Ontology Based Data Access in Statoil

Evgeny Kharlamov[a], Dag Hovland[c], Martin G. Skjæveland[c], Dimitris Bilidas[b], Ernesto Jiménez-Ruiz[c], Guohui Xiao[d], Ahmet Soylu[f], Davide Lanti[d], Martin Rezk[d], Dmitriy Zheleznyakov[a], Martin Giese[c], Hallstein Lie[e], Yannis Ioannidis[b], Yannis Kotidis[g], Manolis Koubarakis[b], Arild Waaler[c]

[a] *University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, OX1 3QD, Oxford, UK.*
[b] *National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, 15784, Athens, Greece.*
[c] *Department of Informatics, University of Oslo, Blindern, 0316, Oslo, Norway.*
[d] *KRDB Center, Free-University of Bozen-Bolzano, P.zza Domenicani 3, 39100, Bolzano, Italy.*
[e] *Statoil ASA, Stavanger, Norway.*
[f] *NTNU – Norwegian University of Science and Technology, Teknologiveien 22, 2815, Gjøvik, Norway.*
[g] *Athens University of Economics and Business, 76 Patission Street, 10434, Athens, Greece.*

Abstract

Ontology Based Data Access (OBDA) is a prominent approach to query databases which uses an ontology to expose data in a conceptually clear manner by abstracting away from the technical schema-level details of the underlying data. The ontology is 'connected' to the data via mappings that allow to automatically translate queries posed over the ontology into data-level queries that can be executed by the underlying database management system. Despite a lot of attention from the research community, there are still few instances of real world industrial use of OBDA systems. In this work we present data access challenges in the data-intensive petroleum company Statoil and our experience in addressing these challenges with OBDA technology. In particular, we have developed a deployment module to create ontologies and mappings from relational databases in a semi-automatic fashion; a query processing module to perform and optimise the process of translating ontological queries into data queries and their execution over either a single DB of federated DBs; and a query formulation module to support query construction for engineers with a limited IT background. Our modules have been integrated in one OBDA system, deployed at Statoil, integrated with Statoil's infrastructure, and evaluated with Statoil's engineers and data.

Keywords: Ontology Based Data Access, Statoil, Optique Platform, System Deployment, Evaluation, Bootstrapping, Optimisations.

1. Introduction

The competitiveness of modern enterprises heavily depends on their ability to make the *right* business decisions by relying on efficient and timely analyses of the *right* business critical data. For example, one of the factors determining the competitiveness of Statoil[1,2] is the ability of its exploration geologists to find in a timely manner new exploitable accumulations of oil or gas in given areas by analysing data about these areas. Gathering such data is not a trivial task in Statoil and in general in data intensive enterprises due to the growing size and complexity of corporate information sources. Such data sources are often scattered across heterogeneous and autonomously evolving systems or has been adapted over the years to the needs of the applications they serve. This often leads to the situation where naming conventions for schema elements, constraints, and the structure of database schemata are very complex and documentation may be limited or nonexistent making it difficult to extract data. As a result, the data gathering task is often the most time-consuming part of the decision making process.

Indeed, Statoil geologists often require data that is stored in multiple complex and large data sources that include EPDS, Recall, CoreDB, GeoChemDB, OpenWorks, Compass, and NPD FactPages (see Section 2 for details about these DBs). These DBs are mostly Statoil's corporate data stores for exploration and production data and Statoil's interpretations of this data. Some of these

Email addresses: evgeny.kharlamov@cs.ox.ac.uk (Evgeny Kharlamov), hovland@ifi.uio.no (Dag Hovland), martige@ifi.uio.no (Martin G. Skjæveland), d.bilidas@di.uoa.gr (Dimitris Bilidas), ernestoj@ifi.uio.no (Ernesto Jiménez-Ruiz), xiao@inf.unibz.it (Guohui Xiao), ahmet.soylu@ntnu.no (
Ahmet Soylu), davide.lanti@unibz.it (Davide Lanti), mrezk@inf.unibz.it (Martin Rezk), dmitriy.zheleznyakov@cs.ox.ac.uk (Dmitriy Zheleznyakov), martingi@ifi.uio.no (Martin Giese), hali@statoil.com (Hallstein Lie), yannis@di.uoa.gr (
Yannis Ioannidis), kotidis@aueb.gr (Yannis Kotidis), koubarak@di.uoa.gr (Manolis Koubarakis), arild@ifi.uio.no (Arild Waaler)

[1] Statoil ASA, is a Norwegian multinational oil and gas company headquartered in Stavanger, Norway. It is a fully integrated petroleum company with operations in thirty-six countries.

[2] https://www.statoil.com/

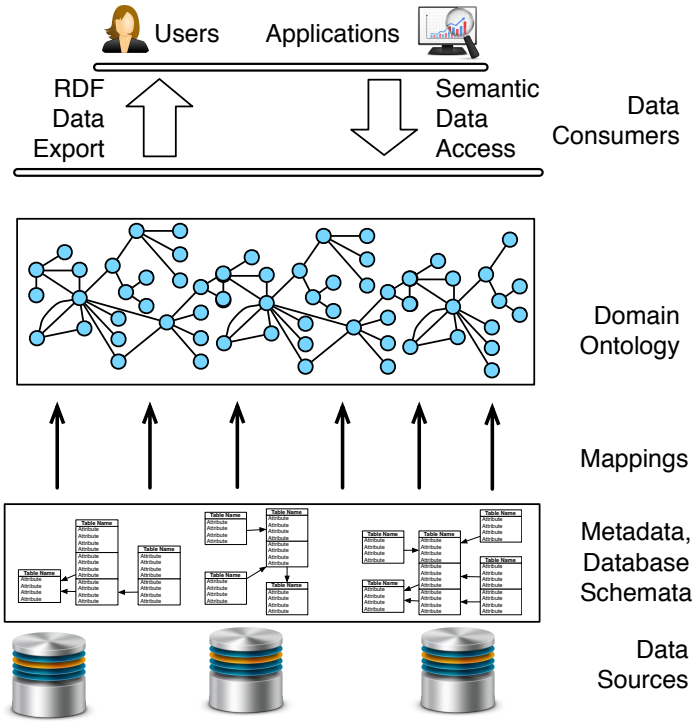


Figure 1: General diagram of OBDA.

DBs has been created long time ago, e.g., EPDS was created about 15 years ago, and can hardly be accessed, i.e., queried, by geologists without a help from IT personnel due to the complexity of their schemata, e.g., EPDS currently has about 3,000 tables with about 37,000 columns, and a common information need of a Statoil geologist corresponds to an SQL query with hundreds to thousands of terms and 50–200 joins. Construction of such queries is not possible for Statoil geologists and thus they have to pass their information needs to IT specialists who then turn the needs into SQL queries. This drastically affects the efficiency of finding the right data that should back decision making (see Section 2 for further details).

Ontology Based Data Access (OBDA) [1] is a prominent approach to data access in which an *ontology* is used to mediate between data consumers and data sources (see a general diagram of OBDA in Figure 1). The ontology provides 'a single point of semantic data access' for data consumers, and allows either to export data in a semantic format or to pose queries over the integrated data sources in terms of a user-oriented conceptual model that abstracts away complex implementation-level details typically encountered in database schemata. Domain experts are thus able to express information needs in their own terms without any prior knowledge about the way the data is organised at the source, and to receive answers in the same intelligible form. The ontology is connected to the data via a set of *mappings*: declarative specifications that relate ontological terms with queries over the underlying data. OBDA systems automatically translate ontological queries, i.e., SPARQL, into database queries, i.e., SQL, and delegate execution of SQL queries to the database systems hosting the data. OBDA is a natural fit to address the Statoil data access challenges described above: if complex databases are presented to users via an ontology, then they can formulate queries in terms of classes and properties in an object-centric fashion, e.g., asking for all *wellbores penetrating* a *rock layer* of a specific *geological age*. Moreover, OBDA is a so-called *virtual* approach, providing an access layer on top of databases while leaving the data in its original stores. Thus, OBDA has the potential to improve data access with a minimal change to existing data management infrastructure.

OBDA has recently attracted a lot of attention and a number of systems have been developed, e.g., [2, 3, 4, 5, 6]. However, to the best of our knowledge, the following three problems have attracted only limited attention and in isolation:

(i) How to *create* ontologies and mappings for a deployment of an OBDA system?

(ii) How to ensure that OBDA query processing is *efficient* in practice?

(iii) How to ensure that the target users are actually able to efficiently *express* their information needs against an OBDA system?

At the same time, these problems have high practical importance for OBDA systems in general and in particular for effective and efficient installation and use of an OBDA system in Statoil. Indeed, deployment of an OBDA system

comes with a high modelling cost due to the complexity of the domain and of the database schemata. Moreover, unoptimised OBDA query processing may become impractical when the ontology and/or database are large [7]. Finally, expressing information needs over an OBDA system as a query written in a query language, e.g., SPARQL, is error prone and requires substantial training which significantly limits the usability of an ODBA system in a company.

In this work we addressed these limitations and developed *(i)* novel semi-automatic techniques to bootstrap new ontologies and mappings from relational databases and to integrate existing ones, and *(ii)* novel optimisation techniques to improve query processing by producing compact and efficient SQL queries, and then by carefully planning their query execution strategy over one or several federated DBs. *(iii)* a novel OBDA oriented visual query formulation interface.

We then implemented these techniques and developed an OBDA deployment, query optimisation, and query formulation systems which were integrated in an end-to-end OBDA system Optique [8, 9].

We applied Optique in Statoil in order to improve the data gathering routine of Statoil geologists. To this end we deployed our OBDA system over seven prominent, complex, and large data sources: EPDS, Recall, CoreDB, GeoChemDB, OpenWorks, Compass, and NPD FactPages (see Section 2 for details about these DBs). These DBs are mostly Statoil's corporate data stores for exploration and production data and Statoil's interpretations of this data, and they are heavily used by Statoil geologists.

Our OBDA solution Optique has been successfully deployed and evaluated at Statoil over the aforementioned seven DBs. We used three metrics to show this success:

- *quality* of the system's deployment,
- *efficiency* of the system's query processing, and
- *effectiveness* and *efficiency* of the system's query formulation support.

In order to objectify the measure of success and facilitate all three metrics, we gathered a catalogue of queries collected from Statoil geologists. These queries cover a wide range of typical information needs, and they are hard to formulate over Statoil databases.

In order to show the quality of our semi-automatic deployment, we showed that the system enables formulation of the queries in the catalog, i.e., it provides enough ontological terms to do so, it covers a wide range of commonly used ontological terms from the geological domain, and the ontology and mappings in combination reflect expectations of geologists, that is, the answers to queries from the catalog correspond to expectations of geologists.

In order to show the efficiency of the Optique's query processing, we conducted a number of experiments. The first set of experiments aimed at showing how our query optimisations lead to significant reduction of query processing time over one DB. To this end we focused on the most complex Statoil DB available for our experiments, EPDS, and showed how our optimisations affect query execution time for queries from the Statoil catalog. Since EPDS is not a public DB, we conducted the same experiments over NPD FactPages, a public Statoil DB. Since NPD FactPages is small, about 105 MB, we developed and applied a procedure to scale this data in such a way that the resulting DB respects the important structural characteristics of the original DB. Finally, we conducted experiments over federated databases in order to show practical benefits of our distributed query planning component. Most of our experiments showed that Optique can handle queries from the Statoil's catalog reasonably well, that is, in time comparable to the time reported by existing Statoil's systems.

In order to show the effectiveness of Optique's query formulation we showed that the semantic queries that geologists have to formulate are much simpler than the data queries over the Statoil databases behind our OBDA deployment. We also conducted a series of user studies to show that the queries from the Statoil's query catalog can be formulated by Statoil's target personnel in a reasonably short time.

Finally, we integrated Optique in Statoil's infrastructure in order to facilitate the update of the system by Statoil's business units. In particular, we integrated Optique with ArcGIS[3] in order to show query results computed by Optique on geological maps.

The paper is organised as follows: in Section 2 we present Statoil's data access challenges. In Section 3 we give an overview of OBDA and then in Section 4 and discuss what are the advantages of OBDA for Statoil and limitations of existing ODBA solutions. in Section 5 we give technical background of our deployment, query processing, and query formulation solutions. In Sections 6–8 we present evaluation of our OBDA deployment at Statoil. Note that in these three sections we introduce several system requirements that we consolidated by interviewing geologists and IT-specialists of Statoil. These requirements were important for us to guide the work and for Statoil colleagues to measure the success of this work. Moreover, gathering and consolidating requirements was compulsory for us to proceed with the system deployment in Statoil. In Section 9 we give details on how we integrated Optique platform in Statoil's infrastructure. In Sections 10 and 11 we summarise the lessons we learned and conclude. Finally, in Appendix A we give an end-to-end real example from our statoil deployment: we present information need of a Statoil geologist, show how it can be formulated over an ontology as a semantic query both in a formal query language and in our visual query system, present mappings relevant to the semantic terms in the semantic query, the

[3] `https://www.arcgis.com/`

SQL query generated from the semantic query with the help of the ontology and mappings, and the distributed query execution plan generated by our query planning system.

Delta from Previous Publications. This submission mainly extends our ISWC'15 paper [10] in several important directions. Some material form Sections 5-7 also appeared in [11, 12, 13, 14, 15]. In short, we now presented deployment in Statoil over a federated scenario of six databases (EPDS, GeoChemDB, Recall, CoreDB, OW, Compass), while previously the deployment was over a single database EPDS (that essentially included NPD FP). Thus, deployment, query optimisation techniques, and evaluation for the federated scenario are new. Also, we present here a new evaluation of the query formulation system with Statoil engineers, a very detailed use-case description, and we first time explain how Optique is integrated in the Statoil's infrastructure. We now give details of what is new in this paper comparing to [10].

- Data Access in Statoil (Section 2): The use-case description of Section 2 significantly extends what we previously reported [10]. Indeed, now we introduce six instead of one database, give Table 1 and give a much more detailed description of the data access routine in Statoil.
- Ontology Based Data Access (Section 3): In order to be self-contained, we added to the current submission an extended section that introduces OBDA and for this purpose we developed a detailed example.
- Technical Background of the Optique Platform (Section 5): Everything in the sub-section on Federated Query Execution (Section 5.3) is new. Sections 5.2 and 5.3 give much more details than the corresponding sections in [10].
- Deployment (Section 6): most of Section 6 is new. The system is now running over seven DBs, while in the previous publications it was only over EPDS. Therefore, most of Section 6.2 is new including Tables 2 and 3 with detailed statistics of bootstrapped ontologies and evaluation of their OWL 2 profiles. Section 6.3 that contains comparison with other systems including ontop, MIRROR, and D2RQ and shows that we are better in most of the cases is new. Finally, we conducted a novel assessment: we checked whether bootstrapped ontology and mappings lead to expected query results.
- Query Answering over OBDA Deployment in Statoil (Section 7): We have significantly extended the evaluation scope. We evaluated the efficiency of Optique for the Statoil's query catalog over federated DBs and this is all new. Moreover, we evaluated Optique over EPDS and scaled NPD FactPages the effect of optimisations based on OBDA constraints techniques and this was only partially presented in our previous publications: results for scaled NPD FactPages are all new, while for EPDS the new aspect is the insight on the evaluation in Table 7.
- Visual Query Formulation at Statoil (Section 8): The evaluation of our query formulation system with Statoil engineers is new.
- Integration in Statoil's Infrastructure (Section 9): The whole section is new. We have integrated our platform with the ArcGIS system and in general the implementation of our system has become much more mature. All this (including Figures 21 and 22) is new.

2. Data Access in Statoil

In this section we describe the data access practices of geologists in Statoil and present challenges they confront.

2.1. Exploration Geology as Data Management

The main task of exploration geologists in oil companies like Statoil, is to find exploitable deposits of oil or gas in given areas and to analyse existing deposits. This is typically done by investigating how parts of the earth crust are composed in the area of interest. In Figure 2 shows an area of investigation around two platforms located offshore; the earth's crust under the platforms has several rock layers. The earth's crust can be analysed indirectly with seismic investigations and more directly by collecting rock samples and so-called log curves from drilled wellbores. In the former case the analyses are conducted by recording the time of echoes of small explosive charges that are fired repeatedly over the area of interest, and using this information to estimate the geological composition of the ground below. In the latter case the analyses is done over rock samples that are taken both during and after drilling and through measurements collected from sensors installed along the wellbore. By combining information from wellbores, seismic investigations, and general geological knowledge, geologists can, for example, assess what types of rock are in the reservoir and intersected along the wellbore. The geologist does this typically in two steps:

(i) find relevant wellbore, seismic, and other data in Statoil databases,

(i) analyse these data with specialised analytical tools.

It has been observed both in Statoil and other companies [16] that step *(i)* is the most time consuming part of the process. The reason is that, on the one hand, the size and complexity of the data sources and access methods make it impossible for end-users to collect the necessary data efficiently. On the other hand, involvement of IT staff that can implement data gathering queries makes the process lengthy. We now describe the Statoil databases that geologists have to access, existing data accessing infrastructure in Statoil, and conclude with desiderata that motivate the solution we applied in Statoil.

Table 1: Database metrics: Showing number of schema constructs for each database schema. Zero-values are removed from the table for readability.

	EPDS	GeoChemDB	Recall	CoreDB	OW	Compass
Overview						
Tables	1595	90	22	15	78	895
Mat. views	27	4				
Views	1703	41	12		1026	1004
Columns	8378	3396	430	63	16668	30638
Tables by no. rows						
0 rows	1130	3	2		15	512
1 row	1152	9	2		4	34
$1 <$ rows ≤ 10	135	9	1	4	15	117
$10 <$ rows ≤ 100	83	20	3	2	17	80
$100 <$ rows $\leq 1\,000$	58	30	3	4	12	87
$1\,000 <$ rows $\leq 10\,000$	63	10	5	1	11	42
$10\,000 <$ rows $\leq 100\,000$	57	4	2	1	2	19
$100\,000 <$ rows $\leq 1000\,000$	35	3	4	2		4
$1\,000\,000 <$ rows	12	3	2	1		
Tables by no. columns						
1 col	4				5	6
$1 <$ cols ≤ 10	586	68	7	11	47	527
$10 <$ cols ≤ 100	1032	23	13	4	26	353
$100 <$ cols $\leq 1\,000$		3	2			9
$1\,000 <$ cols						
Mat. views by no. columns						
1 col						
$1 <$ cols ≤ 10	23	1				
$10 <$ cols ≤ 100	4	3				
$100 <$ cols						
Views by no. columns						
1 col	3	2			3	2
$1 <$ cols ≤ 10	526	12			555	509
$10 <$ cols ≤ 100	1174	14	9		461	471
$100 <$ cols $\leq 1\,000$		13	3		7	22
$1\,000 <$ cols						

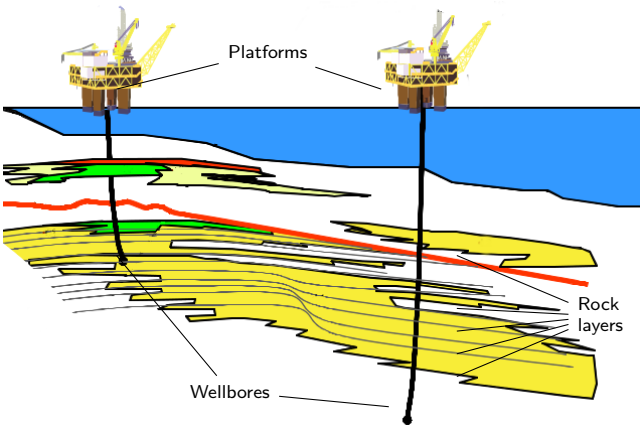


Figure 2: Platforms, wellbores, and rock layers; an illustration of the complex nature of petroleum reservoirs.

2.2. Geological Data at Statoil

Statoil has a number of databases of different formats and sizes that have different content and provided by different vendors and that geologists have access to. In our study we focussed on the following seven databases. Information about the structural size of the Statoil internal databases is found in Table 1.

(1) Exploration and Production Data Store (EPDS) is the Statoil's central repository of geological data of the type described above, i.e., for exploration, production data, and its interpretations. EPDS is stored in an Oracle database. It was created about 15 years ago and it currently has about 3,000 tables and views that have about 37,000 columns all together. As is often the case in large enterprises, naming conventions for schema elements, constraints, and the structure of EPDS's schema are complex and considerable parts of it have limited or no documentation. As a result, the major challenge with accessing EPDS is the schema complexity: writing queries over EPDS requires familiarity with all the variety of its underlying schema components. In terms of size, EPDS is about 700 GB.

(2) Core DB Information about samples taken from the wellbore, and measurements done on them

(3) Openworks Project databases, that is, work in progress. Mostly geological interpretations, but also some measurements. Information should be moved to other databases, like EPDS, when the information is stable.

(4) Recall Wellbore logs, that is, measurements made down along the wellbore, both during drilling, and on later occasions.

(5) GeoChem Measurements from the wellbore in the field of geochemstry. Mostly spectrometry, but also some other measurements.

(6) compass Geometric and geographic infomation about wellbores. E.g. 3d wellbore paths.

(7) Norwegian Petroleum Directorate FactPages (NPD FP) Unlike the others, this is not an internal database at Statoil, but an external data source governed by the NPD, which reports to the Norwegian Ministry of Petroleum and Energy. The directorate performs a wide range of tasks on regulating and monitoring oil and gas activities on the Norwegian Continental Shelf. NPD FP [17] is a dataset that is published online by the NPD and it contains data about oil and gas companies, i.e., their producing fields, ongoing exploratory drilling wellbores, and seismic surveys. NPD FP is frequently used by Statoil domain specialists and we verified with them that it contains a wide range of relevant terms. NPD contains mainly geological and legal information pertaining to wellbores and licenses.

We next describe the approaches currently used in Statoil to access the above and other databases.

2.3. Data Access in Statoil

Following common practices of large enterprises, geologists at Statoil analyse data in two steps: *(i)* they first access and gather relevant data from available databases, and then *(ii)* apply analytical reporting tools on top of the gathered data.

Example 1. *An example and typical Statoil task that would require the two above steps is the following:*

> *Show all the core samples overlapping with the geological unit X, where X is a given lithostratigraphic unit of interest, e.g., the Brent group or the Tarbert formation.*

It has been estimated that in the oil and gas industry [16], and this is confirmed by our Statoil-internal information, 30–70% of the time that geologists spend on analytical tasks is invested in the first step of finding the *right* data for analyses. The focus of this work is to improve Statoil's data access with Semantic Technologies and thus we now give details on why finding the right data is hard.

Access Points. The data access step is typically done via a variety of query interfaces and data extraction tools, such as geographical information system (GIS) tools and specialised data manipulation tools, that we shall collectively refer to as *access points*. The flexibility of the access points is limited and in general users can control them only by inserting values for certain query parameters. When information needs cannot be satisfied with any of the available access points, geologists, possibly with the help of IT staff, try to combine answers obtained from several access points. In some cases, geologists have to contact IT staff to provide a new access point.

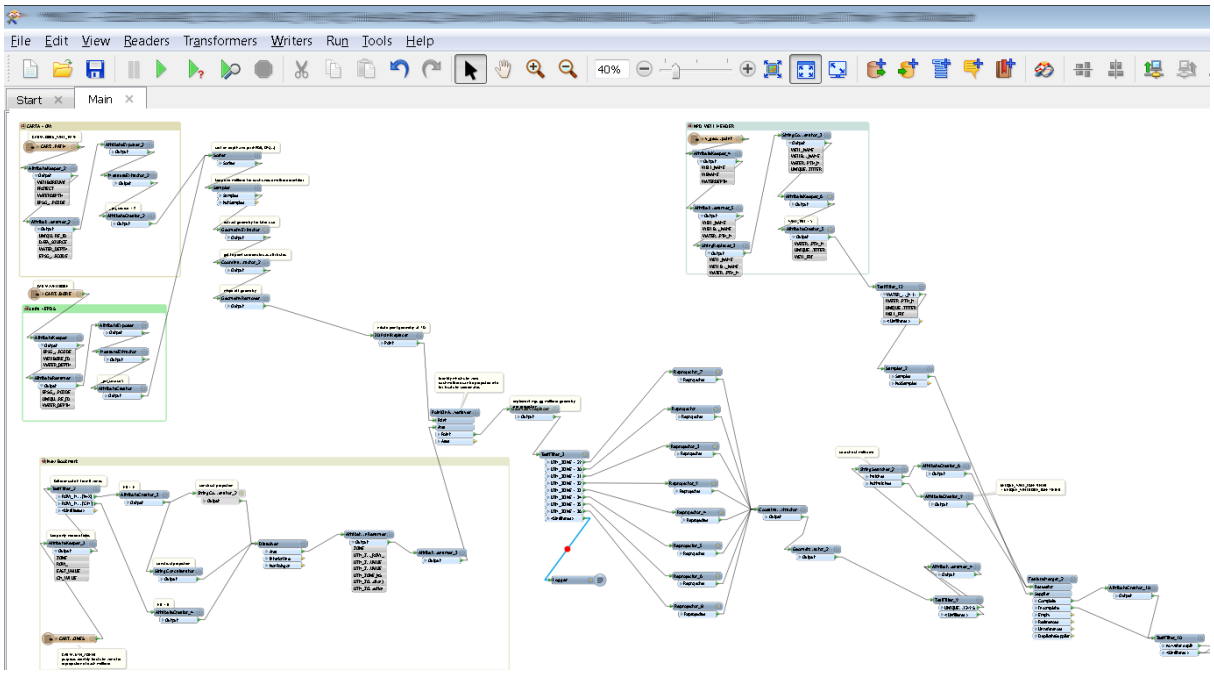


Figure 3: FME: a commercial tool for defining ETL processes. Each box in the diagram describes a data manipulation task. The contents of the boxes are blurred for privacy reasons.

Creating Access Points. Access points are typically based on materialised special purpose database views. To make a new view or to modify an existing one, IT staff typically use external tools, such as Feature Manipulation Engine (FME).[4] The process of making such view consists of the three ETL steps: *(i)* extracting, *(ii)* transforming, and *(iii)* loading data. For Step (i) Statoil IT staff locate relevant data in EPDS, other data sources or existing access points, and produce SQL code for its extraction. This is done using specialised data extraction tools, since directly accessing complex database schemata like EPDS' is prone to error, time consuming, and not always feasible due to its complexity and limited documentation. An SQL query for data extraction generated by the extraction tools are typically large and, e.g., over EPDS they may contain thousands of words and have 50–200 joins. During Step (ii), using specialised tools, IT staff define how to preprocess the data extracted at Step (i). This adds another layer of data processing, this time over the relevant data, to perform projections, filtering, joins, schema renamings and complex computations such as geographical coordinate conversions. In Step (iii), IT staff populate the access point's view with the data: both data extraction and manipulation code is executed which materialises the view. In sum, building an ETL process that establishes an access point for a complex information need consists of a myriad of data access and processing steps, many of which require deep knowledge of the data that is being processed and how it is represented. Figure 3 shows an excerpt of an FME process that establishes an access point for gathering the information for the information need about overlapping core samples as in Example 1.

Data Access Bottleneck. It is common that Statoil geologist have to involve IT staff for data access especially when geologists need to 'explore' the data, e.g., in the case when the concrete information need is not clear and depends on the available data or when a new access point has to be created. Thus, IT staff become the de facto mediators between geologists and databases and this is the case not only for Statoil but for large and data intensive companies in general [16]. In practice, it is often the case that the availability of IT personnel that both understand the information need of the geologist and the inner workings of the data sources and tools necessary to answer the information need is scarce, and such people are often overloaded. Moreover, development of a new access points is a very time consuming process and may take up to several weeks; e.g., in Statoil it commonly takes up to several days to produce an access point that completely answers the required information need. The concrete time of course depends on the complexity of the query task and the degree of the involvement of the IT staff. As the result, the IT staff involvement became the (time) bottleneck of the data access.

2.4. Improving Access to Statoil's Data

There are around 900 geologists and geophysicists in Statoil and accessing data is their routine. Currently, if the access is done via existing access points, then the data can be extracted relatively fast, while the average turnaround for new access points is about four days. Reducing this

[4] `http://www.safe.com/`

time from several days to several hours would potentially bring a significant saving by improving the effectiveness of Statoil's exploration department, which is key to their overall competitiveness and profitability. One way to reduce the data access time is to provide Statoil geologists with a way to express their information needs to the system directly, without an intervention of the IT staff. In the next section we describe an Ontology Based Data Access approach that has the potential to satisfy Statoil's needs.

3. Ontology Based Data Access

Ontology Based Data Access (OBDA) is a prominent approach for end-user oriented access to databases. OBDA relies on Semantic Web technologies, and its application to relational databases has been heavily studied by the Semantic Web community. Using an example from the oil industry domain, we now describe the main concepts of Ontology Based Data Access to relational databases.

3.1. General Idea

The main idea behind OBDA is to provide the user with access to the data store via the use of a domain specific vocabulary of classes, i.e., unary predicates, and properties, i.e., binary predicates, that the user is familiar with. This vocabulary is related to the database schema via view definitions, called *mappings*; thus, technical details of the database schema are hidden from end-users. For example, in the case of Statoil, the majority of EPDS table names are not meant to be read by end-users, e.g., `EXTOBJIND_BKUP`, `RCA_GRDENS`, and `SSRF_RCK_SEG`, while the semantics of others are clearer, e.g., `DOCUMENT`, `WELLBORE`, and `CORE`. The user formulates queries in terms of the classes and properties in an object-centric fashion, e.g., they can ask for all *wellbores penetrating* a *rock layer* of a specific *age*. Queries over the domain vocabulary are then unfolded into queries over the database schemas and executed over the data by DBMS. An important feature of the OBDA approach is that the domain vocabulary is enhanced with a set of formal axioms that constitute an *ontology*, e.g., the ontology may say that each layer of chalk rock has a specific age. In contrast to database constraints, ontological axioms can be exploited to enrich query answers with implicit information. That is, in the OBDA scenario the data is treated under the so-called Open World Assumption and can be incomplete w.r.t. to the ontology axioms. E.g., the data may state that the rock layers in the area surrounding the Ekofisk field are chalk, but not their age, and thus this data would be incomplete w.r.t. the axiom stating that all chalk layers are from Cretaceous era. Enrichment of answers is done via logical reasoning: a user query Q over the domain vocabulary can be rewritten into a new query over this vocabulary that is logically equivalent to Q w.r.t. the ontology and 'absorbs' a fragment of the ontology relevant for answering Q. For example, if the user asks for wellbores that penetrate rock from the Cretaceous era, then this query will not return the wellbores of Ekofisk, while the rewritten query will also ask for wellbores that penetrate chalk, and thus return the Ekofisk's wellbores. To sum up, an OBDA system provides the user with object-centric domain vocabulary enhanced with an ontology that allows queries to be formulated in intuitive terms that the user is familiar with. OBDA systems are able to enrich user queries with ontological knowledge and unfold them into queries over databases. Based on our Statoil inspired example, we now give more details of OBDA components.

3.2. Relational Databases

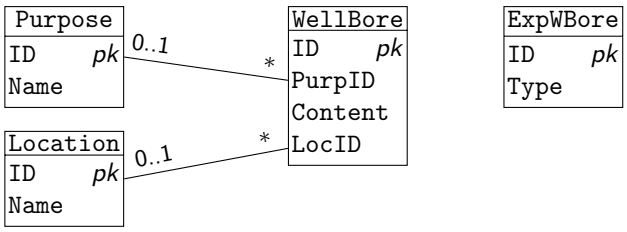
In the relational data model, a database consists of a schema and instance, where the schema is a set of table names with corresponding attribute names and constraints, e.g., primary and foreign keys. The instance 'populates' tables of the schema with tuples by assigning values to the tables' attributes. We illustrate this on our running example.

Example 2. *Consider the sample database schema inspired by our experience with Statoil in Figure 4(a). The schema consists of four tables. Tables* `Purpose` *and* `Location` *are for storing wellbore purposes and locations, respectively. The table* `WellBore` *is for storing information about wellbores and it has four attributes including* `ID` *as the primary key, and* `PurpID` *and* `LocID` *as foreign keys related to the tables* `Purpose` *and* `Location`*, respectively. Finally, the table* `ExpWBore` *is for storing information about exploration wellbores. In Figure 4(b) we present a sample database instance over the sample schema.*

3.3. Ontologies

An ontology is usually referred to as a 'conceptual model' of (some aspect of) the world. It introduces the vocabulary of classes and properties that describe various aspects of the modelled domain. It also provides an explicit specification of the intended meaning of the vocabulary by describing the relationships between different vocabulary terms. These relationships are specified with special first-order formulae, also called axioms, over the vocabulary. An ontology can be thought of simply as a set of such axioms—i.e., a logical theory. Besides axioms, an ontology can contain ontological facts, which can be specified as first-order atoms with constants but not variables. These constants are interpreted as (representations of) objects of the domain. Viewing an ontology as a logical theory opens up the possibility of using automated reasoning for different tasks, including checking an ontology's internal consistency (checking whether the ontology does not entail 'false', i.e., if it is logically consistent), query answering, and other (non-)entailments.

Example 3. *Figure 4(c) lists a vocabulary for describing wellbores with classes such as* Purpose, Location *and classes for different wellbores; and properties such as*



(a) Database Schema

Location:

ID	Name
L1	Norway
L2	UK

Purpose:

ID	Name
P1	Shallow
P2	Injection

Wellbore:

ID	PurpID	Content	LocID
W1	P1	Dry	L1
W2	P2	Oil	L2

ExpWBore:

ID	Type
E1	Active
E2	Discovery

(b) Database Instance

Classes: Location, Purpose, WellBore, ExplorationWellBore, ShallowWellBore,
Properties: hasLocation, hasPurpose, hasName

(c) Ontology Vocabulary

$$\mathsf{ExplorationWellBore} \sqsubseteq \mathsf{WellBore} \quad (1)$$
$$\mathsf{ShallowWellBore} \sqsubseteq \mathsf{WellBore} \quad (2)$$
$$\mathsf{WellBore} \sqsubseteq \exists \mathsf{hasContent} \quad (3)$$

(d) Ontological Axioms

$$\mathsf{ExplorationWellBore}(f(\texttt{ID})) \mapsto \quad (4)$$
```
SELECT ID
FROM ExpWBore
```

$$\mathsf{ShallowWellBore}(f(\texttt{W.ID})) \mapsto \quad (5)$$
```
SELECT W.ID
FROM WellBore W, Purpose P
WHERE W.PurpID = P.ID
  AND P.Name = "Shallow"
```

$$\mathsf{hasLocation}(f(\texttt{ID}), f(\texttt{LocID})) \mapsto \quad (6)$$
```
SELECT ID, LocID
FROM WellBore
```

(e) Mappings

$$\mathsf{ExplorationWellBore}(\mathsf{obj}_{\mathsf{E1}}) \quad (7)$$
$$\mathsf{ExplorationWellBore}(\mathsf{obj}_{\mathsf{E2}}) \quad (8)$$
$$\mathsf{ShallowWellBore}(\mathsf{obj}_{\mathsf{W1}}) \quad (9)$$
$$\mathsf{hasLocation}(\mathsf{obj}_{\mathsf{W1}}, \mathsf{obj}_{\mathsf{L1}}) \quad (10)$$
$$\mathsf{hasLocation}(\mathsf{obj}_{\mathsf{W2}}, \mathsf{obj}_{\mathsf{L2}}) \quad (11)$$

(f) Ontological Facts

Figure 4: Sample Ontology Based Data Access Scenario

hasLocation, hasContent *and* hasName. *Figure 4(d) contains axioms using the vocabulary. Axioms* (1) *and* (2) *state that exploration wellbores and shallow wellbores are wellbores. Axiom* (3) *states that every wellbore has some content. In Figure 4(f) there are ontological facts stating that* $\mathsf{obj}_{\mathsf{E1}}$, $\mathsf{obj}_{\mathsf{E2}}$ *are exploration wellbores,* $\mathsf{obj}_{\mathsf{W1}}$ *is a shallow wellbore, and* $\mathsf{obj}_{\mathsf{W1}}$ *and* $\mathsf{obj}_{\mathsf{W2}}$ *have respectively the locations* $\mathsf{obj}_{\mathsf{L1}}$ *and* $\mathsf{obj}_{\mathsf{L2}}$.

In the last decade, a number of tools for ontological reasoning have been developed. The existence of such tools was an important factor in the design of the OWL ontology language [18] and its basis in description logics [19]. OWL was initially developed to be used in the Semantic Web. The availability of description logic based reasoning tools has, however, contributed to the increasingly widespread use of OWL, not only in the Semantic Web, but as a popular language for ontology development in fields as diverse as biology [20], medicine [21], geography [22], geology [23], astronomy [24], agriculture [25] and defence [26]. Recently ontologies has been recognised as a prominent mechanism for data integration and end-user oriented data access.

3.4. Mappings

Via mappings one can declaratively define how ontological terms are related to terms occurring in the relational schema. Mappings are essentially (read-only) view definitions of the following form that declare how to populate classes with objects—in OWL objects are represented with Uniform Resource Identifiers (URIs)—and to populate properties with object-object and object-value pairs:

$$\mathit{Class}(f_o(x)) \mapsto \texttt{SQL}(x), \mathit{Property}(f_o(x), f_o(y)) \mapsto \texttt{SQL}(x, y),$$
$$\mathit{Property}(f_o(x), f_v(y)) \mapsto \texttt{SQL}(x, y),$$

where $\texttt{SQL}(x)$ and $\texttt{SQL}(x, y)$ are SQL queries with respectively one and two output variables, and f_o, f_v are functions that 'cast' values returned by SQL into respectively objects, i.e, URIs, and values.[5] Classes are populated with URIs $f_o(x)$ computed from the values x returned by $\texttt{SQL}(x)$. Properties can relate two objects, e.g., by stating that a wellbore has a particular location, or assign a value to an object, e.g., stating that a location has a particular name (a string), and they are respectively populated with pairs of objects $f_o(x)$, $f_o(y)$ or pairs of an object $f_o(x)$ and value $f_v(y)$ computed from the values x and y returned by the SQL query. A mapping is *direct* if it relates a table to a class or an attribute to a property. We will rely on direct mappings in automatic mapping generation (see Section 5.2 for details). Given a database and a set of mappings over it, one can execute SQL queries in the mapping definitions and populate the classes and properties of the mappings, thus creating a set of ontological facts. This process is usually referred to as *materialisation* of the ontological facts defined by the mappings.

Example 4. *Figure 4(e) contains three mappings. The first one defines how to populate the class of exploration wellbores with objects computed from IDs of* `ExpWBore` *table. This mapping is direct. Materialisation of facts defined by this mapping over the sample database instance in Figure 4(c), gives Facts* (7) *and* (8) *in Figure 4(f). Note that* $\mathsf{obj}_{\mathsf{W1}} = f_o(\texttt{W1})$ *is the URI computed from the first tuple in the table* `ExpWBore`*. Mapping* (5) *defines which objects can be in* ShallowWellBore *wellbores via a selection on the purpose name. This mapping is not direct and Fact* (9) *is the materialisation of this mapping over the sample database instance. Finally, Mapping* (6) *is direct for the property* hasLocation*, and Facts* (9) *and* (11) *are the materialisation for this property over the sample database instance.*

In contrast to common existing approaches, like e.g. the ETL processes behind access points shown in Figure 3, OBDA mappings are declarative, and their semantics is captured by the ontology. The mappings must still be constructed and maintained, but the potential for automation and tool support is high because of the precise definition of each mapping and their relative low complexity. Indeed, mappings are atomic in the sense that each of them relates to DBs one class or property. Therefore, the SQL query of each mapping much smaller than the typical SQL code of ETL processes behind access points or in collections of SQL views. This makes the SQL code of individual mappings more readable and more maintainable that the one behind access points. Another advantage of OBDA mappings is that they can be reused by many data acces tasks: they 'connect' ontologies to DBs and then can be used for any query over the ontology, while each ETL processes like in Figure 3 is for a concrete data access task and can hardly be reused for other tasks.

[5] These functions should ensure coherent generation of URIs that respects primary and foreign keys.

3.5. Query Answering Over Ontologies and in OBDA

As we discussed above, ontological axioms and facts are a logical theory, and query answering over them is logical reasoning rather than operations on a data instance as in the case of relational databases. To see the difference between relational and ontological query answering, consider the following example.

Example 5. *The following query (in SPARQL notation) selects all objects that have content:*

```
SELECT ?x WHERE { ?x :hasContent ?y. }
```

Considering the ontological facts in Figure 4(f) as a database instance with two unary and one binary tables, it is easy to check that answering the query over this instance yields no answers since the data has no information about hasContent*. At the same time, the facts in Figure 4(f) together with the axioms in Figure 4(d) logically entail three answers to this query:* $\mathsf{obj}_{\mathsf{E1}}$, $\mathsf{obj}_{\mathsf{E2}}$, *and* $\mathsf{obj}_{\mathsf{W1}}$*. These answers are not explicit, but rather implicit, e.g.,* $\mathsf{obj}_{\mathsf{W1}}$ *is a shallow wellbore, thus, due to Axiom* (2) *it is also a wellbore, and therefore, due to Axiom* (3)*, it should have content.*

So, since query answering in OBDA involves logical reasoning, it allows one to retrieve both explicit and implicit query answers. It can be seen as reasoning over the ontology and (virtual) ontological facts that one would have if materialising them from the database over which the mappings are defined. In the case of OBDA, this reasoning is simulated via a two step process of query *rewriting* and *unfolding* as presented in Figure 5. A query Q over the ontological vocabulary is first rewritten w.r.t. the ontology $\mathcal{O}$ into a union Q' of queries s.t. each query of Q' is subsumed by Q. This rewriting is in essence a compilation of relevant ontological information into Q and similar to the resolution procedure in Prolog. We refer the reader to [27] for details on rewriting, and illustrate it with an example below. Then, the query Q' is unfolded into an SQL query $\texttt{SQL}$ using mappings. Finally, $\texttt{SQL}$ is evaluated over the database instance using a database management system which gives the set of answers. It was shown in [27] that the answers computed via rewriting and unfolding, after they are turned into objects or values with the appropriate f_o or f_v, coincide with the answers that are computed over the virtual ontological facts.

Example 6. *Rewriting the query from Example 5 over the sample ontological axioms in Figure 4(d) gives the union*

```
SELECT ?x WHERE { ?x hasContent ?y. }
  UNION  { ?x a WellBore. }
  UNION { ?x a ExplorationWellBore. }
  UNION { ?x a ShallowWellBore. }
```

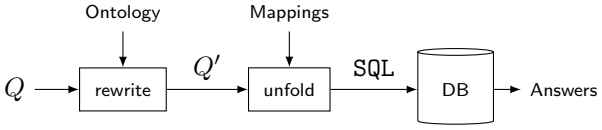


Figure 5: Query processing in OBDA

The next step is unfolding the rewritten query with the sample mappings in Figure 4(e) into the following SQL query:

```
SELECT f(ID) AS x FROM ExpWBore
 UNION
SELECT f(W.ID) AS x FROM WellBore W, Purpose P
WHERE W.PurpID = P.ID AND P.Name = "Shallow".
```

The unfolding is built from Mappings (4) *and* (5)*. As there are no mappings for* hasContent *and* Wellbore*, these parts of the query are simply ignored. Evaluation of this SQL query over the data instance in Figure 4(b) returns* `E1`*,* `E2`*, and* `W1`*. Translation of these values into objects gives the same answers as in Example 5, where we computed the answers over the virtual ontological facts.*

In the next section we discuss advantages and disadvantages of OBDA for Statoil and present technical requirements for OBDA to fulfil Statoil's needs.

4. OBDA: Advantages for Statoil and Limitations

4.1. OBDA: Advantages for Statoil

The main advantage of OBDA systems is that they offer an end-user oriented vocabulary that can be used to *compose* queries. Predefined query templates that are typically available for end-users are not satisfactory (i.e., have limited applicability) in scenarios such as the one of Statoil, where there is a frequent need for queries that do not follow the templates. OBDA, in contrast, offers a vocabulary that can be seen as 'building blocks' that are not specifically tailored to any query but can be used to compose a variety of queries. Thus, OBDA naturally supports data exploration tasks needed in Statoil and allows the users to write new queries themselves without requesting IT support.

Another advantage of an OBDA system is that it can be layered on top of databases without any required customisation. View-based query answering and data warehouses follow a similar approach, but they typically require to move data from the original database to the views, or the data warehouse. OBDA, in contrast, is a so-called *virtual* approach, that is, it does not require to pre-materialise any data at all. Thus, OBDA systems avoid the problem of updating materialised data. The OBDA approach is OLTP rather than OLAP oriented. As we observed in Statoil, geologists typically need OLTP queries, up-to-date answers, and the ability to compose queries themselves. Thus, we see OBDA as more suitable for our use-case than other technologies, e.g., data warehousing.

4.2. OBDA: Limitations for Statoil

The advantages of OBDA come with challenges.

The first challenge is to obtain the required components to install an OBDA system, i.e., ontologies and mappings. To overcome this we developed a bootstrapper that is able to extract ontologies and direct mappings from relational schemata. To address the industrial domain at Statoil, we developed an ontology, called *Subsurface Exploration (SE)* ontology, that is heavily relies on information extracted from the seven databases presented in Section 2 and that widely covers the domain of the petroleum subsurface exploration. Details on our semi-automatic ontology construction techniques can be found in Section 5.2 and our experience with deployment in Statoil in Section 6.

The second challenge is to guarantee that our OBDA system is able to process semantic queries over massive amounts of data, as in Statoil. Since query processing in OBDA requires rewriting, unfolding, and query execution with an RDBMS, this is not a trivial task. In Sections 5.3 and 5.4 we present our query optimisation techniques and in Section 7 results of our experiments.

The third challenge comes with the assumption that the users can formulate queries over the ontological vocabulary. SPARQL [28] is the standard query language over ontologies; however, it is not end-user oriented. To overcome this challenge, we developed an interface—see Section 5.5 for details of our techniques–tailored for users that are not familiar with formal query languages. Our interface is expressive enough to cover most of typical queries we acquired from Statoil geologists and in Section 8 we present results of our user study.

5. Optique Platform: System Overview and Technical Background

We start with an overview of our OBDA platform Optique and then into technical details of four of its components that are the most critical for the deployment in Statoil.

5.1. System Overview

The general Optique platform architecture, shown in Figure 6, has two user roles: the end-user, in our case typically a geoscientist that wants some specific access to data, and the IT-expert, whose job it is in part to provide such end-users with access to data. The Optique platform consists of various components that are integrated in a common framework, provided by the Information Workbench [29]. Ontologies, mappings, queries and other specifications, like SPARQL repository settings, are stored in a central repository that all components can access.

The end-user interacts primarily with our query formulation tool (see Section 5.5 for details) that lets the end-user formulate SPARQL queries using the domain vocabulary defined in the ontology and issue these queries over the data sources connected to the platform. The client

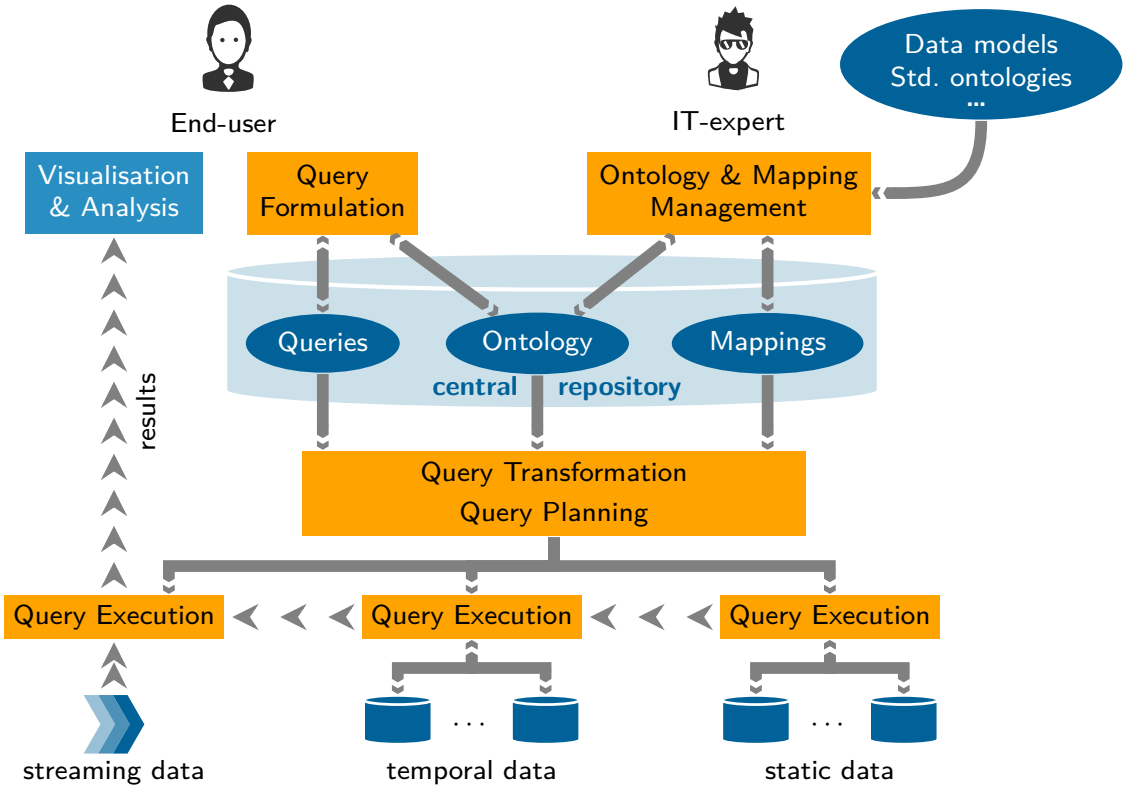


Figure 6: Optique Architecture Overview

side interface of the query formulation tool is driven by information in the ontology, and this information is fed to the client by a server side part that can also exploit ranking using query logs to better the user's efficiency and user experience.

The queries constructed by the formulation tool is processed by our query transformation module (see Section 5.3 for details), which considers the information in the ontology, mappings, data source schemas, and additional optimisation settings, to transform the SPARQL query into an optimised federated SQL query over all the sources connected to the platform. This SQL query is passed on to our query planning and execution module (see Sections 5.3 and 5.4 for details), that deconstructs the SQL query and orchestrates the evaluation of the query parts to the correct underlying data sources. The data sources evaluates the query parts as any other SQL query and return the results back to query execution module, which assembles the query answering results of the sources to form the final result of the end-user query.[6] This result is given back to the query transformation module which delivers the result to the end-user following the SPARQL protocol [31]. Depending on how the end-user query was issued, the results can be visualised differently: as a regular table, in a wiki front-end provided by the Information Workbench, or displayed directly in Statoil end-user expert tools, via a transformation of the SPARQL query result set, see Section 9.

The IT-expert is equipped with interfaces to setup and maintain the platform and its artefacts, in particular the deployment module (see Section 5.2 for details) that can construct ontologies and mappings over the data sources using the data schemas as input. There are also interfaces that allow the IT-expert to add new artefacts to the platform, and examine and edit those stored in the central repository, e.g., queries, ontologies and mappings. The platform relies on existing tools for ontology development like Protégé,[7] but features custom-built modules for mapping construction [32].

We now proceed with more technical background on four Optique modules: deployment module for creating ontologies and mappings, query optimisation and query processing module, federated query execution module, and end-user oriented query construction module.

[6] In the setup of the Optique platform at Statoil we only access static data sources, i.e., regular relational databases. In a different installation of the platform, we also access temporal and streaming sources [30].

[7] `http://protege.stanford.edu`

5.2. Deployment

An OBDA specification is typically defined in the literature as a 4-tuple composed by an ontology vocabulary $\mathcal{V}$, an ontology $\mathcal{O}$, a set of mappings $\mathcal{M}$ and a database schema $\mathcal{S}$. A OBDA instance is an OBDA specification where the database schema $\mathcal{S}$ is replaced by $\mathcal{D}$, a database instance for $\mathcal{S}$.

We support the following tasks for bootstrapping ontologies and mappings, or *assets*, from RDBs in order to create an OBDA instance.

(i) Bootstrapping: Given a relational database $\mathcal{D}$, generate an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$. This task can be naturally divided into two sub-tasks.

- *Vocabulary and Ontology generation*: Given $\mathcal{D}$, create a vocabulary $\mathcal{V}$ and an ontology $\mathcal{O}$ over $\mathcal{V}$.
- *Mapping generation*: Given $\mathcal{D}$, $\mathcal{V}$, and $\mathcal{O}$ create a set of mappings $\mathcal{M}$ relating $\mathcal{D}$ with $\mathcal{V}$.

(ii) Importing: Given an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}_1, \mathcal{M})$ and an ontology $\mathcal{O}_2$, return an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, where $\mathcal{O}$ is the alignment of $\mathcal{O}_1$ and $\mathcal{O}_2$.

Task (ii) is important in applications where ontologies (partially) describing the domain of interest have been already created and users want to incorporate them into their semantic data access system.

The bootstrapping of the ontologies and the mappings enables a (preliminary) deployment of semantic data access system. These mappings and ontologies, however, require further inspection by ontology engineers and domain experts to detect the most promising classes, properties, axioms, and mappings, and to verify their quality. Neverthelesss, the bootstrapped assets should meet some minimal requirements so that they can be usable in practice. We have identified the following *metrics* to measure and guarantee a minimum quality of the generated assets: *(1) Ontology language*: compliance with standard ontology languages with well-defined semantics like OWL 2 and its profiles to enable the use of a wide-range of Semantic Web technologies. Note that the choice of the ontology language (e.g., one of OWL 2 profiles) also affects suitability of the different query answering engines. For example, if the bootstrapped ontology is to be used in a OBDA scenario, as in Optique, OWL 2 QL profile is required by the query rewriting engine. *(2) Mapping language*: compliance with standard directives like the W3C direct mapping specification[8] and standard W3C mapping languages like R2RML.[9] *(3) Query coverage*: suitability of the ontology vocabulary to formulate the queries that the user is interested in. *(4) Query results*: accuracy of the query results obtained with the bootstrapped instance in the sense that the query answer should satisfy the user's expectations.

[8] Direct mappings: `http://www.w3.org/TR/rdb-direct-mapping/`

[9] R2RML language: `http://www.w3.org/TR/r2rml/`

To the best of our knowledge existing bootstrapping systems provide limited or no support for the bootstrapping tasks and quality requirements described above, see, e.g., [33, 34] for an overview of such systems. Most of the state of the art bootstrappers fail to conform with the ontology and mapping language standards, or they do not provide profiling capabilities for the outpout ontology. Furthermore, BootOX outperforms existing systems in bootstrapping that were available for benchmarking (see Section 6.3 for details).

Bootstrapping. The goal of bootstrapping is to find patterns in $\mathcal{D}$, i.e., SQL queries $\texttt{SQL}(x)$ and $\texttt{SQL}(x, y)$ that correspond to meaningful classes and properties. We bootstrap three types of mappings depending on what relational algebra operators can appear in their SQL-parts: *(i) projection*, *(ii) selection*, and *(iii) full mappings* that allow any relational algebra operators.

A special kind of projection mappings, that are recommended by W3C as the standard way to export relational data in RDF, are *direct mappings*. They mirror RDB schemata by essentially translating *(i)* each (non-binary) table into an OWL class; *(ii)* each attribute not involved in a foreign key into an OWL datatype property; *(iii)* each foreign key into an OWL object property.

Generation of axioms, however, is usually a more involved process. Ontological vocabulary extracted by direct mappings can be enriched with axioms by propagating RDB constraints, e.g., a foreign key relating two tables could be propagated into a subclass axiom between the two corresponding classes. BootOX [14] relies on a series of patterns to transform database features into OWL 2 axioms and puts special attention to the OWL 2 profile of the generated ontology (see Appendix B for a complete list of patterns). Database features can be encoded using different axioms which may lead to different OWL 2 profiles. For example, primary keys and unique constraints can be modelled with the OWL 2 construct *HasKey*, which is supported in OWL 2 RL and OWL 2 EL, but must be avoided if the target ontology language is OWL 2 QL as in our OBDA scenario.

BootOX can also discover implicit constraints in databases, e.g., minimal primary keys in tables, candidate foreign keys by checking containment between (samples from) projections of tables. While working with EPDS we found that the discovery of implicit constraints was practically important since prominent tables are materialised views without specified primary or foreign keys, and axioms constructed from such constraints are exploited in query optimisation. Note that the bootstrapped ontology can be quite close to the source schema and we see this as a natural outcome: bootstrapping is the first step in OBDA system deployment, and the resulting assets are by no means perfect, but provide a starting point for post-processing and extension.

Selection mappings are bootstrapped in order to create class and property hierarchies, e.g., we take a class C

bootstrapped with direct mappings and verified by users, and in the corresponding tables we learn attributes whose values give good clustering of tuples; then, for each cluster we create a subclass of C. For example in one of the databases the table *WellBore* contains different types (i.e. clusters) of wellbore according to their purpose and one could bootstrap axioms like '*WellBore_has_purpose_Appraisal* SubClassOf: *WellBore*'.

In order to bootstrap full mappings, we discover chains of tables that are 'well joinable', that is, connected via foreign keys or with good overlap on some sets of attributes, and convert them into candidate classes and properties. For instance, for each chain we detect the 'leading' table T and relate the chain to a class by projecting it on T's primary key; then, we combine names of the tables in the chain and attributes on which they were joined to suggest a name for this class.

In addition, BootOX automatically extends direct mappings with metainformation about provenance. The provenance metainformation is modelled in the mapping assertion, adding, for instance, the source database from which the information is extracted, and more granular information, like table and column identifiers.

URI creation. BootOX follows the good practices of the W3C direct mapping specification to create the URI (templates) for classes, properties and individuals in order to minimise the so-called impedance mismatch problem [1], which is caused due to the fact that databases store data values (e.g strings, integers, etc.) while the ontology includes objects uniquely identified by URIs. We address the problem on the level of object generating functions discussed in Section 3.4. Our functions respect primary and foreign keys and ensures that all the generated objects are unique and the same object is generated when required. Note that BootOX has also been used to assist the manually creation of mappings in order to use common URI templates for the referenced ontology classes, properties and individuals.

The R2RML language provides mechanisms (i.e. templates) to implement the object generating functions. For example, the basic rule to generate the URIs for the subjects in a mapping is the following:

```
http://basens/table/{pk1}/.../{pkn}
```

If the table does not contain any primary key the template URI is created using all columns in the table:

```
http://basens/table/{col1}/.../{coln}
```

When dealing with foreign keys, the URI of the object mapping associated to the referencing table should be generated according to the URIs of the subjects for the referenced table. e.g.:

```
http://basens/reftable/{fkcol1}/.../{fkcoln}
```

Exceptions apply when dealing with inheritance. In these cases URI template generation is a bit more elaborated and requires the use of internal indexes to keep track of the URI template required for the referenced tables.

Alignment. A way to extend a bootstrapped ontology is to align it with an existing high quality domain ontology. Ontology alignments (also called *correspondences*) between entities of two ontologies $\mathcal{O}_1, \mathcal{O}_2$ are represented as a 3-tuple $\{e, e', r\}$ where e and e' are entities of $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively; $r \in \{\sqsubseteq, \sqsupseteq, \equiv\}$ is a semantic relation [35]. If correspondences are automatically created a confidence value c, usually, a real number within the interval $(0 \dots 1]$ is typically added to the mapping tuple. Confidence intuitively reflects how reliable a correspondence is (i.e., 1 = very reliable, 0 = not reliable). Although there are several possible representations for alignments, in this work we represent them through standard OWL 2 axioms.

We have extended the ontology alignment system LogMap [36] that aligns two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ by deriving OWL 2 equivalence and sub-class(property) axioms between the terms from $\mathcal{O}_1$'s and $\mathcal{O}_2$'s vocabularies using the lexical characteristics of the terms and the structure of the ontologies. Our extension [37, 38] is a highly scalable solution that, in the resulting ontology $\mathcal{O}$ after $\mathcal{O}_1$ and $\mathcal{O}_2$ are aligned, minimises the violations of the *conservativity principle* i.e., $\mathcal{O}$ does not entail (many) sub-class(property) axioms over $\mathcal{O}_1$'s and $\mathcal{O}_2$'s vocabulary which are not already entailed by $\mathcal{O}_1$ or $\mathcal{O}_2$. When experimenting with the Statoil databases, we noticed that this logical guarantee is often a necessity since an alignment $\mathcal{O}$ of a bootstrapped $\mathcal{O}_1$ with an imported domain ontology $\mathcal{O}_2$ that does not preserve conservativity gives the following side-effects: query answering over $\mathcal{O}$ produces answers that are unexpected by domain experts, and that would not be obtained if one queries $\mathcal{O}_1$ alone, or $\mathcal{O}$ entails axioms over $\mathcal{O}_2$'s terms that are unexpected and counter-intuitive for domain experts.

5.3. Query Processing Optimisation

Our query processing system relies on the two stage process with rewriting and unfolding as described above. It was observed [7] that a naive implementation of this approach performs poorly in practice; thus, we developed and implemented a number of techniques to optimise both stages of query processing, which we present in detail below. We empirically tested the efficiency of our optimisation techniques over EPDS and will present results on query execution in Section 7. We also evaluated our techniques in a controlled environment and our tests show that thanks to these optimisation techniques our query processing solution can dramatically outperform existing OBDA solutions [39].

We observed that despite these optimizations, the SQL queries `UQ` we produce often return answers with a significant number of duplicate rows; below we will discuss why this is an issue and how we addressed it.

Optimisation of Rewriting. We address two challenges:

(i) redundancy in RQ: fragments of RQ may be subsumed by other fragments, and thus evaluation of UQ over RDBs requires redundant computation;

(ii) inefficiency of rewriting: computation of RQ is in the worst case exponential in the size of Q and $\mathcal{O}$, and thus its online computation is often slow for large Q, $\mathcal{M}$ and $\mathcal{O}$.

The main source of redundancy in RQ is that classes (properties) can participate in multiple mappings either directly, or indirectly via their multiple sub-classes (sub-properties).[10] To avoid this, we minimise both the mappings and the UCQ RQ using query containment. To address the inefficiency, we proposed two novel techniques. Both techniques can be applied in isolation or combined. Our first technique is to improve computation of class hierarchies entailed by the ontology, which the rewriting heavily relies on, by applying graph reachability techniques to a DAG-encoding of dependencies among classes. The second one is to move part of online reasoning offline: for all atomic queries we perform expensive rewriting up front and compile the results of this computation into the existing mappings, and use these enriched mappings when user queries Q are unfolded, see [40] for details.

Optimisation of Unfolding. We address three challenges with query unfolding:

(i) redundant unions due to redundancies in the bootstrapped ontology or mappings;

(ii) redundant joins, that come from the fact that on the ontological level the data is modelled as a graph, i.e., as a ternary relation, while on the data level in RDBs it is modelled with n-ary relations, and thus an unfolding of RQ into SQL over an n-ary table naturally introduces n-1 self-JOIN operations;

(iii) inefficient joins come from the so-called *impedance mismatch*, i.e., on the ontological level objects are represented with object ids URIs while in RDBs with tuples; thus, joins in RQ are unfolded into SQL joins over string concatenation that prevents RDBs from the use of existing indices.

To address these issues, we developed *structural* and *semantic* optimisation techniques. For structural optimisations we push joins inside the unions and special functions (such as URI construction) as high as possible in the query tree; we also detect and remove inefficient joins between sub-queries. Semantic optimisations remove redundant unions, joins, detect unsatisfiable or trivially satisfiable conditions, etc.

OBDA Constraints. The optimizations discussed in the previous paragraphs are based on the additional information coming from the DB schema such as *primary and foreign keys*. Unfortunately, these optimization techniques cannot exploit constraints that go beyond what can be explicitly declared in the schema. For instance, the data stored at Statoil has certain properties that derive from domain constraints or storage policies that are not modeled using DB constraints. For example, the view in the database containing information about wellbores, in fact, satisfies the following *implicit* constraints: (1) it must contain all the wellbores in the ontology; (2) every tuple in the view must contain the information about name, date, and well (no nulls); (3) for each wellbore in the view, there is exactly one date/well that is tagged as 'actual'.

In [12] we show how to use this information to further optimize UQ (please note that details of these technique are beyond the scope of this paper and we refer the reader to [12] for examples and further explanations). We do so by introducing two novel classes of constraints that go beyond database constraints. The first type of constraint, *exact predicate*, intuitively describes classes and properties whose elements can be retrieved without the help of the ontology. The second type of constraint, *virtual functional dependency*, intuitively describes a functional dependency over the virtual RDF graph exposed by the ontology, the mappings, and the database. These notions are used to enrich the OBDA specification so as to allow the OBDA system to identify and prune redundancies from the translated queries. We show in [12] that thanks to these optimizations we can improve the overall performance of query executions by orders of magnitude.

Optimisation of distinct answers. Removing duplicates from query answers raises an interesting problem for OBDA systems. On the one hand, OBDA theory assumes set semantics for computation of query answers, that is, each answer occurs only once in the answer set. On the other hand answering queries over relational databases is typically implemented under bag semantics, that is, any answer can occur multiple times in the answer set. In particular, evaluation of SQL queries produced by our query processing system returns answers with duplicates. From our experience with the OBDA deployment at Statoil, these duplicates bring a number of challenges: duplicate answers appear as noise to most end-users, visualisation systems are significantly slowed down when flooded with repeated answers, and the large number of such answers negatively affects network capacity and database connectivity. Using the `DISTINCT` modifier in SPARQL queries to remove redundant answers is unfolded into the SQL `DISTINCT`, which, in our experiments, was extremely detrimental to performance and led to a number of queries to time out. In order to overcome this problem, we propose to defer the removal of duplicate tuples to the OBDA system rather than the underlying database engine. In particular, we filter out redundant answers using predefined Java hash functions. This simple solution outperforms SQL `DISTINCT` by orders of magnitude in most cases, and opens the door for further optimisation (see Section 7).

[10] This issue is partially tackled at the bootstrapping stage [14].

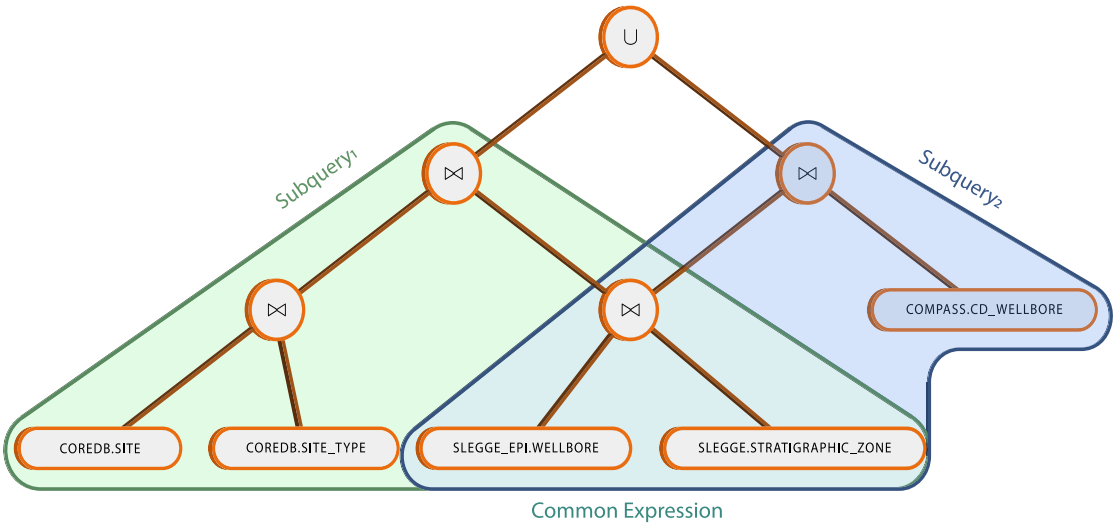


Figure 7: Query Plan.

5.4. Federated Query Execution

The unfolded query produced by the process described in the previous section is a SQL query, that is sent for execution by the underlying RDBMS. As described in Section 2, end users often need to access information that resides in different databases. In the discussed scenario there is need for an underlying system that is capable of importing and processing data from different endpoints. In certain cases it may be preferable to take advantage of the processing capabilities of each endpoint, in order to ship fragments of the query for execution there and only import back an intermediate result, instead of the detailed data. Of course, such a decision should be *cost-based*. For instance a simple policy of sending the largest possible fragment to each end-point is often sub-optimal. Systems that put through these tasks are known as *mediators*, whereas the underlying database systems are known as *endpoints*.

Extending Exareme *With Data Integration Capabilities.* Exareme is an *elastic* execution environment for complex data workflows on the cloud. These data workflows incorporate user computations in the form of *User-Defined Functions* (UDFs). Several extensions have been implemented in order for the system to be able to cope with the demanding requirements of the OBDA installation in Statoil, whereas at the same time preserving its massively parallel processing capabilities:

(i) its optimizer has been re-designed in order to take into consideration common subexpressions coming from different parts of a complex query;
(ii) special data transfer operators have been implemented in order to be able to import data from endpoints;
(iii) a *federated analyzer* module has been implemented, which based on the OBDA mappings, gathers statistics about the external data;
(iv) *pushing data processing* to endpoints as a post-optimization step is considered;
(v) caching and reuse of intermediate results can be enabled to exedite query processing.

In what follows we describe some of these extensions in more detail.

Federated Analyzer. Since the Exareme optimizer is cost-based, the first step in order to be able to make cost estimations for different federated query plans, is to analyze the columns of base tables residing in different endpoints. This is an offline process taking place before query answering. Initially, the OBDA mappings are parsed and a list of all base tables referenced there is obtained. This way Exareme avoids gathering statistics for tables that cannot show up in an unfolded SQL query. This number can be very large for the databases considered in the Statoil environment and this simple optimization saves a lot of computation. For each column of the obtained tables Exareme sends to the corresponding endpoint queries that ask for the different values, the minimum and maximum value and the column size. This way we can obtain basic statistical measures without having to resort to the often unfeasible task of importing all the data.

Common Subexpression Identification. Common subexpression identification refers to the process of identifying the same query fragment in different queries, or in different parts of the same query, and the equally important task of deciding if the specific subexpressions should be computed only once and reused or not. This last decision is not obvious, as reusing a subexpression includes the cost of materializing the intermediate result to disk, whereas if the tuples of the subexpression, as they are produced, can be pipelined to the next query operator, it may be preferable to compute it from the beginning. The decision becomes even more complicated, as when many common

subexpressions exist, the choice for each one possibly affects the cost regarding the choice for the rest. In a parallel environment, the decision to reuse can be even less preferable, as independent query fragments can be computed simultaneously.

Using state of the art techniques in common subexpression identification proved to be crucial in evaluation of OBDA queries, as these contain highly correlated union subqueries. Consider for example the query shown in Figure 7. It consists of two different unions and accesses three different endpoints. Note that the join between tables SLEGGE_EPI.WELLBORE and SLEGGE.STRATIGRAPHIC_ZONE is a common subexpression for these two unions. Exareme includes a Volcano-style optimizer and models the different possible query plans using an AND-OR graph. The optimizer implements a greedy heuristic that was proposed in [41] in order to take the aforementioned decisions.

Pushing processing to endpoints. In a data integration setting where each endpoint is an RDBMS, a mediator can take advantage of the corresponding processing capabilities in order to "push" a query fragment for execution in the endpoint and obtain an intermediate result. One could think of a process of query decomposition where maximal fragments that can be executed in each endpoint are identified and sent for execution. Unfortunately, this approach often leads to inefficient execution plans, as very large intermediate result, that otherwise could be avoided, may be produced. Consider again the example from Figure 7. The join between SLEGGE_EPI.WELLBORE and SLEGGE.STRATIGRAPHIC_ZONE can be pushed to the EPDS endpoint, but if this join leads to a very large intermediate result, then maybe a better query plan would be to import each table separately and use a different join order, by first joining one of the tables with a result coming from another endpoint. The same situation arises for the joins between tables COREDB.SITE and COREDB.SITE_TYPE. For this reason, we examine opportunities for pushing processing towards the endpojnts as a post-optimization step. This is done after an optimized plan, possibly with common subexpression re-usage, has been obtained. In such a plan, we consider pushing fragments that only touch tables from a single endpoint, as long as there is no sub-plan marked as materialized for re-usage. If no plan with more than a single descendant table is found to be beneficial to be pushed to an endpoint, then separate requests for each base table are sent. These requests contain only possible filters and projections for the corresponding base table.

Caching Intermediate Results. Caching of intermediate results refers to the process of keeping results coming from evaluation of a query for future reuse. These results correspond to query fragments imported from endpoints, or results of processing that takes place inside Exareme, for example intermediate results chosen to be materialized from the common subexpression identification optimization or final results of a query. As a subsequent, possibly different query is coming for evaluation, the optimizer should choose a plan, by taking into consideration existing fragments in the cache, and estimate the cost of plans that reuse such fragments accordingly. In the context of data integration this can lead to important savings, as the need of data import can be completely avoided. An eviction policy that guarantees data freshness can be applied, for example by specific timeouts. Also, it is important to provide the user with the option to enable or disable use of the cache on a per query basis.

5.5. Visual Query Formulation

Domain experts could meet their information needs by posing queries via a visual query formulation system, called OptiqueVQS [42, 43, 44, 13], choosing from a collection of existing queries, and writing queries in SPARQL directly. A SPARQL query is automatically generated as a user interacts with OptiqueVQS. In general, OptiqueVQS combines a navigational interaction (i.e., query by navigation) style [45, 46] with graph and form-based representation styles [47]. In this section, we will present OptiqueVQS, which is composed of a front-end (i.e., interface) and a backend component.

Interface. OptiqueVQS is composed of communicating widgets (i.e., user-interface mashup) [48, 13]. Along with flexibility, modularity, and adaptability, such an approach allow us combine multiple representation and interaction paradigms [49, 47].

In Statoil's case, OptiqueVQS is composed of five widgets: The first widget (W1 – menu-based) allows the user to navigate through concepts of an ontology by selecting relationships between them (see the bottom left hand side of Figure 8). The second widget (W2 – diagram-based) presents typed variables as nodes and object properties as arcs and gives an overview of the query formulated so far (see the top side of Figure 8). The third widget (W3 – form-based) presents the attributes of a selected concept for selection and projection operations (see the bottom right hand side of Figure 8). The fourth widget is (W4 – form-based), is a tabular widget (see Figure 9) presenting sample results. It also presents functionality to realise sorting and aggregation operations, such as sum, max, min, and average. The fifth widget (W5 – diagram-based) lets the user constrain attributes by selecting values from a map (see Figure 10). Relevant attributes are annotated so as to bind the map widget to these attributes in W3.

W1 initially lists all the concepts in the ontology and a user starts formulating a query by selecting a starting concept (i.e., kernel). The concept chosen from W1 becomes the active node (i.e., pivot) and appears in W2 as variable node. W1 then lists concept - object property pairs pertaining to the pivot, since there is now an active node. The user can continue adding more typed variables into

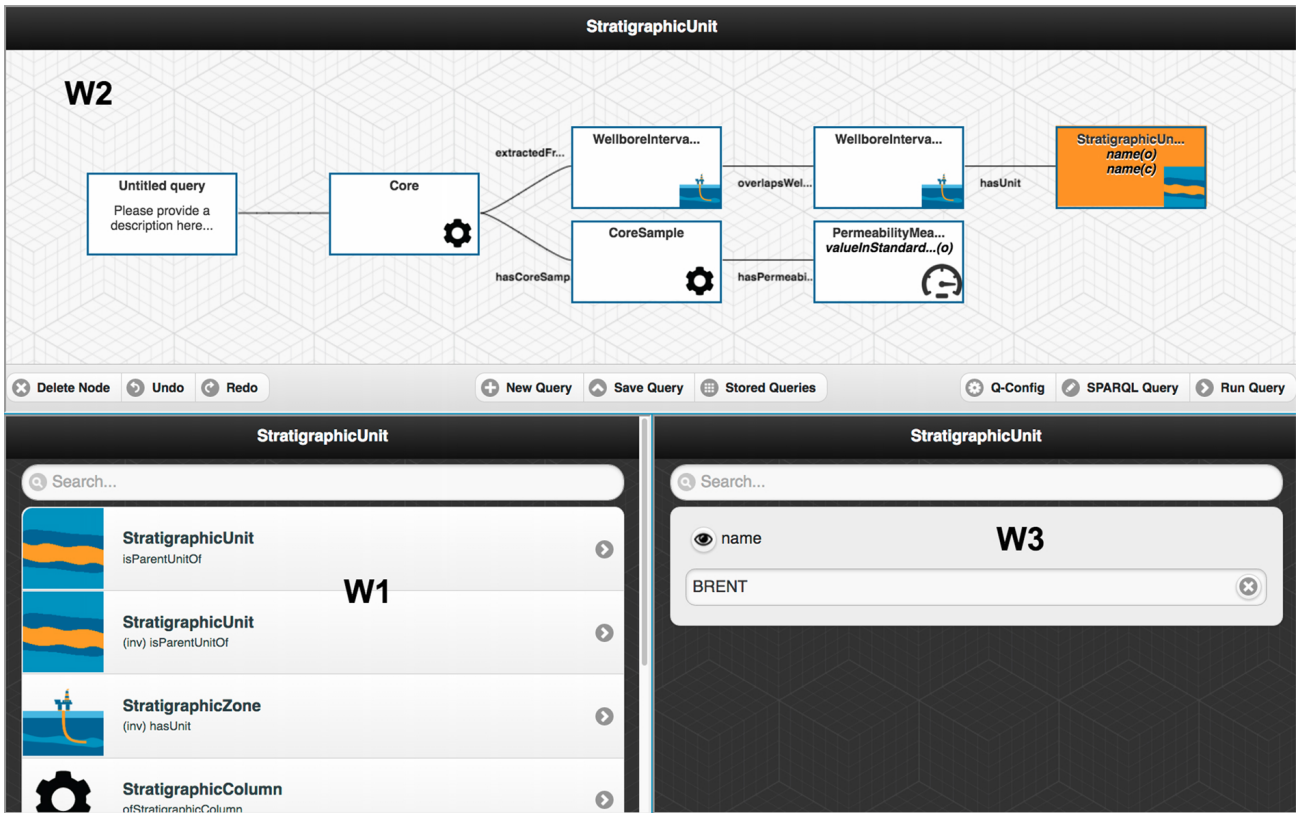


Figure 8: OptiqueVQS interface – an example query is depicted.

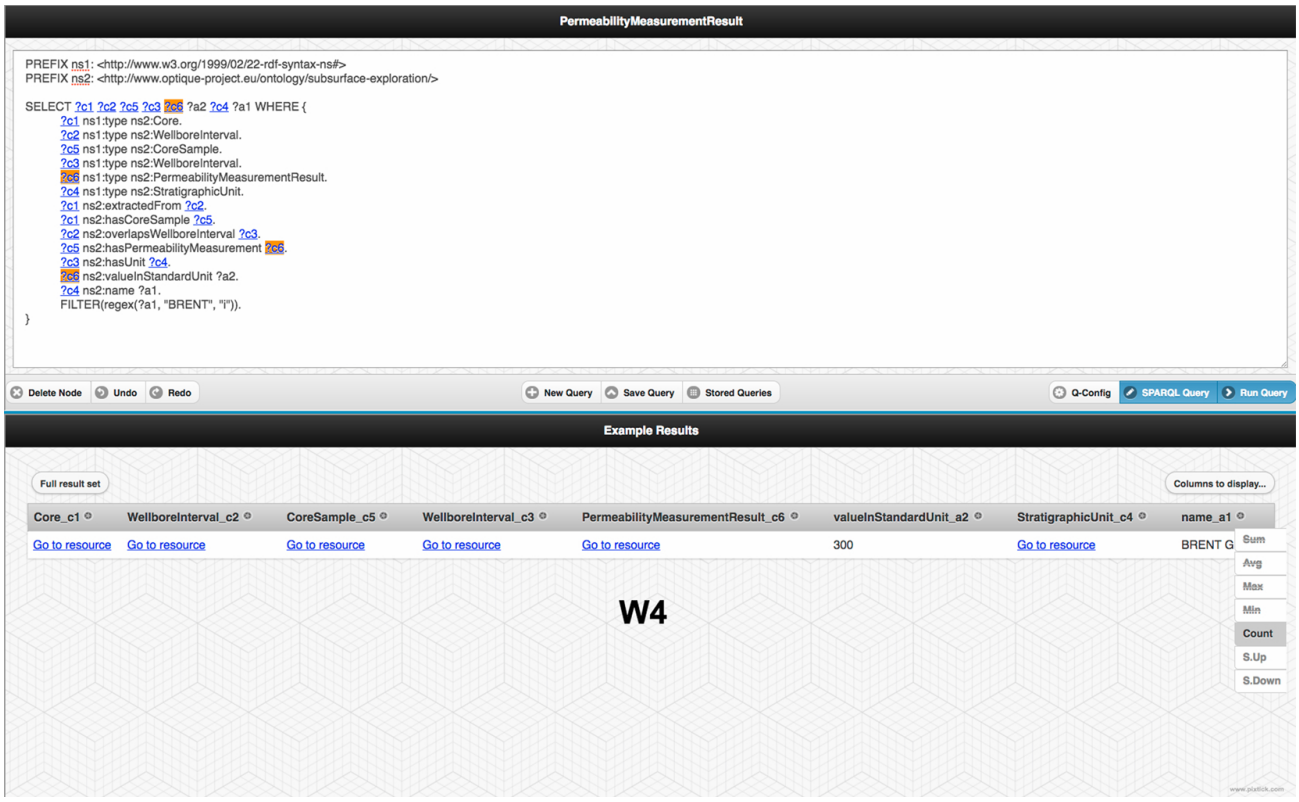


Figure 9: OptiqueVQS interface – tabular result widget and SPARQL view.

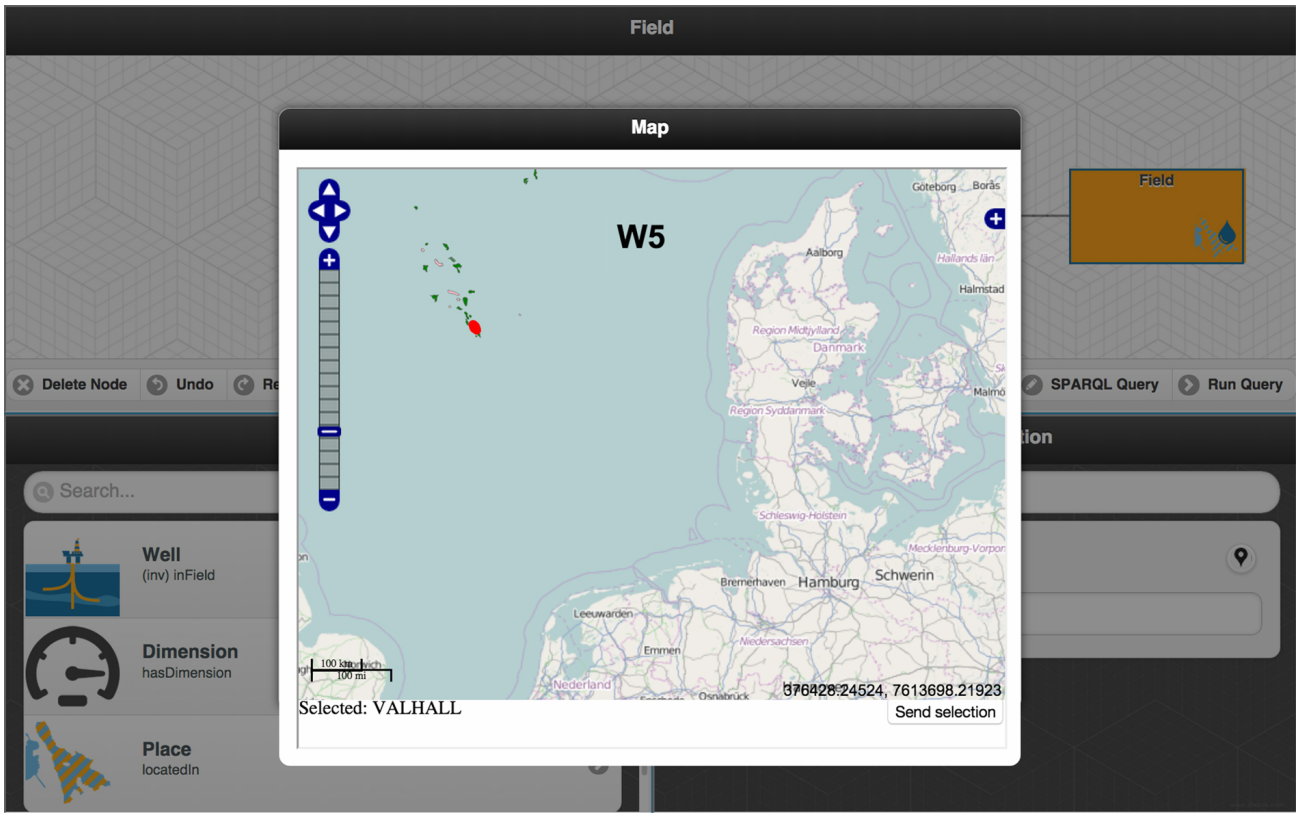


Figure 10: OptiqueVQS interface – another example query where a domain specific map widget is activated.

the query by selecting a pair from W1. Selected concept-object property pair is added to the query over the pivot and the formulated query is presented as a tree in W2. The concept from the last chosen pair automatically becomes the active node (i.e., pivot), and the active node can be changed by clicking on the corresponding variable node in W2. The user can constrain attributes (i.e., using the form elements) and/or select them for output (i.e., using the "eye" icon) through W3. The user can also refine the type of a variable node through a special multi-select form element, called "Type", in W3. A pin icon appears next to each attribute where the map widget could be used (i.e., W5). The user clicks "Run Query" button to see example results and apply sorting and aggregation operations (i.e., W4). The user can view and interact with the query in textual SPARQL mode (i.e., textual and visual modes are synchronised) – see Figure 9. The user can also save, modify, and load queries. An example query is depicted in Figure 8 and the generated SPARQL query is given Figure 9 and Figure 11. Figure 10 presents another example query, where the map widget used to constrain name attribute of a variable node of type "Field".

OptiqueVQS uses a simplified tree-shaped query representation in order to avoid any technical jargon, for example related to OWL and SPARQL. Each widget handles a set of functionality that suits best to its representation and interaction paradigms (e.g., tabular result widget for aggregation operations). OptiqueVQS supports tree-shaped conjunctive queries and queries involving aggregation, while queries involving cycles, negation, and disjunction are not supported.

```
SELECT DISTINCT ?c1 ?c2 ?c5 ?c3 ?c6 ?a2 ?c4 ?a1
WHERE {
    ?c1 ns1:type ns2:Core.
    ?c2 ns1:type ns2:WellboreInterval.
    ?c5 ns1:type ns2:CoreSample.
    ?c3 ns1:type ns2:WellboreInterval.
    ?c6 ns1:type ns2:PermeabilityMeasurementResult.
    ?c4 ns1:type ns2:StratigraphicUnit.
    ?c1 ns2:extractedFrom ?c2.
    ?c1 ns2:hasCoreSample ?c5.
    ?c2 ns2:overlapsWellboreInterval ?c3.
    ?c5 ns2:hasPermeabilityMeasurement ?c6.
    ?c3 ns2:hasUnit ?c4.
    ?c6 ns2:valueInStandardUnit ?a2.
    ?c4 ns2:name ?a1.
    FILTER(regex(?a1, "BRENT", "i")). }
```

Figure 11: A SPARQL query generated by OptiqueVQS.

Backend. The communication between the interface and backend is realised through a REST API. It returns a JSON object according to the interface request. The backend deals with accessing and serving ontology fragments to the interface and harvests the query log and data to improve user experience.

A graph projector is in the core of OptiqueVQS's backend [43, 44]. It feeds OptiqueVQS's widgets to enable a graph-based navigation over an ontology during the query formulation process. Graphs are effective mechanisms to navigate, construct, and communicate complex topological structures for end users. End-user queries could naturally be seen as graphs, since they are mostly conjunctive and we are only dealing with unary and binary predicates. Yet, OWL 2 axioms do not have a direct correspondence to a graph. Even when an axiom can naturally be seen as a graph, to the best of our knowledge there is no standard means to translate it to a graph. Therefore, in order to to extract a suitable graph-like structure from a set of OWL 2, we have adapted a technique called navigation graph [50, 51]. OptiqueVQS uses on the OWL 2 reasoner HermiT [52] to build the navigation graph (e.g., extraction of classification) in order to consider both explicit and implicit knowledge defined in an ontology. Navigation graph approach makes OptiqueVQS domain-agnostic as it could run on any OWL 2 ontology without requiring any domain specific configuration; however, one could still implement domain specific widgets, similar to our map widget, to offer interaction and representation paradigms specific to a domain in order to improve the usability. An example is OptiqueVQS deployment at Siemens for remote monitoring and diagnostics in the context of temporal data sources [53].

A data sampler component is also a part of the backend and it is used to enrich an ontology with additional axioms to capture values from data that are frequently used and rarely changed. This includes the list of values and numerical ranges in an OWL data property range. Such an approach allows presenting attributes in different types, such as sliders, multi-select boxes, date pickers etc, with respect to the underlying data. Moreover, it harvests the query log for ranking and suggesting query extensions as a user formulates a query, that is, the W1 and W3 lists concepts and properties adaptively [54].

6. OBDA Deployment at Statoil

In this section we present our experience in deploying an OBDA instance over Statoil databes. We start with the requirements, then discuss how we developed the ontology and mappings for the Statoil databases, and conclude with a quality assessment of the (automatic) deployment.

6.1. Requirements

Our OBDA solution should enable efficient formulation of information needs from Statoil geologists. In order to achieve this, we conducted interviews with Statoil geologists and IT experts who support them by creating access points. This gave us a few hundreds of information needs expressed in English, that look as follows:

1. In my area of interest, e.g., the Gullfaks field, return wellbores penetrating a specific chronostratigraphic unit, and information about the lithostratigraphy and the hydrocarbon content in the wellbore interval penetrating this unit.
2. Show all the core samples overlapping with the Brent Group.
3. Show all permeability measurements in Brent.

Then, we aggregated thees needs in patterns since many of them asked about essentially the same (or very similar) entities but relied on different concrete 'constants', e.g., several needs were about penetration of stratigraphic layers and they differed only on the names of concrete layers. This aggregation gave us a *Statoil query catalog* of 73 representative Statoil queries in SPARQL, with references to the underlying information needs expressed in natural language. Clearly, our collection of both information needs and corresponding queries is not exhaustive—it is only a sample of what geologist typically ask. At the same time, as we verified with domain experts, the Statoil query catalogue provides a good coverage of topics that are typically of interest for Statoil geologists. From this we derived the first natural minimum requirement for the ontology and mappings:

> **Requirement 1:** *The ontology should enable formulation of queries corresponding to the catalogue's requests and mappings should enable answering these queries.*

To fulfil Requirement 1, the ontology must contain all the terms occurring in the catalogue. For example, Information need 1 contains the terms *wellbores*, *penetrating*, *chronostratigraphic unit*, *lithostratigraphy*, *hydrocarbon content*, and *wellbore interval*. All in all the catalogue contains more than 150 relevant domain terms. As we verified with Statoil geologists, the terms occurring in the catalogue are important, but, as expected, do not provide a sufficient domain coverage; that is, geologists need many more domain specific terms for expressing their information needs. Via interviews with geologists, we determined six domains that should be reflected in the ontology: geospatial, geometrical, enterprise, production, seismic and oil related facilities, which gave the following requirement:

> **Requirement 2:** *The ontology should cover a wide range of geological domain terms including the ones from the catalogue and the six relevant domains.*

A desired requirement of the ontology and mappings is, not only to cover the necessary vocabulary to enable the formulation of queries, but also to enable the correct answering of these queries.

> **Requirement 3:** *The ontology and mappings should lead to the expected query results in the OBDA solution.*

Table 2: Ontology metrics for the Subsurface Exploration (SE) ontology, and for all bootstrapped (Boot) ontologies, which are also aligned (Align) with the SE ontology. The metrics are calculated by the OWLAPI Java API. Zero- and false-values are removed from the table to increase readability.

	SE	EPDS		Recall		GeoChemDB		CoreDB		OpenWorks	
		Boot	Align	Boot	Align	Boot	Align	Boot	Align	Boot	Align
Overview											
Axioms	759	433 624	434 545	15 358	16 263	73 024	73 929	1 140	2 046	212 609	213 519
Logical axioms	520	139 037	139 576	4 895	5 419	22 280	22 804	363	888	63 666	64 195
Classes	106	3 329	3 435	35	141	136	242	16	122	1 472	1 578
Object properties	49	5 560	5 609	17	66	15	64	21	70	3 734	3 783
Data properties	42	63 177	63 219	1 853	1 895	9 329	9 371	117	159	34 581	34 623
Individuals	6	1	8	1	7	1	7	1	7	1	7
Imports	1	1	1	1	1	1	1	1	1	1	1
Profiles											
OWL2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWL2 QL		✓		✓		✓		✓		✓	
OWL2 EL				✓		✓		✓			
OWL2 RL											
Class Axioms											
SubClassOf	150	12 034	12 187	122	275	442	595	46	199	4 988	5 141
Equivalent			16		1		1		2		6
Disjoint	188		188		188		188		188		188
GCI count											
Hidden GCI Count			24		1		2		3		11
Max. superclasses	2		2		2		2		2		2
Avg. superclasses	1.019	1.0	1.001	1.0	1.014	1.0	1.008	1.0	1.016	1.0	1.001
Multiple inheritance	21		21		21		21		21		21
Object Property Axioms											
SubPropertyOf	20	5 001	5 021	10	30	2	22	9	29	1 647	1 667
Equivalent											
Inverse										12	12
Disjoint											
Functional	9		9		9		9		9		9
InverseFunctional											
Transitive											
Symmetric	1		1		1		1		1		1
Asymmetric											
Reflexive											
Irrefexive											
Domain	37	3 506	3 543	14	51	14	51	17	54	1 916	1 953
Range	38	3 256	3 294	17	55	15	53	21	59	2 124	2 162
SubPropertyChainOf											
Data Property Axioms											
SubPropertyOf	12	32 041	32 053	1 421	1 433	5 878	5 890	53	65	14 262	14 274
Equivalent											
Disjoint											
Functional	15		15		15		15		15		15
Domain	24	40 376	40 400	1 458	1 482	6 600	6 624	100	124	18 143	18 167
Range	20	42 823	42 843	1 853	1 873	9 329	9 349	117	137	20 574	20 594
Annotation Property Axioms											
Annotations	39	222 510	222 687	8 547	8 724	41 253	41 430	612	789	109 145	109 322
Domain											
RangeOf											
Individual Assertions											
Class	6		6		6		6		6		6
ObjectProperty											
DataProperty											
NegativeObjectProperty											
NegativeDataProperty											
SameIndividual											
DifferentIndividuals											

6.2. Development of Ontologies and Mappings

In order to meet the aforementioned requirements, the ontology developed for Statoil consists of *(i)* A part which is bootstrapped from the Statoil databases. For example, running the bootstrapper over the EPDS database extracted an ontology comprising 3,329 classes, 68,737 properties, and 139,037 axioms from explicit and implicit constraints. See Table 2 for the complete list of ontology metrics. *(ii)* A second part, the Subsurface Exploration (SE) ontology, developed during the Optique project to cover parts of the petroleum subsurface exploration domain with a special focus on the information needs (i.e. query catalog) to meet Requirement 1. This includes concepts and relations for describing, e.g., fields, wells, wellbores, and

subsurface conditions and environments. The SE ontology contains 106 classes, 49 object properties, 42 datatype properties, and 520 logical axioms.

Figure 12 shows an overview of the resulting OBDA instance for each of the Statoil databases together with the (imported) SE ontology. Next we provide more details about the creation of the OBDA instance.

The bootstrapped part helps us to meet both Requirement 1 and Requirement 2 in order to include a broader set of terms which may be relevant for future information needs covering the six indentified domains. The bootstrapped ontologies and the SE ontology were aligned using the techniques presented in Section 5.2 (see column Align in Table 2). Special care was taken to avoid introducing unwanted consequences: for instance the alignment techniques will avoid adding alignment axioms that would lead to inconsistencies, or faulty consequences like $Well \sqsubseteq WellBore$ that are not supported by the input ontologies [37, 38] and would prevent meeting Requirement 3.

Most of the axioms in the ontologies, including all bootstrapped axioms, fall in the OWL 2 QL profile, which is required for OBDA to guarantee correctness in the query rewritting (see Table 2). Examples of OWL 2 QL axioms are $\mathsf{NaturalGasLiquid} \sqsubseteq \mathsf{Petroleum}$ (natural gas liquids are a sort of petroleum), $\mathsf{Wellbore} \sqsubseteq \exists \mathsf{hasLicense}$ (each wellbore has a license associated to it), and $\mathsf{Company} \sqsubseteq \exists \mathsf{hasName}$ (each company has a name). The SE ontology contains a few non-OWL 2 QL axioms, which we approximated in OWL 2 QL using the techniques of [55].

In order to fulfil Requirement 3, the SE ontology was manually linked via R2RML mappings to the data sources: EPDS, OpenWorks, Recall, CoreDB and GeoChemDB and a total of 75 mappings were created. The bootstrapped mappings were also complemented with manually create complex mappings since there were cases where the bootstrapped mappings did not sufficiently reflect their relation to the correspondent database in order to meet the information needs. See Table 3 for metrics about the manually created and bootstrapped mappings.

The column *Federated* contains mappings where the source SQL query touches more than one database. These mappings are only usable in a federated OBDA setting. All the other mapping rules have SQL queries that each selects data only from a single database.

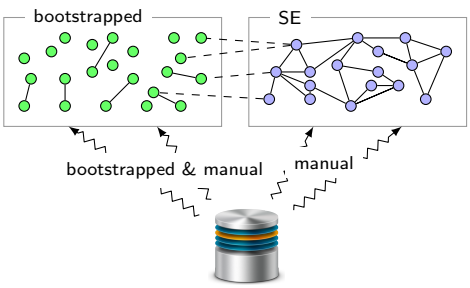


Figure 12: OBDA instance creation for a given database.

We try to avoid such federated mappings because of Statoil policy on SQL queries: Because of the complexities in the SQL schemas, and the rate of change, there is for each database a single team of people who are tasked with the writing of all SQL queries towards that database. Requiring these teams to overlap is organizationally hard, hence, no single person should have to be able to write SQL towards more than one database. This implies that each single SQL query also should only be towards a single database. Future work is to explore the implementation of SWRL rules to replace these federated mappings.

Rows under the headline *Mappings* give the number of possibly non-unique instances of R2RML mapping constructs. Rows under the headline *Database* give data about the contents of the instances of `rr:sqlQuery` and `rr:tableName`, while the row under *Ontology* gives the distinct number of uses of ontological terminology in the mappings.

The queries in the manually created mappings involve up to 6 tables, with an average of 3. The number of output variables SQL queries of mappings ranges from 2 to 4, with the average of 3. In order to develop the mappings we analysed predefined queries used by different information extraction tools over the Statoil databases. In particular, we analysed the results of a set of predefined queries and compared them with the expected results, by interviewing Statoil domain experts. This allowed us to fragment the predefined queries and extract small excerpts useful for OBDA mappings. The relatively small size of the SQL parts of mappings was dictated by two practical reasons: to make maintenance and documenting of the mappings easier, and to ensure efficiency of query processing.

6.3. Assessing the Quality of Our OBDA Deployment

In this section we present results with respect to the quality of the bootstrapped assets (i.e., vocabulary, ontology, and mappings). We have evaluated

(i) the ability of formulating queries (Requirement 1) with the bootstrapped vocabulary (Section 6.3.1), and

(ii) the ability of (enabling the) answering of queries with the bootstrapped ontology and mappings (Requirement 3) in a controlled scenario (Section 6.3.2).

We compared BootOX to five other bootstrapping systems: IncMap [57], Ontop [58], MIRROR [59] and D2RQ [4]. IncMap is designed to directly map a relational database to a target ontology, but is focused on a semi-automatic, incremental/interactive approach rather than direct automated bootstrappings. Ontop, MIRROR and D2RQ follow an approach that is similar to the one employed in BootOX, with respect to the generation of a semantic data access instance (i.e., vocabulary, ontology, and mappings).[11]

[11] Ontop generates only vocabulary, that is, an ontology containing only declaration axioms.

Table 3: Mapping metrics. Each column reports numbers for one mapping collection, which each target one data source. Unless labeled *Boot.* (bootstrapped), the mapping collection is manually constructed.

Datasource:	SE	EPDS	Recall		CoreDB		GeoChemDB		Open Works		Federated
				Boot.		Boot.		Boot.		Boot	
Mappings											
rr:TriplesMap	75	3111	5	34	17	15	11	135	10	1303	1
rr:sqlQuery	75	0	4	0	15	0	11	0	9	0	1
rr:tableName	0	3111	0	34	0	15	0	135	0	1291	0
rr:TermMap	189	0	14	0	39	0	18	0	25	0	2
rr:PredicateMap	0	0	0	0	0	0	0	0	0	0	0
rr:ObjectMap	114	43882	9	1472	22	117	7	6614	15	20071	1
rr:GraphMap	0	0	0	0	0	0	0	0	0	0	0
rr:PredicateObjectMap	114	43882	9	1472	22	117	7	6614	15	20071	1
rr:RefObjectMap	0	137	0	0	0	0	0	0	0	181	0
rr:Join	0	0	0	0	0	0	0	0	0	0	0
rr:subject	0	0	0	0	0	0	0	0	0	0	0
rr:predicate	48	43882	7	1472	12	117	7	6614	12	20059	1
rr:object	0	0	0	0	0	0	0	0	0	0	0
rr:class	26	3111	0	34	7	15	5	135	6	1279	0
Database											
Sum tables	150	3111	4	34	29	15	39	135	15	1291	3
Sum distinct tables	44	3111	1	34	7	15	9	135	7	1291	3
Min. joins per triplemap	0	0	0	0	0	0	0	0	0	0	2
Max. joins per triplemap	6	0	0	0	4	0	3	0	3	0	2
Avg. joins per triplemap	1.0	0.0	0.0	0.0	0.933	0.0	2.545	0.0	0.667	0.0	2.0
Ontology											
Sum distinct ontology terms	74	46993	7	1506	19	132	12	6749	18	21338	1

6.3.1. Coverage of Statoil Query Catalog

The query catalog at Statoil currently includes 73 queries that contain representative information needs from Statoil geologists. For the query coverage experiment we selected a subset of 60 queries for which the required data is stored in the EPDS database.

Since BootOX and IncMap rely on an external domain ontology we selected an state-of-the-art ontology in the oil and gas domain: The Norwegian Petroleum Directorate (NPD) ontology [60]. Note that we did not use for this experiment the SE ontology since its creation was driven by the information needs and hence its coverage with respect to the Statoil query catalog is almost perfect. The target of this experiment is to evaluate an initial deployment with respect to Requirement 1 where a specialised ontology (i.e. SE ontology) may not exixt.

We bootstrapped ontological vocabulary from the relevant parts of EPDS using BootOX, MIRROR, Ontop and D2RQ. Note that IncMap relies on the vocabulary of the available domain ontology. BootOX, unlike Ontop, MIRROR and D2RQ, includes a built-in ontology alignment system which allows to import the vocabulary of the domain ontologies into the bootstrapped ontology.

The results of the query catalog coverage are summarised in Figure 13. The first column represent the coverage of the bootstrapped ontologies computed by BootOX (without importing the domain ontology), Ontop, MIRROR and D2RQ. Since all four systems rely on the direct mapping directives, the bootstrapped vocabulary is, apart from minor differences, basically the same. The middle columns represent the coverage of the vocabulary of the domain ontology, which is equal to the coverage of IncMap. The third column shows the coverage results achieved by the ontology bootstrapped by BootOX including importing. Together with domain experts we performed a manual assessment of each match for both classes and properties. The results of our assessments are also in Figure 13 in the outer circles. For example, 44% of the classes in the query catalog has a good lexical intersection (greater or equal 0.8) with terms of the ontology bootstrapped by BootOX; furthermore, 29% of the classes are fully covered (i.e., true positives), while 19% of the matches are semi-true positives, and 19% are false-positives (i.e. the suggested matches between query catalog classes and ontology classes are wrong for domain experts).

The experiments show that the bootstrapped ontologies

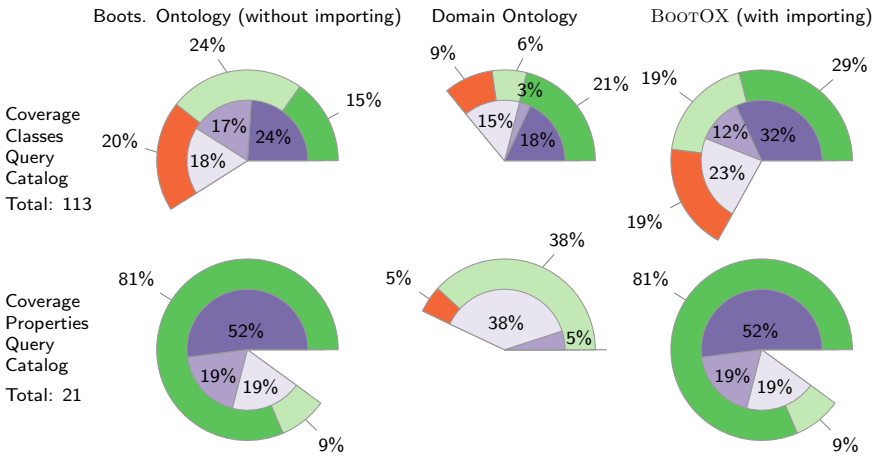


Figure 13: Coverage of terms from the Statoil query catalog with terms from ontologies. Inner pie charts show coverage by lexical confidence using I-SUB [56]: ■ in $[0.9, 1.0]$, ■ in $[0.8, 0.9)$, □ in $[0.6, 0.8)$. Outer pie charts represent the manual verification of the quality of the terms with a coverage above 0.6: ■ *true positive*, ■ *semi-true positive*, ■ *false positive*. Note that *semi-true positives* are not clear-cut cases where the ontology term has a broader or narrower meaning with respect to the query term.

without importing had a higher coverage than the domain ontologies in isolation, e.g., 39% of query class coverage against 27%. These results suggest that there is an adequate number of table and column names with potentially adequate semantic relations with the terms that domain experts at Statoil have in mind when they access data, and thus, the ontology vocabulary computed by the ontology bootstrappers is indeed relevant to query formulation. Nevertheless, the domain ontologies naturally complement the vocabulary obtained from the database and hence BootOX is able to bootstrap an ontology with better coverage over the query catalog than the ones generated by Ontop, MIRROR or D2RQ. For example, 48% of the classes in the catalog are fully or partially covered in the bootstrapped ontology computed by BootOX.

6.3.2. Query Answering in a Controlled Scenario

In this section we assess the quality of the (automatically) bootstrapped ontology and mappings to enable the answering of queries in a controlled scenario.[12]

To this end we ran experiments with a recently released relational-to-ontology benchmark suite, RODI [15, 61], comparing BootOX to IncMap, Ontop, MIRROR, D2RQ, and COMA++ [62]. RODI is designed to test relational-to-ontology mappings end-to-end: it provides an input database and a target ontology and requests complete mappings or mapped data to query. RODI is based on *scenarios*, with each scenario comprising several query tests. While RODI is extensible and can run scenarios in different application domains, it ships with a set of *default scenarios* that are designed to test a wide range of fundamental relational-to-ontology mapping challenges in a controlled fashion. The effectiveness of mappings is then judged by a score that mainly represents the number of query tests that return expected results on mapped data.

[12]Note that assessing the quality of the bootstrapped ontology and mappings in an open scenario like Optique requires a huge involvement of domain experts, thus we leave it for future work.

IncMap is designed to automatically map the target ontology directly to the input database. COMA++ is also an inter-model matching system, while BootOX approached this task in two steps: first, it bootstrapped an intermediate ontology and mappings from the database. Then, it aligned this intermediate, bootstrapped ontology to the target ontology as provided by the benchmark. As mentioned in Section 6.3.1, neither Ontop, MIRROR nor D2RQ include a built-in ontology alignment system to support the importing of the target ontology provided by the benchmark. In order to be able to evaluate these systems with RODI, we aligned the generated ontologies by Ontop, MIRROR and D2RQ with the target ontology using the LogMap system in a similar setup to the one used in BootOX.

Overall, the two most specialized and actively developed systems, BootOX and IncMap, outperform the other existing bootstrapping systems (see [61] for details). In the following we focus on the results obtained in the Oil & Gas scenario. This scenario includes an actual real-world database and ontology, in the oil and gas domain: The Norwegian Petroleum Directorate (NPD) FactPages [60]. Our test set contains a small relational database with a relatively complex structure (70 tables, ≈1,000 columns and ≈100 foreign keys), and an ontology covering the domain of the database. The database is constructed from a publicly available dataset containing reference data about past and ongoing activities in the Norwegian petroleum industry, such as oil and gas production and exploration. The corresponding ontology contains ≈300 classes and ≈350 properties. A total of 443 queries have been compiled in this scenario to cover all of the non-empty fields in the database.

Table 4 shows the average results of the bootstrapping systems for the oil & gas scenario. A closer look to the obtained results by BootOX revealed that, out of the 443 queries, 19 were perfectly answered, 11 returned all the expected tuples but also unexpected tuples, 23 returned

Table 4: Overall scores based on average of per-test F-measure. Best numbers per scenario in bold print.

Scenario	BootOX	IncMap	-ontop-	MIRROR	COMA	D2RQ
Oil & gas domain	**0.14**	0.12	0.10	0.00	0.02	0.08

Table 5: Score break-down for queries that require 1:n class matches.

Scenario	BootOX			IncMap			-ontop-			MIRROR			COMA			D2RQ		
	1:1	**1:2**	**1:3**	**1:1**	**1:2**	**1:3**	**1:1**	**1:2**	**1:3**	**1:1**	**1:2**	**1:3**	**1:1**	**1:2**	**1:3**	**1:1**	**1:2**	**1:3**
Oil & gas domain	0.17	**0.11**	**0.07**	**0.20**	0.01	0.03	0.10	0.09	**0.07**	0.00	0.00	0.00	0.03	0.00	0.00	0.11	0.00	**0.07**

an incomplete set of tuples, and the others returned an empty set of results. A complex feature resulting from the structure of the NPD database and ontology are a high number of 1:n matches, i.e., concepts or properties in the ontology that require a UNION over several relations to return complete results, affected the quality of the results for all bootstrapping systems. Table 5 shows the score break-down for queries that require 1:n class matches. The results, although far from optimal, show that the OBDA assets generated in a completely automated fashion, as in BootOX, can already produce competitive results for a number of queries in a real-world scenario. As mentioned in Section 5.2, these OBDA assets enable a (preliminary) deployment of an OBDA system. However, they require further manual assessment by ontology engineers and domain experts.

7. Query Answering over OBDA Deployment in Statoil

In this section we present query evaluation experiments with our OBDA solution over EPDS and NPD FP. We start with the requirements.

7.1. Requirements

The goal of our OBDA deployment is to make gathering of data from EPDS more efficient; this leads us to the next requirement:

> **Requirement 4:** *Queries from the Statoil catalogue expressed over the ontology should be much simpler than the data queries in corresponding access points.*

We start with analysing the structure of queries from the Statoil catalogue. Most of them, 73%, are either linear or three-shaped conjunctive queries, the others contain aggregate functions and negation. No query in the catalogue has a cycle. Expressing them over the ontology requires from 3 to 13 ontological terms, and the longest query contains 23 terms. In Figure 14, we illustrate the queries from the catalogue that correspond to Information need 3 from Section 6 expressed over the ontology. These queries are quite simple and contain 8 and 13 terms only. At the same time, the Statoil access points corresponding to these two queries are based on two complex SQL queries Q_{IN2} and Q_{IN3}, where Q_{IN2} involves 7 tables and 24 statements in the WHERE clause with 7 technical statements of the form '`T.A is not null`' that ensure correctness of query execution, and 10 joins; Q_{IN3} involves 14 tables and 38 statements in the WHERE clause with 9 technical statements and 18 joins. Due to the space limit, we do not show here Q_{IN2} and Q_{IN3}. This provides clear evidence that the catalogue queries over the ontology are much simpler than the corresponding access point queries. Moreover, expressing catalogue queries over the ontology is relatively easy and could be done quite quickly by IT experts—we did it in one day. Finally, as we will see in Section 8 geologists can formulate these queries relatively quickly using OptiqueVQS.

> **Requirement 5:** *Execution time of queries from the Statoil catalogue over our OBDA deployment should be similar to the data extraction time of the corresponding access points even in the case of data federation scenario.*

As shown in sections 7.2 and 7.4, the OBDA solution at Statoil currently covers most information needs with average execution times of about two minutes for the federation setup, and under one minute for the EPDS setup. This is a huge improvement in comparison to the time needed in order to setup a new access point as described in Section 2, but it is also comparable to the time needed for end users to get information from existing access points, with parameterized queries for example. As effort to cover all the remaining information needs is ongoing, we believe that execution times for the rest will not be much different from the existing ones. In section 10 we discuss some work in progress and ideas for future improvements.

7.2. Running Queries over EPDS

We conducted three sets of experiments with the OBDA deployment using the Statoil query catalogue, and ontology and mappings relevant for the queries in the catalogue. We aim to measure the performance gain given by our optimizations, in particular, in this work we focus on the optimization to eliminate duplicates and the optimizations based on OBDA Constraints (c.f. Section 5.3).

	Information Needs in SPARQL Notation	Textual Description

Q_{IN2}:	`SELECT ?x1 WHERE {`
	`  ?x1 a Core; extractedFrom ?x2.`
	`  ?x2 a WellboreInterval;`
	`  overlapsWellboreInterval ?x3.`
	`  ?x3 a WellboreInterval; hasUnit ?x4.`
	`  ?x4 a StratigraphicUnit; name 'BRENT'.`
	`}`
		Give me all the wellbore cores extracted from a wellbore interval $x2$, such that $x2$ overlaps with another wellbore interval $x3$, whose stratigraphic unit is named "BRENT".

Q_{IN3}:	`SELECT ?x5 ?x6 ?y1 WHERE {`
	`  ?x1 a Core; extractedFrom ?x2.`
	`  ?x2 a WellboreInterval; hasCoreSample ?x5.`
	`  ?x5 a CoreSample;`
	`  hasPermeabilityMeasurement ?x6.`
	`  ?x6 a PermeabilityMeasurementResult;`
	`  valueInStandardUnit ?y1.`
	`  ?x2 overlapsWellboreInterval ?x3.`
	`  ?x3 a WellboreInterval; hasUnit ?x4.`
	`  ?x4 a StratigraphicUnit; name 'BRENT'.`
	`}`
		Give me all the wellbore cores as in Q_{IN2}, along with all the permeability measurements of their samples, and the values of these measurements in standard unit.

Figure 14: Information needs from Section 6 in English and expressed over the ontological vocabulary using SPARQL notation

Experiments Setting. In this section we present the results of testing the Ontop system over the Statoil setting introduced in the previous sections.

In our experiments, the queries were executed sequentially on a HP ProLiant server with 24 Intel Xeon CPUs (X5650 @ 2.67 GHz), 283 GB of RAM. Each query was evaluated three times and we took the average. We consider that a query times out if the execution time is greater than 20 minutes.

Experiments with different `DISTINCT` Strategies. The results are presented in the top two plots of Figure 15.

In the first set of experiments, that we refer to as *noDist*, we executed queries as they appear in the catalogue, while in the other two sets we executed these queries with the `DISTINCT` modifier in the SPARQL query, where in *dbDist* experiments `DISTINCT` is processed by the database engine, while in *obdaDist* by the OBDA engine, as discussed in Section 5.3.

In the *noDist* experiment, 17 out of 60 queries timed out. Out of the successful 43, for 2 queries the time is less than 1s, for 3 queries it is between 2m and 6m, while for most queries it is between 3s and 2m; the average time is 36.5s and the median is 12.5s. These numbers are impressive, as the SQL queries produced by the OBDA system and sent to EPDS are large: they have on average 51k characters and the largest query is 275k characters. Note that execution time here does not include the time for rewriting and unfolding of SPARQL queries, but only the actual execution of the resulting SQL queries; the maximum unfolding time is 187ms which does not add a significant time overhead to the reported query execution time. In order to understand how good the times reported in Figure 15 are for Statoil geologists, we gathered statistics on execution time for queries behind the access points available in Statoil. For the ones that correspond to the Statoil catalogue, execution time over EPDS is on average several seconds, which is comparable to the numbers in our experiments. Moreover, in general for the simplest access points the execution takes from seconds to minutes, and for the ones corresponding to complex analytical tasks it can take overnight; thus, the response time for even the slowest of our queries should not be too slow for Statoil geologists.

We also analysed why some of our queries require longer execution times than others, and why some queries failed, i.e., timed out.

A common pattern in the SQL queries that is responsible for slowing down execution time is the presence of redundant subqueries in the unfolding that are not detected by the optimisation techniques discussed in Section 5.3, and that introduce redundant answers. In fact, we found out that the answer sets for 30 out of 43 queries that did not time out contained duplicates; among them, the mean ratio of redundant answers is 51.6% while the maximum is 99.8% or 83k redundant answers.

In the *dbDist obdaDist* experiments we eliminated the redundant answers at a reasonable cost: execution times of *dbDist* and *obdaDist* are similar to *noDist*. Moreover, *obdaDist* outperforms *dbDist* in 67% cases, and in 2 cases *dbDist* execution gave a time out, while *obdaDist* never did so, which shows the benefits of our approach over *dbDist*.

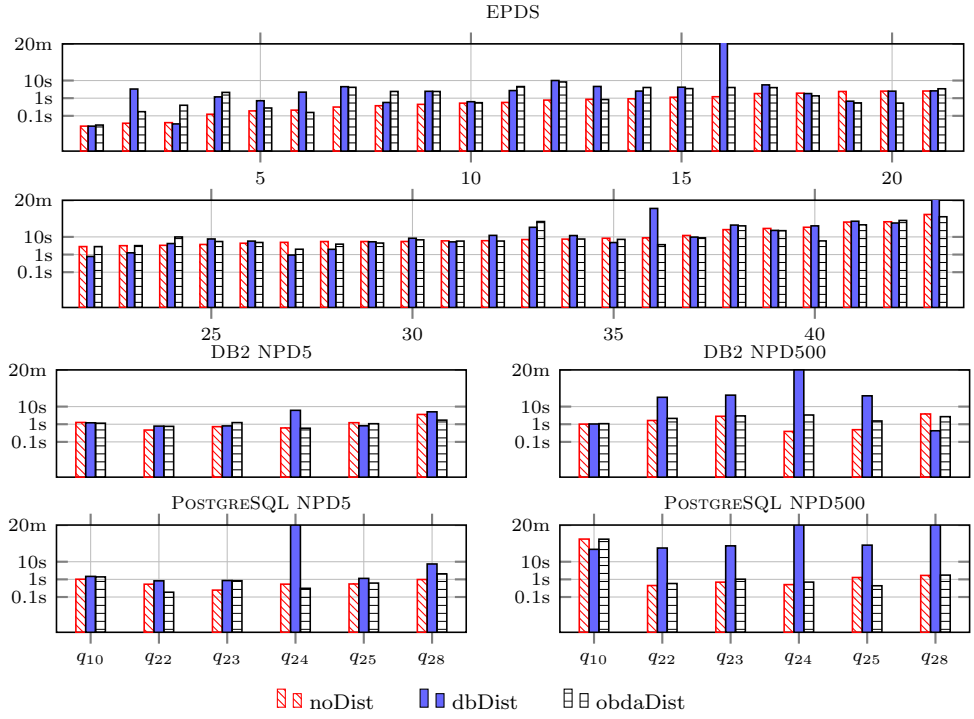


Figure 15: Experiments with different strategies for handling the `DISTINCT` modifier.

Experiments using OBDA Constraints. Observe that by a smart use of `DISTINCT` we removed the redundant answers, but *not* the redundant subqueries in the unfolding that were introducing these redundant answers. More precisely, the main reason for time-outed or slow queries were redundant self-joins and unions in the unfolded query. The problem comes from the fact that the optimization techniques discussed in Section 5.3 cannot exploit constraints that go beyond what can be explicitly declared in the database schema. In the next paragraph we show how this problem can be tackled by using OBDA constraints.

We ran the experiments with 4 exact predicates and 15 virtual functional dependencies, found with automatic tools and validated by database experts. The 60 SPARQL queries have been executed over *Ontop* with and without the optimisations for exact predicates and virtual functional dependencies.

The results are summarized in Table 6 and Figure 16. We can see that the proposed optimisations allow *Ontop* to critically reduce the query size and improve the performance of the query execution by orders of magnitude. Specifically, in Figure 16 we compare standard optimisations with and without the techniques presented in [12]. Observe that the average successful query execution time is higher with new optimisations than without because the number of successfully executed queries increases. With standard optimisations, 17 SPARQL queries time out. With both optimisations enabled, only four queries still time out, two of which do not display a change in the unfolded SQL. These two refer to a data property that is defined by an SQL query that contains a custom user-defined Java function (UDF) that *Ontop* cannot optimise. Without this UDF function, the queries can be executed in less than 2 minutes. In the other 2 queries, the tuning techniques do help to decrease their size but this did not lead to a performance improvement observable within the timeout. We report a 54% improvement of the median time, witnessing the fact that the performance improvement was observed on many queries of the catalogue.

The improvements are due to the fact that the queries sent to EPDS were further optimised due to the constraints. In fact, the number of unions and joins in the unfoldings fell of 34% and 65%, respectively. In other words, 34% and 65% of the joins and unions in the unfoldings from the previous paragraph were redundant. As a result, a total of 27 SPARQL queries get a more compact SQL translation with constraints-based optimisations enabled. The largest proportional decrease in size of the SQL query is 94%, from $171k$ chars, to $10k$. The largest absolute decrease in size of the SQL is $408k$ chars. Note that the number of unions in the SQL may decrease also only with VFD-based optimisation. Since the VFD-based optimisation removes joins, more unions may become syntactically equivalent and are therefore removed. The maximum measured decrease in execution time is on a query that times out with standard optimisations, but uses 3.7 seconds with new optimisations.

The results for this experiment were obtained without the improved handling of `DISTINCT` that was discussed in the previous paragraph. By removing redundancies, in fact, we also eliminated most of the redundant answers

Table 6: Results from the tests over EPDS.

	std. opt.	w/VFD	w/exact predicates	w/both
Number of queries timing-out	17	10	11	4
Number of fully answered queries	43	50	49	56
Avg. SQL query length (in characters)	51521	28112	32364	8954
Average unfolding time	3.929 s	3.917 s	1.142 s	0.026 s
Average total query exec. time with timeouts	376.540 s	243.935 s	267.863 s	147.248 s
Median total query exec. time with timeouts	35.241 s	11.135 s	21.602 s	14.936 s
Average successful query exec. time (without timeouts)	36.540 s	43.935 s	51.217 s	67.248 s
Median successful query exec. time (without timeouts)	12.551 s	8.277 s	12.437 s	12.955 s
Average number of unions in generated SQL	6.3	3.4	5.1	2.2
Average number of tables joined per union in generated SQL	21.0	18.2	20.0	14.2
Average total number of tables in generated SQL	132.7	62.0	102.2	31.4

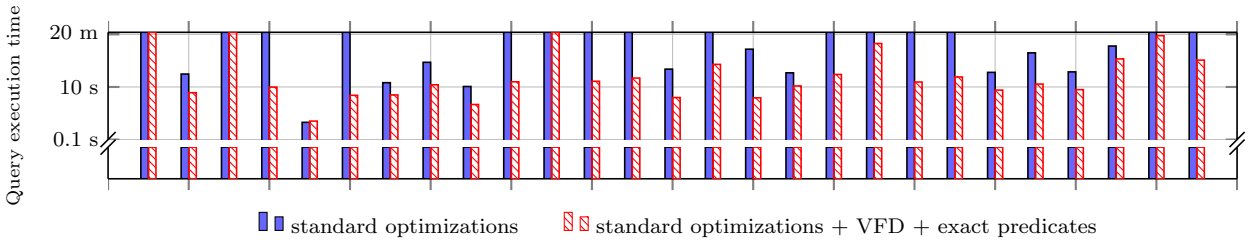


Figure 16: Comparison of query execution time with standard optimisations (Log scale).

for the queries in the catalog, making the impact of the `DISTINCT` modifier irrelevant.

For reference, Table 7 shows some more detailed statistics about executing the query catalogue at statoil over EPDS (with both optimizations enabled). This is unfortunately based on a different experiment than Table 6, with newer mappings and software, so the numbers differ somewhat, although the trends are similar.

7.3. Running Queries in Controlled Environment

We also conducted experiments in a controlled environment, which is important since EPDS is a production server, and hence the load on the server varies constantly and affects the results of experiments. For controlled experiments we used our own server and the NPD benchmark [39], a benchmark for OBDA systems. Compared to EPDS, our controlled setting contains smaller datasets: from 240MB to 25GB with +1.3B triples. We ran the experiments in an HP Proliant server with 24 Intel Xeon CPUs (144 cores@3.47GHz), 106GB of RAM and five 1TB 15K RPM HD using Ubuntu 12.04 64-bit edition. The tests were performed using DB2 and PostgreSQL as underlying database engines.

Experiments with Different `DISTINCT` Strategies. In the lower four plots in Figure 15 we present *noDist*, *dbDist*, and *obdaDist* experiments with 6 NPD queries whose performance was affected by the use of `DISTINCT`. Each experiment was conducted over two datasets: NPD5 and NPD500 corresponding to the scaling of NPD 5 and 500 times, and the results are presented separately for PostgreSQL and DB2. Note that there are 30 queries in the last version of the NPD benchmark (v1.7), while we report experiments with only the ones where the use of `DISTINCT` had an impact (positive or negative) on performance.

Our experiments show that execution of the 6 queries without `DISTINCT` is in most cases under 1s. At the same time if one uses `DISTINCT` in a naive way, by delegating it to the database engines, it is extremely detrimental to performance, leading even more queries to time out (see *dbDist* experiments). Nevertheless, with our treatment of `DISTINCT` (see *obdaDist*), in most cases the queries perform orders of magnitude better than with the naive implementation; as in the EPDS case, the optimisation seems to be more effective when the ratio of redundant answers is high, e.g., the highest ratio of redundant answers is 90% for query q_{24}, and the mean ratio for the queries is around 50%.

Importantly, compared to queries without `DISTINCT`, the overhead of removing redundancy with our techniques is small. In some cases the execution time in *obdaDist* was even better than in *noDist* because the number of the returned results shipped to the end user is significantly reduced. However, there is one interesting exceptional case where the first query in *dbDist* over PostgreSQL (and the last query over DB2) performs better than *noDist* and *obdaDist*. This phenomenon still requires further investigation.

Experiments using OBDA Constraints. In the NPD settings we did not identify constraints able to significantly improve the performance of query evaluation for the queries in the catalog. This suggests that some queries of the benchmark are inherently hard and, and that more sophisticated techniques need to be devised in order to further optimise their unfoldings.

7.4. Experiments for the Federated Setting

In this section we present the query results for running the Statoil query catalog using Exareme as the federation engine. Queries are accessing the following databases

Table 7: Statistics from executing the query catalogue over EPDS with optimizations enabled

	Mean	Median	Max	Min
SPARQL query size (num chars)	462	456	1110	89
SQL query size (num chars)	4283	2547	37788	0
Unfolding time, ms (with timeouts)	23	15	145	0
DB exec time, s (with timeouts)	89	1.3	3600	0
Total time, s (with timeouts)	147	4.2	3600	0.1
Unfolding time, ms (timeouts omitted)	20	13	131	0
DB exec time, s (timeouts omitted)	36	0.8	508	0
Total time, s (timeouts omitted)	58	2.9	1238	0.1

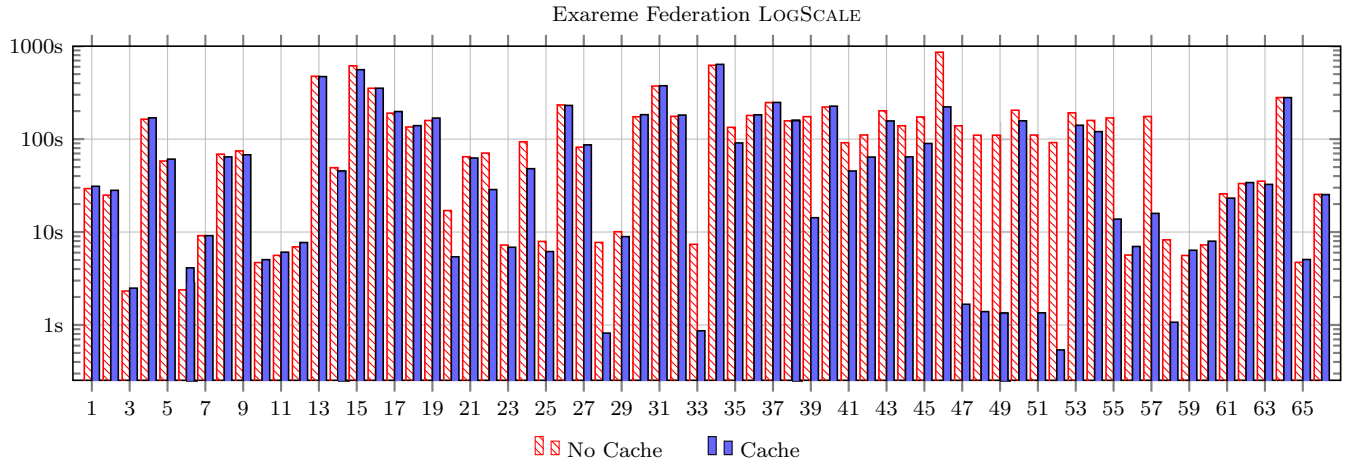


Figure 17: Execution Times for Federated Scenario (With and Without Query Cache)

described in Section 2: *(i)* EPDS *(ii)* Recall *(iii)* CoreDB *(iv)* GeoChemDB *(v)* OpenWorks *(vi)* Compass

In total, 81 queries were executed with a 1000 seconds timeout under two different setups: with and without caching of intermediate results. In the second setup, execution started with empty cache and queries were executed sequentially, according to their numbering in the query catalog. Out of all 81 queries, 66 were executed successfully within the time limit and only seven of them needed more than four minutes. The average time for successful queries in the first setting was 135.6 seconds, whereas in the second setting 101.4 seconds. All results are presented in Figure 17 (Times are in seconds). For the experiments, Exareme was installed on a cluster of eight virtual machines, running in a protected subnet at Statoil. Each virtual machine contains 8 GB of RAM and two processing cores.

8. Visual Query Formulation at Statoil

In this section we evaluate and present our query formulation approach at Statoil.

8.1. Requirements

A set of site visits and user interviews have been conducted to collect design requirements. We have conducted a literature survey [49, 63] to identify best practices and design patterns. Finally, the queries in the query catalogue are translated into actual queries (97 queries in total including alternative formulations), and then analysed [44]. Accordingly, the following three main dimensions [47] have been taken into consideration for the design and implementation of OptiqueVQS:

(i) User synopsis: The target group at Statoil are domain experts, who have extensive domain knowledge. However a majority of domain experts lack technical skills and knowledge such as on databases, programming, and query languages.

(ii) Interaction synopsis: The demand for information is frequent and information needs are varied. Moreover, domain experts often have unpredictable needs in terms of the data they are interested to extract.

(iii) Task synopsis: Domain experts' information needs are often structurally complex, which require joining multiple concepts. According to the query catalogue 73% of queries require join of more than 3 concepts and 41% require join of more than 6 concepts.

Accordingly, the following requirements have been reached:

Requirement 6: *Visual query formulation tool should support a broad range of users, task types, and interaction routines.*

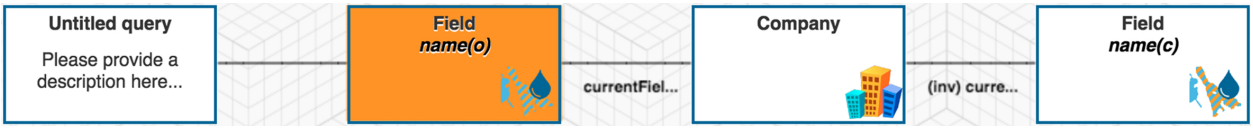


Figure 18: A linear conjunctive query formulated by domain experts in the Statoil experiment.

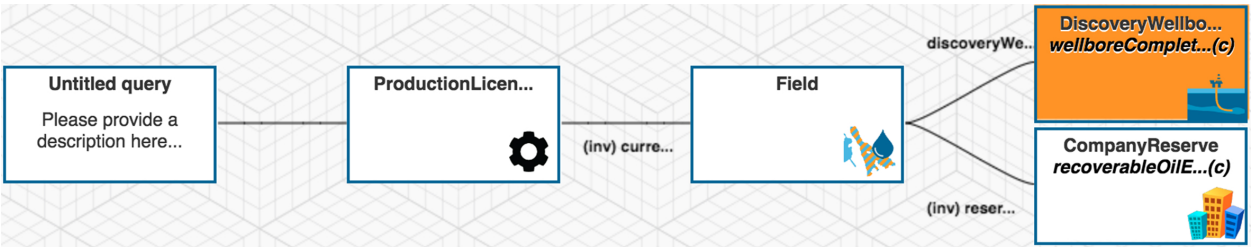


Figure 19: A tree-shaped conjunctive query formulated by domain experts in the Statoil experiment.

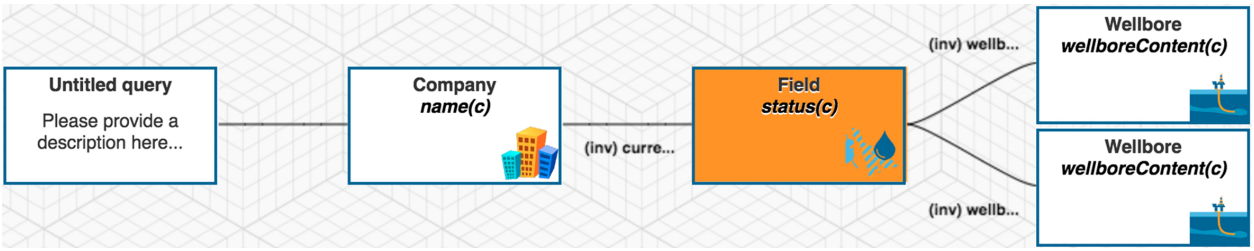


Figure 20: Another tree-shaped conjunctive query formulated by domain experts in the Statoil experiment.

Our survey revealed that a multi-paradigm design combining multiple representation and interaction paradigms is key to address a broad range of user groups having frequent, unpredicted and sophisticated information needs [47, 64]. According to a framework proposed by a Catarci et al. [47], a multi-paradigm user-interface having a diagram-based paradigm in the core supported by form-based and iconic representation paradigms is needed.

> **Requirement 7:** *Domain experts should be able to formulate frequently needed query types that are comparatively less complex.*

According to the Statoil query catalogue, 83% percent of queries are conjunctive and 70% are tree-shaped conjunctive queries. Therefore, tree-shaped conjunctive queries need to be primarily supported.

> **Requirement 8:** *Domain experts should be supported with domain specific components for immediate grasping and innate user reactions.*

Data sources at Statoil have a spatial dimension; therefore, domain experts could greatly benefit from an interaction mechanism where maps are used. This requires us to provide a domain specific map component to address spatial data sources.

OptiqueVQS meets all these requirements as it is combines multiple representation and interaction paradigms through various widgets including a map widget. Currently, 67% of the queries in the query catalogue are supported by OptiqueVQS, that is, tree-shaped conjunctive queries and queries with aggregation (i.e., excluding queries with negation).

8.2. Evaluation

A user experiment has been conducted with a bootstrapped NPD Factpages ontology to measure the efficiency and effectiveness of domain experts with OptiqueVQS [43].

In total, the ontology includes 253 concepts, 208 relationships (including inverse properties), and 233 attributes. A total of three representative participants (geologists) took part in the experiment. None of the participants had knowledge on semantic web technologies and only one of them had advanced IT skills. One participant did not use any similar query tools, while others have some familiarity. Participants completed following tasks during the experiment, given at most three attempts for each task, while being observed by an observer:

(i) List all fields.
(ii) What is the water depth of the "Snorre A" platform (facility)?
(iii) List all fields operated by "Statoil Petroleum AS" company.
(iv) List all exploration wellbores with the field they belong to and the geochronological era(s) with which they are recorded.

(v) List the fields that are currently operated by the company that operates the "Alta" field.

(vi) List the companies that are licensees in production licenses that own fields with a re-coverable oil equivalent over more than "300" in the field reserve.

(vii) List all production licenses that have a field with a wellbore completed between "1970" and "1980" and recoverable oil equivalent greater than "100" in the company reserve.

(viii) List the blocks that contain wellbores that are drilled by a company that is a field operator.

(ix) List all producing fields operated by "Statoil Petroleum AS" company that has a wellbore containing "gas" and a wellbore containing "oil".

Total of 27 tasks were completed by the participants with 84 percent correct competition rate and 69 percent first-attempt correct completion rate. In average, a task took 1.4 attempts and 243 seconds to complete. Figure 18, Figure 19, and Figure 20 represent tasks *v*, *vii* and *ix* respectively. The results suggest that domain experts could translate their information needs into queries with high effectiveness and efficiency by using OptiqueVQS. One prominent wish from the domain experts is the ability to connect concepts which are not directly linked. That would require OptiqueVQS to find and offer possible paths combining two distant concepts. Since OptiqueVQS employs a graph-based approach, such a solution is technically feasible; however, a mechanism to identify most relevant paths is required.

Finally, in Appendix A, we present a query formulated in our OBDA system using OptiqueVQS and also how this query may look like in SQL if it was formulated by an IT expert. An IT expert at Statoil estimated that he would need a full day to extract similar information needs with the existing tools, while it takes less than 10 minutes with OptiqueVQS.

9. Integration in Statoil's Infrastructure

9.1. Installation of the Platform at Statoil

Statoil has made available ten servers exclusive to the Optique project. Two of the servers are dedicated to running, respectively, a stable and a test version of the Optique platform, while the other eight servers are set up for the Exareme database system. One of the eight Exareme servers acts as master and handles the communication with the Optique platform servers and the remaining the servers in the Exareme clustre. The Exareme servers communicate with Statoil databases using the ordinary Java Database Connectivity (JDBC) interface. Statoil users interact with the platform via hypertext transfer protocol (HTTP/HTTPS) and may be authenticated through an lightweight directory access protocol (LDAP) service. An overview of the system architecture of these servers can be seen in Figure 21.

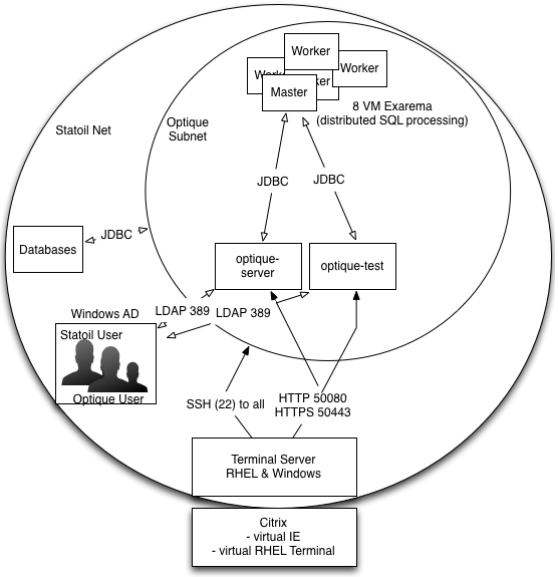


Figure 21: Optique platform server architecture at Statoil

9.2. Integration with GIS tools

The Optique platform is integrated with GIS client tools at Statoil using the Linked Open Data for Web Feature Services Adapter (LOD4WFS)[13][65]. The adapter is used to translate the result of SPARQL SELECT queries into Web Feature Services (WFS) that may be directly read by GIS tools. The adapter installed at Statoil is set up to syncronise with the queries in the collection of queries registered in the Optique platform that contain geographical data. Statoil users may hence use the OptiqueVQS to formulate queries that output geographical data, save the query to this collection, and then immediately execute the query from the GIS client tool where the results of the query will be displayed. Figure 22 shows a screenshot that illustrates what the results of querying from a GIS client tool may look like.

The list of available queries/WFS layers that may be fetched from the Optique platform are also available directly from the GIS tool. This usage pattern shows how the Optique platform can be used to efficently share queries across the enterprise.

10. Lessons Learned and Future Work

During the course of the project with Statoil we had a unique opportunity to deploy OBDA technology in a real industrial setting, to understand the limitations of the current technology, to address these limitations, and to get new ideas for further research. Our OBDA solution for Statoil shows a great potential to improve efficiency of data gathering for geologists by allowing them to *(i)* efficiently express information needs as ontological queries

[13] `https://github.com/jimjonesbr/lod4wfs`

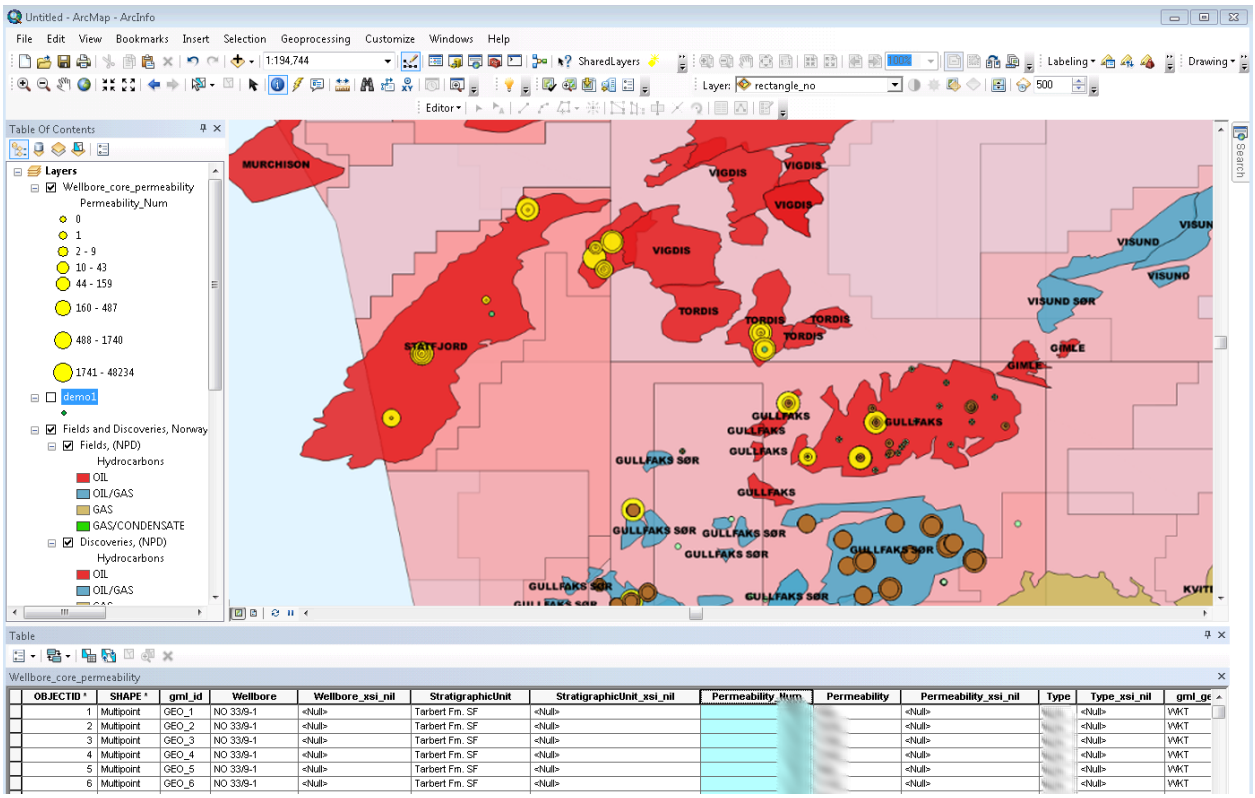


Figure 22: An image of the GIS tool ArcGIS showing how results of queries computed by the Optique platform can be integrated in commonly used client tools at Statoil. The image is manipulated to protect the privacy of Statoil's data.

and *(ii)* efficiently execute these queries over the OBDA deployment.

Regarding Item (i), our experiments show that, although we achieved our objectives, there is still room for improvement. Indeed, our OBDA deployment addresses Requirements 1–3 from Section 6: it has enough ontological terms to express queries in the query catalogue, and they all are 'connected' to the Statoil databases via mappings (thus addressing Req. 1); and the ontology underlying the deployment has a wide range of terms coming from several respected sources (thus addressing Req. 2). Furthermore, our experiments in a controlled scenario (Section 6.3.2) showed competitive results of the fully automatic deployment in a real scenario. The manually created ontology and mappings aimed at complementing the automatically created assets in order to adress Requirement 3 from Section 6. Of course, the resulting ontology by no means gives an exhaustive coverage of the oil and gas domain, but we see it as a good starting point for developing a more sophisticated ontology, and as a suitable point of entry for OBDA to Statoil's databases. Moreover, from the research point of view, we plan to develop ontology bootstrapping techniques that can be more targeted to specific scenarios, e.g., that could work not only with database schemata and data, but also with log files of queries, precomputed views and other assets specific for a given domain or scenario.

Finally, the OptiqueVQS meets all the requirements presented in Section 8. It allows a broad range of users to formulate a range of varied and sophisticated information needs into queries (Req. 6). Domain experts are able to express a significant number of queries with OptiqueVQS (i.e., 67%), that is, tree-shaped conjunctive queries with aggregation (Req. 4 and 7). Moreover, OptiqueVQS could accommodate domain specific widgets, i.e., map widget in the Statoil case, to offer more natural interaction paradigms (Req. 8). On the one hand, we plan to gradually increase the expressiveness level of OptiqueVQS by introducing new widgets or mechanisms; that could include simpler forms of negation, disjunction and cycles. On the other hand, we plan to combine alternative query formulation paradigms (i.e., multi-perspective). Currently, SPARQL mode is editable and synchronised with visual mode and similarly, for example, a natural language query formulation widget could be added and sychnonised with the visual mode.

Regarding Item (ii), the execution time of most queries from the Statoil catalogue was impressive and comparable to the performance of Statoil's existing access points, thus addressing Requirement 5 from Section 7. Our treatment of `DISTINCT` and our use of OBDA constraints led to a significant improvement in performance in comparison to standard processing in OBDA systems. Moreover, our federated query planning techniques showed that even in a federated setting we can answer queries in less then 20 minutes. Note that under 20 minutes for complex federated

queries is fast: currently many complex Statoil queries (behind access points) take (in average) overnight to be executed. We are currently investigating further optimization techniques based on a cost model exploiting statistics of concept and property definitions in the mapping and the structure of the SPARQL query. We are also studying optimization techniques dedicated for UDFs by treating them in isolation from other parts of the query. Besides, we plan to conduct experiments with Statoil queries using other available OBDA query processing engines and compare results with the outcome of experiments reported in this paper.

An important lesson we learned in Statoil was about the role of reasoning which is used by the Optique platform for query enrichment (during query processing). In particular, we did not encounter in Statoil the cases that require reasoning w.r.t existentially quantified individuals. In fact, the most useful features from Statoil ontology are class and property hierarchies, and domain and range axioms. These features correspond the core of RDFS. Non-recursive rules are in principle also useful [66], but a proper evaluation of this for Statoil is our future work.

For future work we also plan to extend and integrate our OBDA deployment in the business processes of Statoil engineers and IT personnel. Moreover, we are working on a better integration of the deployment with existing Statoil analytical and data visualisation tools, and have already integrated our solution with a geospatial data visualisation tool. An important request that we got while evaluating the OBDA deployment with Statoil engineers is to allow for users' interaction with the system: engineers would like to send their feedback to the OBDA system, e.g., by saying that some ontological terms are missing, or that some answers are wrong, and to get explanations from the system, e.g., to get provenance to query answers that include the name of the database where the answers came from as well as the exact tables that were used in the mappings. Enabling such user interaction is also an area of future work. Another important future work is to enable a form of a progress bar that shows an estimation of how much time is still needed to finish query execution. We also work on developing access control mechanisms for our OBDA deployment that can ensure that users can access only the data they are allowed to see [67, 68]. Finally, OBDA systems lack mechanisms that natively address various data quality problems, e.g., data cleaning, entity resolution, etc, which are important in data integration scenarios. This clearly poses practical challenges to adaptation of OBDA solutions in the industrial environment. We have only partially addressed this by developing an entity resolution mechanism that relies on cross-linked tables that explicitly enumerate URIs that refer to the same entity [69]. These techniques were developed during the course of our project with Statoil as a response to a request from Statoil and they were implemented in the Optique platform. At the same time major research advances are needed to develop a robust data quality framework for OBDA.

11. Related Work and Conclusion

11.1. Related Work

We now present related work on OBDA systems in general, and then on the four Optique components emphasised in the paper.

OBDA Systems. There are several academic and industrial systems for OBDA or that are very similar to OBDA in spirit. E.g., Mastro [2], morph-RDB [3], similarly to Optique, support ontology reasoning, while D2RQ [70], OntoQF [5], Virtuoso [71], Spyder [72], and Ultrawrap [6] do not support reasoning. OntoQF, Mastro, and morph-RDB do not offer ontology/mapping bootstrapping. Moreover, Mastro, morph-RDB, and OntoQF lack ontology layering and importing. Ultrawrap does not support ontology importing, but it is extendable with QODI system [73] for query-driven ontology alignment. Moreover, Ultrawrap, Mastro, and morph-RDB lack user-oriented query formulation interfaces; they provide SPARQL end-points and predefined queries. OntoQF considers ontology queries as OWL statements and has no visual query formulation support. Virtuoso, Spyder, and D2RQ have direct mapping bootstrappers and simple user interfaces for navigating the data graph. However, no advanced mapping management is supported.

Bootstrapping Ontology and Mappings. There are many approaches for bootstrapping ontologies and mappings from relational schemata; see [33, 34] for an overview. However, most of the state of the art bootstrappers fail to conform with the ontology and mapping language standards, or they do not provide profiling capabilities for the output ontology. Moreover, BootOX, according to the latest results presented within the RODI benchmark evaluation [61], outperforms most of the existing (automatic) bootstrapping systems.

For historical reasons (i.e., OWL was not yet defined), former bootstrapping systems used RDFS and F-Logic axioms (e.g., [74, 75]). Other systems have also used DLR-Lite based languages (e.g., [76]) and extensions based on SWRL (e.g., [77, 78]). Regarding mapping generation, before R2RML became a W3C recommendation, system typically relied on their own native language to define mappings (e.g., D2RQ [70], Mastro [79]). To the best of our knowledge, currently only IncMap [57], MIRROR [59], *Ontop* [80], Ultrawrap [81, 6], and AutoMap4OBDA [82] produce mappings in the R2RML language. Among the systems using OWL or OWL 2 as the ontology language, only BootOX put special attention to the target ontology expressiveness. BootOX allows to output different ontology axioms to conform to the required OWL 2 profile. Many bootstrapping systems typically use exact or min cardinality restrictions which fall outside the three OWL 2 profiles

(e.g., [83, 84]). Furthermore, other systems, like [85], produce an ontology that falls into OWL 2 Full due to the use of the *InverseFunctional* characteristic in both data and obtect properties. Finally, MIRROR, *Ontop*, Ultrawrap and AutoMap4OBDA are conformant to the OWL 2 QL, but they do not support profiling to the other sublanguages of OWL 2. As BootOX, systems like Automapper [77], Relational.OWL [86] and ROSEX [87] complement the automatically generated ontology with links to domain ontologies. However, none of these systems apply logic-based techniques to assess the consequences of such links to domain ontologies. Special mention require the approaches in [88, 89]. These approaches use (semi-automatic) ontology learning techniques to exploit the data and discover interesting patterns that can be included to enrich the ontology.

Query Rewriting and Optimisation. The first query rewriting algorithm is PerfectRef [90]. This algorithm, implemented in QuOnto, often returned hundreds of thousands of CQs (even for simple ontologies and mappings). To deal with the issue, PerfectRef was later extended by a Semantic Query Optimization (SQO) component, which removes redundant CQs and eliminates redundant self-joins using database integrity constraints (foreign and primary keys) [91]. To further improve the performance, the technique of T-mappings was introduced in [92] and adopted in Ontop [40]. The tree-witness query rewriting algorithm [93] replaced PerfectRef to drastically reduce the size of rewritings and further take advantage of T-mappings.

Federated Query Processing. Federated query processing presents several challenges that have been studied in the database literature. Long and unpredictable data transfer times, unavailability of resources, difficulty to obtain statistics and cost estimations are some of the main difficulties. Examples of solutions proposed include caching of temporary results [94] and adaptive query processing that involves interleaved planning and execution [95]. Regarding query planning, the main idea is to extend an existing method, in order to incorporate decisions as to when it is better to push processing to external sources. The Garlic system [96] uses *grammar-like rules*, extending the method of [97], whereas [98] is based on the Volcano *transformation-based* optimizer [99]. Recent approaches [100, 101, 102] rely on specialized engines that act as endpoints and the mediator can be thought of as a *polystore* or *multistore* that takes advantage of these engines in order to achieve efficient processing depending on the specific kind of underlying data. For example, an endpoint can be a relational database optimized for OLTP tasks, a column store optimized for analytic workloads, a document store etc.

Visual Query Formulation Interfaces over Ontologies. There are many ontology based systems that use visual representations for query formulation, e.g., OntoVQL [103], GRQL [104], TAMBIS [105]; see [106, 63] for detailed related work. To the best of our knowledge, none of the systems that we know about can be used off-the-shelf for query formulation scenarios in Statoil due to specific technical or usability requirements of our industrial use-case [44]. In particular, most of the systems we saw are better suited for exploration of ontologies rather than underlying data, e.g., in contrast to OptiqueVQS they do not offer sufficient treatment of data properties and data values. From the usability perspective, we did not find a system offering a sufficient balance of the view on semantic details of specific query fragments and the query overview [63]. We note that our evaluation of OptiqueVQS in Statoil was a *feasibility* study. That is, our goal was to show that OBDA allowed Statoil's domain experts (without special knowledge of formal query languages and experience in query formulation) to construct (most of) their typical queries which they were not able to construct before (see Requirements 6-8 for further details). Thus, our goal in this paper was to show the benefits of OBDA for domain experts rather than to evaluate OptiqueVQS against other tools and methods.

11.2. Conclusion

In this paper we presented a description of a data analyses routine at Statoil and challenges with access to business critical data required for these analyses. These challenges delay the analytical tasks significantly, thus addressing them is of a high importance for Statoil. Additionally, the challenges are representative for large data intensive industries and a good solution would be beneficial not only for Statoil, but for a wide range of enterprises. We believe that OBDA technology is a promising way to address the challenges while to the best of our knowledge existing off-the-shelf OBDA solutions can not be directly applied to do the job. Thus, we developed an OBDA solution that is capable of dealing with the challenges and is equipped with a deployment module BootOX for semi-automatic creation of ontologies and mappings, a query processing module *Ontop* that ensures efficient OBDA query processing, a federated query execution module Exareme that provides highly optimised query plans, and a query formulation module OptiqueVQS that allows end-users to construct relatively complex queries over ontologies without a prior knowledge of Semantic Technologies. We deployed our solution at Statoil and evaluated it against the requirements we derived from interviews of Statoil engineers, analyses of their business processes, and in particular on a catalogue of typical information needs of Statoil geologists. Although our evaluation was conducted in a control environment and Optique has not been used by Statoil engineers in production, the evaluation results are promising and indicate a great potential of using OBDA in Statoil. We believe that our work opens new avenues for research in the areas of semantic access and semantic integration of federated and distributed relational databases in

large enterprises, since it shows practical benefits of such approach and exhibits important practical challenges that should be addressed in order to ensure success of such technology.

Acknowledgements. This work was partially funded by the EU project Optique (FP7-ICT-318338), the EPSRC projects MaSI^3, DBOnto, ED^3, the BIGMED project (IKT 259055), and the SIRIUS Centre for Scalable Data Access (Research Council of Norway, project no.: 237889).

References

[1] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. on Data Semantics 10 (2008) 133–173.

[2] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D. F. Savo, The MASTRO system for ontology-based data access, Semantic Web J. 2 (1) (2011) 43–53.

[3] F. Priyatna, Ó. Corcho, J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proc. of WWW, 2014, pp. 479–490.

[4] C. Bizer, A. Seaborne, D2RQ - treating non-RDF databases as virtual RDF graphs, in: Proc. of ISWC Posters & Demos Track, 2004.

[5] K. Munir, M. Odeh, R. McClatchey, Ontology-driven relational query formulation using the semantic and assertional capabilities of OWL-DL, Knowl.-Based Syst. 35 (2012) 144–159.

[6] J. F. Sequeda, D. P. Miranker, Ultrawrap: SPARQL execution on relational data, J. of Web Semantics 22 (0) (2013) 19 – 39.

[7] M. Rodriguez-Muro, D. Calvanese, High performance query answering over DL-Lite ontologies, in: Proc. of KR, 2012, pp. 308–318.

[8] M. Giese, D. Calvanese, P. Haase, I. Horrocks, Y. Ioannidis, H. Kllapi, M. Koubarakis, M. Lenzerini, R. Möller, O. Özçep, M. Rodriguez Muro, R. Rosati, R. Schlatte, M. Schmidt, A. Soylu, A. Waaler, Scalable end-user access to big data, in: Big Data Computing, Chapman and Hall/CRC, 2013.

[9] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jimenez-Ruiz, D. Lanti, M. Rezk, G. Xiao, O. Ozcep, R. Rosati, Optique—zooming in on big data access, Computer 48 (3) (2015) 60–67. doi:10.1109/MC.2015.82.

[10] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, I. Horrocks, Ontology based access to exploration data at statoil, in: ISWC, 2015, pp. 93–112.

[11] H. Kllapi, E. Sitaridi, M. M. Tsangaris, Y. Ioannidis, Schedule optimization for data processing flows on the cloud, in: Proc. of ACM SIGMOD, ACM, 2011, pp. 289–300.

[12] D. Hovland, D. Lanti, M. Rezk, G. Xiao, OBDA constraints for effective query answering, in: Proc. of RuleML, Vol. 9718, Springer, 2016, pp. 269–286.

[13] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, I. Horrocks, Experiencing OptiqueVQS – a multi-paradigm and ontology-based visual query system for end-users, Universal Access in the Information Society 15 (1) (2016) 129–152.

[14] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. S. veland, E. Thorstensen, J. Mora, BootOX: Practical mapping of RDBs to OWL 2, in: Proc. of ISWC, 2015, pp. 113–132.

[15] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, W. May, D. Ritze, M. G. Skjæveland, A. Solimando, E. Kharlamov, RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration, in: Proc. of ESWC, 2015, pp. 21–37.

[16] J. Crompton, Keynote talk at the W3C workshop on sem. web in oil & gas industry (2008).

[17] M. G. Skjæveland, E. H. Lian, I. Horrocks, Publishing the norwegian petroleum directorate's factpages as semantic web data, in: Proc. of ISWC, 2013, pp. 162–177.

[18] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: The making of a web ontology language, J. of Web Semantics 1 (1) (2003) 7–26.

[19] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[20] A. S. Sidhu, T. S. Dillon, E. Chang, B. S. Sidhu, Protein ontology development using OWL, in: Proc. of OWLED, 2005.

[21] C. Golbreich, S. Zhang, O. Bodenreider, The foundational model of anatomy in OWL: Experience and perspectives, J. of Web Semantics 4 (3) (2006) 181–195.

[22] J. Goodwin, Experiences of using OWL at the ordnance survey, in: Proc. of OWLED, 2005.

[23] California Inst. of Technology, Semantic web for earth and environmental terminology, `http://sweet.jpl.nasa.gov/` (2006).

[24] S. Derriere, A. Richard, A. Preite-Martinez, An ontology of astronomical object types for the virtual observatory, in: Special Session 3 of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities, 2006.

[25] D. Soergel, B. Lauser, A. C. Liang, F. Fisseha, J. Keizer, S. Katz, Reengineering thesauri for new applications: The AGROVOC example, J. Digit. Inf. 4 (4) (2004) 1–23.

[26] L. Lacy, G. Aviles, K. Fraser, W. Gerber, A. M. Mulvehill, R. Gaskill, Experiences using OWL in military applications, in: Proc. of OWLED, 2005.

[27] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, JAR 39 (3) (2007) 385–429.

[28] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 Query Language, Tech. rep., W3C, `http://www.w3.org/TR/sparql11-query` (2013).

[29] P. Haase, C. Hütter, M. Schmidt, A. Schwarte, The information workbench as a self-service platform for linked data applications, in: Proc. of WWW, 2012.

[30] E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, S. Watson, How semantic technologies can enhance data access at siemens energy, in: Proc. of ISWC, 2014, pp. 601–619.

[31] L. Feigenbaum, G. T. Williams, SPARQL 1.1 protocol, W3C recommendation, W3C, `https://www.w3.org/TR/sparql11-protocol/` (2013).

[32] C. Pinkel, C. Binnig, P. Haase, C. Martin, K. Sengupta, J. Trame, How to best find a partner? an evaluation of editing approaches to construct R2RML mappings, in: Proc. of ESWC, 2014, pp. 675–690.

[33] J. Sequeda, S. H. Tirmizi, Ó. Corcho, D. P. Miranker, Survey of directly mapping SQL databases to the semantic web, Knowledge Eng. Review 26 (4) (2011) 445–486.

[34] D.-E. Spanos, P. Stavrou, N. Mitrou, Bringing relational databases into the semantic web: A survey, Semantic Web J. 3 (2) (2012) 169–209.

[35] P. Shvaiko, J. Euzenat, Ontology matching: State of the art and future challenges, IEEE Trans. Knowl. Data Eng. 25 (1) (2013) 158–176.

[36] E. Jimenez-Ruiz, B. Cuenca Grau, Y. Zhou, I. Horrocks, Large-scale interactive ontology matching: Algorithms and implementation, in: Proc. of ECAI, 2012, pp. 444–449.

[37] A. Solimando, E. Jiménez-Ruiz, G. Guerrini, Detecting and correcting conservativity principle violations in ontology-to-ontology mappings, in: Proc. of ISWC, Vol. 8797, 2014, pp. 1–16.

[38] A. Solimando, E. Jimenez-Ruiz, G. Guerrini, Minimizing conservativity violations in ontology alignments: Algorithms and evaluation, Knowledge and Information Systems (2016) (in

press).
[39] D. Lanti, M. Rezk, G. Xiao, D. Calvanese, The NPD benchmark: Reality check for OBDA systems, in: Proc. of EDBT, 2015, pp. 617–628.
[40] R. Kontchakov, M. Rezk, M. Rodriguez-Muro, G. Xiao, M. Zakharyaschev, Answering SPARQL queries over databases under OWL 2 QL entailment regime, in: ISWC, Vol. 8796, Springer, 2014, pp. 552–567.
[41] P. Roy, S. Seshadri, S. Sudarshan, S. Bhobe, Efficient and extensible algorithms for multi query optimization, in: Proc. of ACM SIGMOD, ACM, 2000, pp. 249–260.
[42] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, OptiqueVQS: towards an ontology-based visual query system for big data, in: MEDES, 2013, pp. 119–126.
[43] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, I. Horrocks, Ontology-based visual query formulation: An industry experience, in: Proceedings of the 11th International Symposium on Visual Computing (ISVC 2015), Vol. 9474 of LNCS, Springer, 2015, pp. 842–854.
[44] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez Ruiz, M. Giese, M. G. Skjæveland, D. Hovland, R. Schlatte, S. Brandt, H. Lie, I. Horrocks, OptiqueVQS: a visual query system over ontologies for industry, Semantic Web J. (under review).
[45] A. H. M. Ter Hofstede, H. A. Proper, T. P. Van Der Weide, Query formulation as an information retrieval problem, Computer Journal 39 (4) (1996) 255–274.
[46] A. Soylu, F. Modritscher, P. De Causmaecker, Ubiquitous web navigation through harvesting embedded semantic data: A mobile scenario, Integrated Computer-Aided Engineering 19 (1) (2012) 93–109.
[47] T. Catarci, M. F. Costabile, S. Levialdi, C. Batini, Visual query systems for databases: A survey, J. of Visual Languages and Computing 8 (2) (1997) 215–260. doi:10.1006/jvlc.1997.0037.
[48] A. Soylu, F. Moedritscher, F. Wild, P. De Causmaecker, P. Desmet, Mashups by orchestration and widget-based personal environments: Key challenges, solution strategies, and an application, Program: Electronic Library and Information Systems 46 (4) (2012) 383–428.
[49] A. Soylu, M. Giese, Qualifying ontology-based visual query formulation, in: Proceedings of the 11th International Conference Flexible Query Answering Systems (FQAS 2015), Vol. 400 of Advances in Intelligent Systems and Computing, Springer, 2015, pp. 243–255.
[50] M. Arenas, B. Cuenca Grau, E. Kharlamov, Š. Marciuška, D. Zheleznyakov, Faceted search over RDF-based knowledge graphs, J. of Web Semantics 37-38 (2016) 55–74.
[51] M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Faceted search over ontology-enhanced RDF data, in: CIKM, 2014, pp. 939–948.
[52] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: An OWL 2 reasoner, JAR 53 (3) (2014) 245–269.
[53] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, E. Kharlamov, O. Ozcep, C. Neuenstadt, S. Brandt, Querying Industrial Stream-Temporal Data: an Ontology-based Visual Approach, Journal of Ambient Intelligence and Smart Environments 9 (1) (2017) 77–95.
[54] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, Towards exploiting query history for adaptive ontology-based visual query formulation, in: MTSR, 2014, pp. 107–119.
[55] M. Console, V. Santarelli, D. Savo, Efficient approximation in DL-Lite of OWL 2 ontologies, in: DL, Vol. 1014 of CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2013, pp. 132–143.
[56] G. Stoilos, G. B. Stamou, S. D. Kollias, A string metric for ontology alignment, in: Proc. of ISWC, 2005, pp. 624–637.
[57] C. Pinkel, C. Binnig, E. Kharlamov, P. Haase, IncMap: Pay as you go matching of relational schemata to owl ontologies, in: OM, 2013.
[58] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web 8 (3) (2017) 471–487.
[59] L. F. de Medeiros, F. Priyatna, O. Corcho, MIRROR: Automatic R2RML mapping generation from relational databases, in: ICWE, 2015, pp. 326–343.
[60] M. G. Skjæveland, E. Lian, I. Horrocks, Publishing the NPD factpages as semantic web data, in: Proc. of ISWC, 2013, pp. 162–177.
[61] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, M. G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, I. Horrocks, RODI: Benchmarking relational-to-ontology mapping generation quality, Semantic Web J. (2016) (accepted for publication).
[62] D. A. et al., Schema and ontology matching with COMA++, in: Proc. of ACM SIGMOD, 2005, pp. 906–908.
[63] A. Soylu, M. Giese, E. Kharlamov, E. Jimenez-Ruiz, D. Zheleznyakov, I. Horrocks, Ontology-based end-user visual query formulation: Why, what, who, how, and which?, Universal Access in the Information Society (2016) (in press).
[64] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, E. Giannopoulou, Ontology visualization methods - a survey, ACM Computing Surveys 39 (4) (2007) 10:1–10:43.
[65] J. Jones, W. Kuhn, C. Keßler, S. Scheider, Making the web of data available via web feature services, in: Connecting a Digital Europe Through Location and Place - International AGILE'2014 Conference, Castellon, Spain, 13-16 June, 2014, 2014, pp. 341–361.
[66] G. Xiao, M. Rezk, M. Rodriguez-Muro, D. Calvanese, Rules and ontology based data access, in: M.-L. Mugnier, R. Kontchakov (Eds.), Proc. 8th International Conference on Web Reasoning and Rule Systems (RR 2014), LNCS, Springer, 2014.
[67] B. Cuenca Grau, E. Kharlamov, E. V. Kostylev, D. Zheleznyakov, Controlled query evaluation for datalog and OWL 2 profile ontologies, in: Proc. of IJCAI, 2015, pp. 2883–2889.
[68] B. Cuenca Grau, E. Kharlamov, E. V. Kostylev, D. Zheleznyakov, Controlled query evaluation over OWL 2 RL ontologies, in: Proc. of ISWC, 2013, pp. 49–65.
[69] D. Calvanese, M. Giese, D. Hovland, M. Rezk, Ontology-based integration of cross-linked datasets, in: ISWC, 2015, pp. 199–216.
[70] C. Bizer, A. Seaborne, D2RQ—treating non-RDF databases as virtual RDF graphs, in: Proc. of ISWC, 2004.
[71] virtuoso.
URL `http://virtuoso.openlinksw.com/`
[72] Spyder.
URL `http://www.revelytix.com/content/spyder`
[73] A. Tian, J. Sequeda, D. P. Miranker, QODI: Query as context in automatic data integration, in: Proc. of ISWC, 2013, pp. 624–639.
[74] L. Stojanovic, N. Stojanovic, R. Volz, Migrating Data-Intensive Web Sites into the Semantic Web, in: SAC, 2002, pp. 1100–1107.
[75] I. Astrova, Reverse Engineering of Relational Databases to Ontologies, in: ESWS, 2004, pp. 327–341.
[76] L. Lubyte, S. Tessaris, Automatic extraction of ontologies wrapping relational data sources, in: DEXA, 2009, pp. 128–142.
[77] M. Fisher, M. Dean, G. Joiner, Use of OWL and SWRL for Semantic Relational Database Translation, in: OWLED, 2008.
[78] D. V. Levshin, Mapping Relational Databases to the Semantic Web with Original Meaning, Int. J. Software and Informatics 4 (1) (2010) 23–37.
[79] C. Civili, M. Console, G. De Giacomo, D. Lembo, M. Lenzerini, L. Lepore, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, V. Santarelli, D. F. Savo, MASTRO STUDIO: Managing Ontology-Based Data Access Applications, PVLDB 6 (12)

(2013) 1314–1317.
[80] Ontop.
URL http://ontop.inf.unibz.it/
[81] J. Sequeda, M. Arenas, D. P. Miranker, On directly mapping relational databases to RDF and OWL, in: Proc. of WWW, 2012, pp. 649–658.
[82] Á. Sicilia, G. Nemirovski, AutoMap4OBDA: Automated generation of R2RML mappings for OBDA, in: Proc. of EKAW, 2016, pp. 577–592.
[83] N. Alalwan, H. Zedan, F. Siewe, Generating OWL Ontology for Database Integration, in: SEMAPRO, 2009, pp. 22–31. doi:10.1109/SEMAPRO.2009.21.
[84] S. H. Tirmizi, J. Sequeda, D. P. Miranker, Translating SQL Applications to the Semantic Web, in: DEXA, 2008, pp. 450–464.
[85] I. Astrova, Rules for Mapping SQL Relational Databases to OWL Ontologies, in: MTSR, 2007, pp. 415–424.
[86] C. P. de Laborda, S. Conrad, Database to Semantic Web Mapping Using RDF Query Languages, in: ER, 2006, pp. 241–254.
[87] C. Curino, G. Orsi, E. Panigati, L. Tanca, Accessing and Documenting Relational Databases through OWL Ontologies, in: FQAS, 2009, pp. 431–442.
[88] F. Cerbah, N. Lammari, Perspectives in Ontology Learning, AKA / IOS Press. Serie, 2012, Ch. Ontology Learning from Databases: Some Efficient Methods to Discover Semantic Patterns in Data, pp. 1–30.
[89] M. G. Skjæveland, M. Giese, D. Hovland, E. H. Lian, A. Waaler, Engineering Ontology-Based Access to Real-World Data Sources, J. Web Semantics.
[90] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, J. of Automated Reasoning 39 (3) (2007) 385–429, doi:10.1007/s10817-007-9078-x.
[91] M. Rodriguez-Muro, Tools and techniques for ontology based data access in lightweight description logics, Ph.D. thesis, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano (2010).
[92] M. Rodríguez-Muro, D. Calvanese, Dependencies: Making ontology based data access work in practice, in: AMW, Vol. 749, 2011.
[93] S. Kikot, R. Kontchakov, M. Zakharyaschev, Conjunctive query answering with OWL 2 QL, in: KR, 2012, pp. 275–285.
[94] S. Adali, K. S. Candan, Y. Papakonstantinou, V. Subrahmanian, Query caching and optimization in distributed mediator systems, in: SIGMOD Record, ACM, 1996, pp. 137–146.
[95] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, D. S. Weld, An adaptive query execution system for data integration, in: SIGMOD Record, ACM, 1999, pp. 299–310.
[96] L. M. Haas, D. Kossmann, E. L. Wimmers, J. Yang, Optimizing queries across diverse data sources, in: Proc. of VLDB, 1997, pp. 276–285.
[97] G. M. Lohman, Grammar-like functional rules for representing query optimization alternatives, SIGMOD Record 17 (3) (1988) 18–27.
[98] J. L. Ambite, C. A. Knoblock, Flexible and scalable cost-based query planning in mediators: A transformational approach, Artificial Intelligence 118 (1) (2000) 115–161.
[99] G. Graefe, W. J. McKenna, The volcano optimizer generator: Extensibility and efficient search, in: Data Engineering, 1993. Proceedings. Ninth International Conference on, IEEE, 1993, pp. 209–218.
[100] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, I. Manolescu, Invisible glue: scalable self-tuning multi-stores, in: Conference on Innovative Data Systems Research (CIDR), 2015.
[101] D. J. DeWitt, A. Halverson, R. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, J. Gramling, Split query processing in polybase, in: Proc. of ACM SIGMOD, ACM, 2013, pp. 1255–1266.
[102] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, S. Zdonik, The bigdawg polystore system, SIGMOD Record 44 (2) (2015) 11–16.
[103] A. Fadhil, V. Haarslev, OntoVQL: A graphical query language for OWL ontologies, in: DL, 2007.
[104] N. Athanasis, V. Christophides, D. Kotzinos, Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL), in: Proc. of ISWC, 2004, pp. 486–501.
[105] P. G. Baker, A. Brass, S. Bechhofer, C. A. Goble, N. W. Paton, R. Stevens, TAMBIS: Transparent access to multiple bioinformatics information sources, in: ISMB, 1998, pp. 25–34.
[106] O. Noppens, Ontology visualization and analysis (2013).
[107] M. Horridge, N. Drummond, J. Goodwin, A. L. Rector, R. Stevens, H. Wang, The Manchester OWL Syntax, in: OWLED, 2006.

Appendix A. Example End-to-End Statoil Data Access Task

To show the Optique solution from end to end, we start with the following information need:

> Give me formation pressure measurements together with the zone they were taken in, and any log curves also covering that zone.

The query as expressed in the VQS is shown in Fig. A.23 and in SPARQL syntax is:

```
SELECT ?w ?wellbore ?wi ?fp ?fp_depth
   ?formation_pressure ?unit ?logged ?curve
WHERE {
?w  a :Wellbore;
    :name ?wellbore.
    :hasWellboreInterval ?wi;
    :hasFormationPressure ?fp.
?fp :hasDepthMeasurement ?fp_depth;
    :valueInStandardUnit ?formation_pressure.
?fp_depth :inWellboreInterval ?wi.
?wi :hasUnit ?unit;
    :overlapsWellboreInterval ?logged.
?logged  :hasLogCurve ?curve.
}
```

The resulting SQL generated by Ontop and sent to Exareme is in Table A.8, and the execution plan used by Exareme to execute the query is in Fig. A.24. The SQL query consists of two union subqueries. The numbers inside the brackets in the resulting plan denote the subquery or subqueries that make use of each node. For example, the part of the plan whose nodes are marked by $[1, 2]$ denote the common subexpression of these two subqueries. Different colors for each base table node denote the database from which it comes from: yellow for EPDS, green for Recall and red for OpenWorks. Some of the mappings used in the unfolding from datalog to SQL are shown in Table A.9.

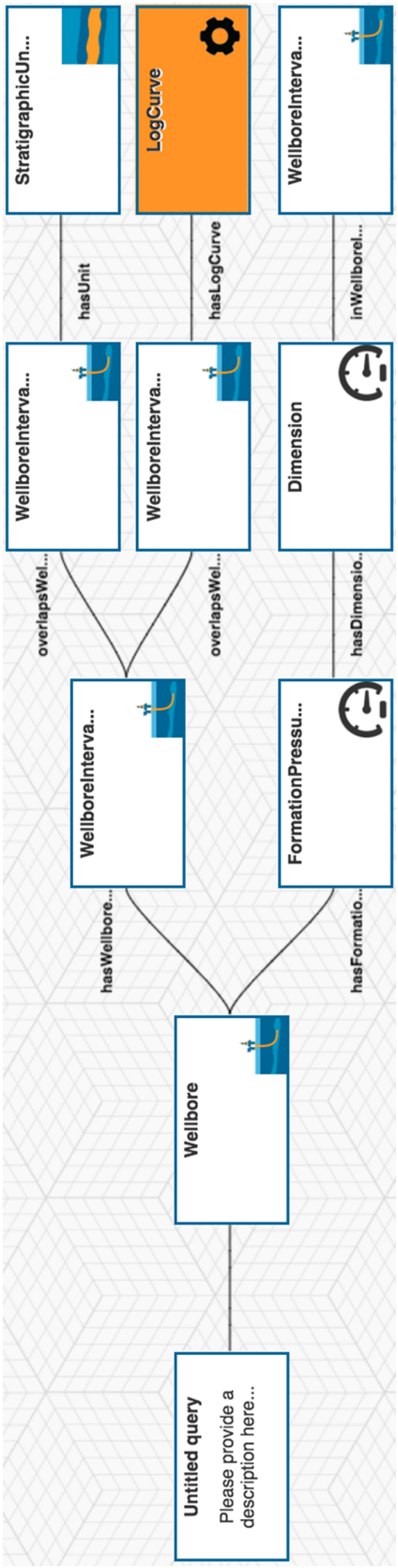


Figure A.23: The end-to-end example query expressed in the OptiqueVQS.

Table A.8: SQL Generated in the end-to-end example

```
SELECT
   'slegge:Wellbore-' || QVIEW2."WELLBORE_ID" AS "wellbore",
   'slegge:StratigraphicZone-' || QVIEW2."WELLBORE_ID" || '-' || QVIEW3."STRAT_COLUMN_IDENTIFIER" || '-' ||
   QVIEW3."STRAT_INTERP_VERSION" || '-' || QVIEW3."STRAT_ZONE_IDENTIFIER" AS "wi",
   QVIEW5."P_PRESSURE_S" AS "fp",
   QVIEW5."DATA_VALUE"  AS "formation_pressure",
   'slegge:PressureMeasuredDepth-' || QVIEW5."P_PRESSURE_S" AS "fp_depth",
   'slegge:StratigraphicUnit-' || QVIEW3."STRAT_COLUMN_IDENTIFIER" || '-' || QVIEW3."STRAT_UNIT_IDENTIFIER" AS "unit",
   'recall:LoggedInterval/recall/' || QVIEW9."MIRROR_ID" AS "logged",
   'recall:LogCurve/recall/' || QVIEW9."MIRROR_ID"  AS "curve"
 FROM
"adp"."SLEGGE1_WELL" QVIEW1, "adp"."SLEGGE1_WELLBORE" QVIEW2,
"adp"."SLEGGE_STRATIGRAPHIC_ZONE" QVIEW3, "adp"."SLEGGE1_ACTIVITY" QVIEW4,
"adp"."SLEGGE1_P_PRESSURE" QVIEW5, "adp"."SLEGGE1_ACTIVITY_CLASS" QVIEW6,
"adp"."SLEGGE1_P_LOCATION_1D" QVIEW7, "adp"."SLEGGE_PICKED_STRATIGRAPHIC_ZONES" QVIEW8,
"adp"."RECALL_LOG_VW" QVIEW9, "adp"."wellbore_sameas_recall_slegge" QVIEW10
WHERE
(QVIEW2."R_EXISTENCE_KD_NM" = 'actual') AND (QVIEW1."WELL_S" = QVIEW2."WELL_S") AND
(QVIEW2."SECURITY_LABELS_S" = 0) AND QVIEW2."WELLBORE_ID" IS NOT NULL AND
(QVIEW2."WELLBORE_ID" = QVIEW3."WELLBORE") AND QVIEW3."STRAT_INTERP_VERSION" IS NOT NULL AND
QVIEW3."STRAT_ZONE_IDENTIFIER" IS NOT NULL AND QVIEW3."STRAT_COLUMN_IDENTIFIER" IS NOT NULL AND
(QVIEW2."WELLBORE_S" = QVIEW4."FACILITY_S") AND (QVIEW5."DATA_VALUE_U" = 'bar') AND
(QVIEW4."ACTIVITY_S" = QVIEW5."ACTIVITY_S") AND (QVIEW4."KIND_S" = QVIEW6."ACTIVITY_CLASS_S") AND
(QVIEW6."CLSN_CLS_NAME" = 'formation pressure depth data') AND
QVIEW5."P_PRESSURE_S" IS NOT NULL AND
(QVIEW5."DATA_VALUE" > 0) AND QVIEW5."DATA_VALUE" IS NOT NULL AND
(QVIEW4."ACTIVITY_S" = QVIEW7."ACTIVITY_S") AND (QVIEW2."WELLBORE_ID" = QVIEW8."WELLBORE") AND
(QVIEW3."STRAT_COLUMN_IDENTIFIER" = QVIEW8."STRAT_COLUMN_IDENTIFIER") AND
(QVIEW3."STRAT_INTERP_VERSION" = QVIEW8."STRAT_INTERP_VERSION") AND
(QVIEW3."STRAT_ZONE_IDENTIFIER" = QVIEW8."STRAT_ZONE_IDENTIFIER") AND
(QVIEW7."DATA_VALUE_1_OU" = QVIEW8."STRAT_ZONE_DEPTH_UOM") AND
((QVIEW3."STRAT_ZONE_IDENTIFIER" IS NOT NULL
AND (QVIEW8."STRAT_ZONE_ENTRY_MD" <= QVIEW7."DATA_VALUE_1_O"))
AND (QVIEW8."STRAT_ZONE_EXIT_MD" >= QVIEW7."DATA_VALUE_1_O")) AND
QVIEW3."STRAT_UNIT_IDENTIFIER" IS NOT NULL AND
(QVIEW9."WELL_NAME" = QVIEW10."recall_id") AND
(QVIEW2."WELLBORE_S" = QVIEW10."slegge_id") AND
((QVIEW9."TOP_DEPTH" <= QVIEW3."STRAT_ZONE_EXIT_MD") AND (QVIEW9."BOTTOM_DEPTH" >= QVIEW3."STRAT_ZONE_ENTRY_MD")) AND
QVIEW9."MIRROR_ID" IS NOT NULL
UNION ALL
SELECT
   'slegge:Wellbore-' || QVIEW2."WELLBORE_ID") AS "w",
   QVIEW2."WELLBORE_ID" AS "wellbore",
   'slegge:StratigraphicZone-' || QVIEW2."WELLBORE_ID" || '-' || QVIEW3."STRAT_COLUMN_IDENTIFIER" || '-' ||
   QVIEW3."STRAT_INTERP_VERSION"  || '-' || QVIEW3."STRAT_ZONE_IDENTIFIER" AS "wi",
   'slegge:FormationPressure-' || QVIEW5."P_PRESSURE_S" AS "fp",
   QVIEW5."DATA_VALUE"  AS "formation_pressure",
   'slegge:PressureMeasuredDepth-' || QVIEW5."P_PRESSURE_S" AS "fp_depth",
   'slegge:StratigraphicUnit-' || QVIEW3."STRAT_COLUMN_IDENTIFIER" || '-' || QVIEW3."STRAT_UNIT_IDENTIFIER" AS "unit",
   'openworks:LoggedInterval_openworksSV4NSEA_' || QVIEW9."LOG_CURVE_ID"  AS "logged",
   'openworks:LogCurve_openworksSV4NSEA_' || QVIEW9."LOG_CURVE_ID" AS "curve"
 FROM
"adp"."SLEGGE1_WELL" QVIEW1, "adp"."SLEGGE1_WELLBORE" QVIEW2,
"adp"."SLEGGE_STRATIGRAPHIC_ZONE" QVIEW3, "adp"."SLEGGE1_ACTIVITY" QVIEW4,
"adp"."SLEGGE1_P_PRESSURE" QVIEW5, "adp"."SLEGGE1_ACTIVITY_CLASS" QVIEW6,
"adp"."SLEGGE1_P_LOCATION_1D" QVIEW7, "adp"."SLEGGE_PICKED_STRATIGRAPHIC_ZONES" QVIEW8,
"adp"."OPENWORKS_SV4NSEA_LOG_CURVE_HEADER" QVIEW9, "adp"."OPENWORKS_SV4NSEA_WELL_MASTER" QVIEW10,
"adp"."OPENWORKS_SV4NSEA_LOG_CURVE_HEADER" QVIEW11
WHERE
(QVIEW2."R_EXISTENCE_KD_NM" = 'actual') AND (QVIEW1."WELL_S" = QVIEW2."WELL_S") AND
(QVIEW2."SECURITY_LABELS_S" = 0) AND QVIEW2."WELLBORE_ID" IS NOT NULL AND
(QVIEW2."WELLBORE_ID" = QVIEW3."WELLBORE") AND QVIEW3."STRAT_INTERP_VERSION" IS NOT NULL AND
QVIEW3."STRAT_ZONE_IDENTIFIER" IS NOT NULL AND QVIEW3."STRAT_COLUMN_IDENTIFIER" IS NOT NULL AND
(QVIEW2."WELLBORE_S" = QVIEW4."FACILITY_S") AND (QVIEW5."DATA_VALUE_U" = 'bar') AND
(QVIEW4."ACTIVITY_S" = QVIEW5."ACTIVITY_S") AND (QVIEW4."KIND_S" = QVIEW6."ACTIVITY_CLASS_S") AND
(QVIEW6."CLSN_CLS_NAME" = 'formation pressure depth data') AND
QVIEW5."P_PRESSURE_S" IS NOT NULL AND (QVIEW5."DATA_VALUE" > 0) AND
QVIEW5."DATA_VALUE" IS NOT NULL AND (QVIEW4."ACTIVITY_S" = QVIEW7."ACTIVITY_S") AND
(QVIEW2."WELLBORE_ID" = QVIEW8."WELLBORE") AND
(QVIEW3."STRAT_COLUMN_IDENTIFIER" = QVIEW8."STRAT_COLUMN_IDENTIFIER") AND
(QVIEW3."STRAT_INTERP_VERSION" = QVIEW8."STRAT_INTERP_VERSION") AND
(QVIEW3."STRAT_ZONE_IDENTIFIER" = QVIEW8."STRAT_ZONE_IDENTIFIER") AND
(QVIEW7."DATA_VALUE_1_OU" = QVIEW8."STRAT_ZONE_DEPTH_UOM") AND
((QVIEW3."STRAT_ZONE_IDENTIFIER" IS NOT NULL AND (QVIEW8."STRAT_ZONE_ENTRY_MD" <= QVIEW7."DATA_VALUE_1_O"))
AND (QVIEW8."STRAT_ZONE_EXIT_MD" >= QVIEW7."DATA_VALUE_1_O")) AND QVIEW3."STRAT_UNIT_IDENTIFIER" IS NOT NULL AND
(QVIEW9."WELL_ID" = QVIEW10."WELL_ID") AND (QVIEW2."WELLBORE_ID" = QVIEW10."WELL_UWI") AND
((QVIEW9."TOP_DEPTH" <= QVIEW3."STRAT_ZONE_EXIT_MD") AND (QVIEW9."BASE_DEPTH" >= QVIEW3."STRAT_ZONE_ENTRY_MD")) AND
QVIEW9."LOG_CURVE_ID" IS NOT NULL AND
(QVIEW9."LOG_CURVE_ID" = QVIEW11."LOG_CURVE_ID")
```

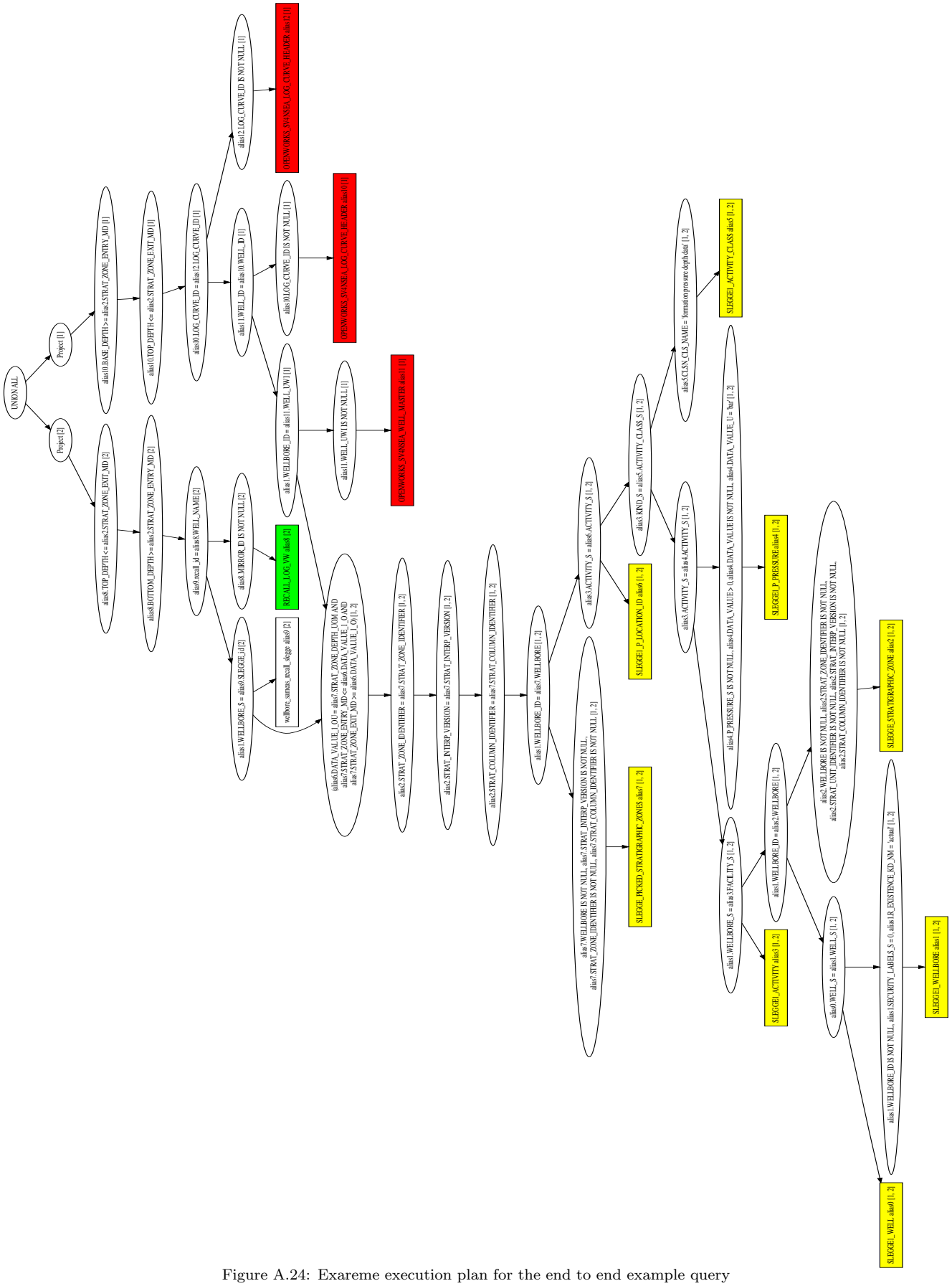


Figure A.24: Exareme execution plan for the end to end example query

Table A.9: Some mappings used in the unfolding of the end-to-end Example query in the Ontop native mapping syntax (see https://github.com/ontop/ontop/wiki/ontopOBDAModel)

```
target slegge:Well-{WID} :hasWellbore slegge:Wellbore-{WELLBORE} .
source select WLB.WELL_LEGAL_NAME as WELLBORE, W.WELL_LEGAL_NAME as WID
       from compass_CD_WELLBORE WLB, compass_CD_WELL W where W.WELL_ID = WLB.WELL_ID


target slegge:Wellbore-{WELLBORE_IDENT} :hasFormationPressure
       slegge:FormationPressure-{PRESSURE_KEY} .
source select WLB.WELLBORE_ID as WELLBORE_IDENT, PRESSURE.P_PRESSURE_S as PRESSURE_KEY
       from slegge1_P_PRESSURE PRESSURE, slegge1_ACTIVITY FP_DEPTH_DATA,
       slegge1_WELLBORE WLB, slegge1_ACTIVITY_CLASS FORM_PRESSURE_CLASS where
       PRESSURE.ACTIVITY_S = FP_DEPTH_DATA.ACTIVITY_S and
       FP_DEPTH_DATA.FACILITY_S = WLB.WELLBORE_S and FP_DEPTH_DATA.KIND_S =
       FORM_PRESSURE_CLASS.ACTIVITY_CLASS_S and
       FORM_PRESSURE_CLASS.CLSN_CLS_NAME = 'formation pressure depth data'


target slegge:Wellbore-{WLB_ID} :hasWellboreInterval
       slegge:StratigraphicZone-{WELLBORE}-{SCI}-{SIV}-{SZI}.
source select W.WELLBORE_ID as WLB_ID, WELLBORE, STRAT_COLUMN_IDENTIFIER as SCI,
       STRAT_INTERP_VERSION as SIV, STRAT_ZONE_IDENTIFIER as SZI
       from slegge_STRATIGRAPHIC_ZONE Z, slegge1_WELLBORE W
       WHERE Z.WELLBORE = W.WELLBORE_ID and W.SECURITY_LABELS_S = 0


target recall:LoggedInterval/recall/{MIRROR_ID} :hasLogCurve recall:LogCurve/recall/{MIRROR_ID};
       :hasTopDepthMeasurement recall:LoggedIntervalTop/recall/{MIRROR_ID};
       :hasBottomDepthMeasurement recall:LoggedIntervalBottom/recall/{MIRROR_ID} .
source select MIRROR_ID from recallTest_LOG_VW


target recall:LoggedInterval/recall/{INT_ID} :overlapsWellboreInterval
       slegge:StratigraphicZone-{BRONNAVN}-{SCI}-{SIV}-{SZI}.
source SELECT LOG.MIRROR_ID as INT_ID, WELLBORE_INTERVAL.WELLBORE as
       BRONNAVN, WELLBORE_INTERVAL.STRAT_ZONE_IDENTIFIER as SZI,
       WELLBORE_INTERVAL.STRAT_INTERP_VERSION as SIV,
       WELLBORE_INTERVAL.STRAT_COLUMN_IDENTIFIER as SCI FROM
       recallTest_LOG_VW LOG, slegge_STRATIGRAPHIC_ZONE WELLBORE_INTERVAL,
       slegge1_WELLBORE WLB, wellbore_sameas_recall_slegge sameas WHERE
       WELLBORE_INTERVAL.WELLBORE = WLB.WELLBORE_ID AND WLB.WELLBORE_S =
       sameas.slegge_id AND LOG.WELL_NAME = sameas.recall_id AND
       LOG.TOP_DEPTH <= WELLBORE_INTERVAL.STRAT_ZONE_EXIT_MD AND
       LOG.BOTTOM_DEPTH >= WELLBORE_INTERVAL.STRAT_ZONE_ENTRY_MD


target slegge:StratigraphicZone-{WLB_ID}-{SCI}-{SIV}-{SZI} a :StratigraphicZone ;
       :name {STRAT_ZONE_ID}^^xsd:string; :hasUnit slegge:StratigraphicUnit-{SCI}-{STRAT_UNIT}.
source select WELLBORE as WLB_ID, STRAT_COLUMN_IDENTIFIER AS SCI, STRAT_INTERP_VERSION AS SIV,
       STRAT_ZONE_IDENTIFIER as SZI, STRAT_UNIT_IDENTIFIER as STRAT_UNIT
       from slegge_STRATIGRAPHIC_ZONE
```

Appendix B. OWL 2 Construction Patterns

BootOX [14] creates ontology vocabulary and a set of OWL 2 axioms from a relational database. Table B.10 presents the encoding of relational database features as OWL 2 axioms followed by BootOX. The encoding, in general, does not lead to only one option, but several possible OWL 2 axioms. One could opt for adding all the axioms associated with a feature or only a selection of them depending on the intended purpose (e.g., Optique requires an OWL 2 QL ontology).

When not stated the contrary in Table B.10, a class C_T, an object property P_f, a data property R_a and a datatype d_t represent the ontology encoding of a table T, a foreign key fk, a data attribute a, and an SQL type t, respectively.

Table B.10: Encoding of relational database features as OWL 2 axioms. OWL 2 axioms are expressed in the Manchester OWL Syntax [107]. * Enumeration with only one literal

#	RDB feature	Ontology feature	OWL 2 axiom	OWL 2 profile		
				QL	RL	EL
(1)	**Non-binary Relation / Table** T	A class C_T for the non-binary table	Class: C_T	✓	✓	✓
(2)	**Binary Relation or Many-to-Many Table** referencing tables T_1 and T_2	A property P (and its inverse Q) associated to the classes C_{T_1} and C_{T_2} with local and/or global constraints	ObjectProperty: P	✓	✓	✓
			Q InverseOf: P	✓	✓	-
			P Domain: C_{T_1}	✓	✓	✓
			P Range: C_{T_2}	✓	✓	✓
			C_{T_1} SubClassOf: P some C_{T_2}	✓	-	✓
			C_{T_1} SubClassOf: P only C_{T_2}	-	✓	-
(3)	**Data attribute** in table T of (sql) type t	A property R_a associated to the class C_T and datatype d_t with local and/or global constraints.	DataProperty: R_a	✓	✓	✓
			R_a Domain: C_T	✓	✓	✓
			R_a Range: d_t	✓	✓	✓
			C_T SubClassOf: R_a some d_t	✓	-	✓
			C_T SubClassOf: R_a only d_t	-	✓	-
(4)	**Foreign Key** in table T_1, referencing T_2, **no intersection with** or **strict subset of** primary key	A property P_f associated to the classes C_{T_1} and C_{T_2} with local and/or global constraints	ObjectProperty: P_f	✓	✓	✓
			P_f Domain: C_{T_1}	✓	✓	✓
			P_f Range: C_{T_2}	✓	✓	✓
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
			C_{T_1} SubClassOf: P_f only C_{T_2}	-	✓	-
(5)	**Foreign Key is the primary key** in T_1, ref. T_2	Class C_{T_1} is subsumed by class C_{T_2}	C_{T_1} SubClassOf: C_{T_2}	✓	✓	✓
(6)	**Foreign Key** in table T **referencing the same table**	A property P_f associated to class C_T with a self-restriction. The property P_f may also be declared with several characteristics	C_T SubClassOf: P_f some C_T	✓	-	✓
			C_T SubClassOf: P_f some Self	-	-	✓
			P_f Characteristics: Transitive	-	✓	✓
			P_f Characteristics: Symmetric	✓	✓	-
			P_f Characteristics: Reflexive	✓	-	✓
(7)	**Primary Key** or **Unique constraint** in table T_1 on a **foreign key** fk referencing T_2	Key axiom for class C_{T_1} and property P_f. P_f is associated to (local and/or global) cardinality constraints	C_{T_1} HasKey: P_f	-	✓	✓
			P_f Characteristics: Functional	-	✓	-
			P_f Characteristics: InverseFunctional	-	✓	-
			C_{T_1} SubClassOf: P_f exactly 1 C_{T_2}	-	-	-
			C_{T_1} SubClassOf: P_f max 1 C_{T_2}	-	✓	-
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
(8)	**Primary Key** or **Unique constraint** on a **data attribute** a of (sql) type t in table T	Key axiom for class C_T and data property R_a. R_a is associated to (local and/or global) cardinality constraints.	C_T HasKey: R_a	-	✓	✓
			R_a Characteristics: Functional	-	✓	✓
			C_T SubClassOf: R_a exactly 1 d_t	-	-	-
			C_T SubClassOf: R_a max 1 d_t	-	✓	-
			C_T SubClassOf: R_a some d_t	✓	-	✓
(9)	**Composed Primary Key** in table T	Key axiom for the class C_T and the data and object properties involved in primary key	C_T HasKey: $R_1 \dots R_n$ $P_1 \dots P_n$	-	✓	✓
(10)	**Not Null Constraint** on a **data attribute** in T, type t	Existential or cardinality restriction over R_a in C_T	C_T SubClassOf: R_a min 1 d_t	-	-	-
			C_T SubClassOf: R_a some d_t	✓	-	✓
(11)	**Not Null Constraint** on a **foreign key** in T_1, ref. T_2	Existential or cardinality restriction over P_f in C_T	C_{T_1} SubClassOf: P_f min 1 C_{T_2}	-	-	-
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
(12)	**Check Constraint** on data attribute a of type t in table T **listing posible values** $v_1 \dots v_n$ (n $\geq$ 1)	Enumeration of literals: in a restriction in class C_T and/or as a range of Ra. Alternatively, one could create subclasses for each of the values	C_T SubClassOf: R_a some $\{v_1 \dots\}$	-	-	✓*
			C_T SubClassOf: R_a only $\{v_1 \dots\}$	-	-	-
			C_T SubClassOf: R_a value v_1	-	-	✓
			R_a Range: $\{v_1 \dots\}$	-	-	✓*
			C_{Tv_i} SubClassOf: C_T	✓	✓	✓
(13)	**Check Constraint** on attrib. a in table T **restricting numerical range** of t	Datatype restriction: in a class restriction in C_T and/or as a range of Ra	C_T SubClassOf: R_a some $d_t[> x]$	-	-	-
			C_T SubClassOf: R_a only $d_t[> x]$	-	-	-
			R_a Range: $d_t[> x]$	-	-	-
(14)	**Several data attributes** $a_1 \dots a_n$ **in different tables** $T_1 \dots T_n$ **with the same name and type** t	Group properties $R_1 \dots R_n$ under new superproperty R_a or merge $R_1 \dots R_n$ into new property R_a	R_i SubPropertyOf: R_a	✓	✓	✓
			R_a Domain: C_{T_1} *or* $\dots$ *or* C_{T_n}	-	-	-
			C_{T_i} SubClassOf: R_a some d_t	✓	-	✓
			C_{T_i} SubClassOf: R_a only d_t	-	✓	-
(15)	T_1 **and** T_2 **not involved in a inheritance relationship**	Class C_{T_1} is disjoint with class C_{T_2}	C_{T_1} DisjointWith: C_{T_2}	✓	✓	✓