dr.ir. P.J.M. van Oosterom
	Drs. C.W. (Wilko) Quak
	Drs. M.E. de Vries
Co-reader:	Dr.ir. F.M. Welle Donker

ABSTRACT

The promotion of the data re-usability was the main motivation behind the conception of linked-data. Spatial data can also be expressed as linked-data. This thesis explores different approaches to utilize spatial data in linked-data format, in order to determine if the technologies have matured enough to assist solving spatial problems. More precisely, it attempts to select a position to build a new shopping center in Delft, Netherlands by using relational data and linked-data. Five different Linked-data implementations are identified and examined; two of them, a Geographic SPARQL Protocol and Resource Description Framework (RDF) Query Language (a recursive acronym for GeoSPARQL) (GeoSPARQL) endpoint and an API, are offered from Publieke Dienstverlening Op de Kaart (PDOK). The other three approaches are designed using Oracle Spatial and Graph. From the latter three, one approach utilizes linked-data stored in a local database while the rest utilize relational data mapped into linked-data. All five approaches are assessed for GeoSPARQL compliance and are compared to a QGIS solution. From the aforementioned approaches, only the two mapping ones have achieved the desired solution. However, none of them fully complies with GeoSPARQL specification hence, further research is necessary.

ACKNOWLEDGEMENTS

I would first like to thank my thesis supervisors, Professor Peter van Oosterom and Drs. Wilko Quak. The door to their office was always open whenever I ran into a problem or had a question about my research. They steered me in the right direction and they made sure I can get all the necessary help I needed, without preventing me from making this research my own.

I would also like to thank Drs. Marianne de Vries who was involved in setting Oracle database and Oracle Spatial and Graph. Without her dedication and her technical knowledge, this graduation project could not have been successful.

I would also like to acknowledge Dr. Frederika Welle Donker as the co-reader of this thesis, for her very valuable comments.

I would also like to thank Dr. Linda van den Brink for dedicating her valuable time to help me during the initial steps of my research.

I would also like to thank Dr. Erwin Folmer from Kadaster for his input. Without his ideas, his quick responses and the support with data it would have been very difficult to reach the end of this research.

I would also like to thank Mr. Matthew Perry and Mr. Souripriya Das for providing their expert knowledge on Oracle and Oracle Spatial and Graph. They took a deep look into my thesis and helped me solve many technical problems.

Finally, I must express my very profound gratitude to my parents and to my big brother Domenico and my girlfriend Eleni for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.
Thank you. Evangelos Theocharous

CONTENTS

LIST OF FIGURES

LIST OF CODE SNIPPETS

LIST OF TABLES

ACRONYMS

1D 1-Dimension . . . 27
API Application Program Interface . . . 1
b. byte . . . 28
BAG Basisregistratie Adressen en Gebouwen . . . 13
BRK Basisregistratie Kadaster . . . 18
BRT Basisregistratie Topografie . . . 18
CLOB Character Large Object . . . 28
CRS Coordinate reference system . . . 22
CSV Comma-Separated Values . . . 20
DAWG Data Access Working Group . . . 8
dbms database management system . . . ix
EPSG European Petroleum Survey Group . . . 22
EU European Union . . . 1
GB Gigabyte . . . 28
GeoSPARQL Geographic SPARQL Protocol and RDF Query Language (a recursive acronym for GeoSPARQL) . . . iii
GIS Geographic Information System . . . xv
GML Geography Markup Language . . . 3
http Hypertext Transfer Protocol . . . 31
https Hypertext Transfer Protocol Secure . . . 31
IRI Internationalized Resource Identifier . . . 33
m meter . . . 23
OBDA Ontology-based Data Access . . . 11
OGC Geospatial Consortium . . . 1
OWL Web Ontology Language . . . 5
PDOK Publieke Dienstverlening Op de Kaart . . . iii
QGIS Quantum Geographic Information System (GIS) . . . 18
R2RML Relational Data Base (RDB) to RDF Mapping Language . . . 2
RDB Relational Data Base . . . xv
RDF Resource Description Framework . . . iii
RDFS RDF Schema . . . 5
REST REpresentational State Transfer . . . 1
RIF Rule Interchange Format . . . 2
RQ Research Question . . . 4
SDI Spatial Data Infrastructure . . . 1
SPARQL SPARQL Protocol and RDF Query Language (a recursive acronym for SPARQL) . . . 1
SQL Structured Query Language . . . 11
SQR Sub-Research Question . . . 4

1 INTRODUCTION

The re-use of public government data is promoted by European Union (EU), in order to increase transparency in government and the growth of information industry, through the implementation of "Directive on reuse of public sector information", also known as the "PSI Directive" (Directive 2003/98/EC) [Kulk and van Loenen, 2012]. Additionally, EU developed the INSPIRE Directive (Directive 2007/2/EC) in May 2007, that establishes an infrastructure for spatial information in Europe [INSPIRE, 2007]. Such an infrastructure enables exchange, sharing, access and use of inter-operable spatial data and spatial data services across the various levels of public authority and across different sectors and it is known as Spatial Data Infrastructure (SDI) [INSPIRE, 2007].

Along with EU, another attempt for publishing and reusing data began, with a completely different background and aim. Tim Berners-Lee, published his idea about the Semantic Web [Berners-Lee et al., 2006]. According to World Wide Web Consortium (W3C), "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries".The Semantic Web, however, is more about creating machine and human readable links between published data that help explore the web of data. Those data along with the links are called linked data.

Linked data can be accessed using SPARQL Protocol and RDF Query Language (a recursive acronym for SPARQL) (SPARQL) and a SPARQL endpoint. An endpoint is "the entry point to a service, a process, or a queue or topic destination in service-oriented architecture" according to WIkipedia [2019], in other words an endpoint can allow accessing data and services from a distant source. Although SPARQL can handle all kinds of data, expressing spatial data as linked data has limited spatial functionality. Hence, Open Geospatial Consortium (OGC) developed GeoSPARQL, which supports spatial reasoning and querying using SPARQL.

The idea of this thesis is to explore if the linked data approaches are mature enough to support spatial problem solving. More specific, this thesis will explore the different linked data approaches to try to combine data from different data sources and solve a spatial problem, by pinpointing a possible position for a future shopping center in Delft. To that use-case, the available solutions are:

- PDOK SPARQL endpoint,
- PDOK's REpresentational State Transfer (REST)ful Application Program Interface (API) and
- Own GeoSPARQL Implementation.

According to GeoSPARQL's Wikipedia web-page "there is (almost) no complete implementation of GeoSPARQL, but some partial or vendor implementations". Namely:

1. Apache Marmotta
2. Parliament
3. Strabon
4. OpenSahara uSeekM IndexingSail Sesame Sail plugin
5. GraphDB

6. Stardog

Those implementations, contain some or all GeoSPARQL classes, properties and extensions. Oracle Spatial and Graph, since version 12c, supports the full implementation of GeoSPARQL [Abhilakh Missier, 2015]. Since, Oracle Spatial and Graph is currently the only complete implementation of GeoSPARQL, this thesis will attempt to solve the use-case spatial problem utilizing this vendor, not only because it supports GeoSPARQL specification, but because it supports an important feature, instead of storing the same data two times, as relational data and linked data, it is possible to store them once as relational data and using RDF Views to query them as linked data which brings out many advantages. Initially, the unnecessary redundancy is avoided as they are only stored once, in relational data form. At the same time those data are easier accessed and re-shared, explored and extended as linked data. Finally, through the connection of different data-sets it is possible to inference new knowledge. This feature will bring a combination of different technologies, databases and linked data.

1.1 SCOPE OF RESEARCH

As already mentioned, this thesis will explore the possible linked data approaches available, in order to solve the aforementioned spatial problem of positioning a shopping center in Delft, Netherlands. The exploration is going to focus on the resulting proposed geometries, the accuracy of the resulting geometries, the degree at which the solution follows the GeoSPARQL specification and the efficiency of solution. The efficiency will be measured in terms of query-running speed and storage requirements. More specifically, before the available approaches are investigated, the criteria for the positioning a shopping center will be determined and additionally the same spatial problem will be solved using GIS so that the solution is known beforehand. After the solution is known, all other approaches will attempt to reproduce it, using exactly the same criteria. The different approaches that will be explored are already existing linked data approaches:

- PDOK's GeoSPARQL Endpoint
- PDOK's API

And approaches designed in Oracle Spatial and Graph which are:

- linked data solution
- Direct mapping solution
- RDB to RDF Mapping Language (R2RML) solution

For each solution, the number of geometries and the similarity with the proposed solution will be compared. Moreover, each approach will be judged based on the degree at which it follows GeoSPARQL implementation, i.e. at which extend it supports the GeoSPARQL conformance classes. *Core* conformance class defines the classes and properties for representing geospatial data. *Topology Vocabulary extension* conformance class defines the vocabulary to assert and query topological relations between spatial objects. *Geometry extension* conformance class defines the vocabulary to assert and query information about geometry data and defines non-topological query functions for operating on geometry data. *Geometry Topology extension* conformance class defines a set of topological query functions that operate on geometry literals. *RDFS Entailment extension* conformance class defines a mechanism for matching implicitly derived RDF triples in GeoSPARQL queries. Finally, *Query Rewrite extension* conformance class defines a set of Rule Interchange Format (RIF) rules that that use topological extension functions to establish the existence of direct topological predicates

[Perry and Herring, 2012]. In other words, *Topology Vocabulary extension* contains the topological relations that are supported e.g. Egenhofer relation family. The *Geometry extension* describes how geometry data are represented and functions that are non-topological between geometry data e.g. distance function. The *Query Rewrite extension* is used to determine whether there is any topological relation, as they were described in *Topology Vocabulary extension*, between geometry literals as they were described in *Geometry extension*. In that way, the *Query Rewrite extension* is used to create the spatial functions in GeoSPARQL. The difference between *Topology Vocabulary extension* and *Geometry Topology extension* is that the former defines how two spatial objects can interact, by defining the topological relations e.g. Egenhofer relation family, and the latter describes the spatial functions that can be used in a spatial query.

Hence, the solutions should support SPARQL querying and should support *spatial object class*, *feature* class which is subclass of *spatial object class* and is disjoint from geometry. It should support Well-known text (WKT) and Geography Markup Language (GML) geometry serializations, topological and non-topological functions with geometry data, allow inference and topological inference. Furthermore, the solution will be compared based on the accuracy of the geometries and based on storage efficiency and speed efficiency.

1.2 CASE STUDY

The following section puts the research into context by presenting and specifying the problem and its details. It focuses on solving spatial problems including spatial aspects using linked data approaches and for that purpose, a case study is going to be defined. In the context of this thesis, a complex problem with spatial aspects refers to a problem that cannot be solved using a simple query, a simple comparison or a spatial function. Such a problem should be solved using multiple data-sets with different characteristics and a series of analyses to reach the required result.

There are many different problems that would comply with the criteria for the definition of a complex spatial problem as it has been presented. The problem which is going to be the case study in this thesis is the identification of a suitable position to facilitate a new shopping center in Delft.

The criteria for the case study will cover some basic requirements that might be important for selecting a position for a new shopping center. Nevertheless, as it is out of scope to measure the actual spatial planning quality of the proposed position or positions for the shopping center, the criteria will be set arbitrary with an aim to allow the comparison between different linked data approaches and will not be chosen based on spatial planning guidelines. Moreover, even though there might be parcels were buildings could be built to facilitate a new shopping center, only existing buildings are going to be considered.

The defined criteria are:

- It should be located in Delft.
  That criterion can be spatial and non-spatial at the same time, depending on the structure of the data e.g. labeled with ”Delft”.
- Distance from train station to be less than 1500 meters.
- Distance from a local main road to be less than 1500 meters.
- It should be allowed by the law to facilitate a shopping center at the building.
- The building must have an area over 1500 sq.meters.

While those are the general criteria, for some approaches they were adjusted to match the structure of the data. For instance the first criterion has been changed to *It should be located within parcels in Delft* to determine that the building is in Delft.

The case study is suitable for the research purpose because it involves multiple selection criteria, both spatial and non-spatial, and multiple data-sets, that can be found both in linked data format and as relational data. During that case study it is possible to determine the extend up to which a certain approach complies with GeoSPARQL, and measure its efficiency and accuracy.

1.3 RESEARCH QUESTION

The following section presents the Research Question (RQ) and the Sub-Research Question (SQR)s that are going to be researched.

The main research question RQ that will be investigated in the context of this thesis is:

RQ 1: ***To what extend is it feasible to solve spatial problems (positioning a shopping center) with a linked data approach using relational spatial data?***

To answer this question, a series of sub-questions should be answered:

SQR_1: What are the possible linked data approaches to position a new shopping center in Delft?

SQR_2: To what extend do the different approaches follow the GeoSPARQL specification?

SQR_3: What benefits and drawbacks do the different linked data approaches offer over approaches with relational data?

SQR_4: How efficient is every approach, in terms of speed of execution and storage requirements?

The efficiency of each approach is going to be compared to others, as well as the GIS solution to provide a qualitative comparison between different solutions.

1.4 THESIS OUTLINE

The use case, i.e. the requirements set for positioning a shopping center and the expected outputs, is presented in details in Section 1.2. In Chapter 2, the relevant theoretical background for linked data, SPARQL, GeoSPARQL, RDF, R2RML and linked data approaches are presented. In Chapter 3, the used tools, the data-sets and the methodology to compare each possible solution is discussed. In Chapter 4, the results produced following the methodology that was designed, are presented. Finally, Chapter 5, elaborates on the results and the efficiency of each solution based on a summary table.

2 LITERATURE REVIEW

The following Chapter presents the relevant literature and links it to the project. More specific, it describes how the linked data were developed, the state of affairs regarding linked data, the relevant published specifications, the relevant technologies and technological improvements and linked data data-sets that have already been published.

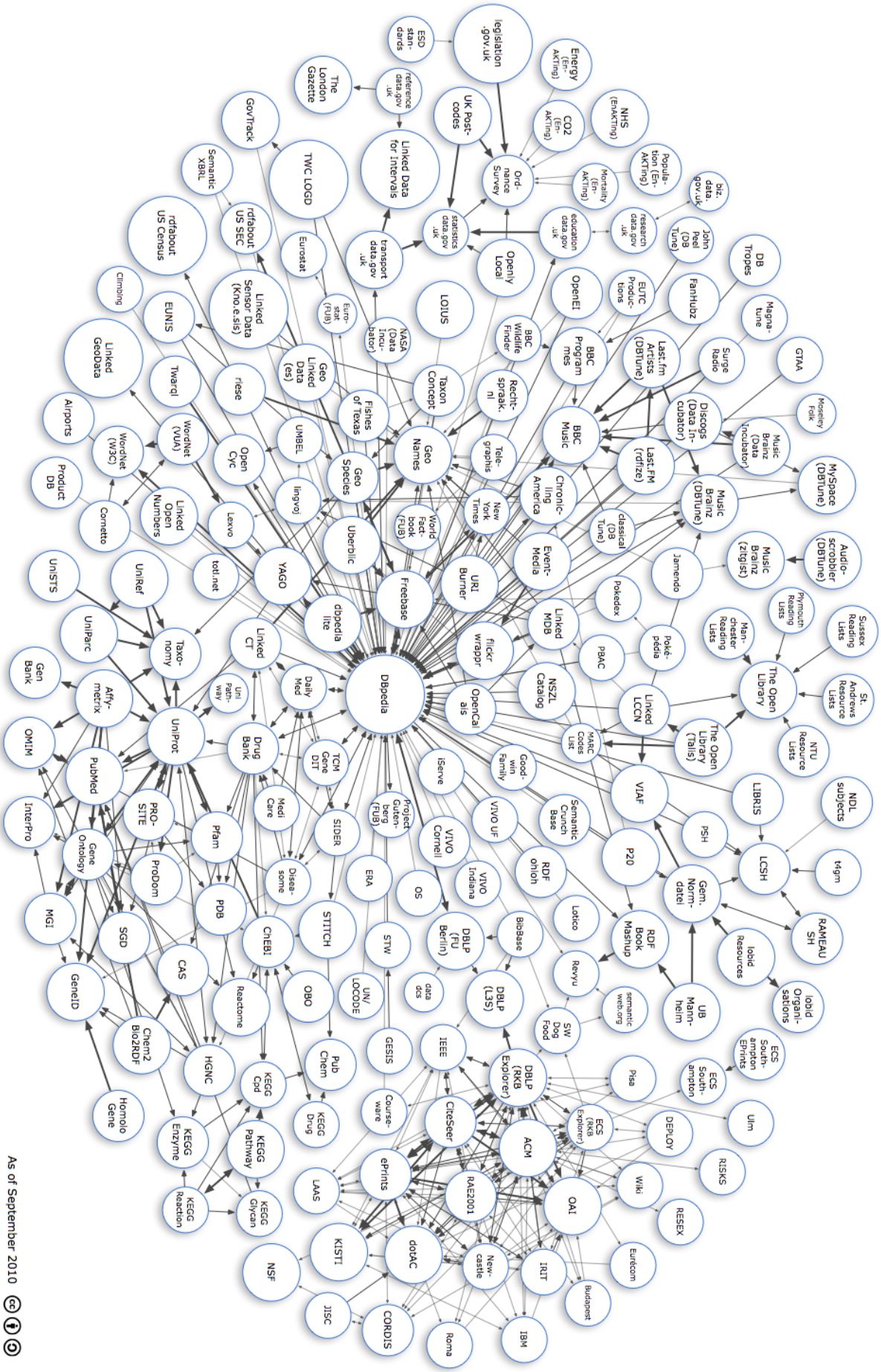
2.1 LINKED DATA

The linked data concept aims to improve the re-usability of data from different types of users. To do so, it specifies that the data should be published on the web, in the form of subject–predicate–object triples format. The triples can be connected to other data with the same way, via subject–predicate–object triples, to enhance them semantically. In that sense, the linked data are not just data published on the web as open data, but are connected to each other semantically.

Nevertheless, linked data cannot be just published with the traditional way one can post a web page. Linked data should be published using the RDF, which was originally developed to be a meta-data data model which is a directed, labeled graph data format for representing information on the web. Linked data can be accessed using the SPARQL and a SPARQL endpoint. The RDF statements contain a subject, a predicate and an object; those three are commonly referred as a triple [Lassila et al., 1998]. The triples are a way to define (predicate) something(object) about someone(subject) in the form of subject–predicate–object. From the triple, the subject must be either a blank node or a Uniform Resource Identifier (URI), the Predicate must be a URI and the Subject can be blank node, URI or a literal. From all three, only the predicate should definitely be a URI that semantically connects Object and Subject. For the purpose of describing the relationships between entities, vocabularies have been developed, using the RDF Schema (RDFS) and the Web Ontology Language (OWL) [Bizer et al., 2009].

Although SPARQL can handle all kinds of data, expressing spatial data as linked data was not fully supported. Hence, OGC, developed a standard for representation and querying geospatial linked data, which contains an ontology based on OGC standards that supports qualitative and quantitative spatial reasoning and querying using SPARQL [Perry and Herring, 2012]. Within the GeoSPARQL standard there are ten spatial functions defined while another five spatial function are defined as possible for development [W3C, 2016]. More specifically, the GeoSPARQL already includes:

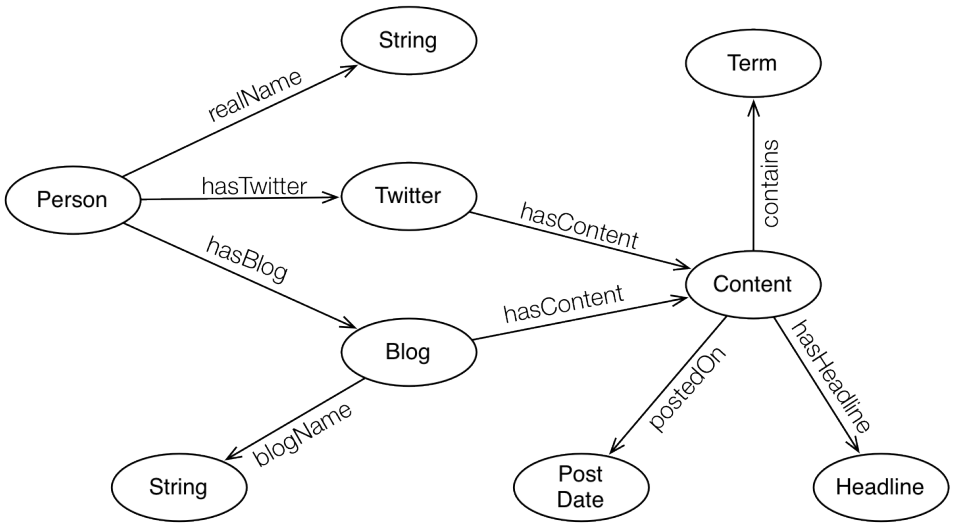
1. Relate
2. Distance
3. Buffer
4. Convex hull
5. Intersection
6. Union
7. Difference



As of September 2010

Figure 2.1: The Cloud of Linked data.

Figure 2.2: Linked data as Directed Graph



8. Symmetric difference
9. Envelope
10. Boundary

While the possibilities for development include the functions:

1. Area. A function that returns the area of an object.
2. Nearest neighbor. A function that returns the nearest neighboring object.
3. Generalization. A function that returns a geometry that is simpler (less vertices) than the initial geometry and it is very similar to it.
4. Centroid. A function that returns the center point of a geometry or a collection of geometries.
5. Bounding box. A function that returns a geometry that contains all the selected geometries.

Those functionalities are some of possible improvements of GeoSPARQL specified by the OGC, that aim to make it more simple, include some more expressive properties and geometry types and make it more specific.

As stated by Berners-Lee et al. [2006], the Semantic Web is an extension to the existing web, which attempts to give computer and human meaning to existing data-set. Due to the global character of World Wide Web (WWW) and the huge volume of information that is published there, one of the initial steps is to connect this information to the according data-set, thus creating "a cloud of linked data", in sort Linked data, which can be seen in figure 2.1. Berners-Lee [2009] also stated four rules that optimize the exploration of data published on the web.

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs. so that they can discover more things.

Code Snippet 2.1: Example of RDF in XML encoding.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>

source: www.w3.org
```

2.2 RDF

The RDF is a family of W3C specifications. It was first introduced in 1999. The RDF 1.0 specification was published in 2004 and the RDF 1.1 specification in 2014. The timeline of W3C standards can be seen in figure 2.3. RDF, which was originally developed to be a metadata data model, is a directed, labeled graph data format for representing information on the web. An example of linked data as RDF is depicted in figure 2.2, where nodes can be Subject, Objects or both, while the arrows represent the Predicates. For example, in triplet (*Person-hasTwitter-Twitter*), *Twitter* is the Object while in another triplet (*Twitter-hasContent-Content*), *Twitter* is the Subject. RDF has been adopted as the medium to post and connect linked data. RDF statements, also called triples, can be encoded in a number of different formats that might be Extensible Markup Language (XML) based (e.g., RDF/XML), shown in Code snippet 2.1, or not (e.g. Turtle), shown in Code snippet 2.2.

2.3 SPARQL

On 2008, SPARQL 1.0 became an official W3C Recommendation and SPARQL 1.1 in 2013. SPARQL is an RDF query language, able to retrieve and manipulate data stored in RDF format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web [Harris et al., 2013]. An example of a SPARQL query that retrieves the Twitter posts of a user after a date is illustrated in Code snippet 2.3.

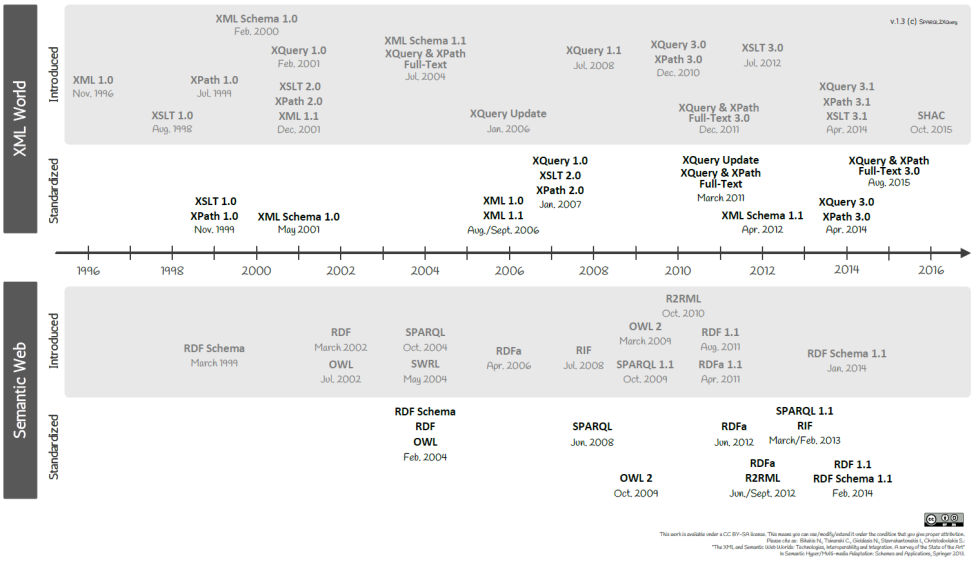
Code Snippet 2.2: Example of RDF in Turtle encoding.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .

source: www.w3.org
```

Figure 2.3: Timeline of W3C Standards



http://www.dblab.ntua.gr/ bikakis/XMLSemanticWebW3CTimeline.pdf

Code Snippet 2.3: Example of a SPARQL query that retrieves the Twitter posts of user "Evangelos Theocharous" that have been posted after 30 March 2019.

```
?p :realName "Evangelos Theocharous"
?p :hasTwitter / :hasContent ?post
?post :postedOn ?dtP
filter(?dtp > "30-03-2019")
```

Figure 2.4: GeoSPARQL query for TU Delft

Default Data Set Name (Graph IRI)

http://dbpedia.org

Query Text

```
select distinct ?ConceptGeom where {?Concept dbo:city dbr:Delft;
rdfs:label "Delft University of Technology"@en;
geo:geometry ?ConceptGeom.
} LIMIT 100
```

(a) Query

ConceptGeom
"POINT(4.3724999427795 52.001667022705)"^^<http://www.openlinksw.com/schemas/virtrdf#Geometry>

(b) Query output

2.4 GEOSPARQL

In the cloud of Linked data, which can be seen in figure 2.1, there are many types of data, such as dates, values, names and places. Some of those data however, might refer to features that also have a spatial dimension. An example would be TU Delft, which is located in Delft, is called "Delft University of Technology" and has a geometry e.g. a position, as shown in the figure 2.4 on page 10. In this query, where data from *dbpedia.org* are accessed, distinct geometries are asked that satisfy 3 criteria in the form of triplets. Initially, Subjects that have Predicate *dbo:city* and Object *dbr:Delft* are queried. *"dbr:"* is a short form for *http://dbpedia.org/resource/* while *"dbo:"* is a short form for *http://dbpedia.org/ontology/* and both of them are in the dbpedia vocabulary. Further, the next triple ask for objects that have the same Subjects as above but also have *rdfs:label* as predicate and *"Delft University of Technology"@en* as Object. The *@en* suffix indicates that the label is in English while an *@nl* would indicate that the label would be in Dutch. Finally, the geometries of those Subjects are accessed with *geo:geometry* Predicate, which leads to the *Point* geometry shown in the figure 2.4.

Features, such as "TU Delft",with spatial dimension started populating the cloud of Linked data, hence geospatial extensions of SPARQL have been developed to help manage them. Such extensions are stSPARQL [Koubarakis and Kyzirakos, 2010] and GeoSPARQL. stSPARQL extends SPARQL to query stRDF meta-data, with stRDF extending RDF with the ability to represent spatial and temporal data. GeoSPARQL has been published as an official OGC standard in 2012 [Perry and Herring, 2012]. It defines classes, properties and extension functions that can be used in SPARQL queries with features that have spatial dimension [Garbis et al., 2013]. In 2016 [W3C, 2016] future developments for GeoSPARQL were published that were aiming to give GeoSPARQL more expressive properties and allow handling of more geometry types, while making it more simple and intuitive. The GeoSPARQL comprises several different components.

- Core component defines top-level RDFS/OWL classes for spatial objects.
- Topology vocabulary component defines RDF properties for asserting and querying topological relations between spatial objects.
- Geometry component defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects and defines topological query functions.

- RDFS entailment component defines a mechanism for matching implicit RDF triples that are derived based on RDF and RDFS semantics.
- Query rewrite component defines rules for transforming a simple triple pattern that tests a topological relation between two features into an equivalent query involving concrete geometries and topological query functions.

2.5 ORACLE SPATIAL AND GRAPH

Oracle Spatial and Graph is a database that supports large-scale Geographic Information Systems, and location-based services applications in the cloud. It is an RDF triplestore that supports significant components of the OGC GeoSPARQL standard. Moreover, it can act as a GeoSPARQL enabled endpoint with federated queries.
Oracle Spatial and Graph uses well-known text serialization and Simple Features relation family to support all the OGC GeoSPARQL standard conformance classes described in page 11 with bullet points.

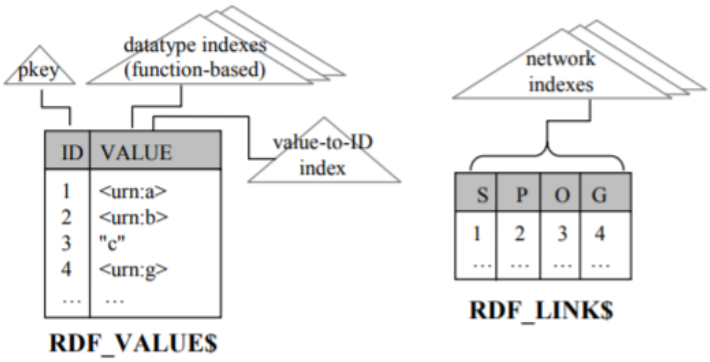
Oracle Spatial and Graph uses a relational schema to store RDF data and processes SPARQL queries by translating them to equivalent Structured Query Language (SQL) queries that are executed by the parallel SQL engine in Oracle Database. A simplified Version of Oracle's relational schema for RDF data consists of an *RDF_LINK* table and an *RDF_VALUE* table. The *RDF_VALUE* table stores and ID to lexical value mapping for RDF terms and the *RDF_LINK* table stores N-Quads of IDs representing subject, predicate, object and graph Perry et al. [2015]. N-Quads are part of the RDF specification by W3C, the difference with normal triples are that they also store a label graph, as described in [W3C, 2014b], and allows to query triples that belong only to specific graphs and describe specific objects. This basic schema of the N-Quads is as shown in figure 2.6a , while in the figures 2.6b and 2.6c, an example triples is shown.

2.5.1 Oracle Views: R2RML mapping language

Another important aspect of Oracle Database is that it supports querying relational data as RDF triplets without having to convert and store them explicitly. This functionality is available through Views build upon the relational data. The views are actually a way to map data and their relationships. What is stored is only the mapping and the metadata. The fact that the RDF triplets are produced on the fly is important, not only for redundancy purposes but for updating the data at the source and at the same time the RDF triplets too. Mapping in Oracle can be performed either by using R2RML language as described in W3C [2012] which is a language created specific for mapping data to RDF triplets, or by direct mapping. R2RML uses Turtle/text file formats. In direct mapping a specific prefix is added to the data. Column names are also converted into URLs and serve as predicates. The relational tables which are going to be used for the mapping, are shown in Figure 2.7. In Figure 2.8a an example of direct mapping in Oracle Spatial and Graph is depicted, while in Figure 2.8b a R2RML mapping example is illustrated.

2.6 ONTOP: ONTOLOGY-BASED DATA ACCESS PLATFORM

Ontop is an Ontology-based Data Access (OBDA) platform which allows to query relational database as RDF graphs using SPARQL [Ontop, 2018]. Ontop currently supports SPARQL 1.0, is compatible with most free and commercial databases and it can act as a SPARQL endpoint. GeoSPARQL compatibility has been achieved with Ontop-spatial extension [Bereta et al., 2016]. A working example of Ontop is illustrated in figure 2.9. Ontop can use R2RML mapping or mapping technique designed for Ontop



(a) Relational Shema RDF

1	<http://www.example.org/family/Cathy>	<http://www.example.org/family/sisterOf>	<http://www.example.org/family/Jack>
2	<http://www.example.org/family/Cindy>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.example.org/family/Female>
3	<http://www.example.org/family/Jack>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.example.org/family/Male>
4	<http://www.example.org/family/Tom>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.example.org/family/Male>
5	<http://www.example.org/family/siblingOf>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
6	<http://www.example.org/family/parentOf>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
7	<http://www.example.org/family/Person>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2000/01/rdf-schema#Class>
8	<http://www.example.org/family/sisterOf>	<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>	<http://www.example.org/family/siblingOf>
9	<http://www.example.org/family/brotherOf>	<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>	<http://www.example.org/family/siblingOf>
10	<http://www.example.org/family/motherOf>	<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>	<http://www.example.org/family/parentOf>
11	<http://www.example.org/family/fatherOf>	<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>	<http://www.example.org/family/parentOf>

(b) *The RDF_VALUES table*

1	1	[MDSYS.SDO_RDF_TRIPLE_S]
2	2	[MDSYS.SDO_RDF_TRIPLE_S]
3	3	[MDSYS.SDO_RDF_TRIPLE_S]
4	4	[MDSYS.SDO_RDF_TRIPLE_S]
5	5	[MDSYS.SDO_RDF_TRIPLE_S]
6	6	[MDSYS.SDO_RDF_TRIPLE_S]
7	7	[MDSYS.SDO_RDF_TRIPLE_S]
8	8	[MDSYS.SDO_RDF_TRIPLE_S]
9	9	[MDSYS.SDO_RDF_TRIPLE_S]
10	10	[MDSYS.SDO_RDF_TRIPLE_S]
11	11	[MDSYS.SDO_RDF_TRIPLE_S]

(c) *The RDF_LINKS Table*

Figure 2.6: An example of Linked data stored in Oracle Spatial and Graph Database

DELFT_PARCELS

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	OBJECT_ID	NUMBER	Yes	(null)	1	(null)
2	PARCEL_GEOM	SDO_GEOMETRY	Yes	(null)	2	(null)

(a) Delft Parcels table definition.

GEBOUW_VLAK_S

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	IDENT	NUMBER	Yes	(null)	1	(null)
2	GEBOUW_GEOM	SDO_GEOMETRY	Yes	(null)	2	(null)

(b) Gebouw table definition.

WEGDEEL_VLAK_S

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	IDENT	NUMBER	Yes	(null)	1	(null)
2	WEGDEEL_GEOM	SDO_GEOMETRY	Yes	(null)	2	(null)

(c) Wegdeel table definition.

SPATIAL_PLANS

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	OBJECT_ID	NUMBER	Yes	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(30 BYTE)	Yes	(null)	2	(null)
3	GEOM	SDO_GEOMETRY	Yes	(null)	3	(null)

(d) Spatial Plans table definition.

Figure 2.7: Relational database tables definition.

platform. According to the developers of Ontop, it has been created because at the time there was no OBDA system with GeoSPARQL support [Bereta et al., 2016].

Ontop-spatial extension

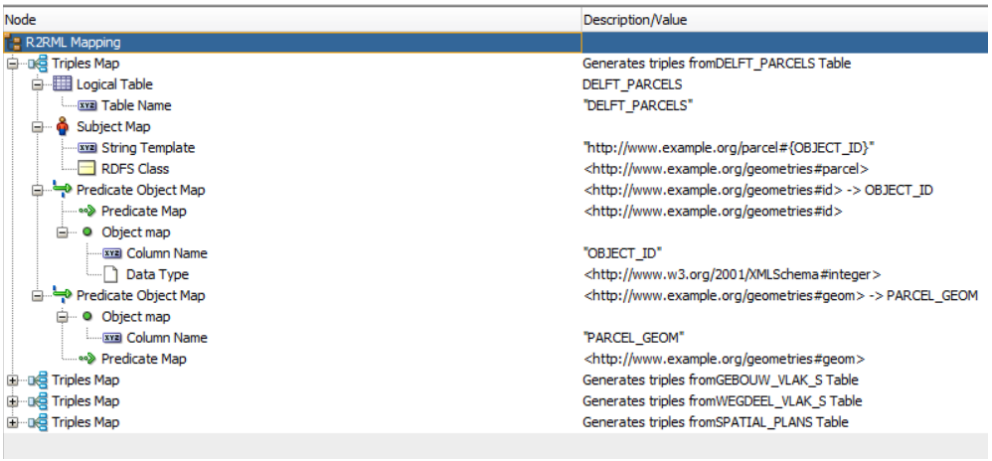
Ontop-spatial, is an extension to Ontop platform, that allows on-the-fly GeoSPARQL-to-SQL translation based on ontologies and mappings [Bereta and Koubarakis, 2016]. In the current version, it compatible with PostGIS, Spatialite and Oracle Spatial and while it supports OGC GeoSPARQL standard conformance classes, it also supports raster data-type format translation [Bereta et al., 2017].

2.7 KADASTER

Kadaster is the the name for the Dutch Cadastre, Land Registry and National Mapping Agency in the Netherlands. From Kadaster it is possible to access data for the case study designed. One of the datasets that is useful for the case study, is the Basisregistratie Adressen en Gebouwen (BAG). BAG contains all official addresses within the Netherlands, is updated by the municipalities and contains a number of different objects. It will be explained thoroughly in Chapter 3.

BAG can also describe non legal spatial object or addresses. Kadaster already made available the Basisregistraties dataset (BAG) and the spatial plans dataset available through direct download and as linked data through a SPARQL endpoint [Kadaster, 2018a] and an API endpoint [Kadaster, 2018b], which will be explained in Chapter 3.

(a) Direct mapping example.



(b) R2RML mapping example.

Figure 2.8: Examples of different mapping methodologies as they are stored in Oracle Spatial and Graph Database.

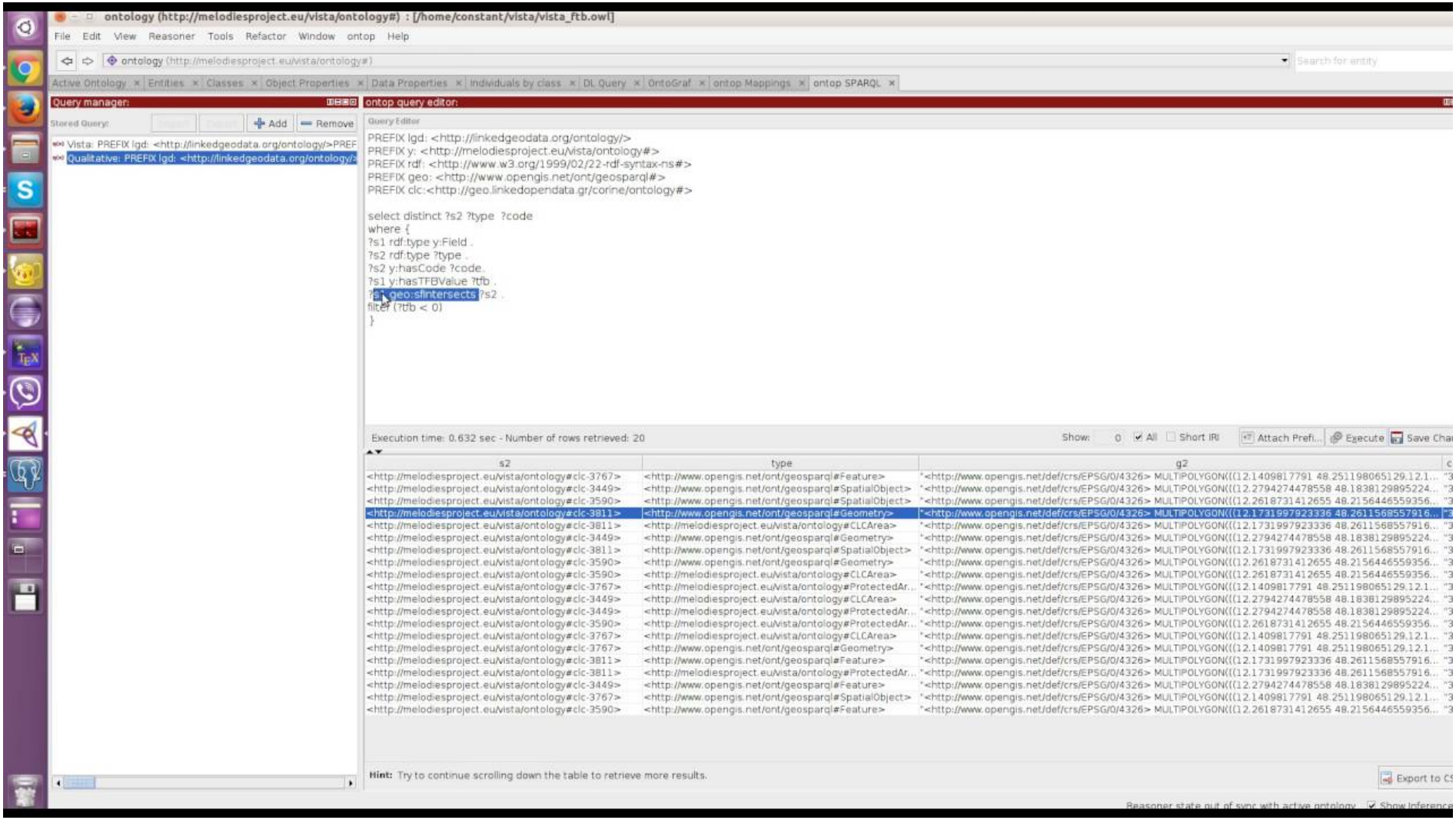


Figure 2.9: Ontop working example.
Source: https://i.ytimg.com/vi/F5_2Zxi5_e8/maxresdefault.jpg

3 DATA AND METHODOLOGY

The following Chapter presents the data (Section 3.1) and the tools (Section 3.2) that have been used and they will also be described in relation to the methodology (Section 3.3) followed and the different approaches used to answer the research question. The different approaches are:

- Benchmark Solution: QGIS approach
- Dutch Kadaster's PDOK approach - GeoSPARQL endpoint
- Dutch Kadaster's PDOK approach - API
- Oracle Spatial and Graph based approach - linked data
- Oracle Spatial and Graph based approach - Relational data to Linked-Data: Direct Mapping
- Oracle Spatial and Graph based approach - Relational data to Linked-Data: R2RML

3.1 DATA

The data that are used for each approach differ in format and structure.

- BAG is part of the government system of basic registers. Municipalities are source holders of the BAG. It contains information on seven object types:
  - NUMMERAANDUIDING (huisnummer en toevoegingen)
    Roughly translates into *NUMBER DESCRIPTION (house number and additions)*
  - OPENBARE RUIMTE (straten)
    Roughly translates into *PUBLIC SPACE (streets)*
  - WOONPLAATS
    Roughly translates into *RESIDENCE*
  - PAND
    It describes the smallest construct that touches the earth and is accessible.
  - VERBLIJFSOBJECT
    Roughly translates into *ACCOMMODATION*
  - STANDPLAATS
    Roughly translates into *PLACE*
  - LIGPLAATS
    Roughly translates into *BERTH*

  The attributes include status, surface, geometry, xy coordinate, year of construction and purpose of use. BAG contains these data for all buildings in the Netherlands. The owned of the dataset is *Dienst voor het kadaster en de openbare registers* or *Service for the land register and the public registers* as it is translated in English.

- Basisregistratie Topografie (BRT) Kadaster is the data manger of the BRT. It consists of digital topographic files on different scale levels. This collection of topographic objects is available as open data. From this dataset roads, spatial planning zones are extracted.
- TOP10NL is the most detailed digital topographic version of BRT and can be used at scale levels between 1: 5.000 and 1: 25.000.
- Basisregistratie Kadaster (BRK) is a dataset that consists of the cadastral registration of immovable property and rights in rem and the cadastral map which map illustrates the location of the cadastral parcels (including parcel number), the borders of the national government, the provinces and the municipalities.

Moreover, for the purpose of familiarizing with the linked data concept and how linked data work within Oracle Spatial and Graph, a set of dummy data (referred as dummy linked data dataset from now on) has been created as relational spatial data in Quantum GIS (QGIS), and then converted to linked data. It consists of 23 Buildings, a train station, eight roads and a main road and a spatial planning zone which allows for building a shopping center. It was converted into linked data triplets following the GeoSPARQL specification, and it is depicted in Figure 3.3.

Benchmark Solution: QGIS approach

For the QGIS approach, the datasets used were stored as relational data in Oracle database, which was connected to QGIS. The tables' definitions can be seen in 2.7 on page 13. For the QGIS approach the datasets that have been used are:

- Buildings in Delft subset of BAG as relational data.
- Roads in Delft subset of BRT as relational data.
- Parcels in Delft subset of BRK as relational data.
- Spatial plans zones in Delft subset of BRT as relational data. That dataset has been retrieved from PDOK's SPARQL endpoint.

Dutch Kadaster's PDOK approach - GeoSPARQL endpoint

The endpoint is reachable online, thus creating own copy of input data is not necessary in this approach. It should be noted that through this approach, the spatial planing zone dataset is downloaded as relational data and converted into linked data to be imported in Oracle Spatial and Data.

Dutch Kadaster's PDOK approach - API

The idea behind reaching the Dutch Kadaster's PDOK API is to access the data remotely and combine it with data in Oracle Spatial and Data database. However, since all the required data are available on the API, no data should be necessary in the database.

Oracle Spatial and Graph based approach - Linked data

For this approach the datasets that have been used are:

- BAG datasets as linked data provided by the dutch Kadaster.
- BRT roads datasets as linked data provided by the dutch Kadaster.

- BRT spatial plans zone datasets downloaded as related data from the dutch Kadaster's PDOK GeoSPARQL endpoint and converted into linked data.
- BRK parcels datasets as linked data provided by the dutch Kadaster.

Oracle Spatial and Graph based approach - Relational data to linked data

For both ***Direct Mapping*** and ***R2RML Mapping*** approaches the datasets that have been used are the same. The tables' definitions can be seen in 2.7 on page 13, and are presented below:

- BAG datasets as relational data provided by the dutch Kadaster.
- BRT roads datasets as relational data provided by the dutch Kadaster.
- BRT spatial plans zone datasets as relational data downloaded from the dutch Kadaster's PDOK GeoSPARQL endpoint.
- BRK parcels datasets as relational data provided by the dutch Kadaster.

3.2 TOOLS

The tools used for each approach in this thesis are presented below:

Benchmark Solution: QGIS approach

QGIS solution is designed to work as a benchmark solution to compare every other solution to this. To create this benchmark solution ***QGIS 2.18*** has been used, which was connected with ***Oracle dbms***. Moreover, ***Spatial Query Plugin*** for QGIS, shown in Figure 3.1, has been used for the spatial queries along with spatial analysis algorithms provided with QGIS.

Dutch Kadaster's PDOK approach - GeoSPARQL endpoint

The GeoSPARQL endpoint [Kadaster, 2018a] has been accessed through ***Mozilla Firefox Browser*** and ***Google Chrome Browser***.

Dutch Kadaster's PDOK approach - API

The PDOK's API has been accessed through ***Oracle's SQL developer***.

Oracle Spatial and Graph based approach - Linked data

The dummy dataset has been designed using QGIS, has been exported to CSV file and using the code snippets in Appendix A has been converted into linked data and imported into Oracle Spatial and Graph. The geometries from CSV file, as illustrated in Figure 3.2, have been converted to linked data *INSERT* statements with Python3, shown in Appendix A. Finally, the *INSERT* statements have been used with Oracle's SQL developer to import the linked data into the database. Similarly, the spatial plan zones that have been downloaded from Dutch Kadaster's PDOK GeoSPARQL endpoint, have been converted into linked data *INSERT* statements using the code snippet in Appendix A, where you can also find the result.

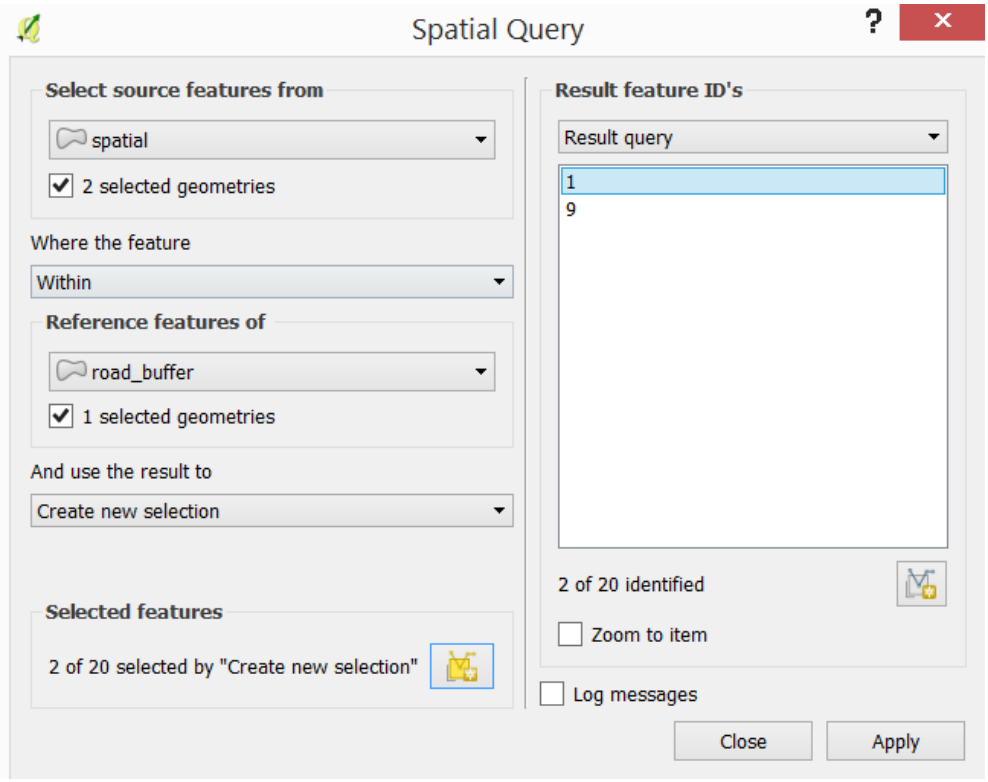


Figure 3.1: QGIS Spatial Query Plugin.

Oracle Spatial and Graph based approach - Relational data to linked data

For the R2RML mapping, the turtle/rdf file was written in ***Sublime text 3*** then converted first into n-triples using ***EasyRdf converter*** [Humfrey, 2018] and then into *INSERT* statements using ***Python3***. The planning laws have been downloaded through the SPARQL endpoint into *Comma-Separated Values (CSV)* format, have been joined into WKT using ***Python3*** and then converted into *INSERT* statements using ***Python3***. The code can be found in Appendix B on page 97.

3.3 METHODOLOGY

During the literature review in Chapter 2, different approaches to handle linked data have been mentioned. However, out of the available vendors, only Oracle Spatial and Graph seems to, according to its specifications, fully support GeoSPARQL.On the other hand, it is not clear if Kadaster's PDOK GeoSPARQL endpoint is designed following the GeoSPARQL specification. Moreover, while there are different approaches to handle linked data and to access relational data as linked data, there is no assessment of the performance of those approaches. Hence, an experimental assessment of those approaches will be carried out. During that assessment, a scale-up approach is going to be followed, going from simple SPARQL queries, that will explore Oracle Spatial and Graph's ability to handle linked data, to more complex GeoSPARQL spatial functions, that will lead to solving the research question. Also, it will deal with smaller and more simple datasets, such as the dummy dataset designed. The dummy dataset comprises of subsets of BAG, BRT and BRK datasets. The datasets used and stored with Oracle Spatial and Graph are going to be sub-sets of BAG, BRT and BRK datasets, including only the data covering the area of Delft, Netherlands, because of the limited processing power of the hardware (laptop) used. The methodology steps mentioned below, will be explained in details in the coming sections.

WKT
POLYGON ((12674.1396695509 308280.243640802,12674.1396695509 308280.254022339,12674.0666394816 308280.254191788,12674.0666394816 308280.243810251,12670.3404419283 308280.252456,12670.3
POLYGON ((12670.3322128704 308300.222756363,12670.3240990239 308319.913461677,12671.8914958125 308319.913461677,12671.8914930332 308319.920206451,12675.7033092857 308319.920206451,1267
POLYGON ((12670.3486690284 308260.286906502,12670.340689168 308279.652453812,12671.9080874552 308279.648817039,12671.9080831773 308279.659198586,12675.67078152 308279.650468145,12675.6
POLYGON ((12678.4761868651 308319.889912311,12682.2430570617 308319.889912311,12682.2430570617 308300.193838371,12678.4731678861 308300.202991175,12678.4761868651 308319.889912311))
POLYGON ((12682.2468113362 308299.593827488,12682.2468113362 308280.224830121,12678.4701055895 308280.233593062,12678.4730758762 308299.60298963,12682.2468113362 308299.593827488))
POLYGON ((12678.4700135796 308279.633591661,12682.2202814849 308279.624890062,12682.2202814849 308260.259295198,12678.4670438801 308260.268024576,12678.4700135796 308279.633591661))
POLYGON ((12682.2240357594 308259.659284844,12682.2240357594 308239.962929478,12678.4639301046 308239.962929478,12678.4669518702 308259.668023167,12682.2240357594 308259.659284844))
POLYGON ((12665.9557434218 308300.233381866,12665.9557434218 308319.928725725,12669.7240926832 308319.928725725,12669.7322122192 308300.224213087,12665.9557434218 308300.233381866))
POLYGON ((12669.7324594591 308299.624210718,12669.7404413037 308280.253848158,12665.9594976963 308280.262620933,12665.9594976963 308299.633370982,12669.7324594591 308299.624210718))
POLYGON ((12669.7489156422 308259.688299801,12669.7570284976 308240.0,12666 308240,12666.0 308259.697019126,12669.7489156422 308259.688299801))
POLYGON ((12665.9962457255 308260.297029481,12665.9962457255 308279.662534052,12669.7406885435 308279.653845969,12669.7486684025 308260.288301999,12665.9962457255 308260.297029481))
POLYGON ((12657.2755943749 308280.290076366,12657.2755943749 308280.300457902,12657.2025643055 308280.300627351,12657.2025643055 308280.290245815,12653.4763667522 308280.298891564,126
POLYGON ((12653.4681376944 308300.269191926,12653.4600238478 308319.959897241,12655.0274206364 308319.959897241,12655.0274178571 308319.966642015,12658.8392341096 308319.966642015,126
POLYGON ((12653.4845938524 308260.333342066,12653.476613992 308279.698889375,12655.0440122791 308279.695252603,12655.0440080013 308279.705634149,12658.806706344 308279.696903709,12658.
POLYGON ((12661.612111689 308319.936347874,12665.3789818856 308319.936347874,12665.3789818856 308300.240273935,12661.6090927101 308300.249426739,12661.612111689 308319.936347874))
POLYGON ((12665.3827361601 308299.640263051,12665.3827361601 308280.271265685,12661.6060304134 308280.280028626,12661.6090007001 308299.649425194,12665.3827361601 308299.640263051))
POLYGON ((12661.6059384035 308279.680027224,12665.3562063088 308279.671325626,12665.3562063088 308260.305730762,12661.6029687041 308260.314460139,12661.6059384035 308279.680027224))
POLYGON ((12665.3599605833 308259.705720407,12665.3599605833 308240.009365041,12661.5998549285 308240.009365041,12661.6028766942 308259.71445873,12665.3599605833 308259.705720407))
POLYGON ((12649.0916682457 308300.279817429,12649.0916682457 308319.975161288,12652.8600175071 308319.975161288,12652.8681370432 308300.270648651,12649.0916682457 308300.279817429))
POLYGON ((12652.868384283 308299.670646282,12652.8763661276 308280.300283722,12649.0954225202 308280.309056496,12649.0954225202 308299.679806546,12652.868384283 308299.670646282))
POLYGON ((12652.8848404662 308259.734735365,12652.8929533215 308240.046435564,12649.1359248239 308240.046435564,12649.1359248239 308259.74345469,12652.8848404662 308259.734735365))
POLYGON ((12649.1321705494 308260.343465044,12649.1321705494 308279.708969616,12652.8766133674 308279.700281533,12652.8845932264 308260.334737563,12649.1321705494 308260.343465044))
POLYGON ((12670.4035997904 308240.298997785,12670.3956199301 308259.664545095,12671.9630182172 308259.660908322,12671.9630139394 308259.671289869,12675.725712282 308259.662559428,1267
POLYGON ((12653.5395246144 308240.345433349,12653.531544754 308259.710980658,12655.0989430412 308259.707343886,12655.0989387633 308259.717725432,12658.861637106 308259.708994992,12658.
LINESTRING (12701.4105889035 308239.46389117,12639.8612276599 308239.461956654)
LINESTRING (12678.1394747002 308238.308080351,12678.1864776362 308341.205799084)
LINESTRING (12665.8066913201 308234.256235345,12665.6419240583 308340.952386794)
LINESTRING (12661.2619176019 308234.780601297,12661.4328342415 308342.166147913)
LINESTRING (12669.983054243 308339.93460348,12669.9773113177 308233.941785828)
LINESTRING (12653.3141662331 308233.602190549,12653.1522970116 308342.69359371)
LINESTRING (12639.8520729584 308259.82244585,12703.0050082814 308260.578667184)
LINESTRING (12640.9791063874 308280.051838176,12703.5376209845 308279.926182568)
LINESTRING (12641.2753390686 308300.004696089,12696.1720136441 308299.890352442)
LINESTRING (12640.8315204905 308320.676571266,12698.0553627381 308319.999122042)

Figure 3.2: Dummy dataset in CSV.

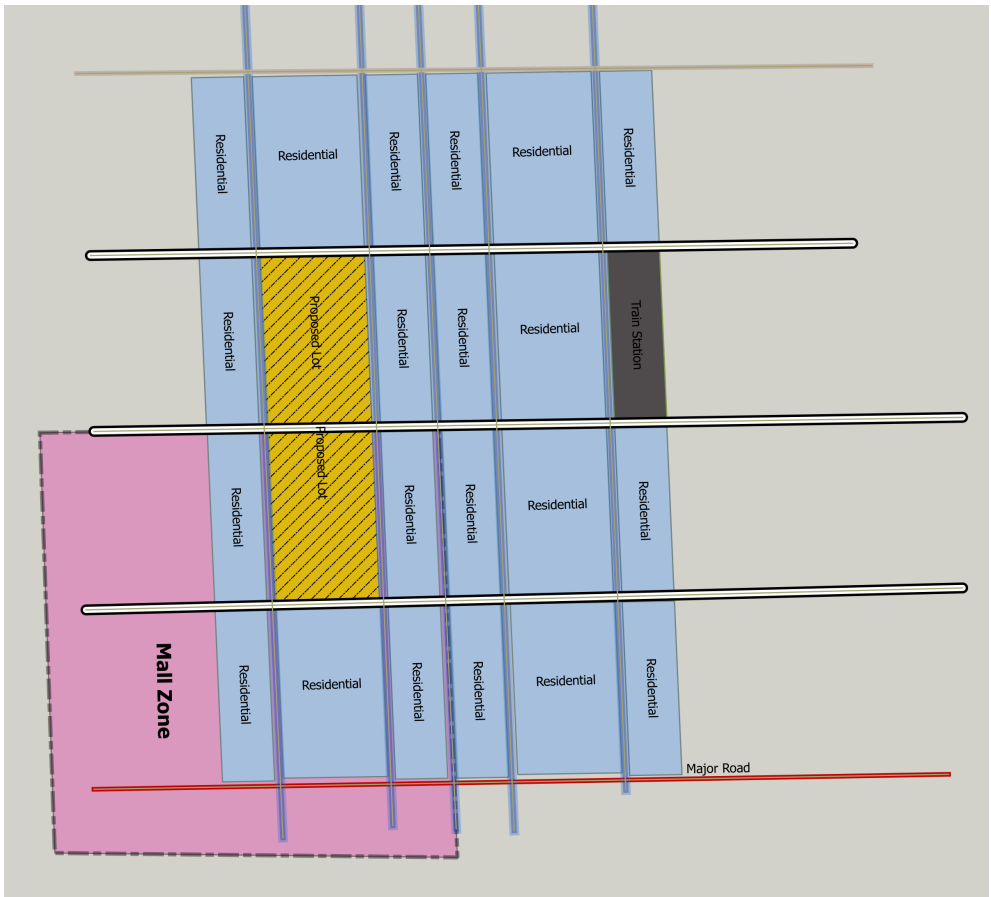


Figure 3.3: Dummy dataset.

Methodology steps:

- Step 1: Create dummy data and familiarize with Oracle Spatial and Graph
- Step 2: Forming a Sample spatial query
- Step 3: Real Data Loading
- Step 4: QGIS solution
- Step 5: PDOK approaches - GeoSPARQL endpoint
- Step 6: PDOK approaches - API
- Step 7: Oracle Spatial and Graph approaches - Direct mapping
- Step 8: Oracle Spatial and Graph approaches - R2RML

3.3.1 Step 1: Dummy data and Oracle Spatial and Graph

Before any other analysis is carried out, familiarization with Oracle Spatial and Graph and its linked data approach was necessary. For that purpose, the dummy data, that are depicted in Figure 3.3, were created using QGIS and then extracted into CSV format and converted into *INSERT* statements using python. The created dummy data were imported into the Oracle dbms following the linked data principles. In that sense, the table that holds the data is created, containing only two columns; the first holds the id of the registry and the second holds a triple, as specified in linked data. Afterwards, a model that holds the triples has been created. Consecutively, the linked data have been added one by one. Part of the Insert statements is illustrated in Figure 3.7.

During the creation of the dummy data, GeoSPARQL specification was followed closely. For that reason, the creation of the dummy data started by creating ontologies for three different types of data, roads, lots and a planning zone as illustrated in Appendix C. An ontology is a model that enables accurate interpretation of data from multiple sources through the explicit definition of terms and relationships in the ontology [Wache et al., 2001]. The ontologies created follow the form of GeoSPARQL ontology for spatial objects as described in [Perry and Herring, 2012] and is presented in the Figure 3.4 on page 23.

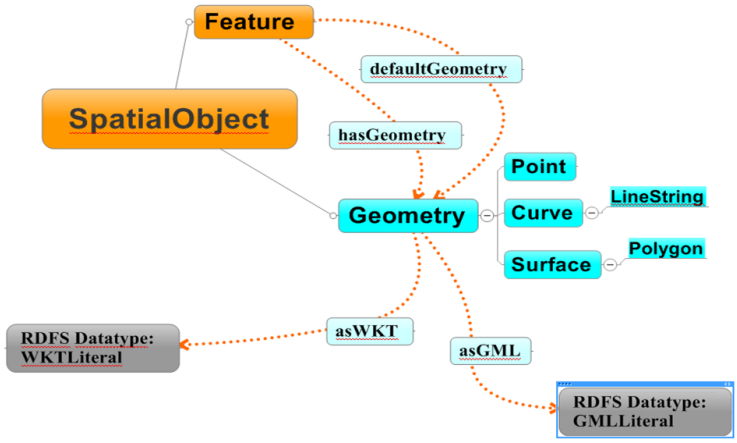
In more details, the dummy data contain twenty four lots (polygons), ten roads (lines) and a planning zone (polygon). The spatial layout is illustrated in the Figure 3.3 on page 21.

Before running any spatial function on linked data in Oracle dbms, a spatial index must be created. The reason behind this is that the spatial index is used to calculate results of spatial functions, especially if there is a large amount of spatial data. However, the spatial index must be present even if it is not used. As described in [Chuck Murray et al., 2013] the spatial index in Oracle Spatial and Graph is an R-tree data structures used for spatial access methods that can index spatial data of up to four dimensions, while our data are on two dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry. As described in [Chuck, 2013], the spatial index requires the Spatial Reference System Identifier (SRID) of the data, a Tolerance that defines the distance where two points will be regarded as different points and the number of Dimensions of the data. In our case, even in the dummy data, the SRID selected is European Petroleum Survey Group (EPSG):28992, which is the *Dutch RD*. The data should be inserted defining their SRID, otherwise, a simple planar Coordinate reference system (CRS) - SRID 262148 Non-Earth (meters) will be used and the coordinates of the geometries will be transformed into the latter SRID. The mismatch between SRIDs of the spatial index and the geometries results in faulty spatial relationships.

Code Snippet 3.1: Validate Deometries

```
BEGIN
sem_apis.validate_geometries(model_name=>'geometries',
SRID=>28992, tolerance=>0.005, options=>'STANDARD_CRS_URI=T RECTIFY=T');
END;
```

Figure 3.4: GeoSPARQL Ontology



After the creation of the index, the first function applied aims to validate the geometries, as shown in Code snippet 3.1. The validation checks for duplicate geometries, that are removed from the view of the data, and for invalid geometries which are fixed if possible. A report is created for the invalid geometries in form of separate tables.

With all the above steps, the database is set for using linked data and functions and the testing can begin. Initially, simple random selections are tested as shown in Code snippet 3.2 on page 23, where the subject-predicate-object relationship of RDF triplets can be seen. The result can be seen in Figure 3.5 on page 24. With this query we make sure the linked data concept works fine.

To familiarize with Oracle Spatial and Graph, the dummy dataset has been used to apply the spatial queries that are available and are described in Appendix B. Having in mind the case study and the GeoSPARQL core conformance classes, the exploration of Oracle's Spatial and Graph functionality revolved around satisfying the requirements set in the scope of research in Section 1.1, page 3.

The first query checks which lots are no more than 30meter (m) from a main road, as seen in Code snippet 3.3. The result depicts the lots, the distance and the main road that the lot is distant from.

As seen in the Figure 3.6 on page 24, the results are as expected. More specific, as seen in the Figure 4.1 on page 40, there is one main road (red line on the bottom of the Figure), and the 30M buffer from that roads contains the twelve buildings on the bottom half part of the Figure.

Code Snippet 3.2: SPARQL Query of ten random triplets

```
SELECT ?s ?p ?o
WHERE { ?s ?p ?o }
LIMIT 10
```

	S	P	O
1	http://www.example.org/geometries/lot2	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
2	http://www.example.org/geometries/lot9	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
3	http://www.example.org/geometries/lot11	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
4	http://www.example.org/geometries/lot13	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
5	http://www.example.org/geometries/lot18	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
6	http://www.example.org/geometries/lot21	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
7	http://www.example.org/geometries/lot23	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
8	http://www.example.org/geometries/lot24	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
9	http://www.example.org/geometries/lot7	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/lot
10	http://www.example.org/geometries/mall zone	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.example.org/geometries/zone

Figure 3.5: Simple selection result

Code Snippet 3.3: GeoSPARQL query lots within 30 meter from main road

```
    SELECT ?oL
    WHERE {
?oL rdf:type :lot. ?oL gsp:hasGeometry/gsp:asWKT ?wkt_l.
?obR rdf:type :road. ?obR gsp:hasGeometry/gsp:asWKT ?wkt_r.
?obR rdfs:label "main road"@EN.
BIND(ogcf:distance(?wkt_r, ?wkt_l,
<http://xmlns.oracle.com/rdf/geo/uom/M>)as ?main_road_dist).
FILTER( ?main_road_dist < 30) }
```

The final step is to inference with topological and non-topological functions. The inference in Oracle Spatial and Graph is executed via *entailment* construction which essentially is a pre-computed set of values for a specific function. Those functions in this thesis will be topological, e.g. *ogcf:sfWithin*, or non-topological, e.g. *ogcf:distance*. The method to do so is to create a table to hold a set of functions which is called rule, and then create the *entailment* using the rule. However, in Oracle Spatial and Graph, the rule even when it is referring to linked data, it is SQL based and it can't have a filter during the creation. The filter can be updated later on, nevertheless, it should use SQL statements and functions. Finally, any attempt to create an entailment with topological function, e.g. relate(within) rule, or non-topological function, e.g. distance rule, was successful during the creation but returned no results. The process can be seen in Code snippet 3.4.

3.3.2 Step 2: Forming a Sample spatial query

In order to be able to compare the performance of different approaches, a single question or query should be posed against all of them. That sample query should include non-spatial querying and both topological and non-topological spatial querying, as per scope of research. It is expected that the sample query won't be able

	OL	MAIN_ROAD_DIST	OBR
1	http://www.example.org/geometries/lot23	.584069318403863	http://www.example.org/geometries/road1
2	http://www.example.org/geometries/lot7	20.7960071752102	http://www.example.org/geometries/road1
3	http://www.example.org/geometries/lot24	20.8723715701837	http://www.example.org/geometries/road1
4	http://www.example.org/geometries/lot11	.537103711193857	http://www.example.org/geometries/road1
5	http://www.example.org/geometries/lot13	.56026514888576	http://www.example.org/geometries/road1
6	http://www.example.org/geometries/lot4	20.8062688718002	http://www.example.org/geometries/road1
7	http://www.example.org/geometries/lot12	20.8254059629737	http://www.example.org/geometries/road1
8	http://www.example.org/geometries/lot1	.513299541675755	http://www.example.org/geometries/road1
9	http://www.example.org/geometries/lot20	.546606953412374	http://www.example.org/geometries/road1
10	http://www.example.org/geometries/lot8	.499641347250107	http://www.example.org/geometries/road1
11	http://www.example.org/geometries/lot16	20.8532344780207	http://www.example.org/geometries/road1
12	http://www.example.org/geometries/lot19	20.842972782362	http://www.example.org/geometries/road1

Figure 3.6: Spatial function:Lots that are no more than 30M from a main road.

Code Snippet 3.4: Creating zoning rule and entailment

```
EXECUTE SEM_APIS.CREATE_RULEBASE('ZONING');

INSERT INTO mdsys.semr_ZONING VALUES(
  'zoning_rule',
  '(?OZ rdf:type :zone) (?OZ gsp:hasGeometry/gsp:asWKT ?WKTZ)
    (?OL rdf:type :lot) (?OL gsp:hasGeometry/gsp:asWKT ?WKTL)',
    NULL,
  '(?OL :inZone ?OZ)',
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/geometries/'),
    SEM_ALIAS('geof','http://www.opengis.net/def/function/geosparql/'),
    SEM_ALIAS('gsp','http://www.opengis.net/ont/geosparql#')));

UPDATE mdsys.semr_ZONING SET
  antecedents = '
  (?OZ rdf:type :zone) (?OZ gsp:hasGeometry/gsp:asWKT ?WKTZ)
  (?OL rdf:type :lot) (?OL gsp:hasGeometry/gsp:asWKT ?WKTL)',
  filter = '(SDO_GEOM.RELATE(SDO_GEOMETRY(WKTZ),
  ''mask=INSIDE'', SDO_GEOMETRY(WKTL), 0.0001)= ''TRUE'')',
  aliases = SEM_ALIASES(
    SEM_ALIAS('','http://www.example.org/geometries/'),
    SEM_ALIAS('geof','http://www.opengis.net/def/function/geosparql/'),
    SEM_ALIAS('gsp','http://www.opengis.net/ont/geosparql#'))
WHERE rule_name = 'ZONING_RULE';

BEGIN
  SEM_APIS.CREATE_ENTAILMENT(
    'zone_geom',
    SEM_Models('geometries'), SEM_Rulebases('ZONING'));
END;

SELECT ol  FROM TABLE(SEM_MATCH(
    '{?ol :inZone ?oz}',
    SEM_Models('geometries'), SDO_RDF_Rulebases('ZONING'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/geometries/')),
    NULL,null));
```

```
execute SEM_APIS.create_sem_model('geometries', 'geometries_rdf', 'triple');


INSERT INTO geometries_rdf VALUES (1,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (2,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot1"@EN'));

 INSERT INTO geometries_rdf VALUES (3,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_1>  '));

 INSERT INTO geometries_rdf VALUES (4,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_1>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> POLYGON ((12674.1071417853 308239
```

(a) *INSERT* statements for dummy dataset.

	SUBJECT	PREDICATE	OBJECT
1	<http://www.example.org/geometries/road_geom_31>	<http://www.opengis.net/ont/geosparql#asWKT>	"<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12639.852072
2	<http://www.example.org/geometries/road_geom_34>	<http://www.opengis.net/ont/geosparql#asWKT>	"<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12640.831520
3	<http://www.example.org/geometries/road_geom_32>	<http://www.opengis.net/ont/geosparql#asWKT>	"<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12640.979106
4	<http://www.example.org/geometries/road_geom_33>	<http://www.opengis.net/ont/geosparql#asWKT>	"<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12641.275339
5	<http://www.example.org/geometries/road_geom_30>	<http://www.opengis.net/ont/geosparql#asWKT>	"<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12653.314166

(b) Dummy dataset stored in RDF model in Oracle dbms.

Figure 3.7: Inserting dummy data on Oracle Spatial and Graph.

	OL	TRAIN_DIST	LOT_AREA	MAIN_ROAD_DIST
1	http://www.example.org/geometries/lot13	17.4728938749963	148.009764500024	.56026514888576
2	http://www.example.org/geometries/lot16	17.4640820986132	145.742449282734	20.8532344780207

Figure 3.8: Lots in dummy dataset that fulfill the sample query criteria.

Code Snippet 3.5: Sample Query fulfilling the criteria

```
?oA rdf:type :lot. ?oA rdfs:label "train station"@EN.
?oA gsp:hasGeometry/gsp:asWKT ?wkt.
?oL rdf:type :lot. ?oL gsp:hasGeometry/gsp:asWKT ?wkt_l.
?oZ rdf:type :zone. ?oZ gsp:hasGeometry/gsp:asWKT ?wkt_z.
?obR rdf:type :road. ?obR gsp:hasGeometry/gsp:asWKT ?wkt_r.
?obR rdfs:label "main road"@EN.
BIND(ogcf:distance(?wkt_r, ?wkt_l,
                <http://xmlns.oracle.com/rdf/geo/uom/M>) as ?main_road_dist).
BIND(ogcf:sfWithin(?wkt_l, ?wkt_z) as ?within_zone).
BIND(ogcf:distance(?wkt, ?wkt_l,
                <http://xmlns.oracle.com/rdf/geo/uom/M>) as ?train_dist).
BIND(orageo:area(?wkt_l, "unit=SQ_M") as ?lot_area).
FILTER(?train_dist > 0 && ?train_dist < 1500 && (?within_zone = true)
                    && ?lot_area > 1500 && ?main_road_dist < 1500)
```

to fit exactly in all approaches and it will be adjusted to fit the approach and the structure of the data, nevertheless it should still be able to comply with the scope of research. For instance, querying buildings that are 1000 meters away from train station will have the same result as creating a buffer around train station and then querying the buildings inside the buffer. However, the *distance* spatial function is a non-topological one that calculates distance and filters it, while the *within* spatial function is a topological one. Hence, it had to be guaranteed that both kind of functions are used in the query if supported by the software.

The criteria set for positioning the shopping center are arbitrary and can be seen in Section 1.2 on page 3.

The Code snippet 3.5 describes the sample query that checks which lots fulfill the aforementioned criteria. As already mentioned, the query should be adjusted to match the structure of the data. The query should return a list of all the lots that fulfill the criteria, however, as seen in Figure 3.8, one of the two lots that are returned is not within the zone that permits building. Nevertheless, that result is not unexpected. The spatial functions are solved utilizing the spatial index that was described before. In that spatial index, a tolerance has been set, which represents the minimum distance at which two points are considered separate. For example, with a tolerance of 0.5 meter, two points that are 30cm far from each other, will be regarded as the same point. Hence, in case that these points are the starting and ending points of a polygon, the polygon is going to be considered closed even if there is a gap of 30cm between the end-points. In other words, the tolerance defines the spatial accuracy of the spatial functions; any function for spatial reasoning at scale below that threshold can not work appropriately. While creating the index the tolerance has been set to 1 meter. Since the road that separates the two buildings is a line (1-Dimension (1D)) narrower than a meter, both buildings appear to have points inside the permitted zone.

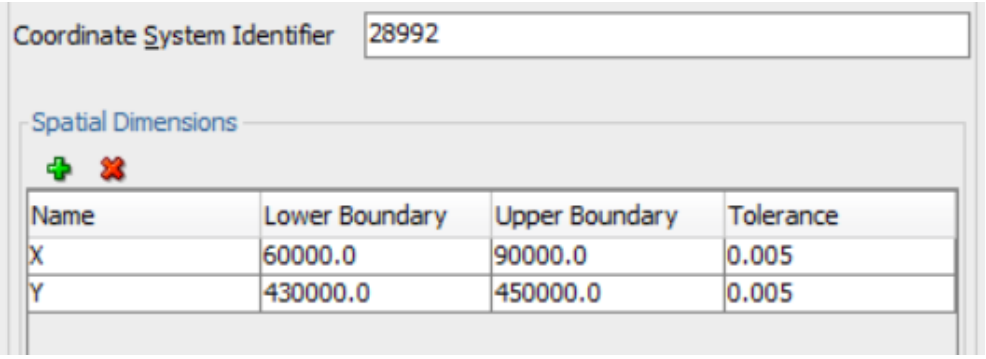


Figure 3.9: Updating the spatial metadata in Oracle dbms tables.

3.3.3 Step 3: Real Data Loading

Progressing from dummy dataset and testing functions into solving the real spatial problem, requires to load into the database all the available datasets. Loading the relational data is a straight forward procedure, where a table is created and then data are loaded using bulk loading Murray et al. [2017]. Three tables are created for that reason, *DELFT_PARCELS*, *GEBOUW_VLAK* and *WEGDEEL_VLAK*. The tables' definitions can be seen in 2.7 on page 13. For each table, spatial metadata should be updated, so that a spatial index can be created for each one of them. The spatial metadata require the SRID of the data, the boundaries of the SRID and a tolerance for each dimension of the data. For all the three datasets, the metadata are the same and are illustrated in Figure 3.9 on page 28. For the spatial plans dataset, a bulk loading is not necessary, as it is a relatively small file with 20 entries and can be loaded using directly the CSV file. Nevertheless, updating the spatial metadata is still a necessity. While updating the spatial metadata of *SPATIAL_PLANS* table, it is very important to notice that the data from PDOK SPARQL endpoint are using SRID: 8307, which corresponds to CRS *ORACLE WGS84*. Loading the full BAG linked data dataset in triplets is not such a trivial task, since it is a file that, in its compressed form, has a size of 13 Gigabyte (GB). A file of such size should be loaded using bulk loading, however, that deemed to be problematic. The reason for that shortcoming is that Oracle Spatial and Graph can support triplets as long as the size of each one of the triplet components (subject-predicate-object) is not be more than 4000 byte (b.)s or 4000 characters. While this is the case for subjects and predicates, the objects that were geometries were sometimes surpassing that size. This can be attributed to the fact that real object geometries have many vertices and the form of the WKT geometry serialization requires many characters to capture them. There are two possibilities to overcome that issue. The first option is to simplify the "problematic" geometries, however due to the size of the data it would be difficult to parse through all of them and simplify them in an error free way. Furthermore, the simplified geometries will lack in precision and will create holes in the dataset. The second way, would be the Oracle's proposed solution to store long geometries. According to that methodology, the geometry-object triplet is substituted with a variable that points to another table and into that table the geometry-object is stored in Character Large Object (CLOB) format. Nevertheless, the huge volume of the data would make it very difficult to identify the long geometries and store them in that way. The process of storing a CLOB is illustrated in Code snippet 3.6. For those reasons, the PDOK's linked data dataset has not been loaded into the database and has not been used at this point.

3.3.4 Step 4: QGIS solution

The QGIS solution is feasible through spatial analysis and querying since all the necessary datasets are loaded into the database as relational data. The data are loaded

Code Snippet 3.6: Storing CLOB variables.

```
-- Create temporary table

create global temporary table local_value$(
 VALUE_TYPE VARCHAR2(10),
 VALUE_NAME VARCHAR2(4000),
 LITERAL_TYPE VARCHAR2(1000),
 LANGUAGE_TYPE VARCHAR2(80),
 LONG_VALUE CLOB)
on commit preserve rows;

CREATE OR REPLACE FUNCTION myGetClobTerm
RETURN MDSYS.SDO_RDF_TERM
AS
 term SDO_RDF_TERM;
BEGIN
 select sdo_rdf_term(
 value_type,
 value_name,
 literal_type,
 language_type,
 long_value)
 into term
 from local_value$
 where rownum < 2;
 RETURN term;
END;

DECLARE
   a CLOB;
begin
a := '
"POLYGON ((244672.317 591105.892,244672.273 591105.892,
                ... ,
                   244687.373 591105.672,244672.317 591105.892))"
               ^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> ';

 -- Insert a row with CLOB geometry

insert into local_value$(
value_type, value_name,
literal_type,language_type,long_value)
values ('LIT','','http://www.opengis.net/ont/
geosparql#wktLiteral','',:a);
-- Use the CLOB constant in a SEM_MATCH query

SELECT *
FROM table(sem_match(
'{ ?cdist ogc:asWKT ?cgeom
 FILTER (
 orageo:withinDistance(?cgeom, oraextf:myGetClobTerm(), 200, "M")) }'
,sem_models('gov_all_vm')
,null, null,null, null, ' ALLOW_DUP=T '));

end;
```

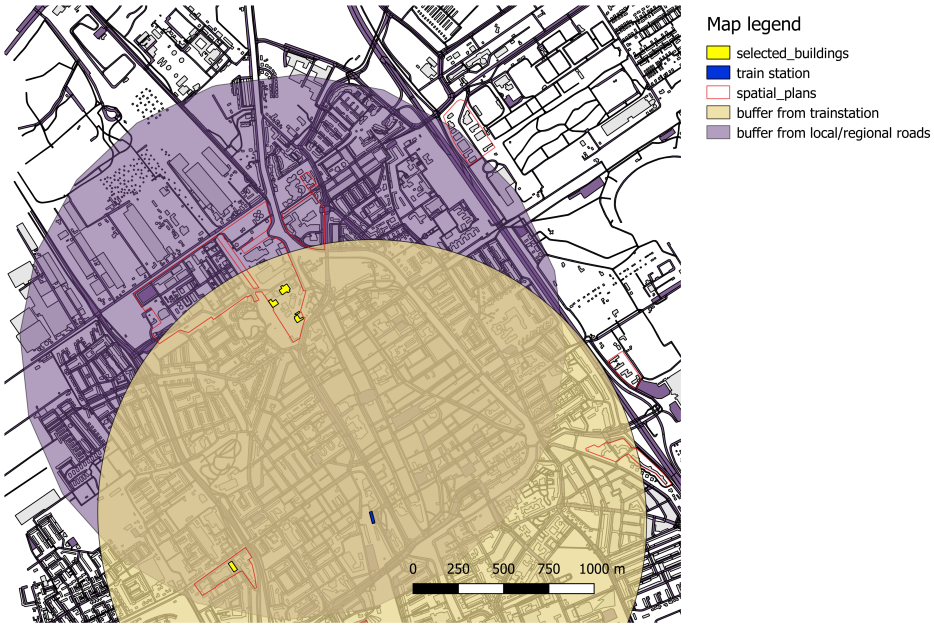


Figure 3.10: QGIS solution.

into QGIS through QGIS-Oracle dbms connection. For the QGIS solution, it is not necessary to follow the specifications that have been set for the linked data approaches, as the solution is not going to be evaluated on its own. It will be used as a benchmark to compare the solutions of the different linked data approaches. For the QGIS solution, the train station is selected through a filter application and then a buffer of 1500 meters is created around it. Likewise, a selection of local main streets is made and a buffer of 1500 meters is created around each one of them. Then, the intersection of those two buffers is calculated. Regarding the buildings, all the buildings that are not inside parcels in Delft are excluded from the selection, as well as buildings that are outside the allowed, by the plan area (the law). Furthermore, a selection upon the previous building-subselection is made, so that only buildings inside the intersection of the buffers is retained. For each one of the remaining buildings, its area is calculated and only those with area over 1500 sq.meters are retained. The selection can been seen in Figure 3.10, where 4 buildings are selected.

3.3.5 Step 5: PDOK approaches - GeoSPARQL endpoint

The first linked data approach offered by Dutch Kadaster PDOK is a GeoSPARQL endpoint, where it is possible for RDF queries to be posed. The endpoint allows for a wide selection of datasets to be queried such as wikipedia's data, however, the most important part is the query of BAG, BRT and BRK datasets, which is currently the only open endpoint to access those data. The endpoint has some pre-written queries that allow the user to familiarize with RDF queries, the linked data Kadaster is offering as well as the different format of results a user can get. More precisely, a user can get a query response in a table, a Google-table, as XML response, as a pivot table or as polygons in Google-maps in case the query returned geometries. The geometries appear to be returning only as WKT, while the geometries in the query can also be given as GML. In case the query returned geometries for which a ***rdf:label*** triplet has also been returned, then those geometries will also appear in Google-maps visualization as a pin with a label. Finally, is is possible to download the results in CSV format containing geometries as WKT. It is important to note that the results are not in triples format and they have the format a normal query response would have.

Code Snippet 3.7: Federated query in ORACLE which access a remote API and returns all the subject-predicate-object triples.

```
'Select ?s ?p ?o
WHERE{
        SERVICE <http://www.example1.org/sparql>{?s ?p ?o}
        }'
```

Before trying the sample query designed for the use case, the very first step is to attempt the limitations of the endpoint in order to determine an approximate of the information that could be drawn out of it. For that purpose, varying amount of data have been queried from it. The next step is to determine the degree it complies with the GeoSPARQL specification and finally, to compare the result produced, to the QGIS benchmark.

3.3.6 Step 6: PDOK approaches - API

PDOK offers access to their linked data datasets via a RESTful API. The RESTful API allows the user to ask the server for services via a protocol such *Hypertext Transfer Protocol (http)* using methods such as **GET** or **POST**. Currently, Dutch Kadaster's API is accessible at ***https://data.pdok.nl/sparql***. It is important to state that, for security reasons, the API is accessed via *https* protocol, which is more secure during transactions as the messages transferred are encoded. In order to post a request to API an endpoint should be used. While there are different endpoints that can be used, e.g. Fuseki, this approach is using Oracle Spatial and Graph, which is capable of posting requests and serving RDF triplets. Oracle Spatial and Graph has been selected because it is supposed to support fully the GeoSPARQL specification, and it will bring a combination of different technologies, databases and linked data. Federated queries have to be used to access a remote API using Oracle Spatial and Graph. Federated queries are **SERVICE** constructs in which the API's Uniform Resource Locator (URL) and the query to be posted on the API, are included. Federated queries are part of normal queries and are executed prior to or after all the other queries have been executed to allow for fine tuning of the performance. By default, federated queries are initially resolved and the full result of them is joined with the results of the other queries. Nevertheless, that can prove to be a poor strategy if the accessed data are huge in volume. The result be narrowed down by executing the rest of the queries and passing their result into the federated query. An example of federated query is illustrated in Code snippet 3.7, which posts a query asking all the triplets on the SPARQL endpoint.

As already discussed, Kadaster's API is using *Hypertext Transfer Protocol Secure (https)* protocol, which is more secure, and the appropriate certificates should be installed prior to querying. The installation of certificates in Oracle Spatial and Graph should be done using the *Oracle Wallet Manager*. The process to install the certificates consists of different steps. Initially, the certificates should be downloaded. For that purpose, *Firefox browser* has been used to download the three necessary certificates (*datapdoknl.crt*, *StaatderNederlandenOrganisatieCA-G2.crt*, *QuoVadisCSP-PKI-OverheidCA-G2.crt*) for *https://data.pdok.nl/sparql*. Furthermore, a wallet for the user has been created and the certificates have been loaded. Finally, using the specific wallet one can query the API. Web page *https://www.w3.org/wiki/SparqlEndpoints* holds an inventory of many open sparql endpoints. Figure 3.11 shows examples accessing two of them.

```
'SELECT *
 WHERE {
  SERVICE <http://dbpedia.org/sparql> {

    SELECT ?s ?p ?o
    WHERE { ?s ?p ?o }
    LIMIT 10
```

	S$RDFTERM	P$RDFTERM	O$RDFTERM
1	<http://www.openlinksw.com/virtrdf-data-formats#default-iid>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
2	<http://www.openlinksw.com/virtrdf-data-formats#default-iid-nullable>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
3	<http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
4	<http://www.openlinksw.com/virtrdf-data-formats#default-iid-nonblank-nullable>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
5	<http://www.openlinksw.com/virtrdf-data-formats#default>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
6	<http://www.openlinksw.com/virtrdf-data-formats#default-nullable>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
7	<http://www.openlinksw.com/virtrdf-data-formats#sql-varchar>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
8	<http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-nullable>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
9	<http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad
10	<http://www.openlinksw.com/virtrdf-data-formats#sql-varchar-dt-nullable>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#Quad

(a) dbpedia endpoint.

```
'SELECT *
 WHERE {
  SERVICE <http://lod.openlinksw.com/sparql/> {

    SELECT ?s ?p ?o
    WHERE { ?s ?p ?o }
    LIMIT 10
```

	S$RDFTERM	P$RDFTERM	O$RDFTERM
1	<http://www.openlinksw.com/schemas/virtrdf#DefaultServiceMap-pquad>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapAT
2	<http://www.openlinksw.com/schemas/virtrdf#DefaultQuadMap-pquad>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapAT
3	<sys:qmv-bfd922693595ebbb4ef4058dc27adbb6-atable-r-DB.DBA.RDF_IRI_RANK_C>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapAT
4	<sys:qmv-e44f89e00951ace16b9a2e9df325f4c8-atable-r-DB.DBA.RDF_IRI_RANK_C>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapAT
5	<http://www.openlinksw.com/schemas/virtrdf#DefaultServiceMap-G-col-G>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo
6	<http://www.openlinksw.com/schemas/virtrdf#DefaultServiceMap-O-col-O>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo
7	<http://www.openlinksw.com/schemas/virtrdf#DefaultServiceMap-P-col-P>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo
8	<http://www.openlinksw.com/schemas/virtrdf#DefaultServiceMap-S-col-S>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo
9	<http://www.openlinksw.com/schemas/virtrdf#DefaultQuadMap-G-col-G>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo
10	<http://www.openlinksw.com/schemas/virtrdf#DefaultQuadMap-O-col-O>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.openlinksw.com/schemas/virtrdf#QuadMapCo

(b) BBC endpoint.

Figure 3.11: Example accessing SPARQL endpoints.

Code Snippet 3.8: Direct Mapping of each table in separate views.

```
exec sem_apis.create_rdfview_model('DELFT_PARCELS_view',
sys.odcivarchar2list('DELFT_PARCELS_S'),
'http://www.example/geometries/',
options=>'CONFORMANCE=T');

exec sem_apis.create_rdfview_model('GEBOUW_VLAK_view',
sys.odcivarchar2list('GEBOUW_VLAK_S'),
'http://www.example/geometries/',
options=>'CONFORMANCE=T');

exec sem_apis.create_rdfview_model('WEGDEEL_VLAK_S_view',
sys.odcivarchar2list('WEGDEEL_VLAK_S'),
'http://www.example/geometries/',
options=>'CONFORMANCE=T');
```

Code Snippet 3.9: Direct Mapping of each table in one view.

```
BEGIN
 sem_apis.create_rdfview_model(
 model_name => 'geometries_dir_view',
 tables => SYS.ODCIVarchar2List('DELFT_PARCELS_S',
 'GEBOUW_VLAK_S','WEGDEEL_VLAK_S','SPATIAL_PLANS' ),
 prefix => 'http://www.example.org/geometries/',
 options => 'CONFORMANCE=T'
 );
END;
```

3.3.7 Step 7: Oracle Spatial and Graph approaches - Direct mapping

Before moving to mapping relational data to RDF triplets, the spatial problem could have been solved by using the linked data dataset provided by Kadaster. Nevertheless, this step is not possible due to problems storing long geometries in Oracle Spatial and Graph, as mentioned in sub-Section 3.3.3, page 28.

Oracle Spatial and Graph supports an important feature, accessing relational data via RDF Views as if they were linked data. In this case, the long geometries problem is not present, since the data are not stored as linked data. The process of creating an RDF View consists of designing the way the relational data are going to be translated into RDF triples, which is called mapping. There are two ways to perform the mapping, either by direct mapping or by performing relational to RDF mapping (RDB2RDF)[W3C, 2012] using Turtle language [W3C, 2014a]. Direct mapping can be seen in Code snippets 3.8 and 3.9, while the R2RML mapping can be seen in Appendix B.

The first and most simple one is the direct mapping. The direct mapping in Oracle Spatial and Graph is performed according to W3C Recommendation [Arenas et al., 2012]. According to the recommendation, the *direct mapping* takes a relational database and generates an RDF graph that is called a "direct graph". In direct mapping, a base Internationalized Resource Identifier (IRI) is used to create the linked data from relational data. To form the triple, the Primary Key of the table is concatenated with the IRI prefix to form the *subject* part of the triple. The *predicate* part is formed out of the column name concatenated with the IRI prefix. Finally, the *object* is formed from the column value concatenated with the IRI prefix. It should be

```
SELECT r, wkt
FROM TABLE(SEM_MATCH( '
SELECT ?r ?wkt where {
        ?r :PARCEL_GEOM ?wkt
FILTER (ogcf:sfTouches(?wkt, "<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83791.28 444861.909, 83765.213
        }
',
SEM_MODELS('DELFT_PARCELS_view' )  , NULL, SEM_ALIASES(SEM_ALIAS('','http://www.example/geometries/DELFT_PARCELS_S#')
```

	R	WKT
1	_:m222mBlankNode20DCFEBC8C47477E	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83765.213 444853.864,
2	_:m222mBlankNode2E10280123A9E15C	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83815.92 444856.296,
3	_:m222mBlankNode76797E0B6627A2F6	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83810.691 444873.255,
4	_:m222mBlankNode8FA97D399946C53F	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83812.194 444868.382,
5	_:m222mBlankNode9241AEC5372A647C	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83768.932 444841.815,
6	_:m222mBlankNodeF988B29223214942	<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83557.153 444769.568,

Figure 3.12: Direct mapping functionality testing.

noted that if the column value is *NULL*, then no triple is generated. Furthermore, in case there are foreign keys, triples from the foreign key column names, the referenced table, and the referenced column names are generated. An illustration of this methodology is depicted in Figure 3.13. If there is no primary key, then the *subject* will be a **blank node**. Another very important aspect of mapping approaches is that while the mapping is stored as an RDF model, it contains only metadata. In other words, it contains information on how to perform the mapping when the data are about to be queried and not the triples themselves.

According to that methodology, an **IRI prefix** was selected *http://www.example.org/-geometries/*. While creating the direct mappings there were two different methodologies followed. Initially, the tables *Delft_Parcels*, *Buildings*, *Spatial_plans*, *Roads* were mapped under a single mapping process, as shown in Code snippet 3.9. What is not very straightforward in this kind of mapping is that while the links defining a class using *rdf:type* have the specified IRI prefix, all the other predicates are formed by the concatenation of the IRI prefix with the column name in the form of *IRI_prefix/Column_name#*, (e.g. parcels will have an *rdf:type http://www.example.org/geometries/DELFT_PAR* and a geometry *pl:GEOM*), where pl is an alias of *http://www.example.org/geometries /DELFT_PARCELS#*. In Figure 3.12, a spatial function is tested against the RDF View to make sure it is working as intended. After the mapping, the sample query was run on the data.

For the second methodology of direct mapping, a mapping process is carried out for each table using the same **IRI** prefix resulting in 3 RDF views for 3 tables. To combine the different RDF views, each one is accessed using a *SERVICE* construct, which in Oracle is used to query RDF views. In other words, the query asks for Oracle database to create a separate query for each different view and combine them afterwards. This process can be seen in Code snippet 3.9. Due to the fact that this methodology results in very slow queries, it was not used with the sample query.

3.3.8 Step 8: Oracle Spatial and Graph approaches - R2RML

The second methodology of mapping is the R2RML mapping. In R2RML mapping the Subject-Predicate-Object triples must specifically be designed for each table and each column.

While Subject and Predicate, according to RDF specification must be or must be converted into URLs, the Object can be a literal or a blank node. In each part of the triple, prefixes and/or suffixes can be added to convert it to the appropriate format. While R2RML mapping offers more freedom for mapping, it is, at the same time, a more complicated process. The mapping in that methodology must be written in

People

PK		→ Addresses(ID)	→ Department(name, city)	
ID	fname	addr	deptName	deptCity
7	Bob	18	accounting	Cambridge
8	Sue	NULL	NULL	NULL

Addresses

PK		
ID	city	state
18	Cambridge	MA

Department

PK	Unique Key		→ People(ID)
ID	name	city	manager
23	accounting	Cambridge	8

(a) Relational Table to be mapped.

```
@base <http://foo.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18> .
<People/ID=7> <People#deptName> "accounting" .
<People/ID=7> <People#deptCity> "Cambridge" .
<People/ID=7> <People#ref-deptName;deptCity> <Department/ID=23> .
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .

<Department/ID=23> rdf:type <Department> .
<Department/ID=23> <Department#ID> 23 .
<Department/ID=23> <Department#name> "accounting" .
<Department/ID=23> <Department#city> "Cambridge" .
<Department/ID=23> <Department#manager> 8 .
<Department/ID=23> <Department#ref-manager> <People#ID=8> .
```

(b) Triples Generated from mapping.

Figure 3.13: Example of direct mapping [Arenas et al., 2012].

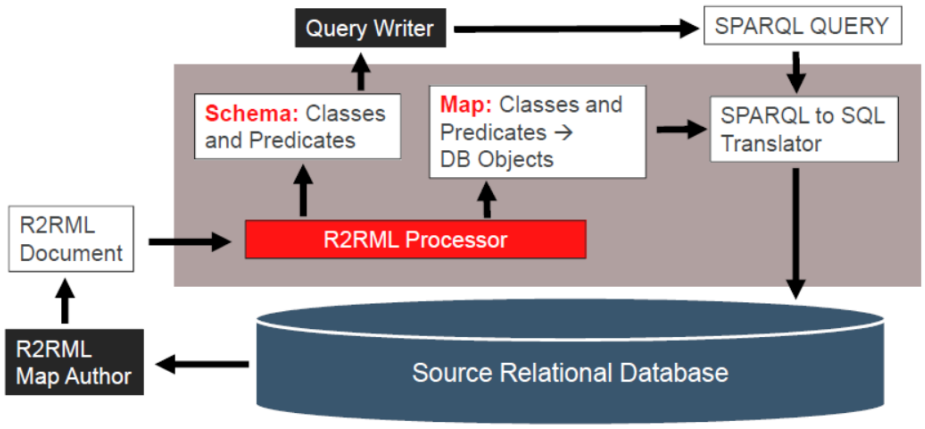


Figure 3.14: R2RML mapping Process [Wu, 2014].

	WKTB$RDFTERM	COMP$RDFTERM
1	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87730.889 447811.1...	"true"^^<http:/
2	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87673.466 447578.2...	"false"^^<http:
3	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87676.439 447597.8...	"false"^^<http:
4	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87673.11 447645.84...	"false"^^<http:
5	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87704.605 447654.5...	"false"^^<http:

```
SELECT wktb$rdfterm, comp$RDFTERM
FROM TABLE(SEM_MATCH( '
SELECT ?wktb ?comp  WHERE {

?b rdf:type ex:building.
?b ex:geom ?wktb.
bind(ogcf:sfIntersects(?wktb, "<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87730.889 447811.157,
```

Figure 3.15: Minimum working example for R2RML mapping with a spatial function that checks if the polygons in the database touch a certain polygon.

Turtle/text format which is, according to W3C [2014a], a language that allows RDF graphs to be written in text form. Turtle/text then should be converted into N-triples, which is another encoding format for RDF graphs [Beckett, 2014]. N-triples, Turtle and RDF graphs are compatible.

The process of how R2RML mapping works is illustrated in Figure 3.14, where it shows that the queries are re-written and solved on relational data-base level, thereby explaining the reason spatial indices are required on the relational data-bases. Since Turtle is text based, it was written using ***Sublime 3*** according to the specification by W3C and has been converted into N-triples using ***EasyRdf*** [Humfrey, 2018]. The N-triples should be inserted into a staging table, which will be used to create the RDF View, hence using ***Python3***, the N-triples are converted into *INSERT* statements and then inserted into the staging table that was created for them. Only after that process the RDF View can be created. The process can be found in Appendix B. In Figure 4.27 on page 62, a minimum working example is depicted, designed to make sure that the RDF view has been correctly designed and that the spatial functions work. Moving on, the sample query has been used on R2RML mapping RDF view.

3.3.9 Testing

Throughout the methodology steps described above, there were tests to determine whether the results from each step were correct.

Initially, as shown in Code snippet 3.1, the geometries of dummy data that were created were validated for invalid geometries. Moreover, inserting a duplicate geometry was attempted, which didn't appear in the queries but was stored in the

```
SELECT b$RDFTERM, area$RDFTERM
FROM TABLE(SEM_MATCH(
'SELECT ?b ?area WHERE { ?b rdf:type :lot.
?b ogc:hasGeometry ?geom.
?geom ogc:asWKT ?wkt.
bind(orageo:area(?wkt, "units=SQ_M") as ?area)}',
SEM_MODELS('geometries') , NULL, SEM_ALIASES(SEM_ALIAS(
```

9	<http://www.example.org/geometries/lot18>	"73.1230047708009"^
10	<http://www.example.org/geometries/lot19>	"72.654884714445"^
11	<http://www.example.org/geometries/lot1>	"148.009764503988"^
12	<http://www.example.org/geometries/lot20>	"74.0470468141117"^
13	<http://www.example.org/geometries/lot21>	"74.3161685908562"^

Figure 3.16: Checking the area of dummy data geometries.

database. Creating triples with non-sequential ids or duplicate ids was also attempted and the triplets were able to be retrieved normally without any error being reported. The purpose of this test was to check Oracle's Spatial and Graph performance on possible invalid data, such as a duplicate triplet. Those tests are illustrated in Code snippet 3.10. To ensure that there is not going to be a future problem, on the table that is holding the id-triplet combination, *id* was selected as primary key, hence no duplicates are allowed. Moreover, the geometry of *lot1* has been designed to be 148 sq.meters, which is an integer and it also fits between the designed roads. The area of the same object has been calculated using GeoSPARQL queries, which can be seen in Figure 3.16 the GeoSPARQL *area function*.

On the second step the performance of all GeoSPARQL and Oracle linked data functions have been tested, as a part of forming the sample query, to ensure that similar functions yield similar results. During the loading of the relational data, the geometries were validated too with many different tolerances, to determine if the errors were due to accuracy or invalid geometries.

In PDOK GeoSPARQL approach there were tests on the error reporting of the endpoint as well as an attempt to get different amount of data in order to determine the point where the server would stop responding. On PDOK API different browsers to download the certificates were used and other APIs were accessed successfully.

Finally, on the mapping approaches the "area test" was performed again. Moreover, the area of the buildings has been calculated on relational data and has been stored in the table and then it has been compared to the results of GeoSPARQL functions to determine if the accuracy has been retained and there was no loss of accuracy.

Code Snippet 3.10: Tests performed on dummy data.

```
--the index of the triple, number 25, is a duplicate, however no error was g

INSERT INTO geometries_rdf VALUES (25,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/address5>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
' <http://www.example.org/geometries/address> '));


--the indices of the triples are not sequential
-- no error was generated and it was successfully queried

INSERT INTO geometries_rdf VALUES (25,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/address5>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
' <http://www.example.org/geometries/address> '));

INSERT INTO geometries_rdf VALUES (2555,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/address5>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
' <http://www.example.org/geometries/address> '));



--That part has been used to test invalid and duplicate geometries
-- Duplicated coordinates - rectifiable

insert into geometries_rdf values
(13, sdo_rdf_triple_s('geometries','<http://www.example.org/geometries/lot4>
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'"POLYGON((1.0 2.0, 3.0 2.0, 1.0 4.0, 1.0 2.0, 1.0 2.0))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'));

-- Boundary is not closed - rectifiable

insert into geometries_rdf values
(14, sdo_rdf_triple_s('geometries','<http://www.example.org/geometries/lot5>
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'"POLYGON((1.0 2.0, 3.0 2.0, 3.0 4.0, 1.0 4.0))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'));

insert into geometries_rdf values
(15, sdo_rdf_triple_s('geometries', '<http://www.example.org/geometries/lot6
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'"POLYGON((1.0 2.0, 3.0 2.0, 1.0 4.0, 1.0 2.0, 1.0 2.0))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'));

EXECUTE sem_apis.validate_geometries(model_name=>'geometries',
SRID=>28992,tolerance=>1.0, options=>'STANDARD_CRS_URI=T RECTIFY=T');

--Selection of object with geometry and geometry representation
  used to check if invalid and duplicate geometries return.
select o, geom, wkt
 FROM TABLE(SEM_MATCH('{
    ?o gsp:hasGeometry ?geom.
    ?geom gsp:asWKT ?wkt }'))
```

4 RESULTS

The results of each step of the methodology will be presented and discussed in this Chapter. One recurring problem that affected the course of this thesis is that in Oracle Spatial and Graph the Administrator is different from the User. Most of the actions required to perform in this thesis, require from the User to either have privileges given by the Administrator, or have the Administrator perform the specific action. In many steps of this thesis instead of seeking Administrator help or privileges, a work-around was attempted using alternative ways; in some steps that was not possible. Going through the results, any time there was a work-around it is going to be mentioned.

4.1 RESULTS OF STEP 1: CREATE DUMMY DATA AND FAMILIARIZE WITH ORACLE SPATIAL AND GRAPH

When designing the dummy data on QGIS, the roads have been designed as lines (1D), subsequently the parcels that were created were separated by approximately 0.5 meters. This fact should be taken into consideration when the threshold of the spatial index is being determined, which will result in different outcomes of the spatial analysis. In the dummy data case, at the first version of the data, the distance between two buildings was approximately 0.5 meters, however, the tolerance of the spatial index has been set at 1 meter. This resulted in strange outcomes of the spatial functions. In Figure 4.1 the highlighted yellow buildings are the result of the *INSIDE* spatial function, nevertheless, the top building was not inside the spatial plan zone and it should not be part of the result. The way the dummy data were created, the 1 meter tolerance, along with the fact that instead of an *inside* spatial function the *touches* spatial function has been used, made that building part of the result. At this step, it was asked from the administrator to re-create the spatial index using a finer threshold. Moreover, the dummy data have been "transported" into coordinates that would be within the Netherlands (EPSG:28992) and they would be covered by the spatial index created and they have been scaled up so that the distance between buildings would be greater than the tolerance. Furthermore, the dummy data was designed in QGIS and exported in CSV format. While it is possible to use bulk load to load the data into a staging table and then load them into the RDF model, due to the small volume of the data, it was regarded as an easier procedure to use *INSERT* statements to load the data. Due to the fact that the dummy data was changed multiple times, a python code, available in Appendix A on page 78, was designed to read the csv file and produce the *INSERT* statements.

Except from the tolerance related problems, there were also SRID related problems. More specifically, when the data that were inserted did not have their SRID specified, Oracle Spatial and Graph regarded those data to have a simple planar SRID 262148 Non-Earth (meters) and any spatial query attempted returned wrong. The reason for that error is that when the coordinates of the data were "on-the-fly" converted into the SRID of the spatial index, their coordinates were outside of the spatial index boundaries, thus the spatial functions were not working as they should. Inserting the data with the SRID epsg:29882 solved that problem. This allowed for more testing to take place. The first test was to insert an invalid geometry (polygon not closed) and a duplicate geometry and then validate the geometries. As expected, the invalid

Figure 4.1: Dummy dataset.

geometry was reported as problematic and was attempted to be fixed, while the duplicate geometry returned no error. Nevertheless, querying the data did not return the duplicate geometry. After removing the invalid geometry and the duplicate geometry, there were triplets inserted that either didn't have a sequential id as the rest, or their id was duplicate. Those two examples did not produce any errors, so to solve the duplicate id problem and protect the process from a future problem, the *id* was declared as primary key in the database.

Finally, the geometries were stored only with the WKT serialization, as it was regarded easier to read and it would be redundant to store both serializations. Nevertheless, a GML representation was stored and retrieved, and a spatial query checking for intersections between a GML and WKT representations, shown in Code snippet 4.1, was tested. This procedure intended to check if Oracle Spatial and Graph can support WKT and GML serializations in spatial functions, and it worked well.

4.2 RESULTS OF STEP 2: FORMING A SAMPLE SPATIAL QUERY

At this step, the sample spatial query was designed. The sample spatial query should contain the comparisons that were mentioned in the scope of the research and at the same time, due to the fact that is going to be used against different linked data approaches, it should be ensured that it could be supported by them as much as possible.

There are 18 spatial functions mentioned in GeoSPARQL specification [Perry and Herring, 2012]. 10 of them are non-topological and 8 of them are topological. Those

Code Snippet 4.1: Intersection spatial function between objects with GML and WKT serializations.

```
SELECT comp FROM TABLE(SEM_MATCH('
SELECT (ogcf:sfIntersects("
<gml:Point srsName=\"SDO:8307\" xmlns:gml=\"http://www.opengis.net/gml\">
gml:pos srsDimension=\"2\">-122.25 37.46 </gml:pos></gml:Point>"
^^"<http://www.opengis.net/ont/geosparql#gmlLiteral>",
"<http://www.opengis.net/def/crs/OGC/1.3/CRS84> POINT (-122.25 37.46)"
^^<http://www.opengis.net/ont/geosparql#wktLiteral>) as ?comp)"
```

Non-topological functions	Topological functions
ogcf:boundary	ogcf:sfWithin
*ogcf:envelope	ogcf:sfContains
*ogcf:symDifference	ogcf:sfCrosses
*ogcf:getSRID	ogcf:sfDisjoint
*ogcf:intersection	ogcf:sfEquals
*ogcf:union	ogcf:sfIntersects
*ogcf:convexHull	ogcf:sfOverlaps
*ogcf:buffer	ogcf:sfTouches
*ogcf:difference	—
*ogcf:distance	***ogcf:relate**

Noted with asterisk (*) are the functions that can be found both in GeoSPARQL and Oracle Spatial and Graph implementations.

Table 4.1: GeoSPARQL Spatial Functions

functions are shown in Table 4.2. Moreover, there is *ogcf:relate* function which is an "umbrella" function that determines the topological relation of two spatial objects. Out of those functions, at least 1 from each category (ogcf:relate excluding) should be used for the sample query.

On the other hand, Oracle Spatial and Graph supports 21 Spatial Functions. Out of them, 20 are Non-topological Functions and only 1 is considered Topological Functions. Oracle, instead of designing a spatial function for each topological relationship, uses the *orageo:relate* function with which all the topological relationships can be determined. While many non-topological functions are the same as GeoSPARQL specification (e.g. *ogcf:distance* and *orageo:distance*), there are many non-topological relationships that are not available in GeoSPARQL specification, such as *orageo:area*, which are useful for our case study. Hence, it will be employed when available. The available Oracle Spatial and Graph spatial functions are shown in Table 4.2

At this point, it is good to remind that the criteria set for the use case are the following:

- the distance from train station to be less than 1500 meters.
- the building should be inside the permitted zone.
- the area of the building should be over 1500 sq.meters .
- the distance from a local main road should be less than 1500 meters.
- the building should be located in a parcel in Delft.

In table 4.3 the different criteria (first column) set for the use case are translated into queries (second column) and the kind of query is described (third column). More specifically, there are 2 Topological and 3 Non-topological queries and there is 1 simple query which satisfies SPARQL querying.

Non-topological functions	Topological functions
orageo::aggrCentroid	*orageo:relate
orageo:aggrConvexHull	
orageo:aggrMBR	
orageo:aggrUnion	
orageeo:area	
*orageo:union	
*orageo:convexHull	
*orageo:buffer	
*orageo:difference	
*orageo:distance	
orageo:centroid	
*orageo:getSRID	
*orageo:intersection	
*orageo:mbr	
orageo:length	
orageo:nearestNeighbor	
orageo:sdoDistJoin	
orageo:sdoJoin	
orageo:withinDistance	
*orageo:xor	

Noted with asterisk (*) are the functions that can be found both in GeoSPARQL and Oracle Spatial and Graph implementations.

Table 4.2: Oracle Spatial and Graph Spatial Functions

Criterion	vocabulary	Kind of query
distance from train station	ogcf:distance	NT
distance from local main road	ogcf:distance	NT
building in parcel	ogcf:sfWithin	T
building inside permitted area	ogcf:sfWithin	T
parcel in Delft	rdf:label	SQ
area of building	orageo:area	NT

NT:Non-topological Relationship, T: Topological Relationship, SQ:SPARQL Querying

Table 4.3: Criteria and Satisfaction of Scope of research

```
SELECT dp.OBJECT_ID ,dp.PARCEL_GEOM,SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(dp.PARCEL_GEOM, 0.005)
   FROM DELFT_PARCELS dp where SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(dp.PARCEL_GEOM, 0.005) != 'TRUE';
```

SQL | All Rows Fetched: 30 in 5.729 seconds

	OBJECT_ID	PARCEL_GEOM	SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(DP.PARCEL_GEOM,0.005)
1	310007453	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <2>] [Edge <1>]
2	310039578	[MDSYS.SDO_GEOMETRY]	13356 [Element <1>] [Coordinate <5>] [Ring <1>]
3	310042603	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <56>] [Edge <57>]
4	310064283	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <180>] [Edge <178>]
5	310095830	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <50>] [Edge <48>]
6	310095831	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <47>] [Edge <46>]
7	310124679	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <39>] [Edge <40>]
8	310136041	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <30>] [Edge <31>]
9	310160761	[MDSYS.SDO_GEOMETRY]	13356 [Element <1>] [Coordinate <2>] [Ring <1>]
10	310177461	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <70>] [Edge <68>]
11	310268771	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <3>] [Edge <4>]
12	310343682	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <145>] [Edge <146>]
13	310344421	[MDSYS.SDO_GEOMETRY]	13349 [Element <1>] [Ring <1>] [Edge <52>] [Edge <54>]

(a) Validate with tolerance 0.005m.

```
SELECT dp.OBJECT_ID ,dp.PARCEL_GEOM,SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(dp.PARCEL_GEOM, 0.00001)
   FROM DELFT_PARCELS dp where SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(dp.PARCEL_GEOM, 0.00001) != 'TRUE';
```

(b) Validate with tolerance 0.00001m.

Figure 4.2: Relational data validation.

4.3 RESULTS OF STEP 3: DATA LOADING

During the loading of the relational data, a testing to validate the loaded geometries was carried out. As mentioned before, Oracle uses *Tolerance* for many of the spatial functions and queries. At the validation step, the tolerance can lead to invalid geometries in case it is not set correct. In Figure 4.2a, the tolerance has been set to *0.005* which led many points, with a distance of less than 0.005m from each other, to be regarded as the same point and resulted in errors with codes 13349 and 13356. These errors are related to geometries self-intersecting and invalid holes. When the tolerance was changed to *0.00001*m there were no such issues.

Apart from validating relational data, loading Kadaster's BAG linked data dataset was attempted. However, for linked data in Oracle Spatial and Graph a column can't have values that are more than 4000b. or 4000 characters. When a value of larger size is required to be loaded into the database, a CLOB data-type should be used. In that case, the long-string value is stored in a table as a pair *pointer-CLOB* and the long-string value is substituted by the pointer. When the long-string value is queried, the database will look for the pointer into the separate table, holding the pairs *pointer-CLOB*, and it will yield the CLOB. Creating a triplet containing a CLOB value in graph *geometries* is shown in Code Snippet 4.2. Furthermore, in real data, there might be values such as descriptions and/or geometries that are considered long-string values. Another simple method to overcome that shortcoming would be to simplify the geometries, thus reducing the vertices and the length of the serialization. However, that would reduce the accuracy of the results and it could leave holes in the dataset and invalid geometries.

Due to the fact that the BAG dataset is large in volume, as it is close to 13 GB in its compressed form, going through each one of the triplets to identify the long-string values and substitute them with a CLOB would be a daunting task. At this point of the thesis, it was decided not to pursue this methodology further.

Code Snippet 4.2: Storing CLOB variables.

```
-- Create temporary table

create global temporary table local_value$(
 VALUE_TYPE VARCHAR2(10),
 VALUE_NAME VARCHAR2(4000),
 LITERAL_TYPE VARCHAR2(1000),
 LANGUAGE_TYPE VARCHAR2(80),
 LONG_VALUE CLOB)
on commit preserve rows;

CREATE OR REPLACE FUNCTION myGetClobTerm
RETURN MDSYS.SDO_RDF_TERM
AS
 term SDO_RDF_TERM;
BEGIN
 select sdo_rdf_term(
 value_type,
 value_name,
 literal_type,
 language_type,
 long_value)
 into term
 from local_value$
 where rownum < 2;
 RETURN term;
END;

DECLARE
   a CLOB;
begin
a := '
"POLYGON ((244672.317 591105.892,244672.273 591105.892,
                ... ,
                   244687.373 591105.672,244672.317 591105.892))"
               ^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> ';

 -- Insert a row with CLOB geometry

insert into local_value$(
value_type, value_name,
literal_type,language_type,long_value)
values ('LIT','','http://www.opengis.net/ont/
geosparql#wktLiteral','',:a);
-- Use the CLOB constant in a SEM_MATCH query

SELECT *
FROM table(sem_match(
'{ ?cdist ogc:asWKT ?cgeom
 FILTER (
 orageo:withinDistance(?cgeom, oraextf:myGetClobTerm(), 200, "M")) }'
,sem_models('gov_all_vm')
,null, null,null, null, ' ALLOW_DUP=T '));

end;
```

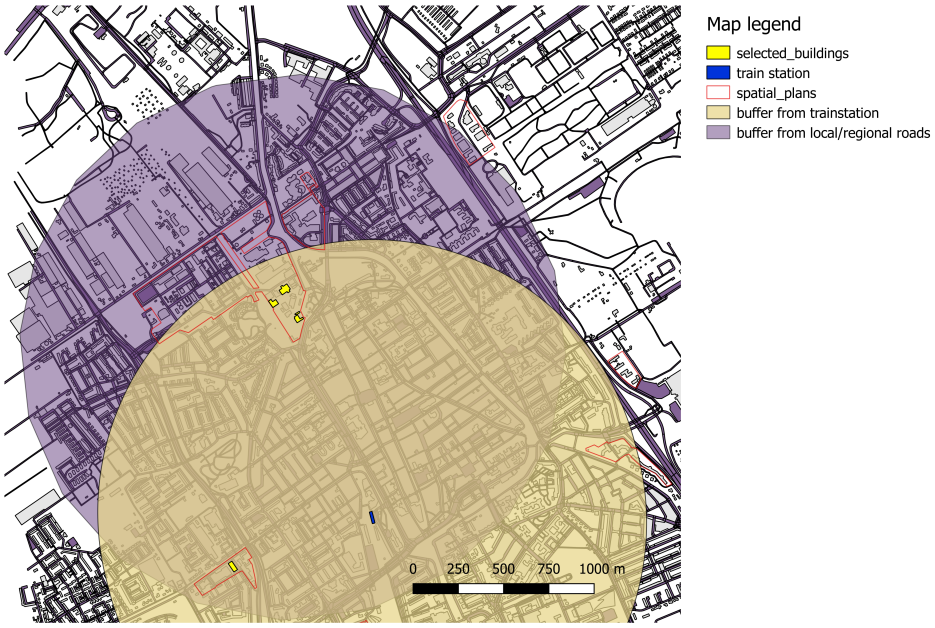


Figure 4.3: The QGIS solution of the use case.

4.4 RESULTS OF STEP 4: QGIS SOLUTION

For implementing the QGIS solution, the order of spatial queries and spatial analyses are selected in order to reduce the crashes due to the visualization of much information. Thus, it might not be the most straight forward way. To solve the spatial problem with QGIS, the data has been loaded into Oracle database, and a connection has been made between Oracle and QGIS to load them as layers. After the loading, a buffer of 1500 meters has been created around train station. At this point a mistake in the data has been noticed. There are two train stations in Delft, namely *Delft centraal* and *Delft Zuid*, however, the data have only one, the central station. The missing *Delft Zuid* station results in one less buffer zone. After the train station buffer, another buffer around local main roads has been created. The categorization of roads was not checked for correctness. The road-buffers were dissolved and the intersection between the road-buffers and the train-buffer was saved. The segments of spatial plans that are inside the previous intersection was the result of another intersection analysis. The buildings that are inside parcels in Delft were saved after a *within* spatial query. Next, the buildings of the previous query were checked for being within the intersection of spatial plans and the buffers. The final part was to filter out buildings that had an area of less than 1500 sq. m. The whole process is visualized in Figure 4.3 and the resulting buildings are shown in Figure 4.4

This was the benchmark solution, the one that every other solution was compared with, which resulted in 4 buildings.

4.5 RESULTS OF STEP 5: PDOK APPROACHES – GEOSPARQL ENDPOINT

The first linked data approach that is available from Dutch Kadaster PDOK is the GeoSPARQL endpoint. In this method a scale-up approach has been used, meaning that instead of using the complete sample query, parts of it are being implemented step by step. The reason the scale-up approach was followed is to be able to deter-

	IDENT	TYPEGEBOUW	area2
1	NL.TOP10NL.102579851	overig	1358.139969
2	NL.TOP10NL.102579466	overig	1282.779743
3	NL.TOP10NL.102571240	overig	2082.219711
4	NL.TOP10NL.102555243	overig	1327.979393

Figure 4.4: The buildings of the QGIS solution of the use case.

mine the structure of the data and the degree it supports the GeoSPARQL implementation. However, the server was timing out during the process and it did not allow for a solution to be reached. The process is going to be explained in more details.

The first successful attempt aims to retrieve the polygon of Delft municipality from PDOK's GeoSPARQL endpoint; the resulting polygon is visualized in Figure 4.5. As a next step, there were attempts to retrieve buildings inside the municipality of Delft. The first attempt (figure 4.6) returned five buildings, however, the second attempt (figure 4.7) that attempted to retrieve more buildings resulted in a server time-out. By trial and error, the maximum number of buildings that were able to be retrieved were around 60 (accessed at 2018, May 18). The server seemed to be more unstable as the number of buildings to be retrieved increased. For this number of buildings sometimes the procedure yielded results and sometimes the server timed-out.

The final step would be to proceed with a more complicated analysis and try to reach the sample query that was build for comparison, as it was mentioned in Code snippet 3.5 on page 27. Cadastre's SPARQL endpoint also offers the plans that determine where it is possible to build a shopping center, thus it is not necessary to connect it with another source or perform post-analysis of the results. In Figures 4.10, 4.11, 4.12 and 4.13, the attempts to use a more complicated query into the SPARQL endpoint are depicted. All four queries attempt to retrieve the buildings that are located in Delft and inside the planning zones. However, all the attempts, even if they were asking for 1 building, have failed due to server time-out. At this point, it is important to note that the endpoint does support GeoSPARQL spatial functions, however, it does not support the Oracle linked data spatial functions, hence there is no area function to measure a building's area. This means that it would not be able to solve the whole spatial problem as described in the use case. It is important to note that the ontology of the spatial objects in the endpoint does not follow the GeoSPARQL ontology for spatial features.

Despite the failed attempts, using the query depicted in Figure 4.8a, it was possible to download the planning zones within the Delft municipality in CSV format, as illustrated in Figure 4.9. The CSV needs to be formatted prior to any use.

Even though the endpoint supports GeoSPARQL functionalities and offers a wide range of datasets, its use is limited by a number of factors. Initially, as mentioned above, there are cases when the server times-out. This is a server side problem where the process of retrieving more data takes more time than the preset time-out time. In any case, it limits the use of the endpoint significantly, to the point that it can be mostly used for data exploration and educational purposes. Another important issue that might arise with the server is that the results it returns are inconsistent and sometimes, when there should be an error produced, there is no indication or warning that something went wrong. A comparison between Figure 4.8a and Figure 4.8b shows that for almost the same query, a different number of results is returned. In the first case, which is the correct one, 20 results are returned while 519 results are re-

Figure 4.5: Delft municipality (within blue zone).

```
9  SELECT  ?wkt2
10 where{
11  ?gemeente a brt:Gemeente .
12   ?gemeente rdfs:label  "Delft"@nl.
13   ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
14
15
16   ?gebouw a brt:Gebouw.
17   ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
18   ?gebouw brt:status ?status.
19   filter(ogcf:sfWithin(?wkt2, ?wkt1))
20 } limit 5
```

Table | Response | Pivot Table | Google Chart | Geo

ip: Add a label variable prefixed with the geo variable name to show popups on the map. E.g. wkt

Figure 4.6: Retrieving five buildings in Delft municipality (marked as blue polygons).

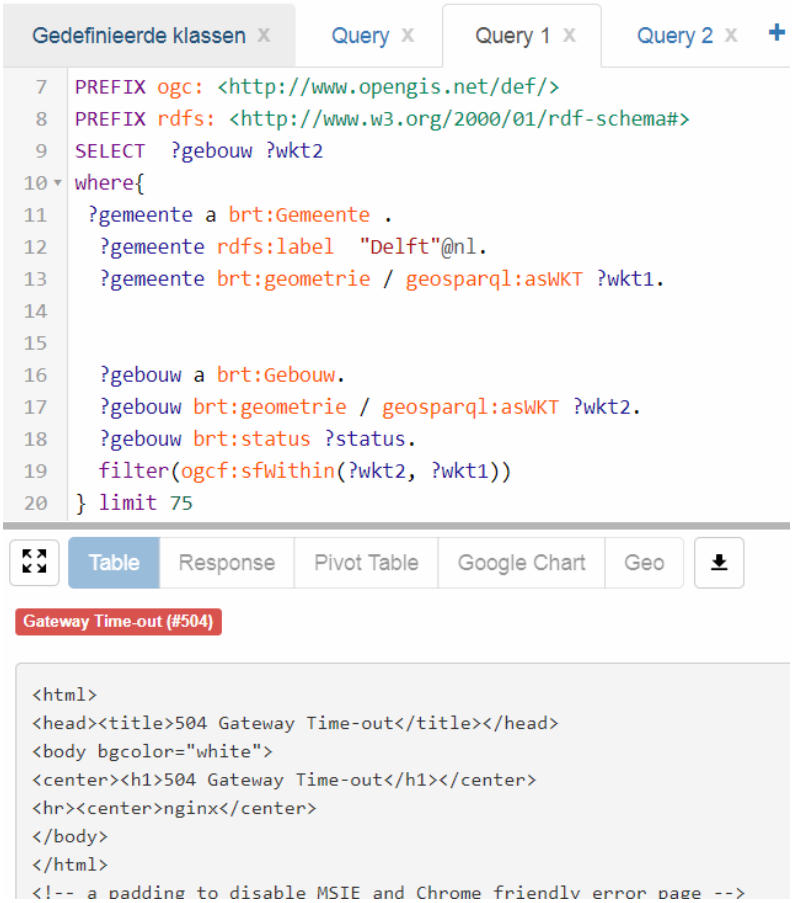


Figure 4.7: Retrieving seventy five buildings in Delft municipality.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX brt: <http://brt.basisregistraties.overheid.nl/def/top10nl#>
PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>
SELECT distinct ?wkt3 WHERE {
  ?gemeente a brt:Gemeente.
  ?gemeente rdfs:label "Delft"@nl.
  ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
  ?plan a brt:Bedrijventerrein.
  ?plan brt:geometrie / geosparql:asWKT ?wkt3.
  FILTER (ogcf:sfWithin(?wkt3, ?wkt1))}
```

Table | Response | Pivot Table | Google Chart | Geo

Showing 1 to 20 of 20 entries (in 2.575 seconds)

(a) Correct Result: only the 20 distinct zones returned.

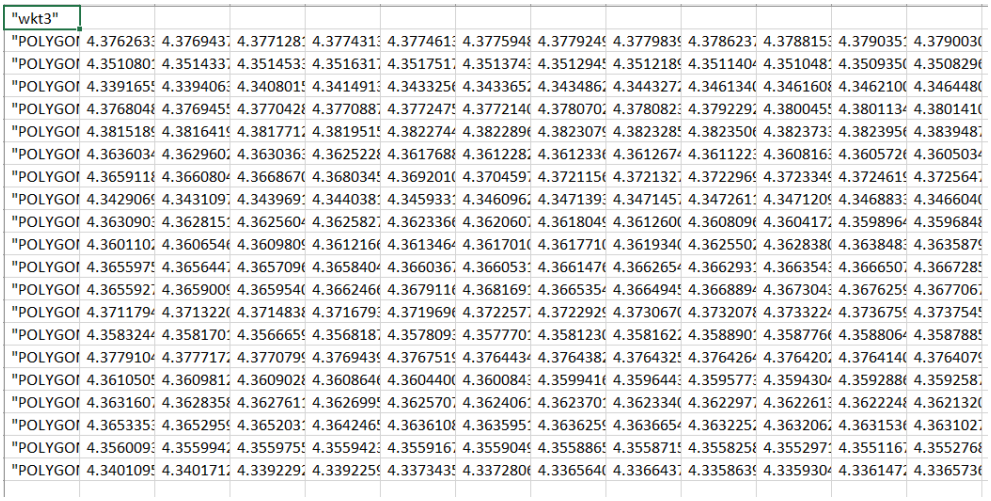
```
PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>
SELECT  ?wkt3 WHERE {
 ?gemeente a brt:Gemeente.
  ?gemeente rdfs:label "Delft"@nl.
  ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
  ?gebouw a brt:Gebouw.
  ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
  ?gebouw rdfs:label ?wkt2Label.
  ?plan a brt:Bedrijventerrein.
  ?plan brt:geometrie / geosparql:asWKT ?wkt3.
  ?plan rdfs:label ?wkt3Label.
  filter(ogcf:sfWithin(?wkt3, ?wkt1)).
}
```

Table | Response | Pivot Table | Google Chart | Geo

Showing 1 to 50 of 519 entries (in 60.175 seconds)

(b) Wrong Result: 519 results where there are 20 distinct zones and 499 duplicates of them.

Figure 4.8: Query to retrieve planning zones in Delft municipality.



"wkt3"												
"POLYGOI	4.376263	4.376943	4.377128	4.377431	4.377461	4.377594	4.377924	4.377983	4.378623	4.378815	4.379035	4.379003
"POLYGOI	4.351080	4.351433	4.351453	4.351631	4.351751	4.351374	4.351294	4.351218	4.351140	4.351048	4.350935	4.350829
"POLYGOI	4.339165	4.339406	4.340801	4.341491	4.343325	4.343365	4.343486	4.344327	4.346134	4.346160	4.346210	4.346448
"POLYGOI	4.376804	4.376945	4.377042	4.377088	4.377247	4.377214	4.378070	4.378082	4.379229	4.380045	4.380113	4.380141
"POLYGOI	4.381518	4.381641	4.381771	4.381951	4.382274	4.382289	4.382307	4.382328	4.382350	4.382373	4.382395	4.383948
"POLYGOI	4.363603	4.362960	4.363036	4.362522	4.361768	4.361228	4.361233	4.361267	4.361122	4.360816	4.360572	4.360503
"POLYGOI	4.365911	4.366080	4.366867	4.368034	4.369201	4.370459	4.372115	4.372132	4.372296	4.372334	4.372461	4.372564
"POLYGOI	4.342906	4.343109	4.343969	4.344038	4.345933	4.346096	4.347139	4.347145	4.347261	4.347120	4.346883	4.346604
"POLYGOI	4.363090	4.362815	4.362560	4.362582	4.362336	4.362060	4.361804	4.361260	4.360809	4.360417	4.359896	4.359684
"POLYGOI	4.360110	4.360654	4.360980	4.361216	4.361346	4.361701	4.361771	4.361934	4.362550	4.362838	4.363848	4.363587
"POLYGOI	4.365597	4.365644	4.365709	4.365840	4.366036	4.366053	4.366147	4.366265	4.366293	4.366354	4.366650	4.366728
"POLYGOI	4.365592	4.365900	4.365954	4.366246	4.367911	4.368169	4.366535	4.366494	4.366889	4.367304	4.367625	4.367706
"POLYGOI	4.371179	4.371322	4.371483	4.371679	4.371969	4.372257	4.372292	4.373067	4.373207	4.373322	4.373675	4.373754
"POLYGOI	4.358324	4.358170	4.356665	4.356818	4.357809	4.357770	4.358123	4.358162	4.358890	4.358776	4.358806	4.358788
"POLYGOI	4.377910	4.377717	4.377079	4.376943	4.376751	4.376443	4.376438	4.376432	4.376426	4.376420	4.376414	4.376407
"POLYGOI	4.361050	4.360981	4.360902	4.360864	4.360440	4.360084	4.359941	4.359644	4.359577	4.359430	4.359288	4.359258
"POLYGOI	4.363160	4.362835	4.362761	4.362699	4.362570	4.362406	4.362370	4.362334	4.362297	4.362261	4.362224	4.362132
"POLYGOI	4.365335	4.365295	4.365203	4.364246	4.363610	4.363595	4.363625	4.363665	4.363225	4.363206	4.363153	4.363102
"POLYGOI	4.356009	4.355994	4.355975	4.355942	4.355916	4.355904	4.355886	4.355871	4.355825	4.355297	4.355116	4.355276
"POLYGOI	4.340109	4.340171	4.339229	4.339225	4.337343	4.337280	4.336564	4.336643	4.335863	4.335930	4.336147	4.336573

Figure 4.9: CSV containing spatial planning zones for business places in Delft municipality.

```
prefix ogcf: <http://www.opengis.net/def/function/geosparql/>
select ?wkt2
where {
  ?gemeente a brt:Gemeente.
  ?gemeente rdfs:label "Delft"@nl.
  ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
  ?gebouw a brt:Gebouw.
  ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
  ?plan a brt:Bedrijventerrein.
  ?plan brt:geometrie / geosparql:asWKT ?wkt3.
  BIND(ogcf:sfWithin(?wkt2, ?wkt3) as ?within).
  filter(ogcf:sfWithin(?wkt2, ?wkt1)).
  filter(?within="true"^^xsd:boolean)
}limit 10
```

Figure 4.10: BIND a *within* spatial function into variable("?within") and use a spatial filter and a boolean filter.

```
prefix ogcf: <http://www.opengis.net/def/function/geosparql/>
select ?wkt2
where {
  ?gemeente a brt:Gemeente.
  ?gemeente rdfs:label "Delft"@nl.
  ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
  ?gebouw a brt:Gebouw.
  ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
  ?plan a brt:Bedrijventerrein.
  ?plan brt:geometrie / geosparql:asWKT ?wkt3.
  filter(ogcf:sfWithin(?wkt2, ?wkt1)).
  filter(ogcf:sfWithin(?wkt2, ?wkt3)).

}limit 1
```

Figure 4.11: Use of two spatial filters.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix ogcf: <http://www.opengis.net/def/function/geosparql/>
select ?wkt2
where {
  ?gemeente a brt:Gemeente.
  ?gemeente rdfs:label "Delft"@nl.
  ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
  ?gebouw a brt:Gebouw.
  ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
  ?plan a brt:Bedrijventerrein.
  ?plan brt:geometrie / geosparql:asWKT ?wkt3.
  filter(ogcf:sfWithin(?wkt2, ?wkt1)&&ogcf:sfWithin(?wkt2, ?wkt3)).

}limit 1
```

Figure 4.12: Use of a spatial filter containing two *within* spatial functions.

```
7  PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
8  PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>
9  SELECT  ?wkt2 WHERE {
10  ?gemeente a brt:Gemeente.
11   ?gemeente rdfs:label "Delft"@nl.
12   ?gemeente brt:geometrie / geosparql:asWKT ?wkt1.
13   ?gebouw a brt:Gebouw.
14   ?gebouw brt:geometrie / geosparql:asWKT ?wkt2.
15   ?plan a brt:Bedrijventerrein.
16   ?plan brt:geometrie / geosparql:asWKT ?wkt3.
17   BIND((ogcf:sfWithin(?wkt3, ?wkt1)) as ?res).
18   BIND((ogcf:sfWithin(?wkt2, ?wkt3)) as ?res2).
19   FILTER(?res=TRUE&&?res2=TRUE)
20 }limit 1
```

Table | Response | Pivot Table | Google Chart | Geo

Gateway Time-out (#504)

```
<html>
<head><title>504 Gateway Time-out</title></head>
<body bgcolor="white">
<center><h1>504 Gateway Time-out</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Figure 4.13: BIND two *within* spatial functions into variables(”?res” and ”?res2”) and use a boolean filter.

turned in the second case. The difference between the two cases is that in the second query, there are three triples non-related to the rest, that yield the geometries of buildings. In the final output, there are 519 results but only 20 of them are unique, while the other 499 are duplicates of them. This is probably happening because the query asks for gemeente (municipality) of Delft, gebouw (building) and bedrijvenenterrein (business places) and joins them in one table. However, since there is no variable to join them on, the query duplicates them to have each possible combination of the three types. Later, the query applies the filter and returns only the WKT records of the business places that are within the municipality of Delft. The filter is being applied to the joined table with all the distinct combinations of "municipality"-"building"-"business place". By selecting only the "business place" out of the distinct combinations, it seems as there are duplicate results. Nevertheless if the whole combination would be selected then it would be unique. However, this might produce incorrect results and it would be preferred not to join the building's table, since it was not used. Finally, in Figure 4.14a, the first two triplets return one result which is the municipality of Delft. Nevertheless, when the commented-out triple is used along with the other two, instead of yielding the geometry of the municipality of Delft, it returns the geometries that are shown in Figure 4.14b. That is because in the commented-out triplet, the subject is misspelled and instead of "?Gemeente" (Gemeente translates to Municipality in English) is written as "?Gemeent". Despite the misspelling error, there are results returned instead of raising an error. In the mentioned case the mistake was obvious however, in other cases it might not be.

4.6 RESULTS OF STEP 6: PDOK APPROACHES – API

The second approach the Dutch Cadastre offers is via a RESTful API. The RESTful API allows the user to ask from the server services via a protocol, such as *http* using methods such as **GET** or **POST**. Currently, Dutch cadastre's API is accessible at ***https://data.pdok.nl/sparql***.

Federated queries have to be used to access a remote API using Oracle's database. Federated queries contain the **SERVICE** construct in which the API's URL and the query to be posted on the API, are included. Federated queries are part of normal queries and are executed prior or after all the other queries have been executed to allow for fine tuning of the performance. By default, federated queries are initially resolved and the full result of them is joined with the results of the other queries.

As already discussed, cadastre's API is using *https* secure protocol, thus to get ready and start posting federated queries in Oracle, the appropriate certificates should be installed. The installation of certificates in Oracle should be done using the *Oracle Wallet Manager*. The process to install the certificates consists of different steps. Initially, the certificates should be downloaded. For that purpose Firefox has been used to download the three certificates (datapdoknl.crt, StaatderNederlandenOrganisatieCA-G2.crt, QuoVadisCSP-PKIOverheidCA-G2.crt) for *https://data.pdok.nl/sparql*. Further, a wallet for the user has been created and the certificates have been loaded. Finally, using the specific wallet one can query the API. Even though the process was correct up to this point, and it was possible to query different endpoints using different protocols and certificates, it was not possible to access this specific API. The reason behind this might have been an added security layer that might be present on the TU Delft server. Another explanation might be a corruption of the certificates that was downloaded. The exact reason is difficult to be pinpointed because there were no error messages and the attempts to access other APIs was successful. Other linked data APIs can be found in *https://www.w3.org/wiki/SparqlEndpoints*.

Gedefinieerde klassen X Query X Query 1 X +

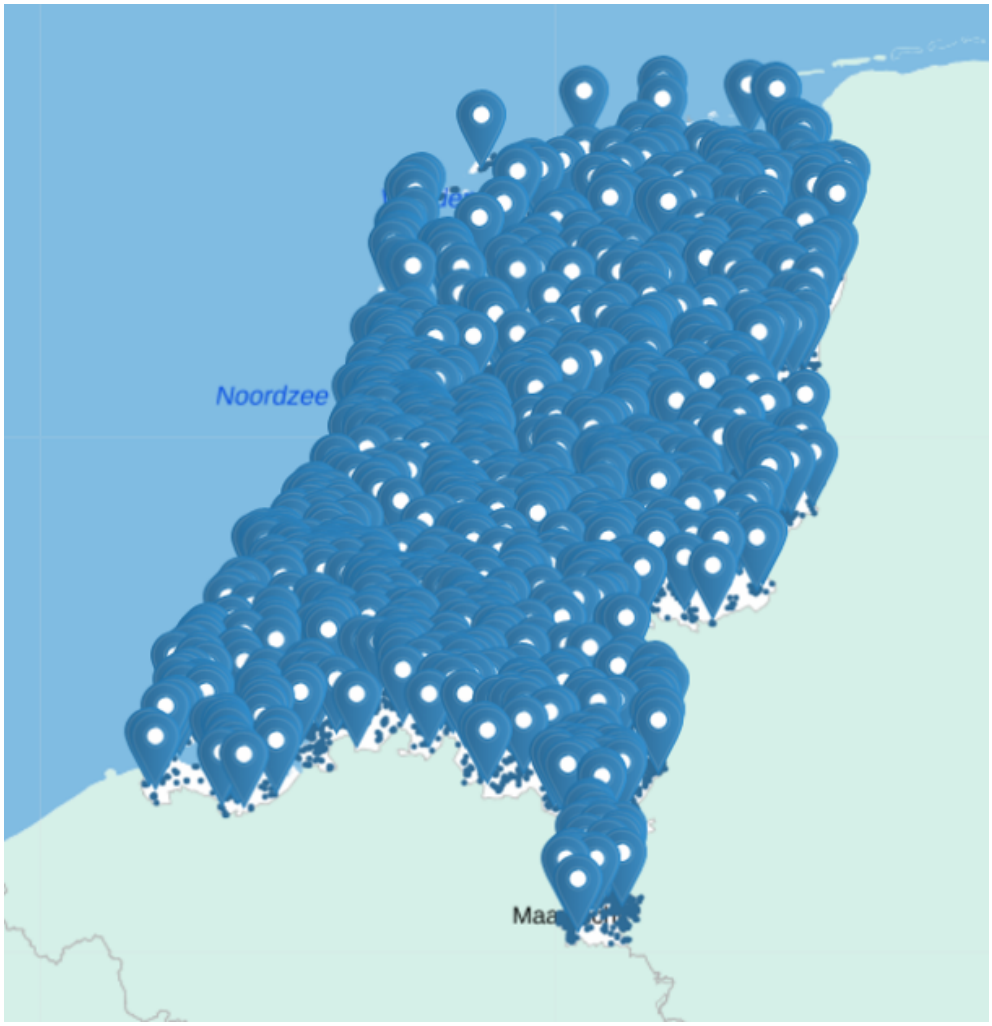
```
3  prefix owl: <http://www.w3.org/2002/07/owl#>
4  prefix brt: <http://brt.basisregistraties.overheid.nl/def/top10nl#>
5  prefix geosparql: <http://www.opengis.net/ont/geosparql#>
6  PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
7  PREFIX og: <http://ogp.me/ns#>
8  PREFIX ogc: <http://www.opengis.net/def/>
9  select  ?gemeente  ?geom ?wkt
10 where {
11     ?gemeente a brt:Gemeente .
12   ?gemeente rdfs:label  "Delft"@nl.
13   #?gemeent brt:geometrie / geosparql:asWKT ?wkt.
14 }
15
16
```

Table | Response | Pivot Table | Google Chart | Geo

Showing 1 to 1 of 1 entries (in 0.041 seconds)

	gemeente
1	http://brt.basisregistraties.overheid.nl/top10nl/id/registratief-gebied/126538680

(a) Malformed query. "?Gemeente" misspelled as "?Gemeent" which is commented out.



(b) Result of the malformed query.

Figure 4.14: Using a malformed query in Dutch cadastre's GeoSPARQL endpoint.

4.7 RESULTS OF STEP 7: DESIGNED APPROACHES – DIRECT MAPPING

This approach, is one of the approaches that bring together two different worlds, the one of relational data with that of linked data and thus getting the advantages of both. The data are stored only in relational data format, hence they are stored more efficiently and without redundant parts (e.g. a column name is stored using a name as a string that is usually up to 2-3 words and that would be 30-40 characters or 30-40b.. In case the data were stored as linked data, each set would contain triplets coming from column names, which would be the same as before plus the added prefix, that could be another 20-30 char. That would make almost double the storage required for one column and one value and it would scale up the more data involved). Instead of storing both formats of data, metadata of how to transform those data into linked data are stored. The relational data are easier to be accessed, explored, re-shared and extended as linked data (e.g. When looking for a geometry of a train station in a dutch dataset it is possible to query it in linked data format and using similarity properties such as *owl:sameAs* [McGuinness et al., 2004], it can be queried using the English translation for it). Moreover, linked data are more easily extended and allow for direct inference both topological and non-topological.

There are two possible methods for the direct mapping. The first one is to map all data-tables under a single mapping and the other is to map each table separately and then query them together. The second option would allow for re-use of mapping, i.e. it will allow the mapping to be of general use rather case-specific. In order to perform the direct mapping on relational tables with spatial data, it is important that the relational tables will also be spatially indexed even if there is a spatial index already in place for linked data.

During the direct mapping using both methods, there were choices that had to be made that are not directly affecting the outcome. The first decision is the prefix that will be used to turn data into linked data. The choice made was *http://www.example.org/geometries/*. The second choice is to use the option "$CONFORMANCE = T$" so that the linked data will be associated with the user creating them. Were this option not chosen, the user-name would appear in between the selected prefix and the table name, resulting in the form *Prefix#UserName.TableName*. In case there are more than one users, it is a good practice to allow the user-name into the linked data form in order to distinguish easily the user that mapped the data. Finally, since the use case is located in the Netherlands, we ensured that the spatial indices were correctly created. At this point, it should be highlighted, that the spatial plans were downloaded from the GeoSPARQL endpoint of Dutch Kadaster, as described at previous step, and their SRID is *WGS-84*. Even though the SRID of spatial plans is different from the other datasets, it was not transformed in order to check the performance of combining datasets with different SRID. That functionality increases the re-usability of the linked data significantly. The mapping inside the database is illustrated in Figure 4.18 on page 57.

Different performance tests were carried out for the query of direct mapping. It should be noted that from experimental results, the Oracle Spatial and Graph, executes first the *SERVICE* constructs, i.e. the federated queries, except if specified otherwise. Next, it executes normal queries and at the end it executes the filter queries. That means that depending on how the query is formed, it can greatly increase execution time. In addition, from the experimental results, it seems that it would load into the memory all the objects that satisfy the normal queries and the federated queries and then it will start filtering them. Due to that, the order of execution of queries and filters is extremely important, as it affects of speed of execution. Moreover, using *BIND* constructs to store variables that are calculated and filtering results out as a next step, requires more time, as the data would be queried twice. On the other hand, if the bind variable is not stored, but pushed into the filter, then it will be filtered at

the time of calculation. That would save even more time in case there are multiple filters, because the next filter execution will have less objects to be calculated for. Hence, the order of queries is very important for the execution time, which might be further improved by using multiple graphs and uniting them through *UNION* option, that would allow to query only the relevant information each time.

One more notable result is that in the case of misspelling in the query, there is no error reported. On the other hand, there is an empty result returned, which in the case that the result is not known, it might seem correct. The only indication of the malfunction was that it returned the empty results very fast (around 0.01 seconds).

For the first method of direct mapping, a misspelled query was initially performed, illustrated in Figure 4.17, where on the second line of the query the geometry of the parcels is queried. While *DELFT_PARCELS* table does not have a *GEOM* column but a *PACEL_GEOM*, no problem was reported. Only by testing line by line, it was possible to figure out the part that was wrong. By fixing the query, there were results, as shown in Figure 4.16 on page 56. In the query illustrated in Figure 4.16a, it is observable that each table has each own alias, making the query difficult and prone to misspelling mistakes. For the reasons mentioned in the previous paragraphs, it was preferred to put all the spatial queries inside the *FILTER* construct. The mapping in the database can be seen in Figure 4.18. The same example using *BIND* constructs and then *FILTER* constructs is remarkably slower in execution speed and is therefor avoided. More precisely, the query solving the use case without *BIND* constructs, shown in Figure 4.16a, took 141.72 seconds which is almost 2.5 minutes, while the same query with *BIND* constructs had to be stopped after 37129 seconds, which is more than 10 hours. For the same reason, as per best practices in Oracle documentation [Perry and Herring, 2012], the negation of queries was avoided wherever possible.

As a final step, querying linked data and relational data (as linked data with direct mapping) at the same query, was attempted. It is an approach that even though in the current use case was not used, in a future step it might be important for using data in both formats. The process and the result is shown in Figure 4.15. The query asks for lots that are stored in linked data format and also for parcels which are stored as relational data and have gone through the direct mapping process to be accessible as linked data. The performance considering speed was decent, however, there was no spatial analysis included.

The second method aims to create a separate mapping for each table of the relational data and connect the RDF views of the relational data under a single query. That would allow the re-use of mappings and make the mappings non use-case specific. For that method, the options and the prefix that was used are the same as the first method.

Although the gains of this methodology would be significant, the problems are many and the performance not as good as the performance of the first method. Despite querying a single view is easy and straightforward, as shown in Figure 4.19, on page 58, querying more than one view is problematic. More precisely, Oracle Spatial and Graph does not allow to use more than one view in a query. A workaround is to access a view via a *SERVICE* construct and query it like it was a federated query, as shown in Figure 4.20, on page 58. However, querying two views at the same time, one normally and one via federated query, is not performing well. The results of that process are shown in Figure 4.21, on page 58, where ten parcels are accessed via a federated query and are checked whether they touch a building. The result required almost eight minutes to be calculated and it was empty. In case the result was reported faster, that would imply that the query malfunctioned. The outcomes of the previous query which were limited to ten parcels happened not to intersect with any building and resulted in an empty response. To prove that assumption, the limit was raised to 200, as illustrated in Figure 4.22 on page 59, and there was one result returned after 45814 seconds, which is almost 13 hours. Due to the low performance, that method was not further tested. On the other hand, using a *UNION* of query

```
SELECT distinct l$rdfterm, p$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?l ?p
WHERE {    {?l rdf:type :lot.
?l ogc:hasGeometry ?geom.
?geom ogc:asWKT ?wkt.} UNION{ SELECT *  WHERE {SERVICE orardbm:DELFT_PARCELS_view { select * where {?p rdf:type ex:DELFT_PARCELS_S.
    ?p p:PARCEL_GEOM ?wktp. } }}
}
}',
SEM_MODELS('geometries' )  , NULL, SEM_ALIASES(SEM_ALIAS('p','http://www.example/geometries/DELFT_PARCELS_S#'),
SEM_ALIAS('ex','http://www.example/geometries/'),
SEM_ALIAS('','http://www.example.org/geometries/')), NULL, NULL, 'PLUS_RDFT=VC')) order by 1,2;
```

SQL | Fetched 200 rows in 0.396 seconds

	L$RDFTERM	P$RDFTERM
16	<http://www.example.org/geometries/lot24>	(null)
17	<http://www.example.org/geometries/lot2>	(null)
18	<http://www.example.org/geometries/lot3>	(null)
19	<http://www.example.org/geometries/lot4>	(null)
20	<http://www.example.org/geometries/lot5>	(null)
21	<http://www.example.org/geometries/lot6>	(null)
22	<http://www.example.org/geometries/lot7>	(null)
23	<http://www.example.org/geometries/lot8>	(null)
24	<http://www.example.org/geometries/lot9>	(null)
25	(null)	_:m222mBlankNode1002900C8B9FE322
26	(null)	_:m222mBlankNode100478CCEDE16473
27	(null)	_:m222mBlankNode100647A735B70BFC
28	(null)	_:m222mBlankNode1006D38F7EFDB3A8
29	(null)	_:m222mBlankNode100DAD9B89294CD7
30	(null)	_:m222mBlankNode100E2E22EEE5BA08
31	(null)	_:m222mBlankNode10107119CD40AC5D
32	(null)	_:m222mBlankNode10194EEBB3EC929
33	(null)	_:m222mBlankNode1019F6EF511FD08A

Figure 4.15: Querying lots stored as linked data and parcels stored as relational data and mapped to linked data format.

```
SELECT distinct b$rdfterm, wktb$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?b ?wktb
WHERE
  { ?p rdf:type ex:DELFT_PARCELS_S.
    ?p pa:PARCEL_GEOM ?wktp.
    ?b rdf:type ex:GEBOUW_VLAK_S. ?b bu:GEOM ?wktb.
    ?b2 rdf:type ex:GEBOUW_VLAK_S. ?b2 bu:TYPEGEBOUW "overig|treinstation". ?b2 bu:GEOM ?wktb2.
    ?r rdf:type ex:WEGDEEL_VLAK_S. ?r ro:TYPEWEG "lokale weg|regionale weg". ?r ro:GEOM ?wktr.
    ?pl rdf:type ex:SPATIAL_PLANS.  ?pl pl:GEOM ?wktpl.
    FILTER( (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500  &&
    (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) > 0 &&
    (orageo:area(?wktb, "unit=SQ_M")) > 1500 &&
    (ogcf:distance(?wktr, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500 &&
    (ogcf:sfWithin(?wktb, ?wktpl)) &&
    (ogcf:sfWithin(?wktb, ?wktp))
    )
    }',
SEM_MODELS('GEOMETRIES_DIR_VIEW') , NULL, SEM_ALIASES(SEM_ALIAS('bu','http://www.example.org/geometries/GEBOUW_VLAK_S#'),
SEM_ALIAS('pa','http://www.example.org/geometries/DELFT_PARCELS_S#'),
SEM_ALIAS('pl','http://www.example.org/geometries/SPATIAL_PLANS#'),
SEM_ALIAS('ro','http://www.example.org/geometries/WEGDEEL_VLAK_S#'),
SEM_ALIAS('ex','http://www.example.org/geometries/')), NULL, NULL, ' PLUS_RDFT=VC ')) ;
```

(a) Correct query for solving the use case using direct mapping.

SQL | All Rows Fetched: 6 in 141.72 seconds

	B$RDFTERM	WKTB$RDFTERM
1	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102571240>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83778.824 448235.705, 83759.065 44
2	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102555243>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83713.676 448155.01, 83697.903 448
3	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102631523>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83375.052 448314.98, 83434.747 448
4	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102637088>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83633.637 448402.686, 83590.984 44
5	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102576876>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83296.529 448127.885, 83294.338 44
6	<http://www.example.org/geometries/GEBOUW_VLAK_S/IDENT=NL.TOP10NL.102579851>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83455.316 446733.893, 83435.339 44

(b) Results for the use case using direct mapping.

Figure 4.16: Query and results for the use case using direct mapping.

```
SELECT distinct b$rdfterm, wktb$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?b ?wktb
WHERE
  { ?p rdf:type ex:DELFT_PARCELS_S.
    ?p pa:GEOM ?wktp.
    ?b rdf:type ex:GEBOUW_VLAK_S. ?b bu:GEOM ?wktb.
    ?b2 rdf:type ex:GEBOUW_VLAK_S. ?b2 bu:TYPEGEBOUW "overig|treinstation". ?b2 bu:GEOM ?wktb2.
    ?r rdf:type ex:WEGDEEL_VLAK_S. ?r ro:TYPEWEG "lokale weg|regionale weg". ?r ro:GEOM ?wktr.
    ?pl rdf:type ex:SPATIAL_PLANS. ?pl pl:GEOM ?wktpl.
    FILTER( (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500  &&
    (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) > 0 &&
    (orageo:area(?wktb, "unit=SQ_M")) > 1500 &&
    (ogcf:distance(?wktr, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500 &&
    (ogcf:sfWithin(?wktb, ?wktpa)) &&
    (ogcf:sfWithin(?wktb, ?wktp)))
    }',
SEM_MODELS('geometries_dir_view') , NULL, SEM_ALIASES(SEM_ALIAS('bu','http://www.example.org/geomet
SEM_ALIAS('pa','http://www.example.org/geometries/DELFT_PARCELS_S#'),
SEM_ALIAS('pl','http://www.example.org/geometries/SPATIAL_PLANS#'),
SEM_ALIAS('ro','http://www.example.org/geometries/WEGDEEL_VLAK_S#'),
SEM_ALIAS('ex','http://www.example.org/geometries/')), NULL, NULL, ' PLUS_RDFT=VC ')) ;
```

Figure 4.17: Misspelled query on Direct Mapping.

Figure 4.18: Direct mapping in the database.

as shown in Figure 4.23 on page 59, it is possible to query the data in a much faster way, which was around 20 seconds. However, the speed of the spatial function is not quite good to allow this method to solve the use case problem. Finally, it should be noted that due to the way the mapping is done, the GeoSPARQL ontology for spatial features is not followed. Moreover, inference (either topological or non-topological) with views is not allowed without materializing the view, i.e. storing the linked data which would negate the purpose of creating it in the first place.

4.8 RESULTS OF STEP 8: DESIGNED APPROACHES – R2RML

The R2RML approach requires a more detailed mapping, as explained in the methodology. Part of this mapping, in Turtle format, is shown in Code snippet 4.4 on page 61. More challenges were encountered following this approach, which was anticipated as this methodology consisted of more steps. Initially, if the GeoSPARQL ontology for spatial features is followed, then the mapping does not seem to work. Apparently, during the mapping, it is not possible to construct new objects. Hence, constructing a geometry object for each spatial object that would hold the geometry representation was not possible, as indicated by the GeoSPARQL specification. Moreover, there were problems with data-types of the relational tables and the final output. More specific, the columns such as "OBJECT_ID" had to be defined specifically as integers,

```
SELECT distinct g$rdfterm, wkt_b$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?g ?wkt_b
WHERE {
{ SELECT *  WHERE {
?g :GEOM ?wkt_b .}}
}',
SEM_MODELS('GEBOUW_VLAK_VIEW' )  , NULL, SEM_ALIASES(SEM_ALIAS(
```

Query Result 8 | SQL | All Rows Fetched: 6201 in 4.504 seconds

	G$RDFTERM	WKT_B$RDFTERM
6192	_:m220mBlankNodeFFB2DDD3FE06E27E	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87072.558 445539.895,
6193	_:m220mBlankNodeFFB47CA97F10CF00	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((85771.044 449247.532,

Figure 4.19: Simple query on RDF View.

```
SELECT distinct r$rdfterm, wkt_p$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?r ?wkt_p
WHERE {
{ SELECT *  WHERE {
SERVICE orardbm:DELFT_PARCELS_view { select * where {?r p:PARCEL_GEOM ?wkt_p }  } .
}}
}',
SEM_MODELS('GEBOUW_VLAK_VIEW' )  , NULL, SEM_ALIASES(SEM_ALIAS('p','http://www.example/geometries/DELFT_PARCELS_S#')
```

Query Result 8 | Query Result 9 | SQL | All Rows Fetched: 19904 in 18.062 seconds

	WKT_P$RDFTERM
1	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83556.494 447898.889, 83555.88 447899.743, 83556.037 447899.851,
2	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((82605.596 447339.28, 82601.397 447336.289, 82609.03 447325.667,
3	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((84203.118 447557.808, 84205.637 447553.432, 84205.733 447553.485,
4	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83604.996 447401.054, 83607.972 447396.825, 83624.593 447408.504,
5	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((82918.072 444625.635, 82917.004 444630.809, 82916.015 444635.605,

Figure 4.20: Simple query on RDF View using SERVICE construct.

```
SELECT distinct r$rdfterm, wkt_p$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?r ?wkt_p
WHERE {
{ SELECT *  WHERE {
SERVICE orardbm:DELFT_PARCELS_view { select * where {?r p:PARCEL_GEOM ?wkt_p } limit 10 } .
?g :GEOM ?wkt_b .
FILTER(ogcf:sfTouches(?wkt_p,?wkt_b))}}
}',
SEM_MODELS('GEBOUW_VLAK_VIEW' )  , NULL, SEM_ALIASES(SEM_ALIAS('p','http://www.example/geometries/DELFT_PARCELS_S#'),
```

Query Result 1 | SQL | All Rows Fetched: 0 in 239.27 seconds

R$RDFTERM | WKT_P$R...

Figure 4.21: Using Spatial function with direct mapping and SERVICE construct to get 10 parcels.

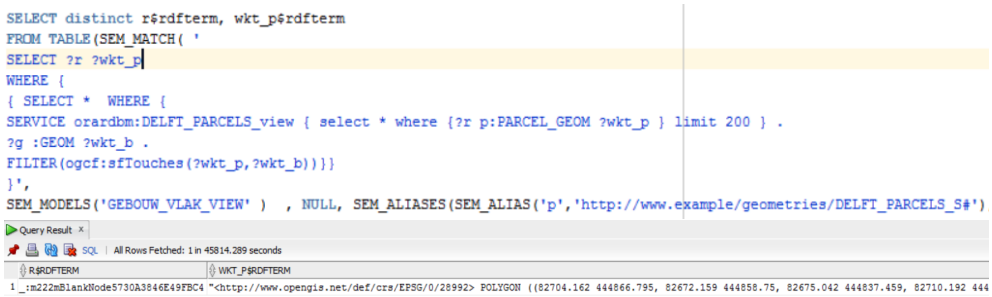


Figure 4.22: Using Spatial function with direct mapping and SERVICE construct to get 200 parcels.

Figure 4.23: Using Union to get Parcels and Buildings with direct mapping and SERVICE construct.

else they were returned as string literals. Furthermore, when a string literal was not coming from a column that had a variable length (e.g. char(50)), then the result was filled with empty spaces (either before or after the string literal to cover the column length), which in linked data format are substituted with a set of *%20*. On top of that, since the variables had different length values, there was a random number of spaces, either preceding or trailing the value, which made impossible to query all the data correctly (e.g. it could return one result labeled as "parcel" even if there were thousands of them, because the spaces got divided that way). A solution to this, was to change the column type to a type with variable length (e.g. varchar(50)), which however, had the same length as the previous one. Then perform a trim operation on spaces preceding or trailing the values, as it is depicted in the example in Code snippet 4.3 on page 60.

After solving those problems, the Turtle text can be copied to **EasyRDF** [Humfrey, 2018] and produce a file with n-triples, part of it is illustrated in Figure 4.25 on page 60 and the full file is available in Appendix B. That file can be directly uploaded into a staging table, which is a table that will hold the mapping and will be used to create the view. Nevertheless, to do so, the n-triples file should be placed inside the loading folder which is located into the server, hence a detour was made. The n-triples have been converted into *INSERT* statements and have been loaded one by one, as triplets into the staging table. That was a result of the python code found in Appendix B. After importing the n-triples into the staging table, the view is created. The mapping inside the database is illustrated in Figure 4.24 on page 60.

To determine whether the mapping works, a simple spatial query is used, as shown in Figure 4.27 on page 62, where a building is checked if it interacts with itself and all the other buildings, returning true only for itself. That query tested that the tolerance of the spatial index is set correctly and that the spatial queries work as they should. Afterwards, there was a test to check the performance of R2RML mapping with different query formatting. For that reason, the sample query was used, but the criteria were stricter, so that less results than the original sample query, will come back after each spatial function and filter. The criteria were changed, so that the distance from the train station should be less than 1200 meter, the distance from main local road should be less than 1300 meter and the area of the buildings more than 50 sq.meters. Those attempts are visible in Figure 4.28a on page 63, where the functions

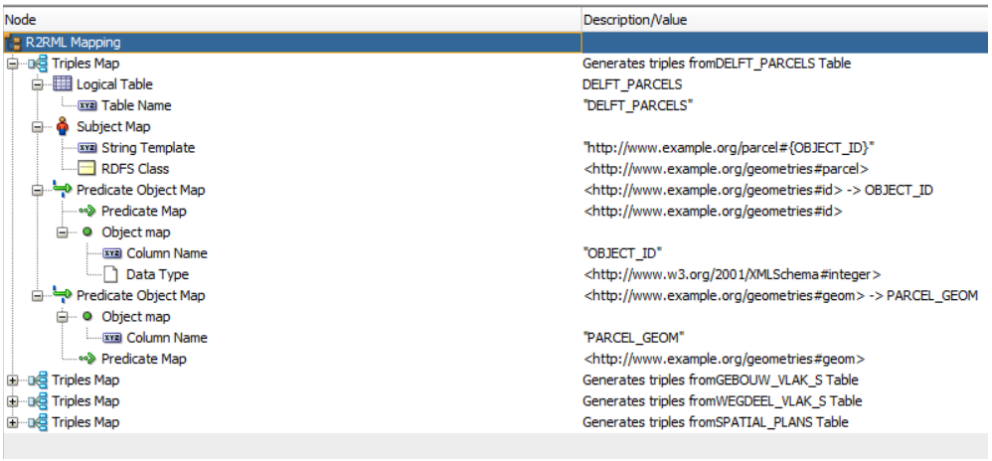


Figure 4.24: R2RML mapping in the database.

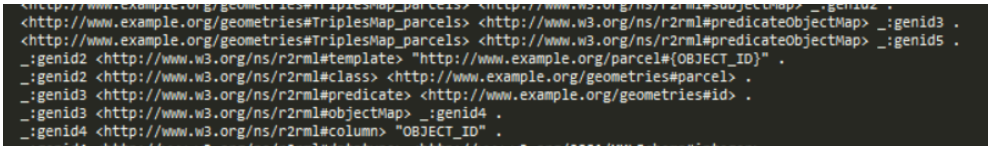


Figure 4.25: Part of the N-triples created.

Code Snippet 4.3: Fixing Roads table for R2RML Mapping.

```
alter table
        WEGDEEL_VLAK_S
modify
(TYPEWEG carchar2(50),
 IDENT carchar2(50));

UPDATE WEGDEEL_VLAK_S
SET TYPEWEG = TRIM(TYPEWEG);

UPDATE WEGDEEL_VLAK_S
SET IDENT = TRIM(IDENT);
```

Code Snippet 4.4: R2RML mapping of Delft Parcels.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix gsp: <http://www.opengis.net/ont/geosparql#>.
@prefix ex: <http://www.example.org/geometries#>.
@prefix orageo: <http://xmlns.oracle.com/rdf/geo/>.
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

ex:TriplesMap_parcels
    rr:logicalTable [ rr:tableName "DELFT_PARCELS" ];
    rr:subjectMap [
        rr:template "http://www.example.org/parcel#{OBJECT_ID}";
        rr:class ex:parcel;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:id;
        rr:objectMap [ rr:column "OBJECT_ID" ; rr:datatype xsd:integer ];
    ];

    rr:predicateObjectMap [
        rr:predicate ex:geom;
        rr:objectMap [ rr:column "PARCEL_GEOM" ];
    ].
```

are inside the *FILTER* and it returned only one building in 31 seconds. On the other hand, as seen in figure 4.28b on page 63, the spatial functions results are bound to variables using *BIND* constructs and then filtered. The result of this process contains 4 buildings, and that is because it does not contain the two spatial functions to check for buildings inside parcels and buildings inside the spatial plans. The reason behind this, is the long process time that it already required to run without those two; more specifically it required 20 minutes to run, and when the two skipped functions have been added it could not produce results before 3 hours. It is clear that pushing the functions inside the filter is much more efficient in terms of computational time.

As a next step, the sample query with the correct values has been run against the mapped data. The results are depicted in Figure 4.29 on page 63. The results are the same as with direct mapping solution, and its speed efficiency is just over one minute and its almost half of the direct mapping. The final solution is visualized in Figure 5.3 on page 69 using QGIS. Same as the direct mapping, it was not possible to infer when querying the RDF views without materializing them.

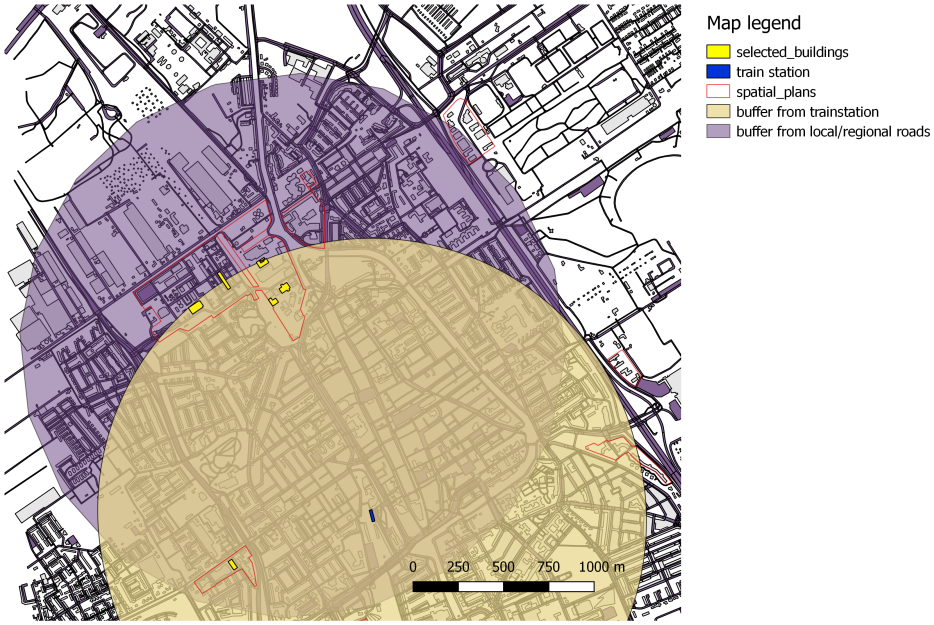


Figure 4.26: The visualization of the solution of the use case with R2RML mapping

```
SELECT wktb$rdfterm, comp$RDFTERM
FROM TABLE(SEM_MATCH( '
SELECT ?wktb ?comp  WHERE {

?b rdf:type ex:building.
?b ex:geom ?wktb.
bind(ogcf:sfIntersects(?wktb, "<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87730.889 447811.157,
```

	WKTB$RDFTERM	COMP$RDFTERM
1	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87730.889 447811.1...	"true"^^<http:/
2	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87673.466 447578.2...	"false"^^<http:
3	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87676.439 447597.8...	"false"^^<http:
4	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87673.11 447645.84...	"false"^^<http:
5	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((87704.605 447654.5...	"false"^^<http:

Figure 4.27: Minimum working example of querying a view created with R2RML mapping.

```
{ ?p rdf:type ex:parcel.
  ?p ex:geom ?wktp.
  ?b rdf:type ex:building. ?b ex:geom ?wktb.
  ?b2 rdf:type ex:building. ?b2 ex:typegebouw "overig|treinstation". ?b2 ex:geom ?wktb2.
  ?r rdf:type ex:road. ?r ex:typeweg "lokale weg|regionale weg". ?r ex:geom ?wktr.
  ?pl rdf:type ex:plan.
  ?pl ex:geom ?wktpl.
  FILTER( (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1200  &&
  (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) > 0 &&
  (orageo:area(?wktb, "unit=SQ_M")) > 50 &&
  (ogcf:distance(?wktr, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1300 &&
  (ogcf:sfWithin(?wktb, ?wktpl)) &&
  (ogcf:sfWithin(?wktb, ?wktp)))
  }',
EM_MODELS('geometries_rv_r2r') , NULL, SEM_ALIASES(SEM_ALIAS('ex','http://www.example.org/geome
```

SQL | All Rows Fetched: 1 in 31.015 seconds

B$RDFTERM	WKTB$RDFTERM
<http://www.example.org/gebouw_vlak#NL.TOP10NL.102579851>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83455.316 446733.893, 83435.339 446720.326, 83466.937

(a)

```
?p rdf:type ex:parcel.
?p ex:geom ?wktp.
?b rdf:type ex:building. ?b ex:geom ?wktb.
?b2 rdf:type ex:building. ?b2 ex:typegebouw "overig|treinstation". ?b2 ex:geom ?wktb2.
?r rdf:type ex:road. ?r ex:typeweg "lokale weg|regionale weg". ?r ex:geom ?wktr.
?pl rdf:type ex:plan.
?pl ex:geom ?wktpl.

BIND(ogcf:distance(?wktr, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>) as ?main_road_dist).
BIND(ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>) as ?train_dist).
BIND(orageo:area(?wktb, "unit=SQ_M") as ?barea).
FILTER(?train_dist > 0 && ?train_dist < 1200  && ?barea > 50 && ?main_road_dist < 1200 && (ogcf:sfWithin(?wktb, ?wktpl))
```

SQL | All Rows Fetched: 4 in 1200.904 seconds

	B$RDFTERM	WKTB$RDFTERM
1	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102516457>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83780.818 448187.886, 83771.167 448148.483, 83775.694
2	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102577798>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83776.11 448118.531, 83747.082 448098.382, 83760.758
3	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102555261>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83500.23 446748.939, 83483.465 446736.284, 83486.704
4	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102580417>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83517.198 446740.225, 83523.827 446744.802, 83512.799

(b)

Figure 4.28: Altered sample query R2RML mapping.

```
SELECT distinct b$rdfterm, wktb$rdfterm
FROM TABLE(SEM_MATCH( '
SELECT ?b ?wktb
WHERE
  { ?p rdf:type ex:parcel.
    ?p ex:geom ?wktp.
    ?b rdf:type ex:building. ?b ex:geom ?wktb.
    ?b2 rdf:type ex:building. ?b2 ex:typegebouw "overig|treinstation". ?b2 ex:geom ?wktb2.
    ?r rdf:type ex:road. ?r ex:typeweg "lokale weg|regionale weg". ?r ex:geom ?wktr.
    ?pl rdf:type ex:plan.
    ?pl ex:geom ?wktpl.
    FILTER( (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500  &&
    (ogcf:distance(?wktb2, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) > 0 &&
    (orageo:area(?wktb, "unit=SQ_M")) > 1500 &&
    (ogcf:distance(?wktr, ?wktb, <http://xmlns.oracle.com/rdf/geo/uom/M>)) < 1500 &&
    (ogcf:sfWithin(?wktb, ?wktpl)) &&
    (ogcf:sfWithin(?wktb, ?wktp)))
    }',
SEM_MODELS('geometries_rv_r2r') , NULL, SEM_ALIASES(SEM_ALIAS('ex','http://www.example.org/geome
```

SQL | All Rows Fetched: 6 in 64.646 seconds

	B$RDFTERM	WKTB$RDFTERM
1	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102555243>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83713.676 448155.01, 83697.903 448177.817, 83680.713
2	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102571240>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83778.824 448235.705, 83759.065 448264.348, 83738.318
3	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102579851>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83455.316 446733.893, 83435.339 446720.326, 83466.937
4	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102631523>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83375.052 448314.98, 83434.747 448224.521, 83449.695
5	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102637088>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83633.637 448402.686, 83590.984 448375.426, 83611.787
6	<http://www.example.org/gebouw_vlak#NL.TOP10NL.102576876>	"<http://www.opengis.net/def/crs/EPSG/0/28992> POLYGON ((83296.529 448127.885, 83294.338 448131.1, 83298.891 4

Figure 4.29: The solution of the use case with R2RML mapping

5 DISCUSSION AND CONCLUSIONS

In this Chapter, we are going to elaborate on the results. Moreover, the research question and the subresearch questions are going to be answered and finally, the limitations of this research are going to be highlighted.

5.1 DISCUSSION

The summary of results is illustrated in the Table 5.1 and the discussion will be based on that table. Since PDOK's API could not be accessed and assessed, it will be excluded from the discussion. At first look, after comparing the results from the rest of the different methodologies, it should be noted that both mapping approaches, out of all the four possible approaches, managed to solve the spatial problem of the use case. The linked data approach is considered to be able to solve the spatial problem, as it supports the functions required, nevertheless its performance and accuracy could not be measured. Moreover, no solution seems to be following the GeoSPARQL implementation to the fullest. Most of them fail to separate the spatial object's geometry from the representation of the geometry, and no solution is able to perform inference of topological or non-topological form. While both Oracle-based solutions and PDOK's GeoSPARQL endpoint support topological and non-topological functions, Oracle-based solutions seem to outclass PDOK's GeoSPARQL endpoint, due to the support of Oracle Spatial and Graph spatial functions, that enable more spatial analysis. However, some of the GeoSPARQL spatial functions are under development as discussed on W3C [2016]. Another point where Oracle-based solution outclass PDOK's GeoSPARQL endpoint, is the support of GML representation and the support of using data with both serializations at a single query.

An important remark is that the two approaches that produced a result for the spatial problem yielded the same result. Nevertheless, an even more important remark is that they brought a different result from the QGIS solution, which was the benchmark solution. Hence, an exploration of the results discrepancies follows. In Figure 5.1, the QGIS solution is illustrated with four buildings as a result of the spatial analysis, while the mapping solution which is the same for the two cases is depicted in Figure 5.3, with six buildings as a result of the queries. Interestingly, there are three buildings that appear only in the proposed solutions and one that is missing from them and it appears on the QGIS solution. The three buildings missing from QGIS solution are located close to the boundary of the train station buffer. One possible explanation for the most right building and the most left building is that they are really close to being inside the buffer, and based on the vertices used to created the buffer on QGIS and the tolerance of the function, they may have been excluded in the QGIS solution, as seen in Figure 5.2. However, this is not the case for the middle building. Hence, the reason of this discrepancy lies in the functions that were used. While in QGIS solution there was a buffer created and then a spatial query selected the buildings inside the buffer, in the proposed solutions, those two functions have been substituted by the distance function, which only requires that the closest point should be inside the buffer area. Besides that, there is one building, which appears on the QGIS solution, but does not appear on the proposed solutions. Since that building is clearly inside the two buffer zones and has an area greater of 1500 sq. meters, there are two criteria that could be different. While that building is located

Table 5.1: Summary table of results

Criteria	Linked data	GeoSPARQL endpoint	PDOK API	Direct Mapping	R2RML Mapping
SPARQL	✓	✗	✗	✓	✓
Spatial object disjoint from geometry	✓	✗	✗	✗	✗
WKT serialization	✓	✓	✗	✓	✓
GML serialization	✓	✗	✗	✓	✓
Topological functions	✓	✓	✗	✓	✓
Non-topological functions	✓	✓(GeoSPARQL)	✗	✓	✓
Topological Inference	✗	✗	✗	✗	✗
Non-topological Inference	✗	✗	✗	✗	✗
Solution on use case	✗(possible)	✗	✗	✓	✓
Speed of solution	✗	✗	✗	140 Second	61 Second
Storage requirements	Linked data	✗	✗	Relational data + Meta-data	Relational data + Meta-data

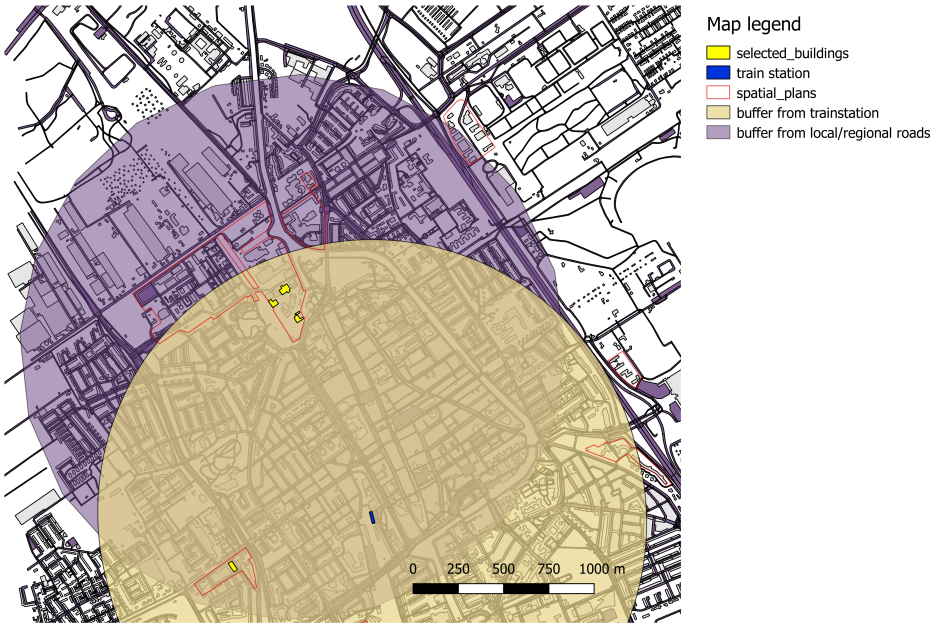


Figure 5.1: The QGIS solution of the use case.

inside the parcels in the proposed solution, not every part of the building appears to be inside the planning zones. The reason for this discrepancy, is assumed to be, the difference of SRID between the spatial plans and the rest of the data-sets. Spatial plans are stored as 8307 SRID, while the rest are stored as 28992 SRID. In QGIS solution, the transformation is executed before the spatial analysis, while on the proposed approaches the transformation is executed on-the-fly by Oracle database. During this research, it is out of the scope to distinguish which of the two transformations has better accuracy, however to prove this assumption, the *SPATIAL_PLANS* data-set has been transformed using QGIS from SRID 8307 to SRID 28992. As a next step, the data-set has been imported as relational data back into the database and re-executed the methodology for the direct and R2RML mappings. With that methodology, the building appeared on the results and inside the spatial plans, which proved that the problem was the SRID transformation.

5.2 CONCLUSIONS ON THE RESEARCH PROBLEM

First, answering the sub-research questions:
SQR_1**: What are the possible linked data approaches to position a new shopping center in Delft?**
The current linked data approaches to position a shopping center in Delft are related to Dutch Kadaster's PDOK, nevertheless, for different reasons, each one of them was not able to deliver a result. For that reason, two other approaches have been proposed, based on mapping of relational data. The approaches attempted are shown in Table 5.2 with their results.

During this thesis research, only the Direct mapping and R2RML approaches achieved a solution that was close to the QGIS solution. The other three solutions, for the reasons that have already been mentioned, could not achieve a solution.

SQR_2**: To what extend do the different approaches follow the GeoSPARQL specification?**

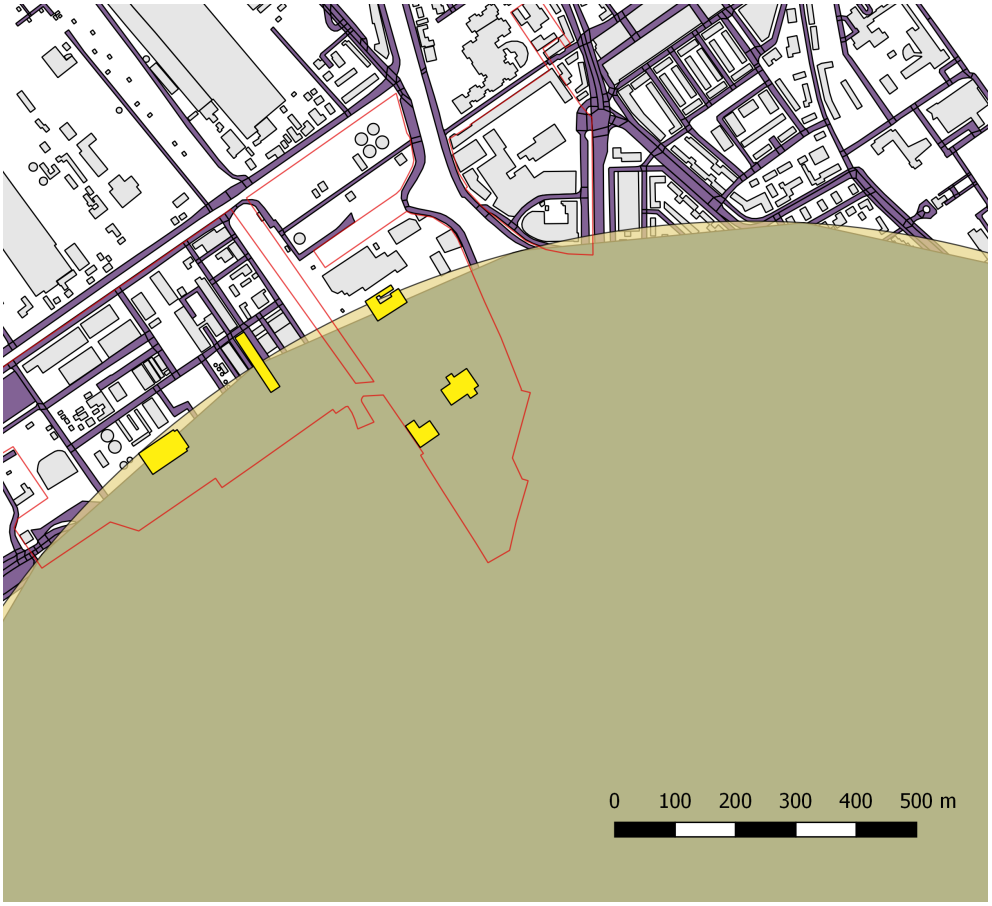


Figure 5.2: The QGIS solution buffer issue.

Table 5.2: Approaches attempted and their results.

Approach	Results
PDOK - GeoSPARQL Endpoint	✗
PDOK - API	✗
Linked data - dummy data	✗
Direct Mapping	✓
R2RML Mapping	✓

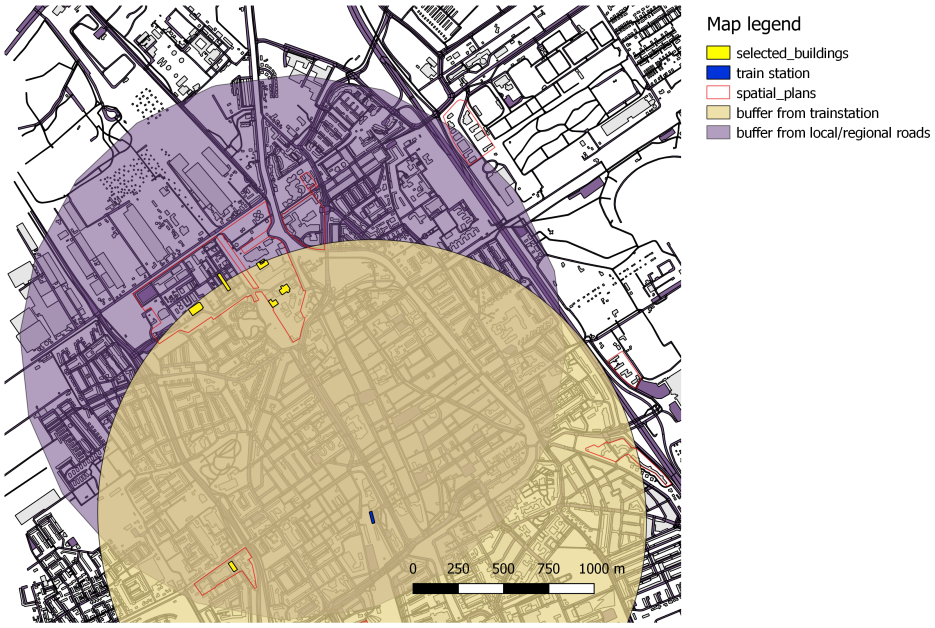


Figure 5.3: The visualization of the solution of the use case with mapping

As illustrated in table in Section 5.1, there is no single approach that can currently fully support the GeoSPARQL specification. More specific, PDOK's approaches do not separate object's geometry from the geometry representation. Moreover, the mapping approaches, when designed to separate object's geometry from the geometry representation, didn't work. Another problem for PDOK's GeoSPARQL endpoint is that it didn't support GML geometries. Finally, as mentioned in Section 5.1, none of the approaches supports inference of any form.

SQR_3: What benefits and drawbacks do the different linked data approaches offer over approaches with relational data?

The linked data approaches can improve reuse of data, as they are accessible though their URL and an update of the linked data does not require for the users to update their own data. Moreover, they are extensible, as any interested party can add information to the linked data, by publishing their linked data and connecting them with existing linked data. On the other hand, this methodology can backfire, since there is no official organization to check the linked data for their validity and if they are connected correctly, which can result in reduced quality or invalid data. On the other hand, in relational data approaches, the data are stored according to a specific data model and their accuracy is checked when accessed from reliable sources. Nevertheless, they are missing features that would allow for improved reuse, sharing and extension of the data. With that in mind, the best approach so far seems to be a combination of the two, so that they can complement and cover each others' vulnerabilities and promote their strengths. In this approach, the relational data can be stored with specific data models and they can be validated and then, with a mapping to be converted into linked data. In that way, the data are stored efficiently, their quality is ensured, they are easier to be reused and extended and if the relational data are updated, then so are the linked data, since the mapping will reference the updated data. Nevertheless, there are parts that still need to be improved, such as the efficient combination of different mapping processes into one query and the inference option without need of materialization of views.

SQR_4: How efficient is every approach, in terms of speed of execution and storage requirements?

Only the two mapping approaches have produced results and it is important to keep in mind that their performance was measured using sub-sets of BAG, BRK and BRT which describe Delft, Netherlands. Based on the query run-time, their performance was assessed and it can be regarded, that they can be used efficiently for solving spatial problems as the one in the case study. Another criterion for assessing each method's efficiency, was the data storage requirements. None of the two approaches requires physical storage of the linked data triplets, as a result they can be assessed as equally efficient. Moreover, they can be regarded equally efficient with the QGIS solution, since all three require physical storage of the relational data. It should be highlighted, that time efficiency corresponds to the run-time of the query and not the preprocessing time, i.e. the time required to load the data, make the mapping and write the query. Another remark is that although the direct mapping is easier to perform, it requires almost twice the run-time compared to the R2RML mapping. Having that in mind, in another case where the volume of data would be bigger, the direct mapping might not be appropriate, as it is less efficient than R2RML mapping. Finally, both mapping approaches can be considered efficient, because their run-time was close to the the QGIS solution's run-time.
One of the main reasons that the approaches might seem inefficient is that the research community is not as familiar with linked data approaches, as it is with relational data approaches. While performing the research for this thesis, the first queries that yielded results required around 10 hours to run, however, within a week of experimentation, their performance improved greatly, as shown in Table 5.1. The solutions, however, were not fine tuned, meaning that with appropriate query re-writing and optimization and with appropriate data modeling, there might have been faster solutions.
Answering the main research question:

RQ_1: To what extend is it feasible to solve spatial problems (positioning a shopping center) with a linked data approach using relational spatial data?

As presented in the Chapters 3 and 4 it is possible for linked data approaches to help solve a spatial problem. However, at the current state, they are considered black boxes and their result needs verification, as with the example of the on-the-fly transformation of the SRID of the spatial plans, where the benchmark QGIS solution was different from the designed mapping solutions.

5.3 LIMITATIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

The limitations can be split into two different categories. The first category is the functional limitations, i.e. certain necessary functionalities missing or not working properly. The second category is the familiarization limitations. Familiarization with linked data is limited or not accelerated because some functionalities have not been implemented. However, this type of limitations are solely related to the user experience and they do not affect the performance of the approaches.

5.3.1 Functional limitations

Initially, the approaches should be adjusted to comply with GeoSPARQL implementation. In that way, different data from different sources will be more easily accessed and re-shared. Additionally, the representation of geometries of real objects can be a

large string, which is problematic for storing them as explicit triplets in Oracle Spatial and Graph. Hence, storing large geometries' representations should be better supported in linked data approaches. Moreover, the performance of queries is fluctuating greatly based on many factors, such as the amount of data, but also the order of the implemented functions. Thus, a query optimization would be beneficial. Furthermore, the way federated queries are formed, combined with the management of credentials, results in difficulties accessing remote data sources. Also, on-the-fly transformation of data SRIDs and spatial index utilization is a "black box" procedure. Finally, inferencing is complicated and ineffective. A rulebase should be designed to hold the rule, then the rule should be designed without filter options and afterwards it should be updated with filter options written in SQL instead of SPARQL or GeoSPARQL. At the end, the rule can be used in a query. Finally, the inferencing during the mapping approaches is not supported without materializing the view, meaning that the data are going to be stored locally.

5.3.2 Familiarization limitations

Improving the error reporting in the process of creating, loading and querying the linked data would help with the familiarization, as currently only hints related to the errors are given and there is no overview over the process of the query. A better error reporting and overview of the process would help the user realize how Oracle Spatial and Graph works with linked data. Finally, visualization of the data would also improve the familiarization process, as users can immediately see the results of their queries.

5.3.3 Recommendations for future research

One of the main points for future improvement to be considered, are the maturation of the linked data technology and the familiarization of the research community with the linked data concept. As already mentioned in Chapter 2, GeoSPARQL specification is planned to be extended with additional spatial functions, however, additional steps towards maturation of linked data technology should be taken. Solving the aforementioned limitations is a step towards that purpose. Nevertheless, in order to get the research community to familiarize and use linked data approaches, there should be more data sources and data-sets published as linked data, and those data-sets should be checked a) for their quality, b) that they follow GeoSPARQL specification and c) that they are connected with each other in a meaningful way. Providing good quality data-sets via good services, APIs and Endpoints, will allow the community to familiarize, use and improve the linked data technology and build applications and products with it.
Technical limitations for future research:

- Query optimization.
- Inferencing capability with filter on SPARQL.
- Federated query inferencing without materialization of views.
- Storage limitation of objects that are more than 4000 bytes.
- Visualization of spatial linked data.
- Simplification of process for federated queries.

REFERENCES

G. Abhilakh Missier. Towards a Web application for viewing Spatial Linked Open Data of Rotterdam, 2015.

M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. A direct mapping of relational data to RDF. *W3C recommendation*, 27, 2012.

D. Beckett. RDF 1.1 N-Triples. *W3C recommendation*, 2014.

K. Bereta and M. Koubarakis. Ontop of geospatial databases. In *International Semantic Web Conference*, pages 37–52. Springer, 2016.

K. Bereta, G. Xiao, M. Koubarakis, M. Hodrius, C. Bielski, and G. Zeug. Ontop-spatial: Geospatial Data Integration using GeoSPARQL-to-SQL Translation. 2016.

K. Bereta, G. Xiao, and M. Koubarakis. Answering geosparql queries over relational data. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42, 2017.

T. Berners-Lee. Tim Berners-Lee on the next web, 2009. URL https://www.ted.com/talks/tim_berners_lee_on_the_next_web.

T. Berners-Lee, J. Handler, and O. Lassila. The semantic web. *Database and Network journal*, 36(3):7, 2006.

C. Bizer, F. U. Berlin, Germany, T. Heath, T. I. Ltd, U. Kingdom, T. Berners-Lee, M. I. of Technology, and USA. Linked data - the story so far. Technical Report context, 2009.

M. Chuck. Oracle database semantic technologies developer's guide, 11g release 2 (11.2), 2013.

D. Chuck Murray, Abugov, N. Alexander, B. Blackwell, R. Chatterjee, D. Geringer, M. Horhammer, Y. Hu, B. Kazar, R. Kothuri, S. Ravada, et al. Oracle Spatial Developer's Guide, 11g Release 2 (11.2), 2013. URL https://docs.oracle.com/cd/E11882_01/appdev.112/e11830/title.htm.

G. Garbis, K. Kyzirakos, and M. Koubarakis. Geographica: A Benchmark for Geospatial RDF Stores. In H. Alani et al., editors, *Geographica: A Benchmark for Geospatial RDF Stores*, volume II of *Part*, page 8219. Springer, 2013.

S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 query language. *W3C recommendation*, 21(10), 2013.

N. Humfrey. EasyRdf converter. http://www.easyrdf.org/converter, 2018. URL http://www.easyrdf.org/converter/.

E. INSPIRE. Directive: Directive 2007/2/ec of the european parliament and of the council of 14 march 2007 establishing an infrastructure for spatial information in the european community (inspire). *official Journal of the European Union, L*, 108 (1):50, 2007.

Kadaster. PDOK GeoSPARQL EndPoint, 2018a. URL https://data.pdok.nl/sparql.

Kadaster. PDOK API, 2018b. URL https://data.pdok.nl/sparql.

M. Koubarakis and K. Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *The Semantic Web: Research and Applications*, pages 425–439, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13486-9.

S. Kulk and B. van Loenen. Brave new open data world? *IJSDIR*, 7:196–206, 2012.

O. Lassila, R. R. Swick, et al. Resource description framework (rdf) model and syntax specification. 1998.

D. L. McGuinness, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

C. Murray, E. I. Chong, S. Das, M. Perry, K. Rieb, J. Srinivasan, S. Sundara, Z. A. Wu, and A. Yalamanchi. Oracle® Spatial and Graph RDF Semantic Graph Developer's Guide 12.2. 2017.

Ontop. Ontop: a platform to query relational databases as Virtual RDF Graphs using SPARQL, 2018. URL https://ontop.inf.unibz.it/.

M. Perry and J. Herring. OGC GeoSPARQL-A geographic query language for RDF data. *OGC Implementation Standard. Sept*, 2012.

M. Perry, A. Estrada, S. Das, and J. Banerjee. Developing GeoSPARQL Applications with Oracle Spatial and Graph. In *SSN-TC/OrdRing@ ISWC*, pages 57–61, 2015.

W3C. R2RML : RDB to RDF Mapping Language, 2012. URL http://www.w3.org/TR/r2rml/.

W3C. RDF 1.1 Turtle: Terse RDF Triple Language, 2014a. URL https://www.w3.org/TR/turtle/.

W3C. RDF 1.1 N-Quads, 2014b. URL https://www.w3.org/TR/n-quads/.

W3C. Further development of GeoSPARQL, 2016. URL https://www.w3.org/2015/spatial/wiki/Further_development_of_GeoSPARQL.

H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing*, volume 2001, pages 108–117. Citeseer, 2001.

WIkipedia. Service-oriented architecture, 2019. URL https://en.wikipedia.org/wiki/Service-oriented_architecture.

Z. Wu. Oracle Graph: Graph Features of Oracle Database 12c, 2014. URL http://www.nocoug.org/download/2014-02/NoCOUG_201402_Zhe_Wu_Oracle_Graph.pdf.

Appendices

A APPENDIX A

In this Appendix, Python3 code for converting relational data to linked data INSERT statements is presented. Also, the INSERT statements of the dummy data and the INSERT statements of the spatial plans are included.

Convert relational data to linked data INSERT statements for Oracle Spatial and Graph

```
#Authort:Evangelos Theocharous
#Date:12-04-2018
import csv
#Definition of the csv file to open
csvfile= 'buildings_wkt.csv'

with open(csvfile, 'r') as f:
    reader = csv.reader(f)
    csv_list = list(reader)
#Creation of empty lists that will be used for storing the csv values
geometry=[]
geometry_holder=[]
fid=[]
buildings_num=0
roads_num=0
types=[]
#vocab is actually a list of the linked data vocabularies used to connect data
#the linked data vocabularies have been chosen manually from http://lov.okfn.org
#with regard to the data they describe

vocab=[
        'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', \
        'http://www.w3.org/2000/01/rdf-schema#label',\
        'http://www.opengis.net/ont/geosparql#hasGeometry', \
        'http://www.opengis.net/ont/geosparql#asWKT']

#parse through the csv_list to separate the data. even though
#the data can be used as they are, their use gets easier if
#separated
for i in range(1,len(csv_list)):
    fid.append(i)
    geometry.append(csv_list[i][0])
    if "POLYGON" in csv_list[i][0] :
        buildings_num+=1
        #geometry_holder is a list with the same number of objects that are
        #all the same. It is used that way for easiness of automation.
        types.append('lot{}'.format(buildings_num))
        geometry_holder.append('http://www.example.org/geometries/lot_geom_')
    else:
        roads_num+=1
        types.append('road{}'.format(roads_num))
        geometry_holder.append('http://www.example.org/geometries/road_geom_')

#creation of a function that will accept the different csv cells, will format
#them accordingly and write them in an open txt file.
def queryWrite(index,fid,rep,vocab,data):
    '''
    queryWrite is a function that gets 5 inputs and writes in a txt file
    that should be already open. The outputs are formatted so they can run
    in Oracle Spatial and Graph
    '''
    print index,fid[rep],rep, vocab,data[rep]
    print "_____"
    #geometries are not assigned directly to the object. According to
    #GeoSPARQL obects have geometries and geometries have representations
    #hence a geometry that will hold the representation is created for each
    #object.
    if vocab == 'http://www.opengis.net/ont/geosparql#hasGeometry':
        s="\n INSERT INTO geometries_rdf VALUES ({0},\n".format(index)
        file.write(s)
        file.write("SDO_RDF_TRIPLE_S('geometries',\n")
        w="'<http://www.example.org/geometries/{0}>',\n".format(types[rep])
        file.write(w)
        file.write("'<{0}>',\n".format(vocab))
        file.write("'<{0}{1}> ')) \n".format(data[rep], rep+1))
    #the representations are assigned to the geometries created for each object.
    elif vocab == 'http://www.opengis.net/ont/geosparql#asWKT':
        s="\n INSERT INTO geometries_rdf VALUES ({0},\n"
        w=s.format(index)
        file.write(w)
        file.write("SDO_RDF_TRIPLE_S('geometries',\n")
        file.write("'<{0}{1}>',\n".format(fid[rep], rep+1))
        file.write("'<{0}>',\n".format(vocab))
        file.write("'\"{0}\"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'));
        \n".format(data[rep]))
    elif vocab == 'http://www.w3.org/2000/01/rdf-schema#label':
        s="\nINSERT INTO geometries_rdf VALUES ({},\n".format(index)
        file.write(s)
        file.write("SDO_RDF_TRIPLE_S('geometries',\n")
        w="'<http://www.example.org/geometries/{0}>',\n".format(types[rep])
```

```
            file.write(w)
            file.write("'<{0}>',\n".format(vocab))

            q="\'\"{0}\"@EN\')); \n".format(types[rep])
            file.write(q)

        #all the other data are treated with the same way
        else:
            s="\nINSERT INTO geometries_rdf VALUES ({},\n".format(index)
            file.write(s)
            file.write("SDO_RDF_TRIPLE_S('geometries',\n")
            w="'<http://www.example.org/geometries/{0}>', \n".format(types[rep])
            file.write(w)
            file.write("'<{0}>',\n".format(vocab))
            if "lot" in w:
                q="'<http://www.example.org/geometries/{0}>' )); \n".format('lot')
            else:
                q="'<http://www.example.org/geometries/{0}>' )); \n".format('road')
            file.write(q)

#open the txt file so the function can write on it.
file = open('query_file.txt','w')
#the index serves to store the triplets in the database. triples are stored
#in a index-triplet combination.
index=0
#call of the queryWrite function for each piece of data
#for each object in the csv.
for j in range(len(fid)):
    index+=1
    queryWrite(index,fid, j,vocab[0], types)
    index+=1
    queryWrite(index,fid, j,vocab[1], geometry_holder)
    index+=1
    queryWrite(index,fid, j,vocab[2], geometry_holder)
    index+=1
    queryWrite(index,geometry_holder, j,vocab[3], geometry)

#close and save the txt file
file.close()

'''
the output is a text, formatted to input the linked data in
Oracle Spatial and Graph database. it creates 10 triplets for each object
'''
```

Create the model to store dummy data into Oracle Spatial and Graph and insert dummy data in it.

```
execute SEM_APIS.create_sem_model('geometries', 'geometries_rdf', 'triple');


INSERT INTO geometries_rdf VALUES (1,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (2,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot1"@EN'));

 INSERT INTO geometries_rdf VALUES (3,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot1>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_1>  '));

 INSERT INTO geometries_rdf VALUES (4,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_1>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12674.1071417853 308239.986478844,12670.3570341201 308239.986478844,12670.3489162682
        308259.686904304,12671.916314559 308259.683258809,12671.9163102775 308259.693649078,12675.6745357946
        308259.6849081,12675.6745357946 308259.67451784,12677.8669520772 308259.669418662,12677.863932171
        308239.976450633,12674.1071417853 308239.976450633,12674.1071417853 308239.986478844),(12674.1071417853
        308239.986478844,12674.1168939741 308239.986478844,12674.1168939741 308239.993223618,12674.1071417853
        308239.993223618,12674.1071417853 308239.986478844))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (5,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot2>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (6,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot2>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot2"@EN'));

 INSERT INTO geometries_rdf VALUES (7,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot2>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_2>  '));

 INSERT INTO geometries_rdf VALUES (8,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_2>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12674.1396695509 308280.243640802,12674.1396695509 308280.254022339,12674.0666394816
        308280.254191788,12674.0666394816 308280.243810251,12670.3404419283 308280.252456,12670.3324601103
        308299.622753994,12671.899858467 308299.618948553,12671.8998541196 308299.629498768,12675.6340334909
```

```
        308299.620432663,12675.6340334909 308299.609882459,12677.8730760925 308299.604446352,12677.8701057959
        308280.234985218,12674.1396695509 308280.243640802))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (9,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot3>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (10,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot3>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot3"@EN'));

 INSERT INTO geometries_rdf VALUES (11,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot3>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_3>  '));

 INSERT INTO geometries_rdf VALUES (12,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_3>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12670.3322128704 308300.222756363,12670.3240990239 308319.913461677,12671.8914958125
        308319.913461677,12671.8914930332 308319.920206451,12675.7033092857 308319.920206451,12675.7033092857
        308319.911921132,12675.6302792163 308319.911921132,12675.6302792163 308319.905176358,12677.8761891987
        308319.905176358,12677.8731681025 308300.204447897,12674.1359152764 308300.213521465,12674.1359152764
        308300.224071669,12674.062885207 308300.224248976,12674.062885207 308300.213698772,12670.3322128704
        308300.222756363))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (13,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot4>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (14,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot4>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot4"@EN'));

 INSERT INTO geometries_rdf VALUES (15,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot4>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_4>  '));

 INSERT INTO geometries_rdf VALUES (16,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_4>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12670.3486690284 308260.286906502,12670.340689168 308279.652453812,12671.9080874552
        308279.648817039,12671.9080831773 308279.659198586,12675.67078152 308279.650468145,12675.67078152
        308279.640086608,12677.870013786 308279.634983817,12677.8670440871 308260.269420071,12674.1131396996
        308260.278150999,12674.1131396996 308260.288541258,12674.1033875107 308260.28856394,12674.1033875107
        308260.278173681,12670.3486690284 308260.286906502))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (17,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot5>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (18,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot5>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot5"@EN'));

 INSERT INTO geometries_rdf VALUES (19,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot5>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_5>  '));

 INSERT INTO geometries_rdf VALUES (20,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_5>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12678.4761868651 308319.889912311,12682.2430570617 308319.889912311,12682.2430570617
        308300.193838371,12678.4731678861 308300.202991175,12678.4761868651 308319.889912311))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (21,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot6>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (22,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot6>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot6"@EN'));

 INSERT INTO geometries_rdf VALUES (23,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot6>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_6>  '));

 INSERT INTO geometries_rdf VALUES (24,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_6>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
```

```
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12682.2468113362 308299.593827488,12682.2468113362 308280.224830121,12678.4701055895
        308280.233593062,12678.4730758762 308299.60298963,12682.2468113362 308299.593827488))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (25,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot7>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (26,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot7>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot7"@EN'));

 INSERT INTO geometries_rdf VALUES (27,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot7>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_7>  '));

 INSERT INTO geometries_rdf VALUES (28,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_7>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12678.4700135796 308279.633591661,12682.2202814849 308279.624890062,12682.2202814849
        308260.259295198,12678.4670438801 308260.268024576,12678.4700135796 308279.633591661))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (29,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot8>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (30,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot8>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot8"@EN'));


 INSERT INTO geometries_rdf VALUES (31,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot8>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_8>  '));

 INSERT INTO geometries_rdf VALUES (32,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_8>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12682.2240357594 308259.659284844,12682.2240357594 308239.962929478,12678.4639301046
        308239.962929478,12678.4669518702 308259.668023167,12682.2240357594 308259.659284844))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (33,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot9>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (34,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot9>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot9"@EN'));

 INSERT INTO geometries_rdf VALUES (35,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot9>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_9>  '));

 INSERT INTO geometries_rdf VALUES (36,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_9>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992>
POLYGON ((12665.9557434218 308300.233381866,12665.9557434218 308319.928725725,12669.7240926832
        308319.928725725,12669.7322122192 308300.224213087,12665.9557434218 308300.233381866))"
^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (37,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot10>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (38,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot10>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot10"@EN'));

 INSERT INTO geometries_rdf VALUES (39,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot10>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_10>  '));

 INSERT INTO geometries_rdf VALUES (40,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_10>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
```

```
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> POLYGON ((12669.7324594591 308299.624210718,12669.7404413037
        308280.253848158,12665.9594976963 308280.262620933,12665.9594976963 308299.633370982,12669.7324594591
        308299.624210718))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (41,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot11>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (42,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot11>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot11"@EN'));

 INSERT INTO geometries_rdf VALUES (43,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot11>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_11>  '));

 INSERT INTO geometries_rdf VALUES (44,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_11>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> POLYGON ((12669.7489156422 308259.688299801,12669.7570284976 308240.0,12666
         308240,12666.0 308259.697019126,12669.7489156422 308259.688299801))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>
        '));

INSERT INTO geometries_rdf VALUES (45,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot12>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (46,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot12>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot12"@EN'));

 INSERT INTO geometries_rdf VALUES (47,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot12>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_12>  '));

 INSERT INTO geometries_rdf VALUES (48,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_12>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12665.9962457255 308260.297029481,12665.9962457255
        308279.662534052,12669.7406885435 308279.653845969,12669.7486684025 308260.288301999,12665.9962457255
        308260.297029481))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (49,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot13>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (50,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot13>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot13"@EN'));

 INSERT INTO geometries_rdf VALUES (51,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot13>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_13>  '));

 INSERT INTO geometries_rdf VALUES (52,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_13>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12657.2430666092 308240.032914408,12653.492958944
        308240.032914408,12653.4848410921 308259.733339868,12655.0522393829 308259.729694372,12655.0522351014
        308259.740084642,12658.8104606185 308259.731343663,12658.8104606185 308259.720953404,12661.0028769011
        308259.715854225,12660.9998569949 308240.022886197,12657.2430666092 308240.022886197,12657.2430666092
        308240.032914408),(12657.2430666092 308240.032914408,12657.2528187981 308240.032914408,12657.2528187981
        308240.039659182,12657.2430666092 308240.039659182,12657.2430666092 308240.032914408))"^^<http://xmlns.oracle.com/
        rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (53,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot14>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (54,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot14>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot14"@EN'));

 INSERT INTO geometries_rdf VALUES (55,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot14>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_14>  '));

 INSERT INTO geometries_rdf VALUES (56,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_14>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12657.2755943749 308280.290076366,12657.2755943749
        308280.300457902,12657.2025643055 308280.300627351,12657.2025643055 308280.290245815,12653.4763667522
        308280.298891564,12653.4683849342 308299.669189558,12655.0357832909 308299.665384117,12655.0357789435
```

```
        308299.675934332,12658.7699583148 308299.666868226,12658.7699583148 308299.656318022,12661.0090009165
        308299.650881916,12661.0060306199 308280.281420782,12657.2755943749 308280.290076366))"^^<http://xmlns.oracle.com/
        rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (57,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot15>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (58,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot15>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot15"@EN'));

 INSERT INTO geometries_rdf VALUES (59,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot15>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_15>  '));

 INSERT INTO geometries_rdf VALUES (60,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_15>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12653.4681376944 308300.269191926,12653.4600238478
        308319.959897241,12655.0274206364 308319.959897241,12655.0274178571 308319.966642015,12658.8392341096
        308319.966642015,12658.8392341096 308319.958356696,12658.7662040403 308319.958356696,12658.7662040403
        308319.951611922,12661.0121140227 308319.951611922,12661.0090929264 308300.250883461,12657.2718401003
        308300.259957028,12657.2718401003 308300.270507232,12657.198810031 308300.27068454,12657.198810031
        308300.260134336,12653.4681376944 308300.269191926))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (61,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot16>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (62,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot16>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot16"@EN'));

 INSERT INTO geometries_rdf VALUES (63,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot16>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_16>  '));

 INSERT INTO geometries_rdf VALUES (64,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_16>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12653.4845938524 308260.333342066,12653.476613992
        308279.698889375,12655.0440122791 308279.695252603,12655.0440080013 308279.705634149,12658.806706344
        308279.696903709,12658.806706344 308279.686522172,12661.00593861 308279.68141938,12661.002968911
        308260.315855634,12657.2490645235 308260.324586562,12657.2490645235 308260.334976822,12657.2393123347
        308260.334999504,12657.2393123347 308260.324609244,12653.4845938524 308260.333342066))"^^<http://xmlns.oracle.com/
        rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (65,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot17>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (66,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot17>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot17"@EN'));

 INSERT INTO geometries_rdf VALUES (67,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot17>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_17>  '));

 INSERT INTO geometries_rdf VALUES (68,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_17>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12661.612111689 308319.936347874,12665.3789818856
        308319.936347874,12665.3789818856 308300.240273935,12661.6090927101 308300.249426739,12661.612111689
        308319.936347874))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (69,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot18>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (70,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot18>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot18"@EN'));

 INSERT INTO geometries_rdf VALUES (71,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot18>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_18>  '));

 INSERT INTO geometries_rdf VALUES (72,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_18>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
```

```
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12665.3827361601 308299.640263051,12665.3827361601
      308280.271265685,12661.6060304134 308280.280028626,12661.6090007001 308299.649425194,12665.3827361601
      308299.640263051))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (73,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot19>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (74,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot19>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot19"@EN'));

 INSERT INTO geometries_rdf VALUES (75,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot19>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_19>  '));

 INSERT INTO geometries_rdf VALUES (76,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_19>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12661.6059384035 308279.680027224,12665.3562063088
      308279.671325626,12665.3562063088 308260.305730762,12661.6029687041 308260.314460139,12661.6059384035
      308279.680027224))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (77,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot20>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (78,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot20>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot20"@EN'));

 INSERT INTO geometries_rdf VALUES (79,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot20>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_20>  '));

 INSERT INTO geometries_rdf VALUES (80,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_20>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12665.3599605833 308259.705720407,12665.3599605833
      308240.009365041,12661.5998549285 308240.009365041,12661.6028766942 308259.71445873,12665.3599605833
      308259.705720407))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (81,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot21>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (82,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot21>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot21"@EN'));

 INSERT INTO geometries_rdf VALUES (83,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot21>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_21>  '));

 INSERT INTO geometries_rdf VALUES (84,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_21>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12649.0916682457 308300.279817429,12649.0916682457
      308319.975161288,12652.8600175071 308319.975161288,12652.8681370432 308300.270648651,12649.0916682457
      308300.279817429))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (85,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot22>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (86,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot22>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot22"@EN'));

 INSERT INTO geometries_rdf VALUES (87,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot22>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_22>  '));

 INSERT INTO geometries_rdf VALUES (88,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_22>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12652.868384283 308299.670646282,12652.8763661276
      308280.300283722,12649.0954225202 308280.309056496,12649.0954225202 308299.679806546,12652.868384283
      308299.670646282))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (89,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot23>',
```

```
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (90,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot23>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot23"@EN'));

 INSERT INTO geometries_rdf VALUES (91,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot23>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_23>  '));

 INSERT INTO geometries_rdf VALUES (92,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_23>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12652.8848404662 308259.734735365,12652.8929533215
      308240.046435564,12649.1359248239 308240.046435564,12649.1359248239 308259.74345469,12652.8848404662
      308259.734735365))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (93,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot24>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/lot>' ));

INSERT INTO geometries_rdf VALUES (94,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot24>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"lot24"@EN'));

 INSERT INTO geometries_rdf VALUES (95,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot24>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/lot_geom_24>  '));

 INSERT INTO geometries_rdf VALUES (96,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/lot_geom_24>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12649.1321705494 308260.343465044,12649.1321705494
      308279.708969616,12652.8766133674 308279.700281533,12652.8845932264 308260.334737563,12649.1321705494
      308260.343465044))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (97,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road1>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (98,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road1>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road1"@EN'));

 INSERT INTO geometries_rdf VALUES (99,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road1>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_25>  '));

 INSERT INTO geometries_rdf VALUES (100,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_25>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12701.4105889035 308239.46389117,12639.8612276599
      308239.461956654)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (101,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road2>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (102,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road2>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road2"@EN'));

 INSERT INTO geometries_rdf VALUES (103,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road2>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_26>  '));

 INSERT INTO geometries_rdf VALUES (104,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_26>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12678.1394747002 308238.308080351,12678.1864776362
      308341.205799084)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (105,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road3>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (106,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road3>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road3"@EN'));
```

```
 INSERT INTO geometries_rdf VALUES (107,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road3>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_27>  '));

 INSERT INTO geometries_rdf VALUES (108,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_27>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12665.8066913201 308234.256235345,12665.6419240583
      308340.952386794)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (109,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road4>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (110,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road4>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road4"@EN'));

 INSERT INTO geometries_rdf VALUES (111,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road4>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_28>  '));

 INSERT INTO geometries_rdf VALUES (112,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_28>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12661.2619176019 308234.780601297,12661.4328342415
      308342.166147913)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (113,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road5>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (114,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road5>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road5"@EN'));

 INSERT INTO geometries_rdf VALUES (115,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road5>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_29>  '));

 INSERT INTO geometries_rdf VALUES (116,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_29>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12669.983054243 308339.93460348,12669.9773113177
      308233.941785828)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (117,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road6>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (118,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road6>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road6"@EN'));

 INSERT INTO geometries_rdf VALUES (119,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road6>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_30>  '));

 INSERT INTO geometries_rdf VALUES (120,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_30>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12653.3141662331 308233.602190549,12653.1522970116
      308342.69359371)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (121,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road7>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (122,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road7>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road7"@EN'));

 INSERT INTO geometries_rdf VALUES (123,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road7>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_31>  '));

 INSERT INTO geometries_rdf VALUES (124,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_31>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
```

```
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12639.8520729584 308259.82244585,12703.0050082814
        308260.578667184)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (125,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road8>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (126,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road8>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road8"@EN'));

 INSERT INTO geometries_rdf VALUES (127,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road8>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_32>  '));

 INSERT INTO geometries_rdf VALUES (128,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_32>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12640.9791063874 308280.051838176,12703.5376209845
        308279.926182568)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (129,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road9>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (130,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road9>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road9"@EN'));

 INSERT INTO geometries_rdf VALUES (131,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road9>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_33>  '));

 INSERT INTO geometries_rdf VALUES (132,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_33>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12641.2753390686 308300.004696089,12696.1720136441
        308299.890352442)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

INSERT INTO geometries_rdf VALUES (133,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road10>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/road>' ));

INSERT INTO geometries_rdf VALUES (134,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road10>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"road10"@EN'));

 INSERT INTO geometries_rdf VALUES (135,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road10>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/road_geom_34>  '));

 INSERT INTO geometries_rdf VALUES (136,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/road_geom_34>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Linestring (12640.8315204905 308320.676571266,12698.0553627381
        308319.999122042)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));


INSERT INTO geometries_rdf VALUES (137,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/mall zone>',
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
'<http://www.example.org/geometries/zone>' ));

INSERT INTO geometries_rdf VALUES (138,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/mall zone>',
'<http://www.w3.org/2000/01/rdf-schema#label>',
'"mall zone"@EN'));

 INSERT INTO geometries_rdf VALUES (139,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/mall zone>',
'<http://www.opengis.net/ont/geosparql#hasGeometry>',
'<http://www.example.org/geometries/zone_geom_35> '));

 INSERT INTO geometries_rdf VALUES (140,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/zone_geom_35>',
'<http://www.opengis.net/ont/geosparql#asWKT>',
' "<http://xmlns.oracle.com/rdf/geo/srid/28992> Polygon ((12637.0836435271 308280.009416401,12665.82920418
        308279.916432824,12665.8123540126 308231.012414785,12636.8463840927 308232.309119963,12636.8463840927
        308232.309119963,12637.0836435271 308280.009416401))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral> '));

BEGIN
  SEM_APIS.CREATE_ENTAILMENT(
    'rdfs_rix_geometries',
    SEM_Models('geometries'),
    SEM_Rulebases('RDFS'));
END;
```

Convert Spatial Plans zone relational data from csv to linked data in text

```
import csv
#Definition of the csv file to open
csvfile= 'zoneplans_fin.csv'

with open(csvfile, 'r') as f:
    reader = csv.reader(f)
    csv_list = list(reader)
#Creation of empty lists that will be used for storing the csv values
geometry=[]
fid=[]
plan_geom=0
num=0
desc=[]
id=[]
types=[]

vocab=[
        'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', \
        'http://www.opengis.net/ont/geosparql#hasGeometry', \
        'http://www.opengis.net/ont/geosparql#asWKT',\
        'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving']

for i in range(1,len(csv_list)):
    fid.append(i)
    id.append('<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/{}>'.format(csv_list[i][0]))
    desc.append(csv_list[i][1])
    geometry.append(csv_list[i][2])
    if "POLYGON" in csv_list[i][2] :
        num+=1
        types.append('<http://www.example.org/geometries/plan_geom_{}>'.format(num))


def queryWrite(index,id,vocab,data):
    '''
    queryWrite is a function that gets 5 inputs and writes in a txt file
    that should be already open. The outputs are formated so they can run
    in Oracle Spatial and Graph
    '''
    print (index,id,vocab,data)
    print ("-------------------------------------------------------------")
    #geometries are not assigned directly to the object. According to
    #GeoSPARQL obects have geometries and geometries have representations
    #hence a geometry that will hold the representation is created for each
    #object.
    s="\n INSERT INTO geometries_rdf VALUES ({0},\n".format(index)
    file.write(s)
    file.write("SDO_RDF_TRIPLE_S('geometries', \n")
    w="'{0}', \n".format(id)
    file.write(w)
    file.write("'{0}',\n".format(vocab))
    file.write("'{0}')) \n".format(data))

file = open('plans_query_file.txt','w')
index=142
for i in range(len(fid)):
    index+=1
    queryWrite(index,id[i], vocab[3], desc[i])
    index+=1
    queryWrite(index,id[i], vocab[0], '"permit_zone_plan"@en')
    index+=1
    queryWrite(index,id[i], vocab[1], types[i])
    index+=1
    queryWrite(index,types[i], vocab[2], geometry[i])

#close and save the txt file
file.close()
```

Insert Spatial Plans Zones Linked data into Oracle Spatial and Graph.

```
 INSERT INTO geometries_rdf VALUES (143,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581608>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (144,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581608>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (145,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581608>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_1>'))

 INSERT INTO geometries_rdf VALUES (146,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_1>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.3762612563397 51.98790702920597, 4.376263350930228 51.987904005060884, 4.376943753574503 51.986948516747326,
        4.377128117953809 51.98672059805304, 4.377431393709545 51.98632844995484, 4.377461398414205 51.986289660201294,
        4.377594878472772 51.98632553831148, 4.377924932732465 51.9864142578217, 4.377983964356868 51.98631862919731,
        4.378623789945699 51.98530098610542, 4.3788153220713335 51.984993782505306, 4.379035197414037 51.98464310910175,
        4.379003089917576 51.98449464220957, 4.379128171359684 51.984425973494695, 4.379835835264583 51.983272621495 72,
        4.380507179382048 51.98322128885373, 4.382003770888215 51.98346953585328, 4.384968360438684 51.98395502670559,
        4.3898233621641145 51.984737994275555, 4.3916346053994975 51.98504618044459, 4.392031295362774 51.98513515463301,
        4.392168727425034 51.98515955078484, 4.392144455796796 51.985503555777335, 4.392067459696317 51.9855885229584,
        4.391908478466101 51.98676442633432, 4.391958333465702 51.98686116822521, 4.39203639268762 51.98689994816308,
        4.392218801396181 51.98699056841684, 4.392566956427445 51.98704875945803, 4.392480338859966 51.987233591001775,
        4.392247136063524 51.98776197362936, 4.392202547475064 51.987861488591406, 4.392123974351311 51.98793098130157,
        4.39206313049736 51.98798159472577, 4.391999377021352 51.988077378977444, 4.3918894288234736 51.98825839032183,
        4.3914429493384715 51.9892832139784, 4.391200136159545 51.98983944160945, 4.391103677692423 51.99007024740849,
```

```
        4.3909392145863535 51.990398490799535, 4.390718304895863 51.990828754817514, 4.390489857459564 51.9912736962092,
        4.390269843176529 51.991708479467, 4.390060859452035 51.99206345472019, 4.38995486006301 51.9922566238396,
        4.389926842168248 51.99226923131506, 4.389898887813399 51.992278980900664, 4.389763143764131 51.99225072015805,
        4.389710802500061 51.992317341636046, 4.389904008722581 51.99236036364927, 4.389631291451751 51.99277472613006,
        4.389295489669865 51.99332268345545, 4.388471255068763 51.99442016760981, 4.3883045249579355 51.994641359281117,
        4.388026837052225 51.994965779500035, 4.387934893015462 51.9950463381502, 4.387880866807081 51.995084405229015,
        4.38778762500149 51.99511928978723, 4.387662030005217 51.9951410095649, 4.387525901113924 51.99514274756856,
        4.387298981573997 51.99513276565479, 4.387235644776673 51.99512731957939, 4.386536177651792 51.99506922072116,
        4.386094471847827 51.99500674907839, 4.384817038170228 51.99474363138934, 4.383719672315073 51.99451868271191,
        4.383746270830003 51.99448050047909, 4.383769453950531 51.99444723294538, 4.383097499448549 51.99428738848072,
        4.383213761535389 51.99409328800732, 4.383673348122012 51.99343621847293, 4.383728867968155 51.99334383786745,
        4.383485291386979 51.99327897382445, 4.38364522080934 51.99305345214256, 4.383756421129515 51.9928966434853244,
        4.383403151374627 51.992805827445, 4.383542615528506 51.992606739245666, 4.383557031026077 51.992592928472824,
        4.383947559454982 51.992035427666714, 4.383948065883799 51.992012789818425, 4.383982966682363 51.99196606147752,
        4.384017166251386 51.991950677753266, 4.384299968084725 51.991542044698164, 4.385107337143495 51.990383719444951,
        4.385113289256775 51.990369836421735, 4.3850773603167665 51.99033643825686, 4.385109477103418 51.9902879437370866,
        4.385180265329812 51.99027653448846, 4.385495122851291 51.98982096599537, 4.385501035877186 51.9898088241928,
        4.385487166383516 51.98979825610667, 4.385195418535407 51.9897243617486, 4.38515353712515 51.98970484663339,
        4.385154081810759 51.98968046698462, 4.385308358348337 51.98946755085408, 4.385264823795386 51.9893957699812,
        4.385028844909943 51.98933300739681, 4.385032452071037 51.98932779220501, 4.385144994856165 51.98916507463969,
        4.384272882043249 51.988934226927, 4.381934728981125 51.9883083388711, 4.380561362593588 51.98793777573522,
        4.380476133302827 51.98806114219291, 4.3803588383467975 51.988028910958924, 4.3798338789416915 51.98876287490449,
        4.377381022564466 51.98812111657934, 4.377201372731958 51.98814454658176, 4.3762612563397 51.98790702920597)) '))

 INSERT INTO geometries_rdf VALUES (147,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581238>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (148,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581238>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (149,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581238>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_2>'))

 INSERT INTO geometries_rdf VALUES (150,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_2>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.351582900410131 52.02246648376157, 4.351080185193309 52.02293532364372, 4.351433743453382 52.023093501548814,
        4.351453370604513 52.02310229447721, 4.351631703893146 52.02294889597935, 4.351751724693181 52.023005457450445,
        4.351374326018492 52.02333230596136, 4.351294523214585 52.02340287616328, 4.351218938230398 52.0234706159917,
        4.351140451843668 52.023540945985225, 4.351048113364927 52.023623666451975, 4.350935014338503 52.023715705305361,
        4.350829671914383 52.02367586556385, 4.350817084420782 52.02368650538647, 4.350808122752974 52.02369409396062,
        4.350618344004182 52.02361014874326, 4.350295124663935 52.023467176130985, 4.350289705471308 52.02346461153859,
        4.351007527970018 52.022824482508675, 4.351248222607836 52.022611228553956, 4.349344552380436 52.021781514777551,
        4.349356684153764 52.02177041265951, 4.348841812947098 52.021564666138424, 4.348919219544711 52.02140823601264,
        4.34908902818442 52.021065066611837, 4.349308011146669 52.020857537350636, 4.349258339467266 52.020832524818207,
        4.349669148073947 52.020567764761026, 4.349836489635789 52.02048895511078, 4.350733221677049 52.020066627079174,
        4.351220751766057 52.01994226104201, 4.35175701374705 52.01988157530402, 4.3523617137631305 52.01987081709676,
        4.352289156348695 52.02189205737399, 4.352288163872925 52.02191968812992, 4.35228427424188 52.02192179159292,
        4.352224357515894 52.021954191503866, 4.352024656659721 52.022088005179526, 4.351582900410131 52.02246648376157)) ')
        )

 INSERT INTO geometries_rdf VALUES (151,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580069>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (152,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580069>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (153,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580069>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_3>'))

 INSERT INTO geometries_rdf VALUES (154,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_3>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.338479959613826 52.015681339065466, 4.339165521650372 52.015088175868506, 4.339406388485761 52.01519352213432,
        4.340801580002899 52.01579075714832, 4.3414913512058035 52.01566345615106, 4.343325679758871 52.016465216762185,
        4.343365238656117 52.01643119422624, 4.343486293368353 52.016327089179924, 4.344327234391659 52.016701457099494,
        4.346134096264867 52.01751552287894, 4.346160829078372 52.017494860245094, 4.346210035782298 52.01745687710063,
        4.346448018126201 52.01755899092689, 4.346495243994461 52.01756426238051, 4.3465955532793625 52.017471251752625,
        4.346666280843197 52.0173735211251, 4.346743266207536 52.01722963418697, 4.347132447316089 52.01733527608163,
        4.3468321201318965 52.0176331190063, 4.346821818084063 52.017654533823456, 4.346822101764357 52.01766992961745,
        4.346827800469469 52.01768049690808, 4.346833499177226 52.017691064198544, 4.346851459585735 52.017706863479965,
        4.346866279717157 52.01771989341093, 4.346899407017632 52.017721315366856, 4.347346105418282 52.017740490552823,
        4.347351109768258 52.017736175148116, 4.347351900415702 52.01773620179466, 4.347435363026748 52.01766352607429,
        4.3483368032654015 52.01687857172027, 4.34826915837955 52.01685107121445, 4.3483717757787544 52.016747652266661,
        4.3484131818727258 52.01670591411895, 4.3488186656000059 52.016319223377295, 4.349636390593478 52.015554454472974,
        4.349933387689767 52.015264377748, 4.350102144848263 52.01532598182923, 4.350298003330204 52.015397469383316,
        4.350376487222375 52.015426169819435, 4.350450406292081 52.01545320302399, 4.350601529045338 52.015887381712666,
        4.350857225247491 52.01645765470749, 4.350873666189351 52.01649511204553, 4.350852195667284 52.01649875228192,
        4.350729705259281 52.01651952576895, 4.3504866339976844 52.016829899986064, 4.350896288588801 52.01780804801051,
        4.35081217094702 52.01781990987416, 4.3507470690739884 52.01782909861957, 4.3506750829753311 52.01783925144459,
        4.350680526133248 52.017853978116264, 4.350165211130991 52.018626487897855, 4.349762229368598 52.01914949383704,
        4.349088095732145 52.020081517275095, 4.34891837894398 52.02023226437245, 4.348482282527163 52.020398818800544,
        4.347956981649939 52.02044689358355, 4.347851331765914 52.02048167355596, 4.34590094329101 52.019625611056526,
        4.3456138448547525 52.01987544172856, 4.347594729195408 52.020743607505416, 4.347695403052623 52.020806537277835,
        4.347888775238392 52.02104511789327, 4.347976359051493 52.02115861982576, 4.348004538924179 52.021240208407356,
        4.347969277798777 52.021364041784196, 4.347899921333242 52.021582525837644, 4.347826257492018 52.02164675412943,
        4.347555673474332 52.02223841828995, 4.345273836249447 52.021221943681304, 4.343974982120345 52.020647081626667,
        4.347119259711697 52.01793609256838, 4.3466752144160195 52.01790873039479, 4.343599043206012 52.020480070534724,
        4.34337695632446 52.02037182281485, 4.343334781043615 52.02035039703286, 4.342481058912576 52.01997299683077,
        4.3421414580170055 52.01982371645725, 4.341534877455957 52.019551964234346, 4.341211433436058 52.01941233479198,
        4.341139850707367 52.019383159213874, 4.340404400344273 52.01905911836466, 4.339704326518566 52.01874787306269,
```

```
        4.338547083042094 52.018249395331374, 4.338435936310483 52.01819960477411, 4.337742777667552 52.01788909221245,
        4.337460138399 6 52.01776798237203, 4.33698361443099 52.01756379323286, 4.337955706254394 52.016709155199194,
        4.338428209223548 52.016891983075226, 4.338540136519056 52.016793105170 9, 4.33928593388155 52.01613424862531,
        4.338560656040111 52.01581659622761, 4.338479959613826 52.015681339065466))'))

 INSERT INTO geometries_rdf VALUES (155,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581573>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (156,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581573>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (157,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581573>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_4>'))

 INSERT INTO geometries_rdf VALUES (158,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_4>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.376006800042569 52.00955266179458, 4.376804845325307 52.00990599626354, 4.3769455123014165 52.009916902461015,
        4.377042817983047 52.00995421401023, 4.37708879438806 52.00991777247433, 4.377247501091794 52.00983142159487,
        4.377214026174578 52.009808888738476, 4.378070269422099 52.009342934767886, 4.378082355941027 52.009351487841094,
        4.3792292598568805 52.00900066604439, 4.380045568125386 52.00845381818664, 4.380113454961921 52.00841638544157,
        4.380141027552595 52.00837425719431, 4.380253142802562 52.00840197592859, 4.380390411407155 52.008442736693354,
        4.3805445229625875 52.00849106607129, 4.380262477910795 52.00870351264183, 4.3798508678512995 52.00901249503598,
        4.379375271359708 52.0092881101813, 4.3787758179226115 52.00962117050722, 4.377983340497593 52.010061562238505,
        4.37765670457606 52.01023373722507, 4.377631180465526 52.010237796992286, 4.377580389148485 52.01023451194091,
        4.3775296476599355 52.01022836886019, 4.377452128496016 52.01028192432537, 4.377307642357553 52.01025040165235,
        4.377298239127278 52.01026174573354, 4.376654454126907 52.01068747614451, 4.376501389012135 52.010707564390344,
        4.375512359084992 52.01049787264758, 4.375396697818221 52.01046646051555, 4.375290888036815 52.01055161871496,
        4.374601173735711 52.01027170951243, 4.374268021965823 52.010381571788805, 4.37359201485345 52.01020186478085,
        4.373618010345859 52.01008487471467, 4.373764271511574 52.01005902376104, 4.373994375043138 52.01000677509828,
        4.374627598942732 52.00986609276888, 4.37567481745486 52.00963692894796, 4.376006800042569 52.00955266179458))'))

 INSERT INTO geometries_rdf VALUES (159,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581607>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (160,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581607>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (161,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581607>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_5>'))

 INSERT INTO geometries_rdf VALUES (162,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_5>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.381480764064989 51.99671316461943, 4.3815189100786665 51.99666019611923, 4.381641992688379 51.996478119423095,
        4.381771203249046 51.99629416237484, 4.381951585281437 51.99603732981704, 4.38227440630549 51.99557712878811,
        4.382289622852218 51.99556674185663, 4.382307937343068 51.995558466775186, 4.382328570112396 51.99555265643598,
        4.382350685879809 51.99554954640233, 4.382373364632172 51.99554925466056, 4.382395673610719 51.99555181818964,
        4.383948728389157 51.99592924936108, 4.384746083699086 51.99612306497764, 4.386069704462636 51.99644477835046,
        4.386105105750622 51.99645383458436, 4.386118007205805 51.996458411758624, 4.3861298312340296 51.99646493931886,
        4.386137314301587 51.996472841187945, 4.386141444500437 51.99648155947941, 4.386140571934233 51.996490639708256,
        4.386135274016828 51.99650422162302, 4.386126713433004 51.9965211016292, 4.386258363049444 51.996555174332414,
        4.3863722737387313 51.99658284882456, 4.386630055652908 51.99794822445669, 4.386519992686232 51.99809510001093,
        4.386100002848796 51.998710719747635, 4.385999012211537 51.99885874169755, 4.385973537617568 51.998864169920786,
        4.385969510071188 51.99886323677797, 4.385505073135387 51.99875541924295, 4.385350869678866 51.99871962566199,
        4.384409443236826 51.9985010813617, 4.384232771904271 51.9984600701274, 4.3834180392912225 51.998269905434604,
        4.383051033697207 51.99817520579665, 4.382259330309521 51.99797091039228, 4.381587500584705 51.99779754868831,
        4.381554076314124 51.99778733945578, 4.381444561420492 51.99775390012623, 4.3808859487281895 51.99762185257406,
        4.380877359002232 51.9976198195682, 4.380867695458273 51.99757349862959, 4.381395363291 51.996831760840436,
        4.381452450269829 51.996706270942745, 4.381480764064989 51.99671316461943))'))

 INSERT INTO geometries_rdf VALUES (163,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580347>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (164,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580347>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (165,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580347>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_6>'))

 INSERT INTO geometries_rdf VALUES (166,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_6>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.363708033297585 51.98969419344394, 4.3636034661113055 51.98990728041982, 4.362960290680144 51.989747884958426,
        4.363036316545492 51.989593777381906, 4.362522850591458 51.98946660071645, 4.361768881646537 51.98928451678167,
        4.361228279969236 51.98917278339424, 4.361233659448096 51.989140012145484, 4.3612674725877065 51.98907609089899,
        4.361122382044578 51.98905342335913, 4.360816341094744 51.98899367515692, 4.360572672169469 51.98893589061153,
        4.3605034701807055 51.98892957891394, 4.359979392983602 51.98880355685028 6, 4.360068146200026 51.98866179603295,
        4.360135163912589 51.98866238104961, 4.360391270534690 6 51.98868174001494, 4.36062638044473 51.98870947258057,
        4.360900727636015 51.98874040536924, 4.361281971292637 51.98881647659632, 4.361401324704731 51.98883512628701,
        4.361769250867307 51.98889260780869 4, 4.361948751300213 51.98891181706523, 4.362073717739046 51.98891932390882 4,
```

```
        4.362227138301618 51.98892661107304, 4.362441436405115 51.988948990128044, 4.363115300040343 51.98905374298995,
        4.364240898517533 51.98922063172508, 4.364276747598682 51.989229410847166, 4.36471065835725555 51.989296724195796,
        4.3647634125199435 51.989347600832374, 4.364762941597986 51.9893484596658, 4.364714627026936 51.989438143626636,
        4.364551044640754 51.98974173021527, 4.363742631426051 51.989589352287396, 4.363708033297585 51.98969419344394))'))

 INSERT INTO geometries_rdf VALUES (167,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581610>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (168,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581610>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (169,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581610>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_7>'))

 INSERT INTO geometries_rdf VALUES (170,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_7>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.365866350248813 51.98935610996536, 4.36591185410775 51.989272747253786, 4.366080491383526 51.9892282151495,
        4.366867010187119 51.98784373337558, 4.368034510139758 51.985625393800134, 4.369201043914086 51.983445094322086,
        4.370459733668443 51.98130549176514, 4.372115607220747 51.98163314031588, 4.372132742571905 51.9816375668294,
        4.372296962121118 51.981679954547296, 4.372334964991484 51.981685990269746, 4.372461924754059 51.981692074214976,
        4.3725647032016814 51.98169438991019, 4.372609553233361 51.981704053145776, 4.372702202812376 51.98174551741089,
        4.372734185762252 51.9817622067331, 4.3728906501526374 51.98174072434858, 4.372997084534593 51.98173522397822,
        4.373159507778211 51.98175730731284, 4.373265315081715 51.981778911466485, 4.373562097631503 51.98183783764354,
        4.374104972068174 51.98194882423992, 4.374491347209178 51.9820351351061046, 4.374924937303447 51.98212141098066,
        4.375306799826134 51.98219889851077, 4.375866872522642 51.98231644282866, 4.376251020259965 51.98239537613088,
        4.376657067833098 51.982477345729166, 4.377099910033881 51.98256676693038, 4.3772622255839595 51.98259385039033,
        4.377130492949081 51.982799632091115, 4.37708191579621 51.98284227125301, 4.376880566592247 51.98320972036105,
        4.375888904614278 51.98434849205663, 4.37560439274302 51.98466710740078, 4.37551156465443 51.98478617462293,
        4.375401131732226 51.98496930624669, 4.37527290320491 51.985223637687504, 4.375137165771608 51.98550357638844,
        4.375069859586266 51.98561858418076, 4.374963557546594 51.985720411475526, 4.374901780683282 51.98579551160428,
        4.374846499094771 51.98588921145525, 4.374771554638451 51.98603555120155, 4.374689681402782 51.986181822292615,
        4.374560608009158 51.98647324252378, 4.374469443524779 51.98662086273464, 4.374361219501961 51.9868082911899,
        4.3739839400042255 51.98724170020825, 4.373925968085334 51.987352500154, 4.373758109127258 51.98743607042427,
        4.373565475508656 51.98767661399367, 4.373467570991064 51.9877988180635, 4.37324614168948 51.98807519405711,
        4.372419606676634 51.989106840311685, 4.372321400615887 51.98928842086526, 4.372207895979376 51.98943249398183,
        4.371900521048937 51.98983938061375, 4.371720338555123 51.99003903094857, 4.371578906068093 51.99016052532999,
        4.371379554843534 51.990389978370686, 4.371397781688954 51.99052057214022, 4.371041937969659 51.99112016260615,
        4.371006225751865 51.99117847914179, 4.370866595991506 51.99140650555543, 4.370634139452312 51.99135655133527,
        4.370310949934704 51.99128709790854, 4.368637279750192 51.99091704338098, 4.36795496296326 51.9907916919185,
        4.367481499247586 51.99070469505568, 4.365711366126788 51.99034120511169, 4.365674413474044 51.99033288427376,
        4.365273468648226 51.99024771249023, 4.36539173022786 51.99002756095149, 4.365609755954537 51.989639178113244,
        4.365777544671265 51.98933595922606, 4.365866350248813 51.98935610996536))'))

 INSERT INTO geometries_rdf VALUES (171,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580342>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (172,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580342>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (173,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580342>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_8>'))

 INSERT INTO geometries_rdf VALUES (174,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_8>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.342340483902607 52.003113406467975, 4.342906980108183 52.002637009787904, 4.343109792351131 52.00271474705241,
        4.343969103216005 52.003044114262465, 4.344038171737692 52.00312274068613, 4.3459331420089455 52.003851375333525,
        4.3460962449028155 52.003883461224376, 4.3471393880757745 52.00301372848215, 4.3471457725123495 52.00301663436574,
        4.347261194224912 52.00306928621958, 4.347120924242157 52.0035873679366, 4.346883314791773 52.00421288669462,
        4.346604085896403 52.00444695055705, 4.346848387600496 52.00455437862495, 4.346740451128668 52.00466289956679,
        4.3464876374706805 52.00489325958594, 4.346409635978667 52.004963240104836, 4.346312832881842 52.00505008815,
        4.346234798678853 52.0050141976922, 4.346190285207054 52.00499366897477, 4.345788497982696 52.00480839604669,
        4.3456899899693955 52.00476296666885, 4.345669200470022 52.00475005449312, 4.345517777088689 52.00468530707819,
        4.345496438190489 52.00467347765837, 4.345284550592746 52.00457834280402, 4.345064592188391 52.00447957651112,
        4.345016526483682 52.004457775422686, 4.344712654127796 52.0043199549019, 4.344357688032347 52.004162885179376,
        4.343996270039328 52.00400028296165, 4.343715295232683 52.003873880836295, 4.343012717907491 52.003553561115254,
        4.3427095619666969 52.00341363825287, 4.342450980089038 52.00329470374665, 4.3424308641262215 52.00328545542208,
        4.342242616967304 52.00319886646398, 4.342341247678149 52.00311375482522, 4.342340483902607 52.003113406467975))'))

 INSERT INTO geometries_rdf VALUES (175,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581605>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (176,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581605>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (177,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128581605>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_9>'))

 INSERT INTO geometries_rdf VALUES (178,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_9>',
```

```
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.363142266175266 52.001115607856086, 4.3630903941607375 52.00121750314813, 4.362815171850674
        52.001164132486664, 4.3625604517650665 52.00111474193449, 4.362582707987276 52.00107125904042, 4.362336664141078
        52.001023966012575, 4.362060766118025 52.00097093825072, 4.361804954693958 52.00092159047519, 4.361260098038067
        52.00081598778191, 4.360809689556828 52.000728689485676, 4.36041720578457 52.00065260947282, 4.35989646739018
        52.00055166027311, 4.359684842457149 52.00050624410016, 4.3597109214096506 52.00046128498324, 4.359998819676116
        51.999920361023385, 4.360180649020064 51.99959373912827, 4.360393311686558 51.99921340812965, 4.360509268171903
        51.999174895907544, 4.360907122229386 51.99845146726128, 4.361669584173857 51.997069791038236, 4.361949231172565
        51.99656545608603, 4.362048151815307 51.99638383586135, 4.362683546507149 51.99521722754561, 4.3629045983452395
        51.99481836748676, 4.363573144622333 51.99357658228026, 4.364172445252265 51.99247213404138, 4.364516532986204
        51.991837719436326, 4.364756689039207 51.99187613018356, 4.36506216265703 51.99131273178285, 4.367075911692352
        51.99183479362112, 4.367105673278637 51.99184694381903, 4.367254448036642 51.991893917938405, 4.367141542934066
        51.99226456679844, 4.367360865327995 51.99230502530436, 4.3673327373771444 51.992349759255966, 4.367208677859082
        51.992587462594 28, 4.367087667030246 51.99282614506104, 4.367047084729555 51.99291711951593, 4.36705044456103
        51.99297233092412, 4.367072502429093 51.99301912910755, 4.367113098845789 51.99306324753108, 4.367177246154108
        51.99308853179468, 4.36725539676035 51.99310823849441, 4.367381253914942 51.993130264224156, 4.367562403836621
        51.99316037346697, 4.367838866743498 51.99320081827636, 4.368794153054014 51.993343220514 42, 4.369147054174013
        51.993410966528444, 4.369373747621502 51.993539453889795, 4.369330629757564 51.993605669987005, 4.369067909511331
        51.99397822132519, 4.368902311995918 51.994212708606725, 4.3686906290129714 51.99450863977992, 4.368660228753856
        51.994557851037804, 4.3686304480213884 51.99458044248359, 4.368605570221566 51.99458971965768, 4.368528066774756
        51.994609974719346, 4.368344373777662 51.994656906024495, 4.368001724903428 51.99518905144763, 4.367574726356733
        51.99587172877714, 4.367287192869913 51.99631731119349, 4.367067095191612 51.996643602808646, 4.366904217376314
        51.996893338078916, 4.3666479548481245 51.9972849606781, 4.366372450256646 51.997709242975745, 4.366281365390899
        51.9978511501904, 4.366221727450538 51.99793197255111, 4.366105267930632 51.99807223028228, 4.365895958075862
        51.99833011873034, 4.365538216706162 51.99876347434576, 4.36552089068548 51.99876332393877, 4.365512037002036
        51.998770977455216, 4.36549427744509 51.99878985259046, 4.365164288204063 51.999236481837485, 4.365021241934641
        51.99942669221099, 4.364675689456675 51.99987972886254, 4.3645546104943636 52.000035639348712, 4.3643072514803425
        52.00036467370032, 4.364098674409929 52.000639931904 45, 4.363924036051199 52.000863754219 98, 4.363734114462939
        52.000793041798836, 4.363344874168175 52.00071754812238, 4.363142266175266 52.001115607856086))'))

 INSERT INTO geometries_rdf VALUES (179,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580343>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (180,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580343>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (181,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580343>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_10>'))

 INSERT INTO geometries_rdf VALUES (182,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_10>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.360031057204453 51.99713751533193, 4.360110283759582 51.996992803651395, 4.360654643280669 51.99599844598608,
        4.360980966818831 51.995402376025936, 4.361216666815774 51.9949717813518, 4.361346488915181 51.994734646513635,
        4.361701058640708 51.994086919527824, 4.36177109609752 51.99395897178116, 4.361934039318899 51.99366128865746,
        4.362550218682345 51.99253557495714, 4.362838004353004 51.992009798937794, 4.36384834593916 51.99222055054972,
        4.363587914968393 51.992700418331, 4.362828436276378 51.99410021730906, 4.36234482642764 51.994991966801926,
        4.36230662379335885 51.995062412036 56, 4.362214821202359 51.99509014298566, 4.3621580043730255 51.99510408414354,
        4.362016826439139 51.99513871073847, 4.361976673793102 51.995212896210184, 4.36162747445165 51.995857974226105,
        4.361304613231574 51.996454417672105, 4.36113940628661 51.99675938822306, 4.361109543233933 51.99684769467654,
        4.361090051634658 51.99690530485121, 4.360950780056857 51.99716812799338, 4.360794099723018 51.997461720682914,
        4.360101583683081 51.9973230488609, 4.360111380148383 51.997304779189065, 4.360172026939607 51.99719179764706,
        4.360172753339343 51.99716707574421, 4.360031057204453 51.99713751533193))'))

 INSERT INTO geometries_rdf VALUES (183,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580351>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Geometrie ingetekend door Kadaster op basis van luchtfoto's. Voor de objectklasse Terrein is het type_landgebruik ook
        gebaseerd op BRP-Gewaspercelen 2017 bron: RVO.nl"'))

 INSERT INTO geometries_rdf VALUES (184,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580351>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (185,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580351>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_11>'))

 INSERT INTO geometries_rdf VALUES (186,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_11>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.365594311742656 51.98828700192974, 4.365597546650875 51.98813057987711, 4.3656447312174835 51.98791932119256,
        4.365709600951493 51.98777718712787, 4.365840484976764 51.98761184284736, 4.366036785109382 51.98744944429757,
        4.366053164 25275 51.98740677727614, 4.366147651662239 51.98731722217358, 4.366265454067965 51.98704949775443,
        4.366293107219294 51.98684996100352, 4.366354392931012 51.98669590390053, 4.3666507873250255 51.98602779654234,
        4.366728557623298 51.98582631633623, 4.367133905999845 51.985109209421715, 4.3672930186848955 51.9847181704546,
        4.367469113041982 51.98442716708582, 4.367602627623261 51.98414530743828, 4.367929091504944 51.98350837659443,
        4.368065262715345 51.983278861619596, 4.368201573733781 51.98317372343314, 4.3683302565785 51.98319123743639,
        4.368328963479 0095 51.98319349029048, 4.368310842634722 51.98322924359351, 4.3683044722326425 51.983241808691126,
        4.367917455953625 51.98397503422564, 4.367666768696027 51.984446625655494, 4.367880719770058 51.98448985493913,
        4.367734671761209 51.984737264601655, 4.36759739676241 51.98499317263787, 4.367418922453184 51.98532584733787,
        4.367247065284971 51.98564461955975, 4.367072792844073 51.98596834142648, 4.366844301851938 51.9863925550898,
        4.366440676688762 51.98713809090131, 4.366323844559033 51.98735421975475, 4.366242909412535 51.987504008023,
        4.366181762391153 51.98761720375474, 4.36608606544984 51.98779440551883, 4.365973683659276 51.98800227601745,
        4.365908546929341 51.98811735160821, 4.365848223995776 51.98821601944136, 4.365781398054705 51.9883253274704,
        4.365594311742656 51.98828700192974))'))

 INSERT INTO geometries_rdf VALUES (187,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580344>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Geometrie ingetekend door Kadaster op basis van luchtfoto's. Voor de objectklasse Terrein is het type_landgebruik ook
        gebaseerd op BRP-Gewaspercelen 2017 bron: RVO.nl"'))

 INSERT INTO geometries_rdf VALUES (188,
```

```
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580344>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (189,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580344>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_12>'))

 INSERT INTO geometries_rdf VALUES (190,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_12>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.366387984148 47 51.99967025864003, 4.365592728603758 51.99946337860446, 4.365900975404673 51.99901513646305,
       4.3659540920476685 51.99900461306221, 4.366246693613885 51.998946657026146, 4.367911614194562 51.99937623629445,
       4.368169196630712 51.99899002463577, 4.36653546041603 51.99855813956735, 4.3664945305003 68 51.99850865177211,
       4.366889476934521 51.99793600177631, 4.36730437745544 51.99727309608662, 4.3676259585560135 51.99676557906118,
       4.3677067117417 53 51.99666149614196, 4.3681019528289236 51.99613854400933, 4.368262597108159 51.99588543500388,
       4.3685517789657 82 51.99543083982426, 4.3686373999338353 51.99529623650658, 4.3698908680326 86 51.9934506127701 54,
       4.370020439731845 51.99327173210268, 4.370075860378391 51.99319523104998, 4.370355474281442 51.99285599223663,
       4.370368607668105 51.99286050124643, 4.370606138476985 51.99252442183913, 4.370626595717388 51.99253021656821,
       4.3706676780246685 51.99247466113414, 4.370829691763472 51.992253695675416, 4.3715443339612 51.99242338122653,
       4.37150209104771 51.99248265729097, 4.371707695102433 51.99252467461417, 4.3724249777187785 51.99269189662306,
       4.372392706695192 51.99273779377645, 4.372207195952845 51.993041839887894, 4.371727786737855 51.99292473264008,
       4.371718204003183 51.99292224094 84, 4.371580460487332 51.9931356867319, 4.37160482361596 51.99314198241506,
       4.37165166795102 51.99315409010753, 4.37210124903972 51.99324847734527, 4.371911602349611 51.9935536559017 35,
       4.371716534515615 51.99384412666834, 4.371644040504594 51.99395514194799, 4.3711780252696695 51.99468803011005,
       4.3709401526521 51.99463150400194, 4.370933302312755 51.994642716798744, 4.37090681730169 51.99468504105429,
       4.370900517321743 51.99469509904702, 4.370831665960667 51.994680580765454, 4.370824314287275 51.99469206787911,
       4.370681474173684 51.99465584970101, 4.370657295950763 51.994649717227276, 4.370590972888273 51.99475426831874 4,
       4.371049983754308 51.99486623313835, 4.371062522122881 51.99486939759634, 4.370839452698298 51.9952576660733,
       4.370580026884445 51.99563531893017, 4.370519476953065 51.99572215799863, 4.370470907841081 51.9958136309035 55,
       4.370293745649101 51.9960713737 3147, 4.369721579866359 51.99707157348738, 4.3692231955016 29 51.99796698877088,
       4.36885048797273 51.99788299992789, 4.3685156146143145 51.998377164677024, 4.368876311678954 51.99846692032368,
       4.368939746906793 51.99848354117218, 4.368865499243213 51.99861454539689, 4.368555554909451 51.99916141165634,
       4.368300639128611 51.99961117392904, 4.368278100536074 51.999646573346965, 4.366645857362554 51.99924753397901,
       4.36638798414847 51.99967025864003))'))

 INSERT INTO geometries_rdf VALUES (191,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580345>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (192,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580345>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (193,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580345>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_13>'))

 INSERT INTO geometries_rdf VALUES (194,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_13>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.3710840672308 65 51.99190068214905, 4.3711794185787 59 51.99178389604652, 4.371322085458464 51.9916081833098 15,
       4.37148388523591 51.991404096134545, 4.371679336477298 51.99114322031865, 4.371969631038768 51.990776132939196,
       4.372257742793326 51.99041907554544, 4.372292945363199 51.990378677 91455, 4.373067069297041 51.98946850441181,
       4.373207842293368 51.98932889020992, 4.373322416753913 51.98921527041535, 4.373675958978379 51.9888102071569,
       4.373754559068065 51.98870671307964, 4.3737339336789365 51.988699398332265, 4.3740226355382985 51.98814675 4985884,
       4.374228080997998 51.98789781756 9646, 4.374284223118787 51.98782420631 2006, 4.374394796876492 51.98767920720798,
       4.3744811776572945 51.98756946953603, 4.374624967473217 51.98740345269 6206, 4.374861361519954 51.98713049289 7016,
       4.375092534447458 51.98687458441 86, 4.375249588912571 51.98667473033577, 4.375451796952435 51.98641961478625,
       4.375503765623702 51.98635207049438, 4.375673213641938 51.98613184601586, 4.375797069631984 51.98596880255977,
       4.375872877350537 51.98588669447 44, 4.375966062873807 51.98579187275755, 4.3763633268014726 51.9853876003371,
       4.376563476256948 51.98515352607 5194, 4.376595968467675 51.98511417364423, 4.376670370591095 51.98501613388207,
       4.376686314315049 51.98498887302089, 4.376687609530764 51.98498889313425, 4.377023619532587 51.98445961593308,
       4.377186866899944 51.98449669396508, 4.377200851836478 51.98449110614845, 4.377208834376305 51.98447666678738,
       4.377236999643 7686 51.98442577141248, 4.377072717852248 51.98437128229057, 4.377066779025983 51.98436930770181,
       4.377156208058611 5 51.98421606176495, 4.377362926359374 51.98386177282988, 4.377574319631316 51.98360816204841,
       4.377677562672035 51.98348263884654, 4.377819046330536 51.98331059416302, 4.378084218797213 5 51.98299832476926 06,
       4.378593925540986 5 51.98309188234818, 4.37860221827346 05 51.9830934006363, 4.378391014457 12 51.98341146429782 4,
       4.378198537411428 51.98370131067049, 4.377997878940638 51.98402145966019, 4.377617330930389 51.98462990795190 6,
       4.377625277616089 51.98463193569 7195, 4.377316783766237 51.98513554508 8724, 4.377182391701347 51.98534790192432,
       4.377163753209803 51.98534420033795, 4.376869885947321 51.98528589287 3725, 4.376768247701218 51.98547748771 7496,
       4.377044273818509 51.98553235242 9544, 4.3770586658475885 51.98553520858516, 4.377016775611577 51.98559754566309,
       4.376927014061966 51.98573114821333, 4.376786804896559 51.98593983222365, 4.376759427063858 51.98598090954 0714,
       4.37665032179219 51.98614461999212, 4.376518448393638 51.98634248091251, 4.3763033972474785 51.98667745208755,
       4.37617992864814 51.98687303881366, 4.376097427534073 51.98700402453 5466, 4.376069931636027 51.98704769845496,
       4.375951494135096 51.98723442931 0334, 4.3752813944717035 51.98829096284 6804, 4.375050132889041 51.98864600836363,
       4.37490320539425 51.98887417421414 6, 4.374862195777506 51.98894366427362, 4.374672726279711 51.98916332899024,
       4.3746437816573716 51.98919688674866, 4.3742247069485956 51.98978914659 3644, 4.373755614879458 51.99075482718521,
       4.373733788026854 51.99074522793202, 4.373579753984123 51.99100247278493, 4.373567875877 13 51.99102231659 7586,
       4.373309125325257 51.99096195687621, 4.3732821887619435 51.99100513166209, 4.372673368798055 51.99086253537 4006,
       4.372384037152819 51.99079477282927, 4.372004982730213 51.99140907837 5916, 4.372801842142345 51.99159464769765,
       4.372877633965568 51.99147464444733, 4.372888476386293 51.99145747942 18, 4.372919584601179 51.99140821031 81,
       4.373075207289025 51.99115692155 5955, 4.373432528147855 51.99124057605 1666, 4.373233261353 38 51.99156632091738 6,
       4.372855317450418 51.99218839611 46, 4.372781525461361 51.99223795344133, 4.372724716239 6345 51.99227890204701,
       4.372706702413138 51.99229188836 6715, 4.371084067230865 51.99190068214905))'))

 INSERT INTO geometries_rdf VALUES (195,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580425>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (196,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580425>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (197,
SDO_RDF_TRIPLE_S('geometries',
```

```
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/128580425>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_14>'))

 INSERT INTO geometries_rdf VALUES (198,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_14>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.358597055444663 51.988162209438144, 4.358324450890514 51.98811962798299, 4.358170152631336 51.988095527928614,
        4.356665975473394 51.9876651830268, 4.356818757594669 51.9872950222401, 4.357809396097418 51.98744946513579,
       4.357770187794037 51.987545311421016, 4.358123028849166 51.987600326391174, 4.358162222306861 51.987504461885656,
       4.358890168006271 51.98761822458334, 4.358776630732481 51.98789474392972, 4.358806429145904 51.98790126958869,
       4.358788504636224 51.98794391765807, 4.358757987708769 51.98794015426949, 4.358723986843401 51.988022940690485,
       4.35875731330402 51.98802735786222, 4.358742430725704 51.988071947132674, 4.358705606209138 51.988067697133566,
       4.358689214355552 51.98810762598741, 4.358672292750129 51.98810652525341, 4.358661048510578 51.98812640909846,
       4.358651609129886 51.98813488393521, 4.358634556456218 51.988139498936995, 4.358619112476846 51.98814126056722,
       4.3586037409031455 51.988139229548125, 4.358597055444663 51.988162209438144))'))

 INSERT INTO geometries_rdf VALUES (199,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673265>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Geometrie ingetekend door Kadaster op basis van luchtfoto's. Voor de objectklasse Terrein is het type_landgebruik ook
       gebaseerd op BRP-Gewaspercelen 2017 bron: RVO.nl"'))

 INSERT INTO geometries_rdf VALUES (200,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673265>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (201,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673265>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_15>'))

 INSERT INTO geometries_rdf VALUES (202,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_15>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.377977451110951 52.01371723832589, 4.377910426426693 52.0137746093743, 4.377717285685752 52.01394172682826,
       4.377079918379685 52.014489654454806, 4.376943929273028 52.014605332810135, 4.376751974407459 52.014768638665295,
       4.376443465288298 52.015031878690024, 4.376438203387706 52.01503394587959, 4.3764325013245 52.01503550591794,
       4.376426476632225 52.015036514870324, 4.376420289728577 52.01503696512498, 4.376414043176896 52.01503683059618,
       4.376407911554326 52.01503612177157, 4.376402025743281 52.01503484876359, 4.375582943793943 52.01469639701533,
       4.375234517291923 52.014551501095596, 4.3759221963146775 52.01404281384219, 4.3764218270704305 52.01373031666496,
       4.376953080613399 52.01345090909255, 4.377268238720507 52.01331947740067, 4.377583466266259 52.01322020004928,
       4.377977451110951 52.01371723832589))'))

 INSERT INTO geometries_rdf VALUES (203,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673269>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (204,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673269>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (205,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673269>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_16>'))

 INSERT INTO geometries_rdf VALUES (206,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_16>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.361073324260292 51.98845610243662, 4.361050562628338 51.98846589049918, 4.360981223695222 51.988491685898204,
       4.36090286118002 51.98849570355578, 4.36086646745817525 51.98849750980422, 4.360440096715902 51.988433292827274,
       4.360084363596958 51.988411347448604, 4.359941672827598 51.98841242984917, 4.359644331916006 51.988423127938844,
       4.359577313274757 51.98842958986221, 4.359430468132074 51.988449314117375, 4.359288605208369 51.98842125223104,
       4.359258712519115 51.988414375309674, 4.359163444395293 51.98839244621228, 4.358871559329293 51.988330272746246,
       4.358914559371575 51.988248050323854, 4.359002652372367 51.988040549325554, 4.359020663901387 51.988025857199624,
       4.359031606940072 51.9880197415546, 4.359044943165475 51.98801529177428, 4.3590592446292895 51.98801255830193,
       4.359074291548161 51.9880116021382, 4.359088435640013 51.98801192348426, 4.359103712231846 51.98801238056764,
       4.359300211740645 51.98803355817687, 4.3593422098967025 51.988013439565606, 4.359473179720299 51.98795070012289,
       4.3595772515804425 51.98790987405507, 4.359691312368095 51.98786135081707, 4.3598159016532785 51.98782034416473,
       4.359872046733714 51.987790865698386, 4.359960202322037 51.98772599008282, 4.360075659304641 51.987765874657924,
       4.3601967867339 71 51.98777939049236, 4.3601931557644455 51.987788545365724, 4.360165208724204 51.98785885461565,
       4.360024015367233 51.98821234761647, 4.360158980237368 51.98823327420909, 4.3602327485683 51.98805725196461,
       4.360612589599733 51.988116063639154, 4.360711585375921 51.98787528896265, 4.36105380789965 51.98798774490257,
       4.361175523425988 51.9881229419007 06, 4.36117506043618 51.9882442312037 94, 4.361073324260292 51.98845610243662))'))

 INSERT INTO geometries_rdf VALUES (207,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673267>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (208,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673267>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (209,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673267>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_17>'))

 INSERT INTO geometries_rdf VALUES (210,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_17>',
'http://www.opengis.net/ont/geosparql#asWKT',
```

```
'POLYGON ((4.363573827167549 51.988915637842624, 4.363160734389073 51.98885118909349, 4.362835855231124 51.988800244047724,
        4.362761197261257 51.988788663658646, 4.362699571204781 51.988778940522735, 4.362570792256072 51.98875861006504,
        4.362406166785185 51.9887338863394, 4.362370158801547 51.9887295457290065, 4.362334009139536 51.9887256712917,
        4.362297746294917 51.988722290241796, 4.362261356120763 51.98871938447855, 4.362224882079928 51.988716963369114,
        4.362132097394921 51.988714330336, 4.362077504715288 51.988711571553395, 4.362046485423297 51.98871000690572,
        4.36200887109713 51.98870715332004, 4.361971386989396 51.98870369860657, 4.361934062830561 51.98869961605817,
        4.361896927118781 51.988694932889565, 4.361860008968284 51.988686949354804, 4.361657848801532 51.98865798171529,
        4.361518633583932 51.98863629171267, 4.36137943305642 51.988614601672225, 4.361379310850562 51.98861485229353,
        4.361360142468169 51.988611994148194, 4.361594493939313 51.98801264745965, 4.361614581311996 51.9880325316453,
        4.362504692202961 51.98817037301301, 4.3625833922236345 51.98818214158293, 4.362484500688157 51.98842967713378,
        4.3629189159279855 51.98849676786512, 4.362957570329057 51.98850316281659, 4.363132102677817 51.988512268550956,
        4.363479024765139 51.988569507981516, 4.363517345513963 51.98857840772222, 4.3635464214658946 51.98859558866124,
        4.363583114600528 51.98863306556943, 4.363597939950899 51.98863990918391, 4.363606490068309 51.98864484651099,
        4.363614449317673 51.98865015622716, 4.363621760088576 51.98865581086489, 4.36362840884987 51.98866176536266,
        4.363634337785926 51.98866800123996, 4.363639841712615 51.98867499746834, 4.363644393526162 51.98868224608931,
        4.3636479360259735 51.988689701661414, 4.363650470236845 51.98869731924948, 4.363651953719432 51.9887050445516,
        4.3636524170215765 51.98871281491202, 4.3636518031474125 51.98872057590196, 4.363647479182472 51.98873588218257,
        4.3636765666245765 51.98874044085302, 4.363672682055077 51.98874989025395, 4.363705844901507 51.9887551405515,
        4.363639319532529 51.98891609073436, 4.363577596956838 51.98890675374091, 4.363573827167549 51.988915637842624))')')

 INSERT INTO geometries_rdf VALUES (211,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673268>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Een orthogerectificeerde fotografische opname van een deel van het aardoppervlak. Gemaakt vanuit een vliegtuig."'))

 INSERT INTO geometries_rdf VALUES (212,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673268>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (213,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/129673268>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_18>'))

 INSERT INTO geometries_rdf VALUES (214,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_18>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.365348641177259 52.02556170513231, 4.365335390518194 52.025601545291416, 4.365295924349102 52.02564210177025,
        4.365203187014508 52.0257183577867, 4.3642465026582835 52.02655386400592, 4.363610853445758 52.02710581579939,
        4.363595124837231 52.027137310608744, 4.3636259875627355 52.02722022174807, 4.363665466520547 52.027306776314354,
        4.36322521229632 52.02739647717618, 4.363206201916354 52.02740195682505, 4.3631536377932765 52.02740720760897,
        4.363102719448864 52.027408670426766, 4.363050474392958 52.027400602562466, 4.363010554480718 52.027392650860456,
        4.362935500309 27 52.02737011037489, 4.362781469564685 52.02729456810156, 4.362701035950335 52.02723774238636,
        4.362492920106717 52.027098942507884, 4.3618447770002655 52.02664022989518, 4.3617811825371495 52.02659520860357,
        4.361777159262638 52.02656853979218, 4.361776072313027 52.02654854825271, 4.361784258067638 52.02652769363715,
        4.361886707538258 52.02637441003643, 4.361961509869009 52.02626206327573, 4.362314269676551 52.02584177113416,
        4.362316282422575 52.025753797403034, 4.362391965893609 52.02564742659008, 4.362666355513691 52.02528118404298,
        4.362915283353454 52.02501697584199, 4.363937088072297 52.0241703173832 66, 4.365950263213056 52.02506828053179,
        4.3659466326267555 52.025071261411014, 4.365864089625921 52.025139029931694, 4.365844873457043 52.02515954364366,
        4.3654128170583455 52.025534374877466, 4.365394175968993 52.025525947373936, 4.365348641177259 52.02556170513231))')
        )

 INSERT INTO geometries_rdf VALUES (215,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130008704>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Geometrie ingetekend door Kadaster op basis van luchtfoto's. Voor de objectklasse Terrein is het type_landgebruik ook
        gebaseerd op BRP-Gewaspercelen 2017 bron: RVO.nl"'))

 INSERT INTO geometries_rdf VALUES (216,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130008704>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (217,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130008704>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_19>'))

 INSERT INTO geometries_rdf VALUES (218,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_19>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.356086692893123 52.00153007431171, 4.356009322699651 52.001423695177486, 4.355994224052146 52.001429234836024,
        4.355975518022042 52.001433951868506, 4.35594232465461 52.00143979140454, 4.355916730714605 52.00144194918098,
        4.35590494401545 52.00144216050925, 4.355886558622214 52.001441801651225, 4.35587156925676 52.00144069052133,
        4.3558258824440514 52.001432514834136, 4.355297142563988 52.00130655800105, 4.3551167879965895 52.00126359200114,
        4.355276895878901 52.00100869753948, 4.3549575896663005 52.00093239579616, 4.3553203476099 84 52.00035401783585,
        4.355018028966968 52.00028013933369, 4.355028548556187 52.000264105639026, 4.354855693479192 52.00022071957279,
        4.354830904775051 52.00021449759361, 4.354923748513728 51.999874015219646, 4.355002055339329 51.999892913402896,
        4.35507720510871 51.999911046762136, 4.3551110880480 95 51.999860655915306, 4.355154921649016 51.99987144937193,
        4.355301196856172 51.99963774611049, 4.3553047647537255 51.99963205151159, 4.355307639891684 51.99962861602271,
        4.355311409885466 51.99962552995518, 4.3553159848935 63 51.999622900387166, 4.355321232635509 51.99962078009164,
        4.355326990883861 51.99961925753442, 4.355333069739197 51.99961835801697, 4.355339322568396 51.99961812519765,
        4.355345545739418 51.999618548302294, 4.355351564533377 51.99961962579904, 4.355393363000006 51.99962961929638,
        4.3554876600029 04 51.99949217986298, 4.355744758604993 51.99856321805291, 4.3557978471809395 51.99837230224215,
        4.3558367100840 28 51.998357928122424, 4.355862971339051 51.99834820763435, 4.3578028732829885 51.998736043572364,
        4.357817853658977 51.99871023285987, 4.3571395521507 92 51.9999639557734, 4.356580733233794 52.000912683924334,
        4.356280893272365 52.001384528479534, 4.356086692893123 52.00153007431171))')')

 INSERT INTO geometries_rdf VALUES (219,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130650985>',
'http://brt.basisregistraties.overheid.nl/def/top10nl#bronbeschrijving',
' "Geometrie ingetekend door Kadaster op basis van luchtfoto's. Voor de objectklasse Terrein is het type_landgebruik ook
        gebaseerd op BRP-Gewaspercelen 2017  bron: RVO.nl"'))

 INSERT INTO geometries_rdf VALUES (220,
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130650985>',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'"permit_zone_plan"@en'))

 INSERT INTO geometries_rdf VALUES (221,
```

```
SDO_RDF_TRIPLE_S('geometries',
'<http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied/130650985>',
'http://www.opengis.net/ont/geosparql#hasGeometry',
'<http://www.example.org/geometries/plan_geom_20>'))

 INSERT INTO geometries_rdf VALUES (222,
SDO_RDF_TRIPLE_S('geometries',
'<http://www.example.org/geometries/plan_geom_20>',
'http://www.opengis.net/ont/geosparql#asWKT',
'POLYGON ((4.3406641999184785 51.984909686708896, 4.340109526495179 51.984827456357955, 4.340171204245872
        51.98466033527007, 4.3392292760885915 51.98450211536589, 4.3392259252587015 51.98448068083253, 4.337343568791171
        51.98418325850614, 4.337280661505889 51.984346316047194, 4.336564096568218 51.9842347134636, 4.33664376505569
        51.983992546231285, 4.335863950948884 51.983877947479094, 4.3359304423609535 51.983845390789455, 4.336147293977145
        51.98375783539835, 4.33657365239432 51.983575484739575, 4.336585680975262 51.98357718318736, 4.336716158513964
        51.983518257892264, 4.336755261777292 51.98350701143957, 4.336794886970769 51.98350083035551, 4.336834275883128
        51.98350474162454, 4.336865812016558 51.98351513564023, 4.336897209103631 51.98353147003422, 4.33693712718748
        51.983553822119234, 4.336996217430914 51.98357992290248, 4.3370583156110945 51.9836012864163, 4.3372501918757935
        51.98363153861879, 4.337487166136571 51.98366872780256, 4.337504305422265 51.98367661113901, 4.337514553489738
        51.98369157013051, 4.33752106987441 51.98370113847262, 4.337688059314531 51.98372552290044, 4.337711968783942
        51.98372901712378, 4.337735600455901 51.983732562797044, 4.339038783258909 51.983936317253765, 4.341164025412135
        51.9842684037326, 4.3412114591582975 51.984317562955525, 4.34124162062549 51.98432079742552, 4.3414669387350715
        51.98434495787615, 4.342082251815703 51.98444218353746, 4.343544835478862 51.98467059328918, 4.3436533482500455
        51.984728931561975, 4.343794483961674 51.984793617528766, 4.343820353287553 51.98482931689596, 4.343824186753043
        51.98499061960425, 4.343809411566345 51.9851836743996, 4.34365312049689 51.98532865517494, 4.34354481937599
        51.98534791043117, 4.343427637597273 51.98533497923838, 4.343313183968495 51.98528758702008, 4.343267773299738
        51.98525269842908, 4.3431682922251165 51.985141224906464, 4.342838345619537 51.98508478488775, 4.3426083052613365
        51.985038744001365, 4.3413262888059405 51.98483097019865, 4.341334852128802 51.98480862646888, 4.340745602613107
        51.98472252488639, 4.3406641999184785 51.984909686708896))'))
```

B | APPENDIX B

In this Appendix, the files related to the mapping processes are illustrated. Initially the mapping on Turtle text is presented. An ”OBJECT ID” is mapped to a label with ”rdfs:label” and a geometry with gsp:hasGeometry. The geometry has a geometry serialization as WKT. Then the Turtle text file is converted into N-triples and stored as .nt file. Finally, the .nt file is converted into INSERT statements with Python3 code and inserted into Oracle Spatial and Graph.

Turtle/text mapping.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix gsp: <http://www.opengis.net/ont/geosparql#>.
@prefix exa: <http://www.example.org/geometries#>.
@PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>.
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .

exa:TriplesMap_delft_parcel
    rr:logicalTable [ rr:tableName "DELFT_PARCELS" ];
    rr:subjectMap [
        rr:template "http://www.example.org/delft_parcels/{OBJECT_ID}";
        rr:class exa:parcel;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template "http://www.example.org/geometries/parcel_{OBJECT_ID}" ; rr:language "en"];
    ];
    rr:predicateObjectMap [
        rr:predicate gsp:hasGeometry;
        rr:objectMap [ rr:template "http://www.example.org/geometries/parcel_{OBJECT_ID}_geom"];
    ].

exa:TriplesMap_delft_parcel_geom
    rr:logicalTable [ rr:tableName "DELFT_PARCELS" ];
    rr:subjectMap [
        rr:template
        "http://www.example.org/geometries/parcel_{OBJECT_ID}_geom";
    ];
    rr:predicateObjectMap [
    rr:predicate exa:asWKT;
    rr:objectMap [rr:column "PARCEL_GEOM" ; rr:datatype gsp:wktLiteral];

].

exa:TriplesMap_gebouw_vlak
    rr:logicalTable [ rr:tableName "GEBOUW_VLAK" ];
    rr:subjectMap [
        rr:template "http://www.example.org/gebouw_vlak/{IDENT}";
        rr:class exa:building;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template
        "http://www.example.org/geometries/building{IDENT}" ; rr:language "en"];
    ];
    rr:predicateObjectMap [
        rr:predicate gsp:hasGeometry;
        rr:objectMap [ rr:template
        "http://www.example.org/geometries/building_{IDENT}_geom" ];
    ].

exa:TriplesMap_delft_building_geom
    rr:logicalTable [ rr:tableName "GEBOUW_VLAK"];
    rr:subjectMap [
        rr:template "http://www.example.org/geometries/building_{IDENT}_geom";
    ];
    rr:predicateObjectMap [
    rr:predicate exa:asWKT;
    rr:objectMap [rr:column "GEOM"; rr:datatype gsp:wktLiteral];

].
```

Turtle/text mapping converted into N-triples.

```
_:genid1 <http://www.w3.org/ns/r2rml#tableName> "DELFT_PARCELS" .
<http://www.example.org/geometries#TriplesMap_parcels><http://www.w3.org/ns/r2rml#logicalTable> _:genid1 .
<http://www.example.org/geometries#TriplesMap_parcels><http://www.w3.org/ns/r2rml#subjectMap> _:genid2 .
<http://www.example.org/geometries#TriplesMap_parcels><http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid3 .
<http://www.example.org/geometries#TriplesMap_parcels><http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid5 .
_:genid2 <http://www.w3.org/ns/r2rml#template> "http://www.example.org/parcel#{OBJECT_ID}" .
_:genid2 <http://www.w3.org/ns/r2rml#class>
<http://www.example.org/geometries#parcel> .
_:genid3 <http://www.w3.org/ns/r2rml#predicate>
```

```
<http://www.example.org/geometries#id> .
_:genid3 <http://www.w3.org/ns/r2rml#objectMap> _:genid4 .
_:genid4 <http://www.w3.org/ns/r2rml#column> "OBJECT_ID" .
_:genid4 <http://www.w3.org/ns/r2rml#datatype>
<http://www.w3.org/2001/XMLSchema#integer> .
_:genid5 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#geom> .
_:genid5 <http://www.w3.org/ns/r2rml#objectMap> _:genid6 .
_:genid6 <http://www.w3.org/ns/r2rml#column> "PARCEL_GEOM" .
_:genid7 <http://www.w3.org/ns/r2rml#tableName> "GEBOUW_VLAK_S" .
<http://www.example.org/geometries#TriplesMap_gebouw>
<http://www.w3.org/ns/r2rml#logicalTable> _:genid7 .
<http://www.example.org/geometries#TriplesMap_gebouw>
<http://www.w3.org/ns/r2rml#subjectMap> _:genid8 .
<http://www.example.org/geometries#TriplesMap_gebouw>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid9 .
<http://www.example.org/geometries#TriplesMap_gebouw>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid11 .
_:genid8 <http://www.w3.org/ns/r2rml#template>
"http://www.example.org/gebouw_vlak#{IDENT}" .
_:genid8 <http://www.w3.org/ns/r2rml#class>
<http://www.example.org/geometries#building> .
_:genid9 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#typegebouw> .
_:genid9 <http://www.w3.org/ns/r2rml#objectMap> _:genid10 .
_:genid10 <http://www.w3.org/ns/r2rml#column> "TYPEGEBOUW" .
_:genid11 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#geom> .
_:genid11 <http://www.w3.org/ns/r2rml#objectMap> _:genid12 .
_:genid12 <http://www.w3.org/ns/r2rml#column> "GEOM" .
_:genid13 <http://www.w3.org/ns/r2rml#tableName> "WEGDEEL_VLAK_S" .
<http://www.example.org/geometries#TriplesMap_wegdeel>
<http://www.w3.org/ns/r2rml#logicalTable> _:genid13 .
<http://www.example.org/geometries#TriplesMap_wegdeel>
<http://www.w3.org/ns/r2rml#subjectMap> _:genid14 .
<http://www.example.org/geometries#TriplesMap_wegdeel>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid15 .
<http://www.example.org/geometries#TriplesMap_wegdeel>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid17 .
_:genid14 <http://www.w3.org/ns/r2rml#template>
"http://www.example.org/wegdeel_vlak#{IDENT}" .
_:genid14 <http://www.w3.org/ns/r2rml#class>
<http://www.example.org/geometries#road> .
_:genid15 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#typeweg> .
_:genid15 <http://www.w3.org/ns/r2rml#objectMap> _:genid16 .
_:genid16 <http://www.w3.org/ns/r2rml#column> "TYPEWEG" .
_:genid17 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#geom> .
_:genid17 <http://www.w3.org/ns/r2rml#objectMap> _:genid18 .
_:genid18 <http://www.w3.org/ns/r2rml#column> "GEOM" .
_:genid19 <http://www.w3.org/ns/r2rml#tableName> "SPATIAL_PLANS" .
<http://www.example.org/geometries#TriplesMap_plans>
<http://www.w3.org/ns/r2rml#logicalTable> _:genid19 .
<http://www.example.org/geometries#TriplesMap_plans>
<http://www.w3.org/ns/r2rml#subjectMap> _:genid20 .
<http://www.example.org/geometries#TriplesMap_plans>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid21 .
<http://www.example.org/geometries#TriplesMap_plans>
<http://www.w3.org/ns/r2rml#predicateObjectMap> _:genid23 .
_:genid20 <http://www.w3.org/ns/r2rml#template>
"http://brt.basisregistraties.overheid.nl/top10nl/id/functioneel-gebied#{ID}" .
_:genid20 <http://www.w3.org/ns/r2rml#class>
<http://www.example.org/geometries#plan> .
_:genid21 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#description> .
_:genid21 <http://www.w3.org/ns/r2rml#objectMap> _:genid22 .
_:genid22 <http://www.w3.org/ns/r2rml#column> "DESCRIPTION" .
_:genid23 <http://www.w3.org/ns/r2rml#predicate>
<http://www.example.org/geometries#geom> .
_:genid23 <http://www.w3.org/ns/r2rml#objectMap> _:genid24 .
_:genid24 <http://www.w3.org/ns/r2rml#column> "GEOM" .
```

Python3 code to convert an .nt file to INSERT statements for Oracle Spatial and Graph

```
file = open('nt2triples.txt','w')
with open('r2rml_2.txt') as fin:
    for line in fin:
        triple = str.split(line)
        q="INSERT INTO st_table VALUES (\n '{}',\n '{}',\n '{}');\n".format(triple[0],triple[1],triple[2])
        file.write(q)


#the index serves to store the triplets in the database. triples are stored
#in a index-triplet combination.

file.close()
```

N-triples converted into INSERT statements.

```
INSERT INTO st_table VALUES (
'_:genid1',
'<http://www.w3.org/ns/r2rml#tableName>',
'"DELFT_PARCELS"');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_parcels>',
'<http://www.w3.org/ns/r2rml#logicalTable>',
'_:genid1');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_parcels>',
'<http://www.w3.org/ns/r2rml#subjectMap>',
'_:genid2');
```

```
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_parcels>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid3');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_parcels>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid5');
INSERT INTO st_table VALUES (
'_:genid2',
'<http://www.w3.org/ns/r2rml#template>',
'"http://www.example.org/parcel#{OBJECT_ID}"');
INSERT INTO st_table VALUES (
'_:genid2',
'<http://www.w3.org/ns/r2rml#class>',
'<http://www.example.org/geometries#parcel>');
INSERT INTO st_table VALUES (
'_:genid3',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#id>');
INSERT INTO st_table VALUES (
'_:genid3',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid4');
INSERT INTO st_table VALUES (
'_:genid4',
'<http://www.w3.org/ns/r2rml#column>',
'"OBJECT_ID"');
INSERT INTO st_table VALUES (
'_:genid4',
'<http://www.w3.org/ns/r2rml#datatype>',
'<http://www.w3.org/2001/XMLSchema#integer>');
INSERT INTO st_table VALUES (
'_:genid5',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#geom>');
INSERT INTO st_table VALUES (
'_:genid5',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid6');
INSERT INTO st_table VALUES (
'_:genid6',
'<http://www.w3.org/ns/r2rml#column>',
'"PARCEL_GEOM"');
INSERT INTO st_table VALUES (
'_:genid7',
'<http://www.w3.org/ns/r2rml#tableName>',
'"GEBOUW_VLAK_S"');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_gebouw>',
'<http://www.w3.org/ns/r2rml#logicalTable>',
'_:genid7');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_gebouw>',
'<http://www.w3.org/ns/r2rml#subjectMap>',
'_:genid8');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_gebouw>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid9');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_gebouw>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid11');
INSERT INTO st_table VALUES (
'_:genid8',
'<http://www.w3.org/ns/r2rml#template>',
'"http://www.example.org/gebouw_vlak#{IDENT}"');
INSERT INTO st_table VALUES (
'_:genid8',
'<http://www.w3.org/ns/r2rml#class>',
'<http://www.example.org/geometries#building>');
INSERT INTO st_table VALUES (
'_:genid9',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#typegebouw>');
INSERT INTO st_table VALUES (
'_:genid9',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid10');
INSERT INTO st_table VALUES (
'_:genid10',
'<http://www.w3.org/ns/r2rml#column>',
'"TYPEGEBOUW"');
INSERT INTO st_table VALUES (
'_:genid11',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#geom>');
INSERT INTO st_table VALUES (
'_:genid11',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid12');
INSERT INTO st_table VALUES (
'_:genid12',
'<http://www.w3.org/ns/r2rml#column>',
'"GEOM"');
INSERT INTO st_table VALUES (
'_:genid13',
'<http://www.w3.org/ns/r2rml#tableName>',
'"WEGDEEL_VLAK_S"');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_wegdeel>',
'<http://www.w3.org/ns/r2rml#logicalTable>',
'_:genid13');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_wegdeel>',
'<http://www.w3.org/ns/r2rml#subjectMap>',
'_:genid14');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_wegdeel>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
```

```
'_:genid15');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_wegdeel>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid17');
INSERT INTO st_table VALUES (
'_:genid14',
'<http://www.w3.org/ns/r2rml#template>',
'"http://www.example.org/wegdeel_vlak#{IDENT}"');
INSERT INTO st_table VALUES (
'_:genid14',
'<http://www.w3.org/ns/r2rml#class>',
'<http://www.example.org/geometries#road>');
INSERT INTO st_table VALUES (
'_:genid15',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#typeweg>');
INSERT INTO st_table VALUES (
'_:genid15',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid16');
INSERT INTO st_table VALUES (
'_:genid16',
'<http://www.w3.org/ns/r2rml#column>',
'"TYPEWEG"');
INSERT INTO st_table VALUES (
'_:genid17',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#geom>');
INSERT INTO st_table VALUES (
'_:genid17',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid18');
INSERT INTO st_table VALUES (
'_:genid18',
'<http://www.w3.org/ns/r2rml#column>',
'"GEOM"');
INSERT INTO st_table VALUES (
'_:genid19',
'<http://www.w3.org/ns/r2rml#tableName>',
'"SPATIAL_PLANS"');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_plans>',
'<http://www.w3.org/ns/r2rml#logicalTable>',
'_:genid19');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_plans>',
'<http://www.w3.org/ns/r2rml#subjectMap>',
'_:genid20');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_plans>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid21');
INSERT INTO st_table VALUES (
'<http://www.example.org/geometries#TriplesMap_plans>',
'<http://www.w3.org/ns/r2rml#predicateObjectMap>',
'_:genid23');
INSERT INTO st_table VALUES (
'_:genid20',
'<http://www.w3.org/ns/r2rml#template>',
'"http://brt.basisregistraties.overheid.nl/top10nl/id/
functioneel-gebied#{ID}"');
INSERT INTO st_table VALUES (
'_:genid20',
'<http://www.w3.org/ns/r2rml#class>',
'<http://www.example.org/geometries#plan>');
INSERT INTO st_table VALUES (
'_:genid21',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#description>');
INSERT INTO st_table VALUES (
'_:genid21',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid22');
INSERT INTO st_table VALUES (
'_:genid22',
'<http://www.w3.org/ns/r2rml#column>',
'"DESCRIPTION"');
INSERT INTO st_table VALUES (
'_:genid23',
'<http://www.w3.org/ns/r2rml#predicate>',
'<http://www.example.org/geometries#geom>');
INSERT INTO st_table VALUES (
'_:genid23',
'<http://www.w3.org/ns/r2rml#objectMap>',
'_:genid24');
INSERT INTO st_table VALUES (
'_:genid24',
'<http://www.w3.org/ns/r2rml#column>',
'"GEOM"');
```

C | APPENDIX C

In this Appendix the ontologies designed for the dummy data are presented.

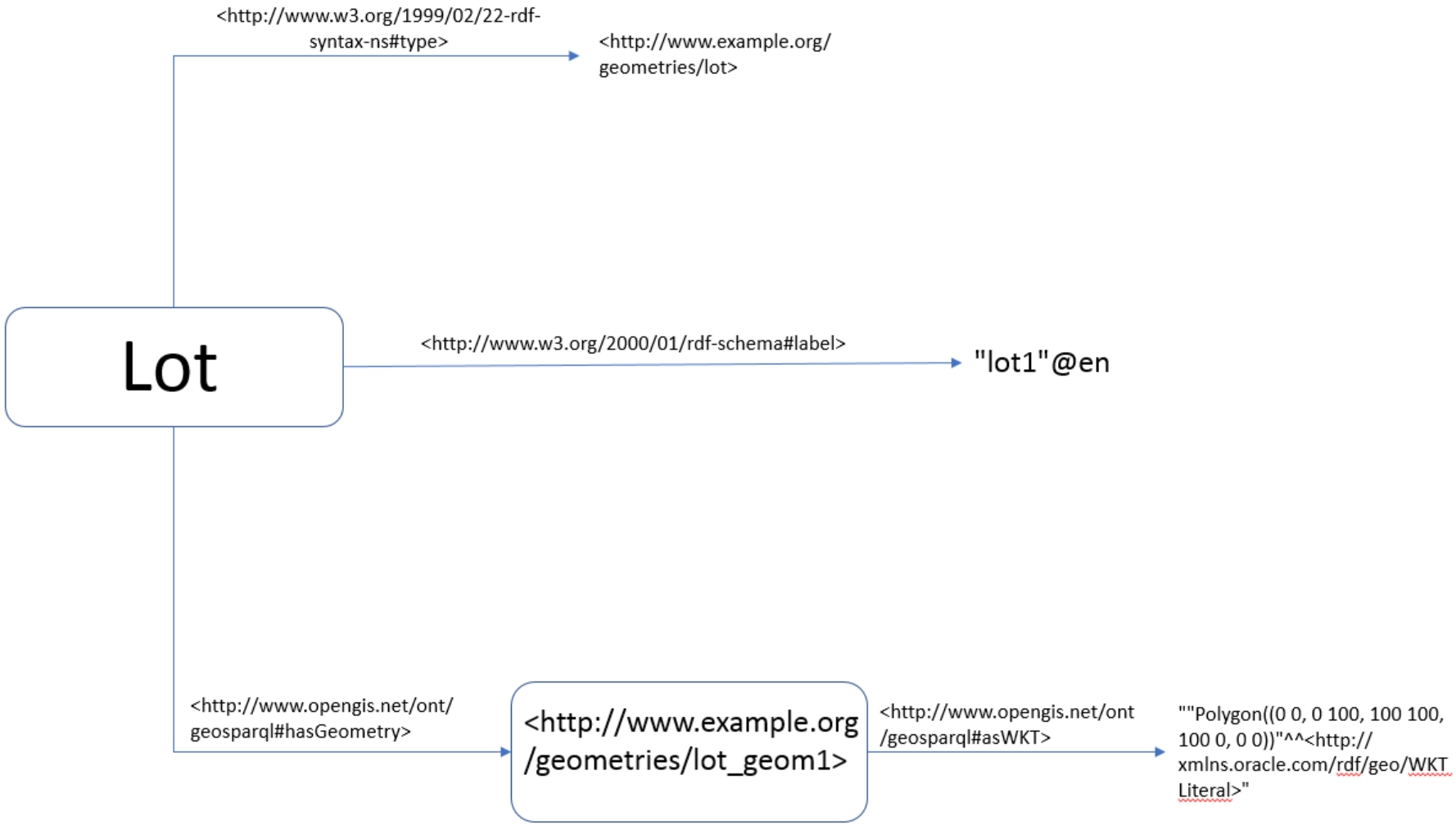


Figure C.1: Designed lot ontology.

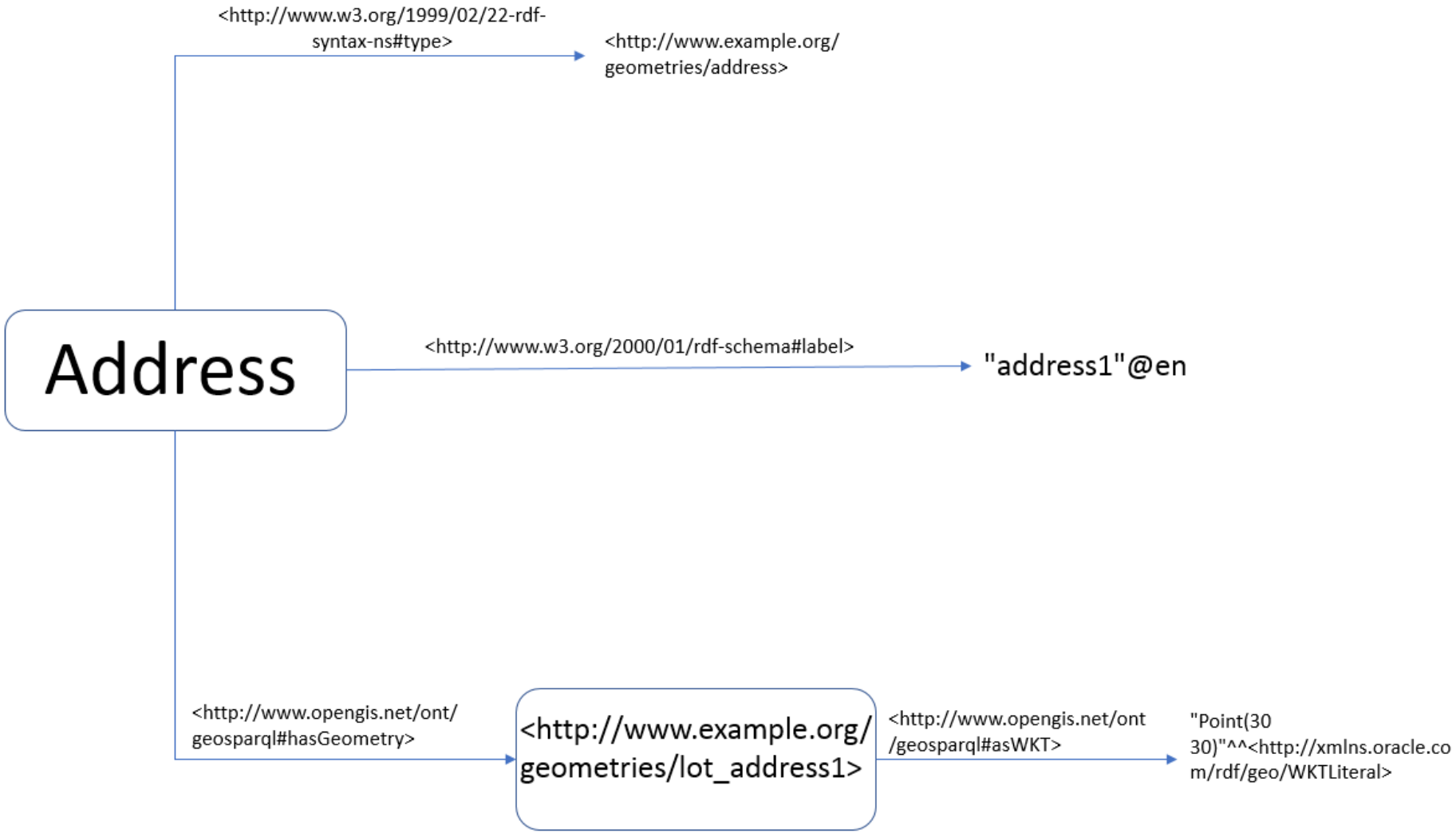


Figure C.2: Designed address ontology

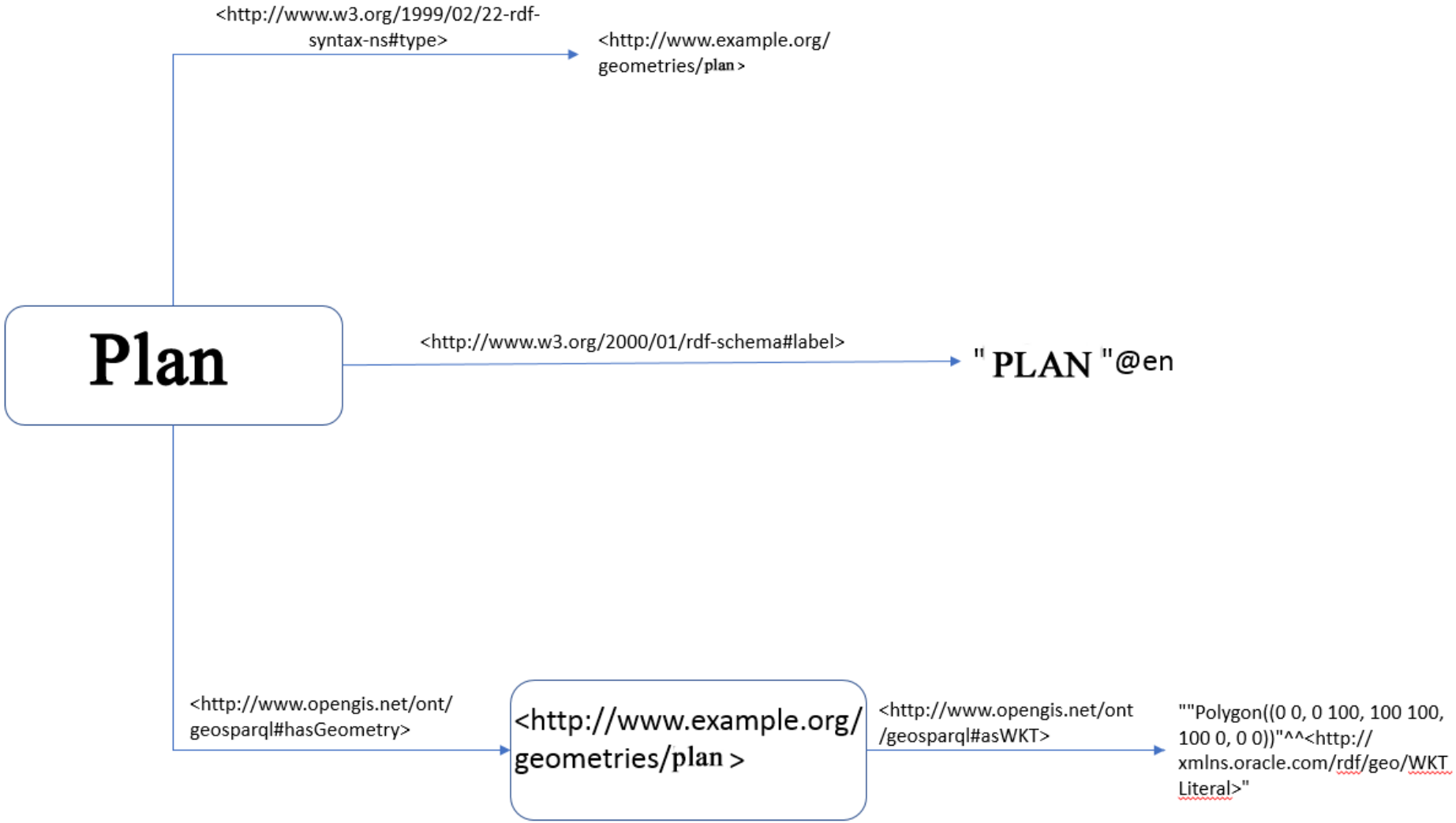


Figure C.3: Designed spatial plan ontology

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.