

Querying the Global Cube: Integration of Multidimensional Datasets from the Web

Benedikt Kämpgen¹, Steffen Stadtmüller¹, and Andreas Harth¹

Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany,
{benedikt.kaempgen, steffen.stadtmueller, harth}@kit.edu

Abstract. National statistical indicators such as the Gross Domestic Product per Capita are published on the Web by various organisations such as Eurostat, the World Bank and the International Monetary Fund. Uniform access to such statistics will allow for elaborate analysis and visualisations. Though many datasets are also available as Linked Data, heterogeneities remain since publishers use several identifiers for common dimensions and differing levels of detail, units, and formulas. For queries over the Global Cube, i.e., the integration of available datasets modelled in the RDF Data Cube Vocabulary, we extend the well-known Drill-Across operation over data cubes to consider implicit overlaps between datasets in Linked Data. To evaluate more complex mappings we define the Convert-Cube operation over values from a single dataset. We generalise the two operations for arbitrary combinations of multiple datasets with the Merge-Cubes operation and show the feasibility of the analytical operations for integrating government statistics.

1 Introduction

Given the Open Data policy of governments and intergovernmental organisations, citizens can access many statistical datasets online. For example, one can find the Gross Domestic Product of countries per year from Eurostat, the World Bank and the International Monetary Fund. Integrating such multidimensional datasets will allow for more complete answers and detailed comparisons of indicators. For example, the GDP of a country from one and the population from another dataset allow analysts to compute the GDP per Capita and to cross-check these derived values with values from other publishers.

Towards providing uniform access, many datasets are also made available – directly or by third-parties – as Linked Data reusing the RDF Data Cube Vocabulary (QB), the quasi-standard for publishing multidimensional datasets. Although analytical operations over QB datasets have been defined [5] and indicators from two datasets can be compared in visualisations [3] integration is still difficult [13] if datasets:

- contextualise their indicators with different dimensions, e.g., “geo-location”, “time” or “gender”,
- use different names for the same dimensions, e.g., “geo” and “location” or dimension values, e.g., “DE” and “Germany”,

- provide different levels of detail, e.g., regional or national level,
- use different units of measurement, e.g., “Million Euro” and “Euro”,
- and publish datasets derived from other datasets, e.g., “GDP per Capita” computed via “Nominal GDP” divided by “Population”.

Defining a global schema over heterogeneous datasets published as Linked Data is challenging since related work has so far concentrated on relational settings [2] with few sources that are centrally integrated. Relationships between different datasets are often buried in informal descriptions, and the routines to resolve semantic conflicts are provided in code or external background information [10]. After introducing preliminaries and a motivating scenario in Section 2 and before describing related work in Section 6, and concluding with Section 7, we provide the following contributions:

- We define the Global Cube using the Drill-Across operation [6] over datasets published as Linked Data. We describe how to derive previously unknown values in the Global Cube using OWL axioms as well as conversion [10] and merging correspondences [2] (Section 3).
- We analyse the complexity of generating the Global Cube (Section 4) and show the feasibility of our approach for government statistics (Section 5).

2 Preliminaries and Government Statistics Scenario

We use the common Multidimensional Data Model (MDM) of Data Cubes as a conceptualisation of QB datasets since we then can apply analytical operations such as Slice and Dice [5, 7]: An MDM consists of data cubes (instances of `qb:DataSet`) with facts (instances of `qb:Observation`). Every data cube defines measures (instances of `qb:MeasureProperty`) and dimensions (`qb:DimensionProperty`). Every fact in the data cube has a value for each of the measures and dimensions. The values of the measures are functionally dependent on the values of the dimensions, i.e., for every possible combination of dimension values, only one fact can be contained in the data cube. Members are the possible dimension values and may be grouped into levels along hierarchies (e.g., instances of `qb:CodeList`).

For instance, see Table 1 for an overview of data cubes from existing datasets to which an integration system should provide access. The table shows in the rows all data cubes and in the columns all their dimensions. The cells give example members for a dimension, “-” if the dimension is not and “...” if the dimension may be used.

For readability reasons, we describe URIs with namespaces¹, slightly abusing the W3C CURIE syntax for expressing compact URIs. Relative URIs such as `:DE` are defined by the data source in the respective context.

Since most publishers – also of our scenario data cubes – follow the practice of using an unspecific measure `sdmx-measure:obsValue` and a dimension indicating the measured variable, e.g., `estatwrap:indic_na` and `gesis:variable`,

¹ Use <http://prefix.cc/> to look up prefix definition.

Table 1. Overview of data cubes in rows with their dimensions in columns and dimension members available in the scenario in cells.

Cube \ Dimension	estat-wrap:geo	estatwrap:unit	dcterms:date	geis:geo	geis:variable	estatwrap:indic_na	estat-wrap:sex	estat-wrap:age
eurostat:id/tec00115#ds (GDP Growth)	:DE...	:PCH_PRE.	2001...	-	-	-	-	-
allbus:ZA4570v590.rdf#ds (Unemploy. Fear)	-	-	2004...	:00...	:v590.1...	-	-	-
eurostat:id/tsdcc310#ds... (EU 2020 Indicator)	:DE...	...	2001...	-	-
eurostat:id/nama_aux_gph#ds (GDP Per Capita)	:DE...	:EUR_HAB.	2001...	-	-	:NGDPH...	-	-
eurostat:id/nama_gdp_c#ds (GDP Components)	:DE...	:MIO_EUR.	2001...	-	-	:B1G, :D21_M-D31	-	-
eurostat:id/demo_pjan#ds (Population)	:DE...	-	2001...	-	-	-	:F...	:Y18...

and since cubes with multiple measures can be transformed to this form by introducing a new measure dimension, for the remainder of this paper we assume data cubes to have only one general measure, `sdmx-measure:obsValue`.

Every multidimensional element is published by a data source identified by the namespace. The `eurostat` namespace² makes available thousands of data cubes with indicators about European countries from Eurostat. For instance, the *GDP Growth* with the growth rate of the gross domestic product of all European countries per year, with the unit “percentage change on previous period” (`:PCH_PRE`) and for the geo dimension denoting Germany as `:DE`. Also, Eurostat provides citizens with EU 2020 Indicators, e.g., the energy dependence, productivity, and intensity as well as the greenhouse gas emission. Table 1 gives an example of one of those datasets; every *EU 2020 Indicator* cube exhibits the geo and time dimension and can contain other dimensions from the same data source. *GDP Components* provides granular values from which other indicators can be computed. For instance, the Nominal GDP (GDP at market prices) can be computed from adding the “Total gross value added” (`estatwrap:indic_na` of `:B1G`) and “Taxes less subsidies on products” (`:D21_M-D31`); similarly, the Nominal GDP divided by the *Population* should result in the *GDP Per Capita*.

The `allbus` namespace³ provides information about “attitudes, behaviour and social structure in Germany” from the Cumulated German General Social Survey (ALLBUS). Among others, we can retrieve a survey where German employees were asked about their fear of becoming unemployed (*Unemploy. Fear*). The measure describes the number of answers for a given question. The `allbus:variable` dimension denotes the type of answers given “No fear”,

² <http://estatwrap.ontologycentral.com/>

³ <http://lod.geis.org/lodpilot/ALLBUS/>

“Yes, of becoming unemployed”, “Yes, of having to change employer”⁴. The `:geo` dimension describes the participants’ country, e.g., Germany is denoted via `:00`.

An analytical query over a data cube can be described as a nested set of OLAP operations and executed using SPARQL over QB datasets [6, 5, 7]: **Projection** selects measures from a data cube; **Dice** filters for facts with certain members as dimension values; **Slice** sets dimensions to an implicit *ALL* member so that they can be aggregated over and removed; and **Roll-Up** aggregates dimension members to a higher level of the dimension hierarchy. Measures are aggregated using an aggregation function, e.g., described by the measure or by the user in the query [5]. We now describe three example queries that citizens may want to pose over the available datasets.

Unemployment Fear and GDP Growth (UNEMPLOY): Citizens want to compare the indicator about unemployment fear with the “GDP Growth” over time for Germany to get insights about the relation between GDP and employees’ perceived situation.

Comparing EU 2020 - Indicators (EU2020): Here, citizens want to aggregate and compare important metrics about European countries by average for all countries and to show the aggregated numbers per year, so that trends of important indicators for European countries become visible.

GDP per Capita from Different Sources (GDP_CAP): Here, citizens may want to confirm that the GDP per Capita per country and year provided by different institutions and derived from different datasets is equal to increase their trust in Open Data.

3 Building the Global Cube

In this section, we introduce an integrated view over data cubes, the Global Cube, and show how to increase its size using mappings between data cubes.

We describe the data of a cube $ds \in \text{DataCube}$ as a relation $ds(D1, D2, \dots, Dn, M)$ with $\text{dimension}(ds)$ the set of dimensions used by a cube and M the unspecific measure `sdmx-measure:obsValue`. The relation contains all possible dimension-member combinations on a specific level of detail, possibly with M an empty value such as “null” or “” We use functional datalog for describing rules about relations.

Definition 1 defines Drill-Across, the basic operation for integrating cubes.

Definition 1 (Drill-Across). *Given two data cubes $ds1(D11, D12, \dots, D1n, M1)$ and $ds2(D21, D22, \dots, D2n, M2)$, we define Drill-Across: $\text{DataCube} \times \text{DataCube} \rightarrow \text{DataCube}$ [6] and $\text{Drill-Across}(ds1, ds2) = ds3$ as follows: If $\text{dimension}(ds1) \neq \text{dimension}(ds2)$ then $ds3(D31, \dots, D3n, M3)$, with $\text{dimension}(ds3) = \text{dimension}(ds1) \cup \text{dimension}(ds2)$ empty, i.e., its relation contains no tuples; else then $D1i = D2i$, $1 \leq i \leq n$ and the following rule holds: $ds3(D1, \dots, Dn, M) :- ds1(D1, \dots, Dn, M1), ds2(D1, \dots, Dn, M2), M = f(M1, M2)$, with $f(M1, M2)$ defined as follows: If $(M1 \neq \text{null}$*

⁴ `allbus:variable.rdf#v590.1` to `allbus:variable.rdf#v590.3`


```

8         estatwrap:geo, {eurostat-geo:DE}),
9     {estatwrap:unit, gesis:variable})
10 )

```

However, this query only returns results if Unemploy. Fear and GDP Growth – despite their heterogeneous structures according to Table 1 – exhibit a dimension `estatwrap:geo` and a member `eurostat-geo:DE`. Therefore, in the following, we present two possibilities to reduce heterogeneities between data cubes to increase the number of answers returned for queries over the Global Cube: 1) Slicing of dimensions and mappings between shared dimensions and members, and 2) converting and merging of data cubes.

3.1 Drill-Across with Shared Dimension Mappings in Linked Data

To execute the analytical query given in Listing 1.1, we need to evaluate the query plan of a nested set of OLAP operations over QB datasets.

Every sub-query-plan of OLAP operations not including the Drill-Across operation we can evaluate using the *OLAP-to-SPARQL algorithm* [8] where every analytical operation is evaluated using parts of a SPARQL query. Similarly, given the RDF describing two input cubes re-using QB and sharing all their dimensions, we can evaluate the Drill-Across operation using SPARQL. See Listing 1.2 for an example SPARQL query for our previous OLAP query (Listing 1.1).

Listing 1.2. SPARQL for UNEMPLOY Drill-Across query

```

1 select ?geo0 ?date0 f(avg(?obsValue1), avg(?obsValue2))
2 where {
3   OPTIONAL { ?obs1 qb:dataSet eurostat:id/tec00115#ds;
4     estatwrap:geo ?geo0;
5     dcterms:date ?date0;
6     sdmx-measure:obsValue ?obsValue1 .
7   FILTER (?geo0 = eurostat-geo:DE) }
8   OPTIONAL { ?obs2 qb:dataSet allbus:ZA4570v590.rdf#ds;
9     estatwrap:geo ?geo0;
10    dcterms:date ?date0;
11    sdmx-measure:obsValue ?obsValue2 .
12  FILTER (?geo0 = eurostat-geo:DE)
13 } } group by ?geo0 ?date0

```

Here, for each of the two input cubes, we query for observations linked via `qb:dataSet` to the respective QB dataset URI (line 3 and 8); the observations from both datasets we join on the values of their dimension properties (4,5 and 9,10) and bind the values of their measures to separate variables (6,11) and combine them with `f(M1, M2)` with `f` resolving possible integrity constraint violations (1)⁵. Various optimisations such as materialisation [9] are possible but not the topic of this paper. The integration of more than two data cubes is possible by chaining Drill-Across operations.

⁵ In this example, no conflict resolution would be done but the values of the two cubes displayed to the user, e.g., using the `concat` function.

Drill-Across requires data cubes to use the same dimensions and members. Dimensions and members are shared if the same URIs or literal values are used; for instance, `:dim` is shared by `:ds1` and `:ds2` if the following patterns bind:

```

1 :ds1 qb:structure/qb:component/qb:dimension :dim .
2 :ds2 qb:structure/qb:component/qb:dimension :dim .

```

For instance, since GDP Growth and Unemploy. Fear both use `dcterms:date` and literal values for years such as 2006, drill-across over the time dimension can directly be done. However, different RDF terms may be used and only implicitly represent shared dimensions and members. For instance, the GDP Growth and Unemployment Fear cubes use different geo dimensions, `estatwrap:geo` and `gesis:geo`, as well as different members representing Germany, `:DE` and `:00`.

To allow the implicit definition of shared dimensions and members, we assume that the standard OWL semantics hold. OWL axioms can either be loaded from existing Linked Data or manually added to the system. There are different ways to indicate shared dimensions and members in Linked Data, e.g., `owl:sameAs`, `owl:equivalentProperty`, and `rdfs:subPropertyOf`.

For instance, GDP Growth and Unemploy. Fear share `estatwrap:geo` and `gesis:geo` if the dimensions are linked via `owl:equivalentProperty`; after also stating `eurostat-geo:DE owl:sameAs gesis-geo:00`, the query from Listing 1.1 will bring together GDP Growth and Unemploy. Fear for Germany.

3.2 Conversion and Merging Correspondences Between Data Cubes

We now define more complex mappings between data cubes. A Conversion Correspondence according to Definition 3 describes relationships between two cubes in terms of their dimension-member combinations (`inputmember`, `outputmember` $\in 2^{Dimension \times Member}$), i.e., how facts with certain members on dimensions in an `inputcube` can be converted to facts with other members on such dimensions in an `outputcube`. The actual conversion is described using a conversion function `f` \in `Function` that describes how the value of the measure of the `outputcube` can be computed by the value of the measure of the `inputcube` and that may be implemented in any programming language [10].

Definition 3 (Conversion Correspondence). *We define a Conversion Correspondence adapted from correspondences over relational data [2] and conversion functions [10] as follows: $ConversionCorrespondence = \{(inputmembers, outputmembers, f) \in 2^{Dimension \times Member} \times 2^{Dimension \times Member} \times Function\}$ with $Function: String \rightarrow String$. Given two data cubes $ds1(D11, \dots, D1n, M1)$ and $ds2(D21, \dots, D2n, M2)$ with $D1i = D2i$, $1 \leq i \leq n$. A conversion correspondence cc between the cubes, $cc(ds1) = ds2$, holds if the following rule holds: $ds2(D21, \dots, D2n, M2) :- ds1(D11, \dots, D1n, M1)$, $inputmember \in inputmembers$ hold for $ds1$, $outputmember \in outputmembers$ hold for $ds2$, $D1i \in dimension(ds1) \setminus inputmembers : D1i = D2i$, $f(M1) = M2$.*

We define a Convert-Cube operation with `Convert-Cube: DataCube \times ConversionCorrespondence \rightarrow DataCube` to denote the application of a conversion correspondence to an input data cube to result in a new derived cube

with the same structure as the input cube: `Convert-Cube(ds1, cc) = ds2 <=> cc(ds1) = ds2`.

For instance, the relationship between the member “Million Euro” and “Euro” in Eurostat can be described with the following correspondence: `MIO2EUR = ({(estatwrap:unit, eurostat-unit:MIO_EUR)}, {(estatwrap:unit, eurostat-unit:EUR)}, "f(x) = 1,000,000·x")` The application of `MIO2EUR` over the GDP Components data cube is denoted by `Convert-Cube(estatwrap:id/nama_gdp_c#ds, MIO2EUR)` and returns a new data cube containing values with unit “Euro”. To allow the consecutive application of conversion correspondences to a data cube in a nested set of `Convert-Cube` operations, each `Convert-Cube` operation we evaluate using a SPARQL 1.1 `CONSTRUCT` query generating the RDF of the derived cube to which in turn another `Convert-Cube` operation can be applied. Listing 1.3 shows the SPARQL query for our example.

Listing 1.3. Evaluation of `MIO2EUR` over GDP Components data cube.

```

1 CONSTRUCT {
2   ds12c44:ds qb:structure ?dsd .
3   _:outputobs qb:dataSet ds12c44:ds;
4     estatwrap:unit eurostat-unit:EUR;
5     gesis:geo ?gesisgeo;
6     estatwrap:geo ?estatwrapgeo;
7     estatwrap:indic_na ?estatwrapindicna;
8     dcterms:date ?dctermsdate;
9     sdmx-measure:obsValue ?outputvalue1 .
10  } where { {
11    select ?dsd ((1000000 * ?inputvalue1) as ?outputvalue1)
12      ?gesisgeo ?estatwrapgeo ?dctermsdate ?estatwrapindicna
13      ?inputvalue1
14  } where {
15    estatwrap:id/nama_gdp_c#ds qb:structure ?dsd .
16    ?inputobs qb:dataSet estatwrap:id/nama_gdp_c#ds;
17      estatwrap:unit eurostat-unit:MIO_EUR;
18      gesis:geo ?gesisgeo;
19      estatwrap:geo ?estatwrapgeo;
20      estatwrap:indic_na ?estatwrapindicna;
21      dcterms:date ?dctermsdate;
22      sdmx-measure:obsValue ?inputvalue1 .
23  } } }

```

The SPARQL `CONSTRUCT` query can be divided by triple patterns in the body (line 13 to 20 in Listing 1.3) that provide bindings for triple patterns in the head (line 2 to 9) that in turn define the constructed triples. Since in our implementation no functions are possible in triple patterns (see `1000000 * ?inputvalue1`), we surround body triple patterns with a SPARQL `SELECT` query. The query generates for every fact in the input cube with unit `eurostat-unit:MIO_EUR` a new fact with unit `eurostat-unit:EUR` in an output cube with the same structure (dimensions and measures); the value of the (generic) measure is 1,000,000 times the value of the input cube’s measure.

Along this example, we explain how a cube and a conversion correspondence as input to a Convert-Cube operation can be translated to the respective SPARQL CONSTRUCT query. The body triple patterns are created in the following steps:

1. **Dataset Triples:** We bind the data structure definition and observations from the dataset URI of the input cube (line 13 and 14).
2. **Inputmembers Triples:** For each dimension-member combination in `inputmembers`, we add a respective triple pattern (15).
3. **Dimensions Triples:** For each dimension from the input cube which is not contained in `inputmembers`, we bind from the observation the value for the dimension URI to a variable derived from the dimension URI to refer back to it in the head triple patterns later (16 to 19). Since the data cubes share their geo dimensions, there are triple patterns for `geis:geo` and `estatwrap:geo`.
4. **Measures Triples:** For each measure in `inputcube`, we bind from the observation the value to a variable that is unique per measure for referral in other parts of the rule (20).
5. **Function Triples:** For each measure in `inputcube`, we bind a variable for the derived dataset's measure with an expression for function `f` with the input variable of `f` replaced by the respective measure variable (11).

Similarly, we create the triple patterns in the head:

1. **Dataset Triples:** We introduce a URI for the derived output dataset (e.g., by a combination of unique IDs for the original dataset and the conversion correspondence); we add the data structure definition of the input dataset to the output dataset (line 1); we add new observations to the output dataset using a blank node (line 2)
2. **Outputmembers Triples:** For each dimension-member combination in `outputmembers`, we add a respective triple pattern (line 4).
3. **Dimension Triples:** For each dimension from the input cube which is not contained in `outputmembers`, we add to the new observation the dimension values of the observation in the body (line 5 to 8).
4. **Measure Triples:** For each measure in `inputcube`, we assign to the respective measure in the derived observation the variable describing the converted value from the body (9)

The Dimension Triples make sure that the derived data cube has the same dimensions as the input cube and copy all dimension values not touched by the conversion correspondence. For that, contrary to the Open-World assumption in Linked Data, we have to assume all dimensions stated by the data structure definition of the input dataset to be known.

The SPARQL query is evaluated over the RDF representing the data cube to generate the derived cube. To answer a query over the Global Cube, we need to take into account all derived data cubes, including those derived by nested Convert-Cube operations. Given an OLAP query with nested Convert-Cube operations, any Convert-Cube operation is evaluated using one evaluation of the respective SPARQL CONSTRUCT query over the input data cube's RDF.

Iteratively, the RDF of the input data cube may first need to be derived by the evaluation of another Convert-Cube operation. In the next section, we will describe an analysis of the number of derived data cubes in the Global Cube.

Conversion correspondences we can extend to merging correspondences to combine values from two cubes. A merging correspondence according to Definition 4 describes how facts with certain members on dimensions in two data cubes can be merged to facts in a third data cube with members on such dimensions and with the same structure as the first input cube.

Definition 4 (Merging Correspondence). *We define **MergingCorrespondence** = $\{(inputmembers1, inputmembers2, outputmembers, f) \in \mathcal{P}^{Dimension \times Member} \times \mathcal{P}^{Dimension \times Member} \times \mathcal{P}^{Dimension \times Member} \times Function\}$ with **Function**: $String \times String \rightarrow String$. Given three data cubes $ds1(D11, D12, \dots, D1n, M1)$, $ds2(D21, \dots, D2n, M2)$, and $ds3(D31, \dots, D3n, M3)$ with $D1i = D3i$, $1 \leq i \leq n$. A merging correspondence mc between the three cubes, $mc(ds1, ds2) = ds3$ holds if the following rule holds: $ds3(D31, \dots, D3n, M3) :- ds1(D11, \dots, D1n, M1), ds2(D21, \dots, D2n, M2), inputmember1 \in inputmembers1$ hold for $ds1$, $inputmember2 \in inputmembers2$ hold for $ds2$, $outputmember \in outputmembers$ hold for $ds3$, $D1i \in dimension(ds1) \setminus inputmembers1 : D1i = D3i, f(M1, M2) = M3$.*

We define a Merge-Cubes operation with $Merge-Cubes: DataCube \times DataCube \times MergingCorrespondence \rightarrow DataCube$ to denote the application of a merging correspondence to two input data cubes to result in a derived cube.

The following example computes the Nominal Gross Domestic Product (NGDP) from the sum of two GDP component indicators: $COMPUTE_GDP = (\{(estatwrap:indic_na, eurostat-indic_na:B1G)\}, \{(estatwrap:indic_na, eurostat-indic_na:D21_M_D31)\}, \{(estatwrap:indic_na, eurostat-indic_na:NGDP)\}, "f(x_1, x_2) = x_1 + x_2")$. And the following example computes the GDP per Capita in Euro per Inhabitant from the Nominal GDP and the Population: $COMP_GDP_CAP = (\{(estatwrap:indic_na, eurostat-indic_na:NGDP), (estatwrap:unit, eurostat-unit:EUR)\}, \{(estatwrap:sex, eurostat-sex:T), (estatwrap:age, estatwrap-age:TOTAL)\}, \{(eurostat:indic_na, eurostat-indic_na:NGDPH), (eurostat:unit, eurostat-unit:EUR_HAB)\}, "f(x_1, x_2) = x_1/x_2")$. Here, from the second input cube only facts are selected that contain measures for all genders and age groups, assuming they describe the population. The algorithm to evaluate Convert-Cube using a SPARQL CONSTRUCT query can be extended to the Merge-Cubes operation.

4 Analysis of the Global Cube

Given a set of data cubes and conversion and merging correspondences, an algorithm to generate all derived data cubes to answer a query over the Global Cube may not terminate since correspondences can be infinitely nested.

If we require that the same correspondence is not applied repeatedly we can give an upper bound estimation of the number of (derived) cubes based on ds datasets and mc merging and conversion correspondences: $noderivedds(ds, mc)$.

For that, we define a recursive function $noc(dp, ds, mc)$ that distinguishes the depth dp of a nested set of correspondence applications with $noc(0, ds, mc) = ds$, $noc(dp, ds, mc) = mc^2 * noc(dp-1, ds, mc-1)^2 + 2 * mc^2 * \sum_{0 <= i < dp-1} noc(dp-1, ds, mc-1) * noc(i, ds, mc-1)$. In the recursion, we need to consider the ordering of inputs to the merging, i.e., the consecutive application of merging correspondences in the left, right and both inputs. Then, $noderivedds(ds, mc) = noc(0, ds, mc) + noc(1, ds, mc) + \dots + noc(mc, ds, mc)$.

As an example, for our GDP_CAP query we assume the GDP Per Capita, the GDP Components, and the Population as data cubes, MIO2EUR as conversion, and COMP_GDP and COMP_GDP_CAP as merging correspondences. The maximum number of (derived) datasets is given by: $noderivedds(3, 3) = noc(0, 3, 3) + noc(1, 3, 3) + noc(2, 3, 3) + noc(3, 3, 3) = 3 + 81 + 13,608 + 3,003,480 = 3,017,172$.

However, most derived data cubes are empty, e.g., MIO2EUR(GDP Per Capita) since GDP Per Capita does not contain values with unit “Million Euro”. Therefore, if we require in a nested application of correspondences that **outputmembers** of the first correspondence fit **inputmembers** of the second correspondence, the number of derived datasets in many cases will be reduced.

To generate all possible derived data cubes in our GDP_CAP query, we use a functional Datalog program, where we define the datasets using relation **dataset(Ds)**, dimensions using **dimension(Ds, Dim)** and dimension-member combinations using **dimensionmember(Ds, Dim, Mem)**; also, we define every conversion and merging correspondence using four rules for 1) generating the dataset, 2) copying over the dimensions, 3) copying over the dimension-members, and 4) setting the new dimension member. For instance, rule 1) for MIO2EUR is as follows: `dataset(mio2eur(X)) :- dataset(X), dimension(X,unit), (\+ dimensionmember(X,unit,Z); dimensionmember(X,unit,mioeur))`.

Running the program in XSB Prolog, we now only get 54 derived datasets – including a computation of the GDP per Capita via `comp_gdp_per_cap(comp_gdp(mio2eur(gdpcomponents), mio2eur(gdpcomponents)), population)`, to evaluate using SPARQL. The computation would take milliseconds on commodity hardware, since the program would only contain 14 atoms and 12 rules; the facts do not need to be represented in the program since we find matches between datasets to convert or merge only by looking at the definition of correspondences, i.e., output- and inputmembers.

We may want to allow cycles in the definition of input-/outputmembers, e.g., conversion correspondences MIO2EUR and EUR2MIO. In this case we can design an algorithm that terminates with the application of conversion and merging correspondences when no new facts can be added to the Global Cube:

Sketch of Proof According to Definition 1 of Drill-Across and Definition 2 of the Global Cube, an “Integrity Constraint Violation” is returned for a measure if two input cubes during the computation of the Global Cube contain different measure values for identical dimension-member combinations.

Only different dimension-member combinations can provide new facts, all other data cubes either provide facts with the same measure value or an “In-

egrity Constraint Violation”. If dimension-member combinations are limited, so are derived datasets that provide new facts to the Global Cube.

The number of dimension-member combinations is limited: Considering ds datasets as special combinations and the order of combining combinations, cc conversion and mc merging correspondences provide at max $(ds + cc + mc)!$ combinations of dimension-member combinations. \square

In our example, this would result in $(3 + 1 + 2)! = 720$ possible combinations.

5 Evaluation: Integrating Government Statistics

Figure 1 illustrates how a client issues the *GDP_CAP* query (using the query language MDX) to the designed integration system over a Global Cube defined by data cubes GDP Per Capita, GDP Components and Population. The integration engine 1) loads and validates available data cubes defined by QB datasets (*Base-Cube*), 2) translates the query over the Global Cube to a logical operator query plan over the available as well as all derived data cubes, 3) transforms the logical to a physical operator query plan with iterators that 4) are then executed. Here, *MIO2EUR* converts “Million Euro” to “Euro”, *COMP_GDP* computes the Nominal GDP, and *COMP_GDP_CAP* computes the GDP Per Capita which in the Global Cube is brought together with values from the GDP Per Capita data cube.

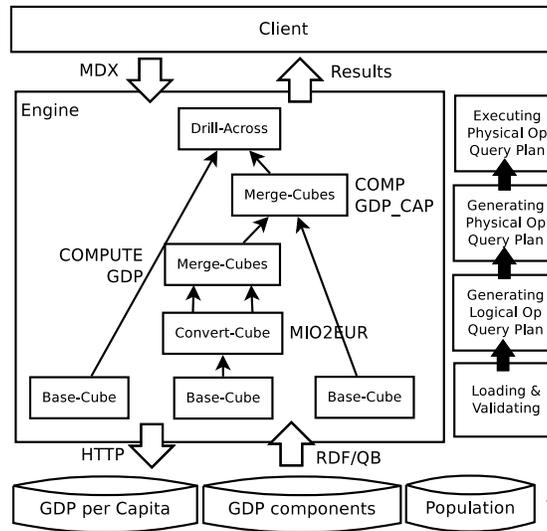


Fig. 1. Overview of integration system

We implemented all operations, including the Drill-Across, Convert-Cube, and Merge-Cubes operations, in *OLAP4LD*, an Open-Source Java engine for

analytical queries over multidimensional datasets published as Linked Data⁶. OLAP4LD uses a directed crawling strategy to load and validate all data cubes into a Sesame Repository (v2.7.10) as embedded triple store. Due to lack of space, in the further descriptions we assume all available data cubes loaded⁷.

Given a query and a set of available datasets, the logical query plan is automatically generated per Definition 2 of the Global Cube, and by automatically applying mappings between shared dimensions and members as well as convert and merging correspondences.

Drill-Across is implemented as a nested loop join directly over the results of the *OLAP-to-SPARQL algorithm* [8]. OWL semantics we evaluate using a duplication strategy by repeatedly executing SPARQL INSERT queries implementing entailment rules of equality⁸.

Every SPARQL query defined by a Convert-Cube and Merge-Cubes operation is evaluated once and the result is loaded in the triple store for usage by consecutive operations.

Setup: For the *UNEMPLOY* query, we created `owl:sameAs` mappings between the `:geo` dimensions and the identifier for Germany `:DE` and `:00` from Eurostat and Gesis. For *EU2020*, we selected four and eight EU 2020 Indicator datasets that surely overlap, e.g., the energy dependence, productivity, and intensity. For *GDP_CAP*, we created a ConversionCorrespondence for *MIO2EUR* and MergingCorrespondences for *COMPUTE_GDP*, and *COMP_GDP_CAP*.

Every query motivated in our scenario we executed five times on an Ubuntu 12.04 workstation with Intel(R) Core(TM) i5 CPU, M520, 2.40GHz, 8 GB RAM, 64-bit on a JVM (v6) with 512M initial and 1524M maximum memory allocation.

Results from Evaluating Drill-Across: Table 2 gives an overview of experiment results. We compare query results from previous experiments executing Drill-Across with the OLAP engine Mondrian over MySQL for query processing (*UNEMPLOY_1*, *EU2020_1a*, *EU2020_1b*) [7] and from executing our Drill-Across implementation (*UNEMPLOY_2*, *EU2020_2a*, *EU2020_2b*). The experiments are comparable since we used the same machine.

We successfully integrated the GDP Growth from Eurostat⁹ and the Unemployment Fear from ALLBUS and found overlaps for Germany in 2004 and 2006. Mappings between implicitly shared dimensions and members were considered. Also, for the EU2020 query, we successfully integrated four and eight datasets.

Loading and validating datasets takes much less time than with our previous implementation which results from the directed crawling strategy and from a switch from *qcrumb.com* to Sesame as a SPARQL query engine. L&V also includes the time for reasoning. Most time is spent in executing several queries for multidimensional elements (MD) and in generating the logical and physical

⁶ <https://code.google.com/p/olap4ld/>

⁷ Additional information can be found on the evaluation website of the paper:

http://linked-data-cubes.org/index.php/Global_Cube_Evaluation_EKAW14

⁸ <http://semanticweb.org/OWLLD/#Rules>

⁹ Apparently, dataset <http://estatwrap.ontologycentral.com/id/tsieb020#ds> in Eurostat was replaced by dataset `tec00115` after conducting these experiments.

Table 2. For every experiment, number of integrated datasets #DS, triples #T, observations #O, look-ups #LU, and average elapsed query times in sec for loading and validating datasets (L&V), executing (MD) a certain number of metadata queries (#MD), generating the logical query plan (LQP), generating the physical query plan (PQP), executing the physical query plan (EQP), and total elapsed query time (T).

Experiment	#DS	#T	#O	#LU	L&V	MD	#MD	LQP	PQP	EQP	T
UNEMPLOY_1	2	20,268	350	22	273	-	-	-	-	0.073	273
UNEMPLOY_2	2	3,897	362	12	11	5	41	3	3	0.036	22
EU2020_1a	4	24,636	1,247	26	654	-	-	-	-	0.161	654
EU2020_2a	4	19,714	2,212	12	18	15	67	3	3	0.094	39
EU2020_1b	8	35,482	2,682	34	1,638	-	-	-	-	0.473	1,638
EU2020_2b	8	38,069	3,992	20	47	40	103	6	10	0.151	103

query plans (LQP+PQP). Our experiments with 2, 4, and 8 datasets indicate that MD, LQP, PQP and EQP increase linearly. LQP includes the time for interpreting the query language (MDX) and building a nested set of OLAP operations. PQP includes the time to run the *OLAP-to-SPARQL algorithm* [8]. Executing the SPARQL queries and Drill-Across operations (EQP) only takes a fraction and is similar to processing time in the OLAP engine [7].

Results from Evaluating Convert-Cube and Merge-Cubes: To show the feasibility of conversion and merging correspondences, we manually built the logical query plan as illustrated in Figure 1 for comparing GDP per Capita from different datasets. Consequently, no time is spent for metadata queries and generating the logical query plan. For an efficient materialisation of all derived data cubes, a more scalable RDF engine would be needed.

We successfully executed the GDP_CAP query, the resulting cube allows us to compare computed and the given GDP per Capita as stored in Eurostat. By the small divergence between the computed and the given values, we presume that the computations are correct; for instance, for UK in 2010, the Nominal GDP per Capita is directly given as 27,800 and computed as 27,704 Euro per Inhabitant.

On average, the query takes 246sec. We load 1,015,044 triples. Note, these do not only come from 10 lookups to the three integrated datasets, but also from loading derived datasets to the embedded triple store. Similarly, in total, the engine loads or creates 126,351 observations. The long time of 119sec for the 10 look-ups, loading, and validating results from the fact that integrated datasets per se are larger than the datasets we have loaded in previous experiments, e.g., the Population data cube is described by more than 22MB of RDF/XML.

Generating the physical operator query plan with on average 16sec is fast, but executing the query plan with an average of 111sec takes as long as loading and validating. This is because different from the pipelining strategy of the Drill-Across iterator to directly process the results of a previous iterator, for Convert-Cube and Merge-Cubes the physical query plan involves materialising data cubes as derived cubes and storing them in the embedded triple store for

the next iterator. From 111sec needed for processing the physical query plan, on average 91sec (82%) was spent on generating and storing the derived datasets.

6 Related Work

Some authors [1, 6] allow drill-across over cubes with not fully shared dimensions, e.g., dimensions with different granularity such as monthly and yearly, and for which one dimension can be defined as the association of several ones for which a mapping is needed, e.g., latitude/longitude versus point geometry.

Tseng and Chen [13] define several semantic conflicts, e.g., implicitly shared dimensions and inconsistent measures, that are manually solved using XML transformations. Different from these approaches to overcome heterogeneities, we keep a strict Drill-Across definition and allow solving of semantic conflicts with abstract conversion and merging relationships.

For finding relationships between multidimensional datasets common ontology matching approaches are less suitable [15]. Torlone [12] automatically match heterogeneous dimensions by checking requirements of shared dimensions such as coherence and soundness; similar to our work, they use joins and materialisation approaches. We focus on more complex mappings explicitly given by experts.

Wilkinson and Simitsis [14] propose flows of hypercube operators as a conceptual model from which ETL processes can be generated. The Linked-Data-Fu language [11] uses N3 rules for describing complex data processing interactions on the Web. A rule engine could possibly improve our query processing approach, e.g., by bulk-loading, crawling and query processing in parallel threads and if backtracking from a query is supported. However, N3 does not support functions such as needed in our conversion and merging correspondences. Also, we provide an abstraction layer specific to multidimensional datasets published as Linked Data. Etcheverry and Vaisman [5] map analytical operations to SPARQL over RDF but do not define multi-cube operations and mappings.

Siegel et al. [10] introduce the notion of semantic values – numeric values accompanied by metadata for interpreting the value, e.g., the unit – and propose conversion functions to facilitate the exchange of distributed datasets by heterogeneous information systems. Calvanese et al. [2] describe a rule-based approach to automatically find the matching between two relational schemas. We extend their approaches to data cubes published as Linked Data.

Diamantini et al. [4] suggest to uniquely define indicators (measures) as formulas, aggregation functions, semantics (mathematical meaning) of the formula, and recursive references to other indicators. They use mathematical standards for describing the semantics of operations (MathML, OpenMath) and use Prolog to reason about indicators, e.g., for equality or consistency of indicators. In contrast, we focus on heterogeneities occurring in terms of dimensions and members, and allow conversions and combinations.

7 Conclusions

As the number of statistical datasets published as Linked Data is growing, citizens and analysts can benefit from methods to integrate national indicators, despite heterogeneities of data sources. In this paper, we have defined the Global Cube based on the Drill-Across operation over cubes published as QB datasets. The number of answers of queries over the Global Cube can be increased via OWL mappings as well as more complex conversion and merging relationships between datasets. Results from a scenario integrating government statistics indicate that – if a more scalable RDF engine is used – the operations can provide the foundations for an automatic integration of datasets.

Acknowledgements. This work was partially supported by the German Research Foundation (I01, SFB/TRR 125 “Cognition-Guided Surgery”), the German Federal Ministry of Education and Research (Software Campus, 01IS12051), and the EU’s 7th Framework Programme (PlanetData, Grant 257641).

References

1. Abelló, A., Samos, J., Saltor, F.: Implementing Operations to Navigate Semantic Star Schemas. In: Proceedings of DOLAP. ACM Press (2003)
2. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data Integration in Data Warehousing. *International Journal of Cooperative Information Systems* 10, 237–271 (2001)
3. Capadisli, S., Auer, S., Riedl, R.: Linked Statistical Data Analysis. In: ISWC Sem-Stats (2013)
4. Diamantini, C., Potena, D., Storti, E.: A Logic-Based Formalization of KPIs for Virtual Enterprises. *Advanced Information Systems* pp. 274–285 (2013)
5. Etcheverry, L., Vaisman, A.A.: Enhancing OLAP Analysis with Web Cubes. In: Proceedings of the 9th ESWC. Springer (2012)
6. Gómez, L.I., Gómez, S.A., Vaisman, A.A.: A Generic Data Model and Query Language for Spatiotemporal OLAP Cube Analysis Categories and Subject Descriptors. In: Proceedings of EDBT (2012)
7. Kämpgen, B., Harth, A.: Transforming Statistical Linked Data for Use in OLAP Systems. In: Proceedings of the 7th I-Semantics (2011)
8. Kämpgen, B., Harth, A.: No Size Fits All - Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views. In: Proceedings of ESWC (2013)
9. Shukla, A., Deshpande, P.M., Naughton, J.F.: Materialized View Selection for Multi-cube Data Models. In: Proceedings of EDBT. pp. 269–284 (2000)
10. Siegel, M., Sciore, E., Rosenthal, A.: Using semantic values to facilitate interoperability among heterogeneous information systems. *Transactions on Database Systems* (1994)
11. Stadtmüller, S., Harth, A.: Data-Fu : A Language and an Interpreter for Interaction with Read / Write Linked Data. In: Proceedings of WWW (2013)
12. Torlone, R.: Two approaches to the integration of heterogeneous data warehouses. *Distributed and Parallel Databases* 23(1), 69–97 (2007)
13. Tseng, F., Chen, C.: Integrating heterogeneous data warehouses using XML technologies. *Journal of Information Science* 31 (2005)
14. Wilkinson, K., Simitsis, A.: Designing Integration Flows Using Hypercubes. In: Proceedings of EDBT/ICDT (2011)

15. Zopilko, B., Mathiak, B.: Object property matching utilizing the overlap between imported ontologies. In: *The Semantic Web: Trends and Challenges*. pp. 737–751. *Lecture Notes in Computer Science* (2014)