Publishing the Norwegian Petroleum Directorate's FactPages as Semantic Web Data

Martin G. Skjæveland[1], Espen H. Lian[1], and Ian Horrocks[2,1]

[1] Department of Informatics, University of Oslo, Norway
{martige,elian}@ifi.uio.no
[2] Department of Computer Science, University of Oxford, UK
ian.horrocks@cs.ox.ac.uk

Abstract. This paper motivates, documents and evaluates the process and results of converting the Norwegian Petroleum Directorate's FactPages, a well-known and diverse set of tabular data, but with little and incomplete schema information, stepwise into other representations where in each step more semantics is added to the dataset. The different representations we consider are a regular relational database, a linked open data dataset, and an ontology. For each conversion step we explain and discuss necessary design choices which are due to the specific shape of the dataset, but also those due to the characteristics and idiosyncrasies of the representation formats. We additionally evaluate the output, performance and cost of querying the different formats using questions provided by users of the FactPages.

1 Introduction

The Norwegian Petroleum Directorate (NPD) is a governmental specialist directorate and administrative body which reports to the Ministry of Petroleum and Energy. NPD's main objective is to "contribute to creating the greatest possible values for society from the oil and gas activities by means of prudent resource management based on safety, emergency preparedness and safeguarding of the external environment" [17]. This objective is met, they state, by performing four functions, of which we highlight one: "The NPD has a national responsibility for data from the Norwegian continental shelf. The NPD's data, overview and analyses constitute a crucial factual basis on which the activities are founded" [17]. One of the datasets that the NPD manages is the *NPD FactPages* [5], or *FactPages* for short. The FactPages contain data about petroleum activities on the Norwegian continental shelf (NCS), e.g., about companies that own or operate petroleum fields, results of tests taken during drilling, geographical data for physical installations and the areas of fields and seismic surveys, transfers of shares of fields between companies, and production results measured in volumes of petroleum. Some of this data dates back as far as the start of oil production on the NCS, in the early 1970s. The data in the FactPages is collected from companies that operate on the NCS; the NPD is entitled to all information the companies have regarding their activity in Norway, and may formulate detailed

routines for reporting this information. This information forms the basis for the authorities' planning of future activity and their judgement of existing activity. Additionally, an important purpose of the FactPages is to secure efficient sharing of information between the companies, and to provide sufficient information to the public. Hence, the FactPages act as a national reference data library for information regarding the NCS, both for historical data and for data about current activities such as ongoing seismic surveys and active exploration wells. Needless to say, the FactPages are an important and heavily used dataset documenting what is by far most important industry in Norway.

The FactPages are made available as a web application serving a set of HTML pages in a "factsheet format", meaning in this case that data elements are structured into predefined categories and subcategories, and data for each individual data element is displayed as a list of simple tables according to its category. Most of the background data for these reports is also available for download in bulk as tabular data in CSV, Excel and XML format. The FactPages are updated daily and their content can be freely used as long as the source is properly referenced [17].

We argue that the NPD does not fully achieve its main objective with its management of the FactPages. There is a lot of unrealised potential in the way the FactPages are published, with a resulting loss of value to the operating companies on the NCS, the general public and thus also to the authorities and the NPD themselves. Our biggest concerns are that, firstly, the FactPages data is available only in the presentational form that is currently implemented by the publication system, i.e., data cannot be categorised or joined in arbitrary ways by the user; this means that a lot of information from the FactPages is practically unavailable since collecting, joining and aggregating the necessary data would require considerable manual effort. Secondly, the current representation of the FactPages makes it impossible to properly integrate it with other datasets or vocabularies, such as relevant industry standards, and to make simple references to individual data items. This is a problematic situation for a reference data library.

This paper presents a case study of using semantic technologies to address the above-mentioned problems—specifically, of converting the FactPages into a semantically enriched RDF dataset, and supporting SPARQL query answering via an RDF triple store with different levels of OWL reasoning enabled. The transformation process can be split into three stages:

1. Convert and represent all bulk tabular data as a regular relational database. The biggest effort in this step has been to create a relational schema for the database. This step gives us the possibility to answer queries over the dataset using a standard relational database system, and provides a baseline for a comparison of different levels of semantic querying.
2. Transform and export the relational data into RDF format. In this step considerable work has been put into cleaning the data and representing the dataset as is appropriate for semantic web data. After this step we are able to serve the dataset through a SPARQL endpoint and publish it according Linked

Open Data (LOD) principles [2, 9], with query answers being computed by an RDF triple store.
3. Build a suitable OWL ontology. An initial ontology was produced using the same data used to create the relational schema in the first step combined with the RDF transformation in step two; this was then manually extended by adding axioms that capture information that is not available to the automatic translation. After this step we have a dataset that can be more easily integrated with other existing datasets and vocabularies, and we can use a reasoning enabled triple store to enrich query answers with implicit information.

We tested query answering at each stage using a set of questions provided by regular users of the FactPages. In each case we consider the ease with which the questions can be formulated as queries, the performance of the query answering system and the quality of the resulting answers.

Although our exposition is driven by the NPD use case, our results and the lessons learnt are of a general character, and are likely to be applicable to many other similar cases.

2 Background and Motivation

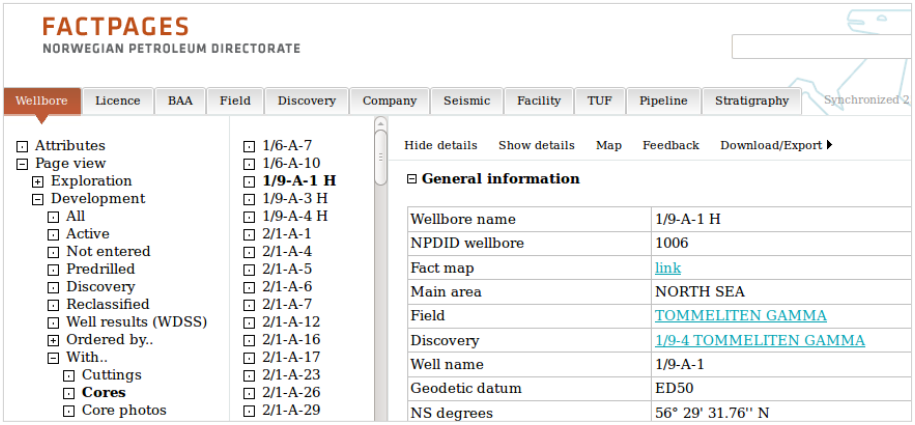
The FactPages are made available as a web application powered by Microsoft's SQL Server Reporting Services. The web pages are browsed by selecting one out of 11 main categories.[3] These offer a hierarchical menu of different *views*,[4] each of which may contain sub categories for investigating a particular feature of the chosen main category, e.g., a sub category under *Wellbore* is *Core Photos*, which contains photographs of core samples taken during drilling of the wellbore. Upon reaching a leaf node in this menu, a list of members which fit the selection appears, and selecting a member displays information about it in the most prominent part of the application. This information is called a *report*. See Fig. 1 for a screenshot of the FactPages application.

What we are missing in the FactPages application is the ability to pose arbitrary queries to the dataset, allowing users to combine and aggregate information differently than what is made available by the application; the description given above explains the only way to access data with the web application. For example, it is practically impossible to find the wellbores for which there are core samples from a specific lithostratigraphic unit,[5] a question which is relevant for geologists

[3] The categories are *Wellbore*, *Licence*, *BAA* (Business Arrangement Area), *Field*, *Discovery*, *Company*, *Seismic* [Surveys], *Facility*, *TUF* (Transportation and Utilisation Facility), *Pipeline*, and *Stratigraphy*.

[4] Possible values are *Page view*, *Statistics*, *Table view*, *Attributes*, and *Geography*. *Table view* contains the data which is available for download in tabular format. *Attributes* gives a description and an SQL datatype for some of the columns found in the tabular formats.

[5] Roughly speaking, a geological strata or layer.



A screenshot of the FactPages showing information about the wellbore 1/9-A-1 H with core samples (the core samples fall outside the page on the screenshot). The information is reached by selecting *Wellbore*, *Page view*, *Development*, *With..*, *Cores*, and *1/9-A-1 H*. On the top of the page one can see the main entry points of the FactPages, the panes labelled with *Wellbore* on the left to *Stratigraphy* on the right. In the left margin the subcategories are listed. The next list shows the members in the selection. The largest part of the window contains the report for the selected instance.

Fig. 1. A screenshot of the NPD FactPages.

who want to explore a particular area on the NCS. Another disadvantage of the application is its poor use of URIs and links. Individual data items do not have a URI, and although reports do have a URI, these are tightly coupled to the implementing system and it is explicitly stated that they may change without notice.[6] So, e.g., to identify a core sample a user would need to invent an identifier local to the report where information about it is found, making sharing of this data with others more difficult and error-prone. Moreover, there are few links between reports: in cases where a report mentions an asset for which there exists a report, a direct link to this report is not always provided.

The main motivation for semantically enriching the FactPages according to the three-step plan is the added value it gives the dataset in terms of availability and usability, but also a general increase in data quality. The advantages of publishing data according to linked open data and semantic web principles is assumed to be known to the reader, so we only briefly present the most important and relevant ideas:

Global identifiers URIs provide a schema for assigning identifiers which are likely to be globally unique. This is crucial for data integration. Publishing

[6] See `http://factpages.npd.no/factpages/Parameters.aspx`. The URI for the page shown in Fig. 1 is `http://factpages.npd.no/FactPages/Default.aspx?nav1=wellbore&nav2=PageView|Development|All&nav3=1006`.

according to LOD principles combines the identifier of a data item with the address to use to retrieve information about the item.

Generic data model RDF [10] provides a simple, uniform and universal data model, simplifying data exchange and consumption. RDF's triple structure, and predicate especially, make the data "schemaless", and to some extent self describing—particularly when the vocabulary used in the dataset is further described in an ontology. Again, this simplifies data exchange and integration, but also querying, since the dataset is independent of a particular schema or database system. Adding new data and extending the dataset with new vocabularies is easy due to the simple and self describing nature of RDF.

Query interface The SPARQL standards [6] define both a powerful query language for RDF data, and how to provide services which expose such data to the Internet through SPARQL endpoints using existing infrastructure. This makes it possible to safely allow arbitrary queries over the dataset.

Semantics OWL [12] provides a language for formally describing the semantics of the vocabulary used in the dataset, unlike a database schema which is geared towards describing *how* data is stored. An ontology can be used to introduce and explicate the semantics of domain-centric vocabulary, making the data more accessible to a wider range of end-users, and also acting as machine readable documentation for the dataset. An ontology additionally provides a sophisticated means for integrating datasets, and the semantics of the ontology can be used by a reasoner to infer new facts from the dataset, and to explain logical consequences and query answers.

There are several reasons for converting the tabular data from the FactPages via a relational database rather than directly into semantic web data. Firstly, W3C recommendations provide standardised specifications for converting from relational databases to semantic web data [4, 1], and the tools available for performing such a conversion are greater in number, and in many cases more mature and more actively maintained and developed (see, e.g., [15, 16]) than their spreadsheet conversion counterparts (such as, e.g., [11, 7]). Secondly, having the dataset represented as a relational database allows us to compare how well suited SQL and SPARQL are for expressing the questions posed by our users and the performance of relational and RDF stores in answering the resulting queries. Thirdly, converting from a relational database also makes our comparison more relevant to those (many) cases where data is kept in such a system; in fact there are good reasons to believe that the master data for the FactPages is kept in a relational database. Finally, a relational database is a very useful tool for handling and investigating tabular data, and this has been very convenient when implementing the conversion procedure. Moreover, as we shall see, the apparatus for importing the tabular files into a relational database is similar to the process that we use to design an appropriate semantic web representation of the relational database. Thus the overhead of having an extra conversion step does not exceed the added value of having a relational database available.

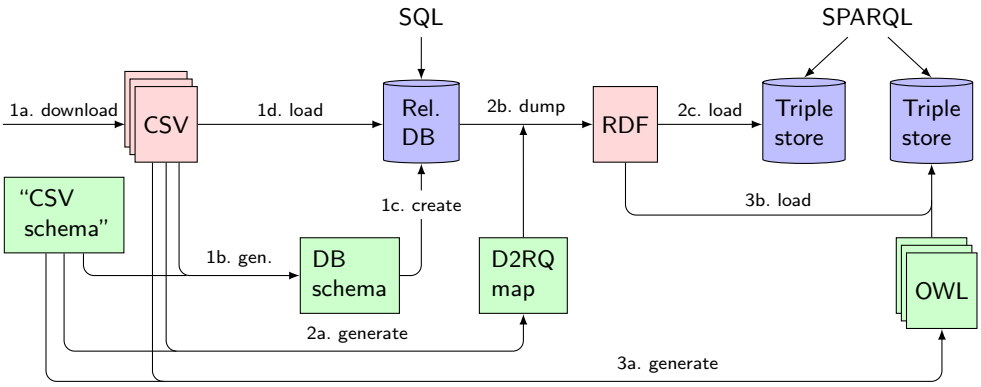
We are also interested in investigating the effect that ontologies have in enriching the dataset, and on query performance. Therefore, we have built a suitable

OWL 2 ontology, mainly derived from the relational to RDF transformation, but also enhanced with hand-crafted axioms capturing, e.g., class hierarchies. We then evaluated our standard SPARQL queries over the ontology-extended dataset using triple stores that support reasoning with RDFS [8] and with the OWL 2 profiles OWL 2 QL, OWL 2 EL and OWL 2 RL [13]. The goal is to determine if the added semantics adds anything to our query answer, and if so at what computational cost.

Finally, it is worth mentioning that publishing the data in this way not only makes it easier to use, and hence more valuable, but it can also contribute to improving the quality of the original data: converting the dataset into a semantic web framework can reveal potential errors and suggest good information representation principles, and increased use of the dataset may help to identify any additional errors.

3 Methodology

In this section we explain our procedure for transforming the FactPages CSV files into semantic web data: first, loading the files into a relational database, then exporting the contents of the database to RDF and loading the RDF into a triple store, and finally, building an ontology from the available schema data. An overview of the process is found in Fig. 2.



1. (a) Download CSV files from FactPages website.
   (b) Generate relational database schema using data from CSV files, attribute descriptions, and additional specifications (the "CSV schema").
(c, d) Create database and load CSV files into database.
2. (a) Generate D2RQ map from CSV files.
(b, c) Dump database to RDF file using D2RQ map, load RDF file into triple store.
3. (a) Generate ontology from CSV files and schema. Manually add additional axioms.
   (b) Load RDF dump and ontology files in to triple store setup for reasoning.

Fig. 2. Schematic overview of the conversion process.

3.1 Step 1: Loading CSV files into a relational database

In order to load the CSV files into a relational database, we need to build a database schema. A database schema usually specifies the database's tables, columns, column datatypes, primary and foreign keys, and indexes. As table names and column names we use the CSV filename and the column names that are given in the first line of the CSV files. The *Attribute* view under each main category in the FactPages application lists many of columns used in the CSV files, giving each a label, description and SQL datatype. However, not all columns occurring in the CSV files are found under *Attributes*, and *Attributes* contains many columns which are not used in the CSV files. In cases where an SQL datatype is missing for a column, we determine the datatype by examining the data. Primary and foreign keys are identified by investigating temporary versions of the database created without keys. Usually the column names and column descriptions, when available, provide hints as to how to build proper keys. For the FactPages data, column names containing `idNPD` are strongly indicative of a key. For tables where no key exists, we add a numeric auto increment column and set this as the primary key. For each column, we also record whether or not null values occur, which is again determined by a simple inspection of the data. The specification for how to build the database schema is kept in what we call the "CSV schema". This is a collection of CSV files which is used in all of the three conversion steps to determine how to produce the outcome of the conversion. For the first step, these files are used to record column datatypes and comments, primary and foreign keys, and whether columns may contain `NULL` values or not.

The relational database we are using is a MySQL database.[7] Prior to loading the FactPages CSV files into the database, they need to be cleaned. During cleaning all files are converted to UTF-8 character set, problematic linebreaks are removed, and date and time values are converted into the correct format for MySQL. Values which we believe are meant to be null values, e.g, `NA`, `not available`, `n/a`, `N/A`, `""` and `"NULL"`, are all set to the database null value `NULL`, and values which we guess indicate an unknown value, e.g., `?`, `"?"` and `Not known`, are changed to the string value `UNKNOWN`. Null valued date columns are set to the maximum value `9999-12-31`. We also correct minor variations in spelling when a column name is used in multiple CSV files, e.g., setting `wlbnpdid_wellbore` and `wlbnpdidwellbore` to the more commonly used pattern `wlbNpdidWellbore`. The dataset contains extensive geographical data, described using the well-known text (WKT) markup language;[8] this needs to be taken special care of—using the MySQL function `GeomFromText`—in order to be correctly imported into the database. The database is created and the CSV files loaded into the database with a single SQL script; an excerpt which creates and adds foreign keys for the table `licence_petreg_licence_oper` and loads the CSV file `licence_petreg_licence_oper.csv` into this table is found in Fig. 3.

It is worth noting that we have created the database with the aim of having a faithful representation of the original FactPages CSV files and to have a practical

[7] `http://www.mysql.com/`, version 5.1.35

[8] `http://en.wikipedia.org/wiki/Well-known_text`

```
CREATE TABLE licence_petreg_licence_oper (
    ptlName VARCHAR(40) NOT NULL COMMENT "Tillatelse",
    cmpLongName VARCHAR(200) NOT NULL COMMENT "Company name",
    prlNpdidLicence INTEGER NOT NULL COMMENT "NPDID production licence",
    cmpNpdidCompany INTEGER NOT NULL COMMENT "NPDID company",
    ptlOperDateUpdated DATE COMMENT "Dato oppdatert",
    dateSyncNPD DATE NOT NULL,
    PRIMARY KEY (prlNpdidLicence)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
LOAD DATA LOCAL INFILE "csv/data/licence_petreg_licence_oper.csv"
    INTO TABLE licence_petreg_licence_oper
    -- [...]
ALTER TABLE licence_petreg_licence_oper ADD FOREIGN KEY (prlNpdidLicence)
    REFERENCES licence(prlNpdidLicence);
ALTER TABLE licence_petreg_licence_oper ADD FOREIGN KEY (cmpNpdidCompany)
    REFERENCES company(cmpNpdidCompany);
```

Fig. 3. Creating and loading table `licence_petreg_licence_oper`.

and efficient tool for working with them, not to create a perfectly modelled database. The resulting database is highly denormalised and contains many duplicate values; we assume that the CSV files are constructed in this way on purpose so as to carry more useful information for the (human) user. For example, tables which are related to the company table, i.e., that include the column `cmpNpdidCompany` (NPD's key for companies), usually also contain the name of the company and possibly additional duplicate data from the company table. Poor database modelling/design is also evident in the fact that one-to-one relationships are sometimes unnecessarily represented by a separate table. The translation can also result in missing foreign keys, for example in the case of the `wellbore_exploration_all` table. This table contains a column `fclNpdidFacility` which refers to the facility that drilled the wellbore; facilities are, however, distributed across two facility tables (`facility_fixed` and `facility_moveable`), so we cannot express this relationship using a foreign key. One way to solve this would be to create a table or view which collects all facilities, but, as mentioned, we have decided to postpone such corrections till the next step in the process.

3.2 Step 2: Dumping the relational database to RDF

To dump the contents of the relational database to RDF, we use D2RQ.[9] D2RQ makes it easy to expose relational databases as virtual RDF graphs by using a *map* which specifies a translation between terms in the database, and RDF vocabularies and OWL ontologies. This map determines how SPARQL queries over the virtual graph are rewritten to SQL queries over the relational database. Given a database as input, D2RQ can automatically generate such a map, and it is also equipped with a server which sets up a SPARQL endpoint and a LOD frontend serving the virtual graph. The virtual graph can also be dumped to a file. The generated map that D2RQ produces roughly maps each table to a class, with each row being made an instance of the class and given a URI pattern which is a concatenation of the values that constitute the primary key. Each column is

[9] `http://d2rq.org`, version 0.8.1

mapped to a property—an object property if the column is a foreign key, and otherwise a datatype property. For the all the common SQL datatypes, datatype properties are given an XSD datatype according to their SQL datatype. Less standard datatypes, like those holding geographical data, are ignored by D2RQ with a warning to the user. Classes are identified by a URI with the table name as localname and columns with the tablename and column name as localname.

As expected, this makes a very crude RDF representation: the identifiers of classes and properties are too tightly related to the database terms, and the URIs of individuals are not as informative as they could be. Additionally, partly due to an imprecise and poorly modelled database schema, there are effectively too many datatype properties which should be object properties, the resulting XSD datatypes are not always correct, and many values do not convey the intended meaning in the best manner. Examples of the latter are properties which clearly have a Boolean value domain, but where the values are variations of "yes" and "no" in different languages and with different capitalisation, or the values are, e.g., `ACTIVE`/`INACTIVE`, or one specific value and `NULL`. To fix these shortcomings we extend the CSV schema mentioned in the previous section and use this to generate a D2RQ map. We also map tables to classes and columns to properties, but in the CSV schema we specify the resulting URIs of the classes and properties, and allow extra classes and properties to be created by the map, as there are some cases where we want to build more than one class from a single table. We also specify whether tables or columns should be mapped at all. In this way we can remove duplicate data from database in the RDF output, creating a normalised dump. For each class we specify informative URI patterns which often are built by extending existing URI patterns, e.g., three URIs for a wellbore, a wellbore core and a wellbore core photo are respectively,

```
http://sws.ifi.uio.no/data/npd-v2/wellbore/5
http://sws.ifi.uio.no/data/npd-v2/wellbore/5/core/6
http://sws.ifi.uio.no/data/npd-v2/wellbore/5/core/6/photo/7607/2714-2718m
```

which indicate that the second URI is a core sample from the wellbore identified by the first URI, and that the core photograph is of the given core sample. For the first parts of the URI pattern we generally use one of the main categories found in the FactPages application.[10] This provides a partial solution to the problem of interlinking resources even when there is no foreign key—in these cases a URI can usually be built from values in the database. For the example we gave in the previous section, we simply let fixed facilities and moveable facilities share the same URI pattern and export the values from the column `wellbore_exploration_all.fclNpdidFacility` as the URIs `.../facility/[fclNpdidFacility]`. Dates of value `9999-12-31` are not included in the dump as they represent `NULL` values in the database, however, they are still useful since the columns they appear in are used in for many URL patterns, which do not accept `NULL` values. To represent the geographical information in the dataset we make use of the GeoSPARQL vocabulary,[11] which is designed for

[10] See footnote 3 on page 3.

[11] `http://schemas.opengis.net/geosparql/`

```
map:licence_petreg_licence_oper
  d2rq:dataStorage map:Adatabase;  a d2rq:ClassMap;  d2rq:class ptl:ProductionLicence;
  d2rq:classDefinitionLabel "Production Licence: Petroleum register, Operators";
  d2rq:uriPattern "/URIPATTERN62//petreg/licence/@@licence_petreg_licence_oper.prlNpdidLicence@@".
map:licence_petreg_licence_oper__cmpNpdidCompany__ref
  a d2rq:PropertyBridge;  d2rq:belongsToClassMap map:licence_petreg_licence_oper;
  d2rq:join "licence_petreg_licence_oper.cmpNpdidCompany => company.cmpNpdidCompany";
  d2rq:property ptl:licenceOperatorCompany;  d2rq:refersToClassMap map:company .
map:licence_petreg_licence_oper__ptlOperDateUpdated
  a d2rq:PropertyBridge;  d2rq:belongsToClassMap map:licence_petreg_licence_oper;
  d2rq:column "licence_petreg_licence_oper.ptlOperDateUpdated";
  d2rq:condition "licence_petreg_licence_oper.ptlOperDateUpdated <> '9999-12-31'";
  d2rq:property ptl:dateUpdated;  d2rq:datatype xsd:date;
```

Fig. 4. D2RQ map of parts of table `licence_petreg_licence_oper`.

this purpose [14]. For other changes to exported values we use D2RQ's translation table feature. It allows us to specify a series of one-to-one mappings between database values and RDF values, where the RDF values may be any legal RDF resource. We use translation tables to convert all the different Boolean values to the values `xsd:true` and `xsd:false`, to translate country names and codes into the correct resource representative in DBPedia, and to make minor adjustments to some oddly shaped values.[12] We also translate values from columns like `fclKind` (Facility kind/type) into nicely formatted URLs which are added as types to the relevant row individuals.[13] These specific translation tables are created by querying the database when the map is generated, a task which is easy to set up with a database at hand. Fig. 4 contains a snippet from the generated D2RQ map showing a mapping of parts of the database table `licence_petreg_licence_oper`, whose definition is found in Fig. 3. It illustrates a table which contains one-to-one relationships between a production licence and the operator of the licence. Instead of mapping this table to a separate class, we map the table to an existing class, `ptl:ProductionLicence`, and simply add a property from the licence to the operating company. The results from dumping to RDF using this map are exemplified in Fig. 5. After dumping the database to RDF we post process the RDF file, making changes which we are not able to represent in the D2RQ map, or at least not easily. The most important change we do is to remove a token from all URIs generated after our identifier schema. These tokens are added by us to all such URIs to ensure that all patterns are distinct, this can be seen in Fig. 4 as `d2rq:uriPattern "/URIPATTERN62//...`. This is a simple workaround for a bug in D2RQ which causes problems for its query rewriter if a URI pattern is a sub pattern of other URI patterns. As already shown, we make heavy use of URIs which are of this form. In the database there is one case of a column containing more than one value, hence breaking first normal form. In the post process, we split this in to multiple atomic values. To our knowledge, there is no built in

[12] E.g., the unit of measure values `[m ]` and `[ft ]` are translated to the `xsd:string`-s `m` and `ft`.

[13] E.g., the strings `MULTI WELL TEMPLATE` and `CONCRETE STRUCTURE` are translated to the resources `npdv:MultiWellTemplateFacility` and `npdv:ConcreteStructure-Facility`.

```
<http://sws.ifi.uio.no/data/npd-v2/petreg/licence/21559426>
    a ptl:ProductionLicence;
    ptl:dateUpdated   "2013-02-20"^^xsd:date;
    ptl:licenceOperatorCompany  <http://sws.ifi.uio.no/data/npd-v2/company/23173852> .
```

Fig. 5. RDF result of dumping a row from table `licence_petreg_licence_oper`.

way of achieving this with D2RQ alone. Lastly, we change all `UNKNOWN` values into blank nodes. This is possible to do in the D2RQ map, but would require that we add a special case for all column maps where `UNKNOWN` occurs.

3.3 Step 3: Building an OWL ontology

The generated ontology should define the same vocabulary as indirectly generated in the previous step, so every class and property mentioned by the map, which is in the namespace of the dataset, is declared as an `owl:Class` or a property of the correct kind: `owl:ObjectProperty`, `owl:DatatypeProperty` or `owl:AnnotationProperty`. We also record, using a separate set of annotation properties, the SQL table or column that the resource was generated from, and add any name and comment that is associated with the table or column in the database. We do this with all axioms that can be directly traced back to the database so as to help with debugging and further development of the ontology.

For each foreign key property, we add an existential restriction as a super class of the class representative of the table to which the foreign key applies, and we qualify the restriction with the class representative of the table that the key references. For each column that is `NOT NULL`, we add an unqualified existential restriction on the corresponding property as a super class of the class representative of the table where the column is found. For each object property, we set a domain and range for the property if the property is used with only one class as, respectively, domain or range; we set the domain of datatype properties in the same way, but we use the range value from the D2RQ map.

Fig. 6 illustrates the procedure with an excerpt of the result of generating an OWL ontology from the table given in Fig. 3. We can see that the foreign key `licence_petreg_licence_oper` is translated into the qualified existential restriction `ptl:licenceOperatorCompany some npdv:Company`. The other subclass axioms originate from a different SQL table, which is indicated by the fact that the class has two instances of the annotation property `sql:table`. (Each subclass axiom is also annotated, but this is not shown in the figure.) Indeed, as noted for the listing in Fig. 4, the table `licence_petreg_licence_oper` is mapped to a class for which another table is mapped. In this other table the columns corresponding to the properties `ptl:dateLicenceValidFrom` and `ptl:dateLicenceValidTo` are declared as `NOT NULL`. Finally, the `ptl:licenceOperatorCompany` property's column is only used once, and as a foreign key, so it is safe to set its domain and range.

The generated ontology is later extended manually by adding axioms which capture information that is not available to the automatic process. This amounts largely to adding atomic general superclasses to the generated classes, e.g.,

```
Class: ptl:ProductionLicence
    Annotations:
        sql:table "licence_petreg_licence_oper",
        sql:table "licence_petreg_licence",
        sql:columns "ptlName,cmpLongName,[...] (table: licence_petreg_licence_oper)",
        sql:columns "ptlName,ptlDateAwarded,[...] (table: licence_petreg_licence)"
    SubClassOf:
        ptl:dateLicenceValidFrom    some rdfs:Literal,
        ptl:dateLicenceValidTo      some rdfs:Literal,
        ptl:licenceOperatorCompany  some npdv:Company
ObjectProperty: ptl:licenceOperatorCompany
    Domain:  ptl:ProductionLicence
    Range:   npdv:Company
DataProperty: ptl:dateUpdated
    Annotations:
        sql:datatype "DATE (column: ptlOperDateUpdated)",
        sql:column "tuf_petreg_licence_oper.ptlOperDateUpdated"
    Range:  xsd:date
```

Fig. 6. An excerpt of the generated OWL ontology.

`npdv:Facility` is set as the superclass of the generated classes `npdv:FixedFacility` and `npdv:MoveableFacility`, and introducing a set of mutually disjoint top level classes like `npdv:Agent`, `npdv:Area` and `npdv:Point`. A small selection of classes and properties that model geographical data are mapped to the GeoSPARQL vocabulary. The generated ontology (called `npd-v2-db`), the set of SQL annotation properties (`npd-v2-sql`), the added superclasses ontology (`npd-v2-hm`), and the geographical mappings (`npd-v2-geo`) are kept in separate ontologies and files which are all imported by a central "hub" ontology (`npd-v2`).

3.4 Results

The outcome of the conversion steps are summarised in the tables below. 70 CSV files containing a total of 963 columns are downloaded from the FactPages application. These become 70 tables and 276 *distinct* columns in the relational database. To map the database to RDF we use a D2RQ map containing 79 class maps and 859 property bridges. Dumping the database to RDF produced (indirectly) 119 classes, 351 properties and 2.342.597 triples; this process takes approximately 4,5 hours. The complete ontology contains 209 classes and 375 properties.

	CSV	SQL	D2RQ Map	RDF	OWL
Files/Tables/Classes	70	70	79	119	209
Columns/Properties	963	276	859	341	375

The table below lists some numbers and the expressivity of the different ontologies, excluding imports, as reported by Protégé.[14] Figures in the last row of the table include axioms from imported "external" ontologies; the other rows relate only to "local" `npd-v2-*` ontologies.

[14] `http://protege.stanford.edu/`, version 4.3

Ontology	Axioms	Logical ax.	Classes	Obj. prop.	Da. prop.	Expressivity
npd-v2-db	3355	1006	109	87	221	$\mathcal{ALE}(\mathcal{D})$
npd-v2-hm	81	71	75	6	1	$\mathcal{ALCHI}(\mathcal{D})$
npd-v2-geo	7	3	3	2	0	$\mathcal{ALI}$
npd-v2-sql	7	0	0	0	0	-
npd-v2 (local)	3450	1080	132	89	221	$\mathcal{ALCHI}(\mathcal{D})$
npd-v2 (all)	4463	1271	209	131	229	$\mathcal{SHIF}(\mathcal{D})$

All files, together with our LOD representation of the FactPages, are published at the project website `http://sws.ifi.uio.no/project/npd-v2/`.

4 Query Evaluation

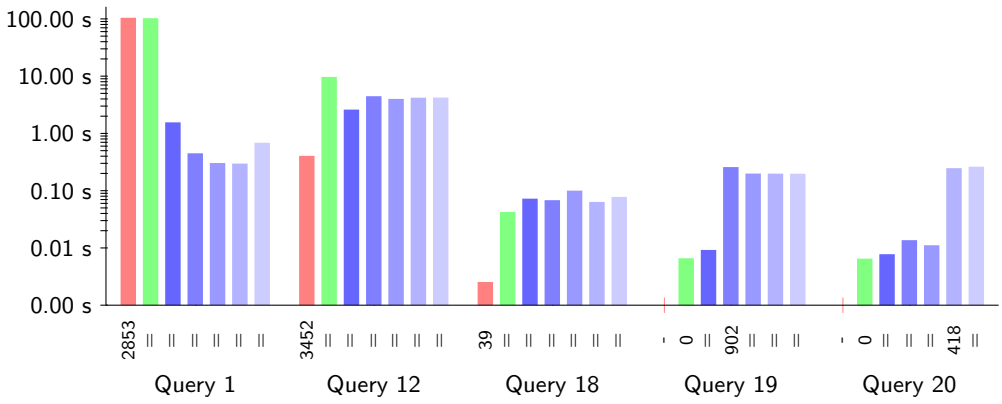
We have asked users of the FactPages to provide us with questions they would like to have answered by the FactPages. A subset of these questions has been translated to a total of 20 SPARQL and/or SQL queries. As part of our analysis of the costs and benefits of using the different representation formats and database systems, we want to evaluate how suited the two query languages are for representing the questions, and whether the added semantics coupled with reasoning gives more results and at what cost. That is, we wish to compare the systems with different levels of semantic querying, not to benchmark them.

When translating the queries we found that, although the query languages are very similar, formulating queries in SPARQL was slightly easier. This is in part due to the simple format of RDF, and that creating graph patterns seems to be a simpler and more natural process than finding the right tables and how to correctly join them. (There is no need for `FROM` in SPARQL.) An advantage for the SPARQL queries in our setup is, of course, that the somewhat oddly shaped relational database is restructured in the conversion step to RDF and OWL, making the graph model and class and property names more intuitive. Moreover, as discussed in Section 3, the modelling of the database is not ideal, and this may hamper query formulation.

The queries were executed against the MySQL database, a Fuseki[15] triple store with the post processed dump produced in step 2, and against a Stardog[16] triple store that was also loaded with the ontologies created and referenced in step 3. We tested Stardog using five different reasoning configurations: none (i.e., RDF only), RDFS, OWL 2 QL, OWL 2 EL and OWL 2 RL—note that, although our ontology does not fit into any of the OWL profiles, Stardog simply ignores axioms that fall outside the enabled profile. For each query and each system (configuration), we measured the execution time and the number of results returned. All questions, queries, execution times and (number of) results are published on the project website, and we present a representative selection in Fig. 7.

The results of our experiment can be summarised as follows.

[15] `http://jena.apache.org/documentation/serving_data/`, version 0.2.6
[16] `http://stardog.com/`, version 1.2



The chart shows the running times and the number of results returned for five different queries. For each query the database systems from left to right are MySQL, Fuseki, Stardog with no reasoning, then Stardog with respectively RDFS, QL, EL and RL reasoning enabled. The running times are given in seconds on a logarithmic scale, and the number of results returned are listed under each bar. A bar is marked by '=' if its number of results are the same as the bar to its left. MySQL does not have any query for Query 19 and 20, as these questions require vocabulary which does not exist in the database schema.

Fig. 7. Selected queries with running times and results returned.

- Queries are useful! This is of course well-known, but still true. We were able to answer all the questions we were given, most of which result in information which is not practically possible to retrieve from the FactPages in their current official representation.
- The only queries that return more answers when reasoning is enabled are queries which use vocabulary defined in the manually created ontology axioms, cf. Queries 19 and 20. In these cases, however, reasoning made a dramatic difference, increasing the number of answers from zero to several hundred. This could clearly be important in some applications. In the case of Query 19, RDFS reasoning produced the same number of answers as the OWL 2 profiles; for Query 20, however, RDFS and QL reasoning did not produce any answers, and EL and RL reasoning produced the same number of answers. This suggests that careful choice of profile could also be important in some applications.
- For most queries in the experiment, MySQL is faster than the triple stores, and Stardog is slightly faster than Fuseki. However, the worst performance result for any query is Query 1 running on MySQL. This is due to a join on a column which is not indexed, and illustrates a weakness with relational databases which triple stores do not suffer from.
- Running times for queries with reasoning enabled are only significantly different when more results are returned, as in Queries 19 and 20.

5 Conclusion

We have presented the results of a case study in which data published as CSV files was transformed into relational data, RDF data, and RDF data augmented with an ontology. Our goal was to analyse the costs and benefits of such transformations, with anticipated benefits being easier query formulation and enriched query answers, and anticipated costs being the transformation process itself and increased query answering times.

Simply translating the CSV files into relational data brought with it significant benefits in being able to retrieve information that was otherwise almost impossible to access. Additionally transforming the data into (LOD compliant) RDF brought with it a range of additional benefits, including better availability and (re)usability. Augmenting the data with an ontology adds semantic clarity, and can both extend the range of possible queries, and improve the quality of query answers.

Regarding transformation cost, the transformation process was greatly facilitated by the range and quality of available tools, with D2RQ being used to good effect in our case. Significant amounts of additional (largely manual) effort were needed to produce good quality RDF data, but this was due at least in part to quality issues relating to the source data. The cost of building the ontology was less than might have been expected, and although relatively simple in structure, the ontology was able to exploit existing work by mapping relevant classes and properties to the GeoSPARQL vocabulary. Regarding query answering cost, the results here were very encouraging, with query answering times only significantly longer in those cases where use of the ontology resulted in greatly enriched query answers.

For future work we are planning to improve the quality of the current ontology, and to make further developments with help from domain experts; we may also develop versions of the ontology specifically designed for one or more of the OWL 2 profiles. We anticipate that a more developed ontology will enable users to pose more sophisticated queries, and we plan to conduct a new evaluation experiment using such queries. Secondly, we intend to apply the ontology-based data access (OBDA) [3] methodology to the FactPages dataset, using mappings between the relational database and the ontology to produce a highly scalable query answering system. We believe that our existing conversion methodology can relatively easy be adapted to produce mappings for this purpose.

Acknowledgements. We wish to thank the NPD for making the FactPages publicly available. This research was partially funded by the EU FP7 grant "Optique", and the Norwegian Research Council through the Semicolon II project.

References

1. Marcelo Arenas et al., eds. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. Sept. 2012. URL: `http://www.w3.org/TR/rdb-direct-mapping/`.

2. Tim Berners-Lee. *Linked Data*. 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html.
3. Diego Calvanese et al. "Ontologies and Databases: The DL-Lite Approach". In: *Reasoning Web*. Ed. by Sergio Tessaris et al. Vol. 5689. LNCS. Springer, 2009, pp. 255–356.
4. Souripriya Das, Seema Sundara, and Richard Cyganiak, eds. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. Sept. 2012. URL: http://www.w3.org/TR/r2rml/.
5. *FactPages - Norwegian Petroleum Directorate*. 2013. URL: http://factpages.npd.no/factpages/.
6. The W3C SPARQL Working Group, ed. *SPARQL 1.1 Overview*. W3C Recommendation. Mar. 2013. URL: http://www.w3.org/TR/sparql11-overview/.
7. Lushan Han et al. "RDF123: From Spreadsheets to RDF". In: *International Semantic Web Conference*. Ed. by Amit P. Sheth et al. Vol. 5318. LNCS. Springer, 2008, pp. 451–466.
8. Patrick Hayes. *RDF Semantics*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/rdf-mt/.
9. Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. 1st ed. Morgan & Claypool, 2011.
10. Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/rdf-concepts/.
11. Andreas Langegger and Wolfram Wöß. "XLWrap - Querying and Integrating Arbitrary Spreadsheets with SPARQL". In: *International Semantic Web Conference*. Ed. by Abraham Bernstein et al. Vol. 5823. LNCS. Springer, 2009, pp. 359–374.
12. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia, eds. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second edition)*. W3C Recommendation. Dec. 2012. URL: http://www.w3.org/TR/owl2-syntax/.
13. Boris Motik et al., eds. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. Dec. 2012. URL: http://www.w3.org/TR/owl2-profiles/.
14. Matthew Perry and John Herring, eds. *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. OGC Implementation Standard. Sept. 2012. URL: http://www.opengis.net/doc/IS/geosparql/1.0.
15. Satya S. Sahoo et al. *A Survey of Current Approaches for Mapping of Relational Databases to RDF*. Tech. rep. W3C, 2009. URL: http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.
16. Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. "Bringing relational databases into the Semantic Web: A survey". In: *Semantic Web* 3.2 (2012), pp. 169–209.
17. *The Norwegian Petroleum Directorate - Norwegian Petroleum Directorate*. Accessed April 23th 2013. 2011. URL: http://www.npd.no/en/About-us/.