Mapping Patterns for Virtual Knowledge Graphs

Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Department of Computing Science
Umeå University, Sweden

IEEE International Conference on
Artificial Intelligence & Knowledge Engineering (AIKE 2020)
9 December 2020 – Virtual
Outline of the presentation

1. Challenges in Data Access
2. Virtual Knowledge Graphs for Data Access
3. Query Answering in VKGs
4. Mapping Patterns
5. Conclusions
Challenges in the Big Data era

The FOUR V’s of Big Data

Volume
- It’s estimated that 2.5 quintillion bytes of data are created each day.
- 6 billion people have cell phones.
- 40 zettabytes (43 trillion gigabytes) of data will be created by 2020, an increase of 300 times from 2005.

Variety
- Variety is different forms of data.
- 30 billion pieces of content are shared on Facebook every month.
- 4 billion+ hours of video are watched on YouTube each month.
- 400 million tweets are sent per day by about 200 million monthly active users.

Velocity
- Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure.
- The New York Stock Exchange captures 1 TB of trade information during each trading session.
- By 2015, 4.4 million IT jobs will be created globally to support big data, with 1.9 million in the United States.

Veracity
- In one survey, 27% of respondents admitted how much of their data was inaccurate.
- 1 in 3 business leaders don’t trust the information they use to make decisions.
- Poor data quality costs the US economy around $3.1 trillion a year.

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As of 2011, the global size of data in healthcare was estimated to be 150 exabytes (161 billion gigabytes).

By 2014, it’s anticipated there will be 420 million wearable, wireless health monitors.

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors, and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

Diego Calvanese (unibz + umu)
Variety, not volume, is driving Big Data initiatives

MIT Sloan Management Review (28 March 2016)

http://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives/
How much time is spent searching for the right data?

Important problem: searching for data and establishing its quality

Example: in oil&gas, engineers spend 30–70% of their time on this (Crompton, 2008)
Challenge: Accessing heterogeneous data

Statoil (now Equinor) Exploration

Geologists at Statoil, prior to making decisions on drilling new wellbores, need to gather relevant information about previous drillings.

Slegge relational database:

- Terabytes of relational data
- 1,545 tables and 1,727 views
- each with dozens of attributes
- consulted by 900 geologists
Problem: Translating information needs

**Information need expressed by geologists**

"In my geographical area of interest, return all pressure data tagged with key stratigraphy information with understandable quality control attributes, and suitable for further filtering."

To obtain the answer, this needs to be translated into SQL:

- Main table for wellbores has 38 columns (with cryptic names).
- To obtain pressure data requires a 4-table join with two additional filters.
- To obtain stratigraphic information requires a join with 5 more tables.
Problem: Translating information needs

We would obtain the following SQL query:

```sql
SELECT WELLBORE.IDENTIFIER, PTY_PRESSURE.PTY_PRESSURE_S,
       STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER, STRATIGRAPHIC_ZONE.STRAT_UNIT_IDENTIFIER
FROM WELLBORE,
     PTY_PRESSURE,
     ACTIVITY FP_DEPTH_DATA
LEFT JOIN (PTY_LOCATION_1D FP_DEPTH_PT1_LOC
        INNER JOIN PICKED_STRATIGRAPHIC_ZONES ZS
            ON ZS.STRAT_ZONE_ENTRY_MD <= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
            ZS.STRAT_ZONE_EXIT_MD >= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
            ZS.STRAT_ZONE_DEPTH_UOM = FP_DEPTH_PT1_LOC.DATA_VALUE_1_OU
        INNER JOIN STRATIGRAPHIC_ZONE
            ON ZS.WELLBORE = STRATIGRAPHIC_ZONE.WELLBORE AND
            ZS.STRAT_COLUMN_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER AND
            ZS.STRAT_INTERP_VERSION = STRATIGRAPHIC_ZONE.STRAT_INTERP_VERSION AND
            ZS.STRAT_ZONE_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_ZONE_IDENTIFIER)
    ON FP_DEPTH_DATA.FACILITY_S = ZS.WELLBORE AND
    FP_DEPTH_DATA.ACTIVITY_S = FP_DEPTH_PT1_LOC.ACTIVITY_S,
    ACTIVITY_CLASS FORMPRESSION_CLASS
WHERE WELLBORE.WELLBORE_S = FP_DEPTH_DATA.FACILITY_S AND
    FP_DEPTH_DATA.ACTIVITY_S = PTY_PRESSURE.ACTIVITY_S AND
    FP_DEPTH_DATA.KIND_S = FORMPRESSION_CLASS.ACTIVITY_CLASS_S AND
    WELLBORE.REF_EXISTENCE_KIND = 'actual' AND
    FORMPRESSION_CLASS.NAME = 'formation pressure depth data'
```

This can be very time consuming, and requires knowledge of the domain of interest, a deep understanding of the database structure, and general IT expertise. This is also very costly! Equinor loses 50,000,000 eur per year only due to this problem!!
Problem: Translating information needs

We would obtain the following SQL query:

```sql
SELECT WELLBORE.IDENTIFIER, PTY_PRESSURE_PTY_PRESSURE_S,
     STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER, STRATIGRAPHIC_ZONE.STRAT_UNIT_IDENTIFIER
FROM WELLBORE,
     PTY_PRESSURE,
     ACTIVITY_FP_DEPTH_DATA
LEFT JOIN (PTY_LOCATION_1D_FP_DEPTH_PT1_LOC
           INNER JOIN PICKED_STRATIGRAPHIC_ZONES_ZS
           ON ZS.STRAT_ZONE_ENTRY_MD <= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
           ZS.STRAT_ZONE_EXIT_MD >= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
           ZS.STRAT_ZONE_DEPTH_UOM = FP_DEPTH_PT1_LOC.DATA_VALUE_1_OU
           INNER JOIN STRATIGRAPHIC_ZONE
           ON ZS.WELLBORE = STRATIGRAPHIC_ZONE.WELLBORE AND
           ZS.STRAT_COLUMN_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER AND
           ZS.STRAT_INTERP_VERSION = STRATIGRAPHIC_ZONE.STRAT_INTERP_VERSION AND
           ZS.STRAT_ZONE_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_ZONE_IDENTIFIER)
           ON FP_DEPTH_DATA.FACILITY_S = ZS.WELLBORE AND
           FP_DEPTH_DATA.ACTIVITY_S = FP_DEPTH_PT1_LOC.ACTIVITY_S,
     ACTIVITY_CLASS_FORM_PRESSURE_CLASS
WHERE WELLBORE.WELLBORE_S = FP_DEPTH_DATA.FACILITY_S AND
     FP_DEPTH_DATA.ACTIVITY_S = PTY_PRESSURE.ACTIVITY_S AND
     FP_DEPTH_DATA.KIND_S = FORM_PRESSURE_CLASS.ACTIVITY_CLASS_S AND
     WELLBORE.REF_EXISTENCE_KIND = 'actual' AND
     FORM_PRESSURE_CLASS.NAME = 'formation pressure depth data'
```

This can be very time consuming, and requires knowledge of the domain of interest, a deep understanding of the database structure, and general IT expertise.

This is also very costly!

Equinor loses **50.000.000€** per year only due to this problem!!
# Outline

1. Challenges in Data Access
2. Virtual Knowledge Graphs for Data Access
3. Query Answering in VKGs
4. Mapping Patterns
5. Conclusions
Outline

1. Challenges in Data Access
2. Virtual Knowledge Graphs for Data Access
3. Query Answering in VKGs
4. Mapping Patterns
5. Conclusions
Solution: Virtual Knowledge Graphs (VKGs) – Also known as OBDA

Ontology $O$
- data is viewed as a graph, formulated in the vocabulary of the user

Mapping $M$
- specifies how to populate the ontology from the data

Data Sources $S$
- autonomous and heterogeneous

Greatly simplifies the access to information, and frees end-users from the need to know the precise structure of information sources.

Diego Calvanese (unibz + umu)
The choice of the right languages needs to take into account the tradeoff between expressive power and efficiency of query answering.

The W3C has standardized languages that are suitable for VKGs:

1. **Knowledge graph**: expressed in **RDF**  
   ([W3C Rec. 2014] (v1.1))

2. **Ontology** $\mathcal{O}$: expressed in **OWL 2 QL**  
   ([W3C Rec. 2012])

3. **Mapping** $\mathcal{M}$: expressed in **R2RML**  
   ([W3C Rec. 2012])

4. **Query**: expressed in **SPARQL**  
   ([W3C Rec. 2013] (v1.1))
RDF – Data is represented as a graph

The graph consists of a set of **subject-predicate-object triples**:

Object property:

<\text{A-1}> \text{ore:describes} <\text{ReM-1}> .

Data property:

<\text{ReM-1}> :\text{created} "2008-02-07" .

Class membership:

<\text{A-1}> \text{rdf:type} :\text{JournalArticle} .
The OWL 2 QL ontology language

- **OWL 2 QL** is one of the three standard profiles of OWL 2. [W3C Rec. 2012]

- Is considered a lightweight ontology language:
  - controlled expressive power
  - efficient inference

- Optimized for accessing large amounts of data [C., De Giacomo, et al. 2007]
  - Queries over the ontology can be rewritten into SQL queries over the underlying relational database (First-order rewritability).
  - Consistency of ontology and data can also be checked by executing SQL queries.
Main constructs of OWL 2 QL

Class hierarchy: rdfs:subClassOf \( (A_1 \sqsubseteq A_2) \)

Example: :MovieActor rdfs:subClassOf :Actor.

Inference: \(<\text{person}/2> \text{ rdf:type :MovieActor .} \)
\[\implies <\text{person}/2> \text{ rdf:type :Actor .} \]

Domain of properties: rdfs:domain \( (\exists P \sqsubseteq A) \)

Example: :playsIn rdfs:domain :MovieActor.

Inference: \(<\text{person}/2> :\text{playsIn} <\text{movie}/3> . \)
\[\implies <\text{person}/2> \text{ rdf:type :MovieActor .} \]

Range of properties: rdfs:range \( (\exists P^- \sqsubseteq A) \)

Example: :playsIn rdfs:range :Movie.

Inference: \(<\text{person}/2> :\text{playsIn} <\text{movie}/3> . \)
\[\implies <\text{movie}/3> \text{ rdf:type :Movie .} \]

...
Other constructs of OWL 2 QL

Class disjointness: `owl:disjointWith` \((A_1 \sqsubseteq \neg A_2)\)


Inference: `<person/2> rdf:type :Actor`.
   `<person/2> rdf:type :Movie`.
   \[\Rightarrow\] RDF graph inconsistent with the ontology

Inverse properties: `owl:inverseOf` \((P_1 \sqsubseteq P_2^- \text{ and } P_2 \sqsubseteq P_1^-)\)

Example: `:actsIn owl:inverseOf :hasActor`.

Inference: `<person/2> :actsIn <movie/3>`.
   \[\Rightarrow\] `<movie/3> :hasActor <person/2>`.

Property hierarchy

Property disjointness

Mandatory participation
There is a close correspondence between OWL 2 QL and conceptual modeling formalisms, such as UML class diagrams and ER schemas [Berardi, C., and De Giacomo 2005; Bergamaschi and Sartori 1992; Borgida 1995; C., Lenzerini, and Nardi 1999; Lenzerini and Nobili 1990; Queralt et al. 2012].

In fact, to visualize an OWL 2 QL ontology, we can use standard UML class diagrams.
SPARQL query language

- Is the standard query language for RDF data. [W3C Rec. 2008, 2013]
- Core query mechanism is based on **graph matching**.

```
SELECT ?a ?t
WHERE { ?a rdfs:typeof Actor .
  ?a playsIn ?m .
  ?m rdfs:typeof Movie .
  ?m rdfs:title ?t .
}
```

Additional language features (SPARQL 1.1):

- **UNION**: matches one of alternative graph patterns
- **OPTIONAL**: produces a match even when part of the pattern is missing
- complex **FILTER** conditions
- **GROUP BY**, to express aggregations
- **MINUS**, to remove possible solutions
- property paths (regular expressions)
- ...
Use of mappings

In VKGs, the **mapping** $M$ encodes how the data $D$ in the sources should be used to create the virtual knowledge graph.

**Virtual knowledge graph** $V$ defined from $M$ and $D$

- Queries are answered with respect to $O$ and $V$.
- The data of $V$ is not materialized (it is virtual!).
- Instead, the information in $O$ and $M$ is used to translate queries over $O$ into queries formulated over the sources.
- Advantage, compared to materialization: the graph is **always up to date** w.r.t. data sources.
### Mapping language

The **mapping** consists of a set of assertions of the form

\[
\Phi(\vec{x}) \mapsto t(\vec{x}) \text{ rdf:type } C \\
\Phi(\vec{x}) \mapsto t_1(\vec{x}) \ p \ t_2(\vec{x})
\]

where

- \(\Phi(\vec{x})\) is the **source query** in SQL,
- the **right hand side** is the **target**, consisting of a triple pattern involving an ontology concept \(C\) or a (data or object) property \(p\), and making use of the answer variables \(\vec{x}\) of the SQL query.

### Impedance mismatch: values in the DB vs. objects in the knowledge graph

In the **target**, we make use of **iri-templates** of the form \(t(\vec{x})\), which transform database values into object identifiers (IRIs) or literals.

**Note:** When convenient, we group multiple mapping assertions with the same **source query** into a single assertion where the **target** is a set of triple patterns.
Mapping language – Example

Ontology $O$:

![Ontology diagram]

Database $D$:

<table>
<thead>
<tr>
<th>MOVIE</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mcode</td>
<td>mtitle</td>
<td>myear</td>
<td>type</td>
<td></td>
</tr>
<tr>
<td>5118</td>
<td>The Matrix</td>
<td>1999</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>8234</td>
<td>Altered Carbon</td>
<td>2018</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>2281</td>
<td>Blade Runner</td>
<td>1982</td>
<td>m</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTOR</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pcode</td>
<td>acode</td>
<td>aname</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5118</td>
<td>438</td>
<td>K. Reeves</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5118</td>
<td>572</td>
<td>C.A. Moss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2281</td>
<td>271</td>
<td>H. Ford</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The mapping $M$ applied to database $D$ generates the (virtual) knowledge graph $V = M(D)$:

$m_1$: SELECT mcode, mtitle FROM MOVIE WHERE type = "m"

~~> :m/{mcode} rdf:type :Movie .
   :m/{mcode} :title {mtitle} .

$m_2$: SELECT M.mcode, A.acode FROM MOVIE M, ACTOR A WHERE M.mcode = A.pcode AND M.type = "m"

~~> :a/{acode} :playsIn :m/{mcode} .
Outline

1. Challenges in Data Access
2. Virtual Knowledge Graphs for Data Access
3. Query Answering in VKGs
4. Mapping Patterns
5. Conclusions

**VKG specification** $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ and **VKG instance** $\langle \mathcal{P}, \mathcal{D} \rangle$

- $\mathcal{O}$ is an ontology (expressed in OWL 2 QL),
- $\mathcal{M}$ is a set of (R2RML) mapping assertions,
- $\mathcal{S}$ is a (relational) database schema with integrity constraints,
- $\mathcal{D}$ is a database conforming to $\mathcal{S}$.

**Semantics** is specified in terms of first-order logic interpretations of the ontology.

A FOL interpretation $\mathcal{I}$ of the ontology predicates is a model of $\langle \mathcal{P}, \mathcal{D} \rangle$ if

- it satisfies all axioms in $\mathcal{O}$, and
- contains all facts in $\mathcal{M}(\mathcal{D})$, i.e., retrieved through $\mathcal{M}$ from $\mathcal{D}$.

Note:

- In general, $\langle \mathcal{P}, \mathcal{D} \rangle$ has infinitely many models, and some of these might be infinite.
- However, for query answering, we do not need to compute such models.
In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKG and a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$. 

**Certain answers**

Given a VKG instance $\mathcal{J}$ and a query $q$ over $\mathcal{J}$, the certain answers to $q$ are those answers that hold in all models of $\mathcal{J}$. 
First-order rewritability

To make computing certain answers viable in practice, the VKG setting relies on reducing it to evaluating SQL (i.e., first-order logic) queries over the data.

Consider a VKG specification $\mathcal{P} = \langle O, M, S \rangle$.

First-order rewritability [Poggi et al. 2008]

A query $r(\vec{x})$ is a **first-order rewriting** of a query $q(\vec{x})$ with respect to $\mathcal{P}$ if, for every source DB $\mathcal{D}$, certain answers to $q(\vec{x})$ over $\langle \mathcal{P}, \mathcal{D} \rangle$ = answers to $r(\vec{x})$ over $\mathcal{D}$.

For OWL 2 QL ontologies and R2RML mappings, (core) SPARQL queries are first-order rewritable.

In other words, in VKGs, we can compute the certain answers to a SPARQL query by evaluating over the sources its rewriting, which is a SQL query.
Query answering by query rewriting

- Ontological Query $q$
- Rewritten Query
- Ontological Answer
- Relational Answer
- Result Translation
- Unfolding
- SQL
- Evaluation
- Mappings
- Data Sources
Ontop [C., Cogrel, et al. 2017]

- State-of-the-art VKG system
- Compliant with the RDFS, OWL 2 QL, R2RML, and SPARQL standards.
- Supports all major relational DBs
  - Oracle, DB2, MS SQL Server, Postgres, MySQL, Denodo, Dremio, Teiid, etc.
- **Open-source** and released under Apache 2 license
- Development of **Ontop**:
  - Development started in 2009
  - Already well established:
    - +200 members in the mailing list
    - +9000 downloads in the last 2 years
  - Main development carried out in the context of several local, national, and EU projects
Architecture of **Ontop**

**Application Layer**
- Protege
- RDF4J Workbench & SPARQL Endpoint
- CLI (Docker)

**API Layer**
- OWL API
- RDF4J API

**Ontop Core**
- Ontop SPARQL Query Answering Engine
- OWL API (OWL Parser)
- R2RML API
- JDBC
- RDF4J API (SPARQL Parser)

**Inputs**
- OWL 2 QL Ontologies
- R2RML Mappings
- Relational Databases
- SPARQL Queries

---

Diego Calvanese (unibz + umu)

Mapping Patterns for Virtual Knowledge Graphs

AIKE – 9/12/2020 (24/46)
Virtual approach for query answering in *Ontop*

![Diagram of the virtual approach for query answering in *Ontop*]

1. **SPARQL parsing**
2. **Rewriting**
3. **Unfolding w.r.t. mappings**
4. **Structural/semantic optimization**

**OFFLINE**

1. **Ontology parsing**
2. **Ontology classification**
3. **Mapping compilation**
4. **Mapping analysis**

**Online**

1. **Post-processing**
2. **Evaluation**
3. **RA-to-native query translation**
4. **Normalization/Decomposition**

**Inputs:**
- **Ontology file**
- **SPARQL string**
- **SPARQL Q**
- **q1, q2, q3**

**Outputs:**
- **Native results**
- **Native queries**
- **RA result**
Some use cases of *Ontop* – Research projects

- EU FP7 project **Optique** “Scalable End-user Access to Big Data” (11/2012 – 10/2016)
  - 10 Partners, including industrial partners **Statoil, Siemens, DNV**.
  - *Ontop* is core component of the Optique platform.

- EU project **EPNet** (ERC Advanced Grant) “Production and distribution of food during the Roman Empire: Economics and Political Dynamics”
  - Access to data in the cultural heritage domain.

- Euregio funded project **KAOS** “Knowledge-aware Operational Support” (06/2016 – 05/2019)
  - Preparation of standardized log files from timestamped log data for the purpose of process mining.
  

- EU H2020 project **INODE** “Intelligent Open Data Exploration” (11/2019 – 10/2022)
  - Development of techniques for the flexible interaction with data.
Commercial use cases of *Ontop* in which we are involved

- **NOI Techpark in Bolzano** – Development of knowledge graph of South Tyrol data [Ding et al. 2020]
  - Tourism data
  - Mobility data

- **Collaboration with SIRIS Academic (Barcelona)** – Development of data integration and dashboards for data analysis over open data from public institutions
  - Tuscany’s Observatory of Research and Innovation
  - Sorbonne University


See [Xiao, Ding, et al. 2019] for a survey on VKG systems and use cases.
First spin-off of the Free University of Bozen-Bolzano.

Incorporated in April 2019.

Product: *Ontopic Suite* based on the *Ontop* engine.

Services around *Ontop* and the *Ontopic Suite*.

Consultancy for VKG-based data integration projects.
Outline

1. Challenges in Data Access

2. Virtual Knowledge Graphs for Data Access

3. Query Answering in VKGs

4. Mapping Patterns

5. Conclusions
VKG mappings:

- Map complex queries to complex queries – cf. GLAV relational mappings [Lenzerini 2002].
- Overcome the abstraction mismatch between relational data and target ontology.
- Are inherently more sophisticated than mappings of schema matching [Rahm and Bernstein 2001] and ontology matching [Euzenat and Shvaiko 2007].

As a consequence

- Management of VKG mappings is a labor-intensive, essentially manual effort.
- Requires highly-skilled professionals [Spanos, Stavrou, and Mitrou 2012].
- Writing mappings is challenging in terms of semantics, correctness, and performance.

Designing and managing mappings is the most critical bottleneck for the adoption of the VKG approach.
VKG mapping patterns

Several approaches supporting the creation of mappings have been proposed, several of them based on patterns.

But there is no comprehensive approach for VKG mapping patterns exploiting all of:

- the relational schema with its constraints
- extensional data stored in the DB
- the TBox axioms that constitute the ontology
- the conceptual schema at the basis of the relational schema

I would like to address this issue by presenting:

- a catalog of VKG mapping patterns,
- design scenarios for VKG mapping patterns, and
- an analysis of practical VKG use cases wrt the use of mapping patterns.
Catalog of mapping patterns

We build on well-established methodologies and patterns studied in:

- data management – e.g., W3C Direct Mapping Specification [Arenas et al. 2012] and extensions
- data analysis – e.g., algorithms for discovering dependencies, and
- conceptual modeling

In specifying each pattern, we consider:

- the three components of a VKG specification: DB schema, ontology, mapping between the two;
- the conceptual schema of the domain of interest;
- underlying data, when available.

We do not fix what is given as input and what is produced as output, but we simply describe how the elements relate to each other, on a per-pattern basis.
Types of mapping patterns

Patterns are organized in two major groups:

**Schema-driven patterns**
Are shaped by the structure of the DB schema and its explicit constraints.

**Data-driven patterns**
- Consider also constraints emerging from specific configurations of the data in the DB.
- For each schema-driven pattern, we identify a data-driven version: The constraints over the schema are not explicitly specified, but hold in the data.
- We provide also data-driven patterns that do not have a schema-driven counterpart.

- We use also additional semantic information from the ontology \(\sim\) **Pattern modifiers**
- Some patterns come with **views over the DB-schema**:
  - Views reveal structures over the DB-schema, when the pattern is applied.
  - Views can be used to identify the applicability of further patterns.
Constraints on the data

When defining the mapping patterns, we consider the following types of constraints:

- **Primary key constraint**: $T(K, A)$
- **Key constraint**: $key_T(K)$
- **Foreign key constraint**: $T_1[A] \subseteq T_2[K]$, where $K$ is a (typically primary) key of relation $T_2$.

We use the notation:

$$T_1(A, B) \rightarrow T_2(K, A')$$

*Note*: Normal math font (e.g., $A$) is used for single attributes, while boldface is used for sets of attributes (e.g., $A$ or $K$).
Types of mapping patterns

We have identified several different mapping patterns:

1. Entity (MpE)
2. Relationship (MpR)
3. Relationship with Identifier Alignment (MpRa)
4. Relationship with Merging (MpRm)
5. 1-1 Relationship with Merging (MpR11m)
6. Reified Relationship (MpRR)
7. Hierarchy (MpH)
8. Hierarchy with Identifier Alignment (MpHa)
9. Clustering Entity to Class / Data Property / Object Property (MpCE2X)

Each mapping pattern is specified through four different components that we illustrate on an example.

**Note:** The patterns make use of IRI-templates of the form “:\E/\{K\}”, where we assume that “:\E/” is a prefix that is specific for the instances of a class \(C_E\).
Mapping pattern: Entity (MpE)

Relational schema and constraints:

\[ T_E(\underbrace{\text{K}}, \underbrace{\text{A}}) \]

Mapping assertion:

\[
\begin{align*}
  s : T_E \\
  t : & :E/\{\text{K}\} \text{ rdf:type } C_E . \\
  & \{ :E/\{\text{K}\} \ d_A \ \{A\} . \} \_{A \in \text{K} \cup \text{A}}
\end{align*}
\]

Ontology axioms:

\{ \exists d_A \sqsubseteq C_E \}_{A \in \text{K} \cup \text{A}}

For the application of the mapping pattern, we observe the following:

- This fundamental pattern considers a single table \( T_E \) with primary key \( \text{K} \) and other relevant attributes \( \text{A} \).
- The pattern captures how \( T_E \) is mapped into a corresponding class \( C_E \).
- The primary key \( \text{K} \) of \( T_E \) is used to construct the objects that are instances of \( C_E \), using a template \( :E/\{\text{K}\} \) specific for \( C_E \).
- Each relevant attribute of \( T_E \) is mapped to a data property of \( C_E \).
Mapping pattern: Entity (MpE) – Example

Consider a TClient table containing ssns of clients, together with name, dateOfBirth, and hobbies as additional attributes.

\[
\text{TClient}(\text{ssn}, \text{name}, \text{dateOfBirth}, \text{hobbies})
\]

Mapping: TClient is mapped to a Client class using the attributes ssn to construct its objects. In addition, the ssn, name, and dateOfBirth are used to populate in the object position the three data properties ssn, name, and dob, respectively. The attribute hobbies is ignored.

\[
\text{mappingId} \quad \text{MClient} \\
\text{source} \quad \text{SELECT ssn, name, dateOfBirth FROM TClient} \\
\text{target} \quad :C/{\text{ssn}} \text{ rdf:type :Client ;} \\
\quad \quad :\text{ssn} \ {\text{ssn}} ; \\
\quad \quad :\text{name} \ {\text{name}} ; \\
\quad \quad :\text{dob} \ {\text{dateOfBirth}} .
\]
Mapping pattern: Relationship (MpR)

Relational schema and constraints:

\[
\begin{align*}
T_E(K_E, A_E) & \quad T_F(K_F, A_F) \\
T_R(K_{RE}, K_{RF}) & \quad \uparrow
\end{align*}
\]

Mapping assertion:

\[
\begin{align*}
s : T_R \\
t : :E/\{K_{RE}\} \quad p_R : F/\{K_{RF}\}.
\end{align*}
\]

Ontology axioms:

\[
\begin{align*}
\exists p_R \sqsubseteq C_E \\
\exists p_R^- \sqsubseteq C_F
\end{align*}
\]

For the application of the mapping pattern, we observe the following:

- This pattern considers three tables \(T_R\), \(T_E\), and \(T_F\).
- The primary key of \(T_R\) is partitioned into two parts \(K_{RE}\) and \(K_{RF}\) that are foreign keys to \(T_E\) and \(T_F\), respectively.
- \(T_R\) has no additional (relevant) attributes.
- The pattern captures how \(T_R\) is mapped to an object property \(p_R\), using the two parts \(K_{RE}\) and \(K_{RF}\) of the primary key to construct respectively the subject and the object of the triples in \(p_R\).
An additional `TAddress` table in the client registry stores the addresses at which each client can be reached, and such table has a foreign key to a table `TLocation` storing locations using attributes `city` and `street`.

```
TClient(ssn, name, dateOfBirth, hobbies)
TLocation(city, street)
TAddress(client, locCity, locStreet)
FK: TAddress[client] -> TClient[ssn]
FK: TAddress[locCity, locStreet] -> TLocation[city, street]
```

**Mapping:** The `TAddress` table is mapped to an `address` object property, for which the ontology asserts that the domain is the class `Person` and the range an additional class `Location`, corresponding to the `TLocation` table.

```
mappingId  MAddress
source      SELECT client, locCity, locStreet FROM TAddress
target      :C/{client} :address :L/{locCity}/{locStreet} .
```
Mapping pattern: Relationship with Identifier Alignment (MpRa)

Relational schema and constraints:

\[
\begin{align*}
T_E(K_E, A_E) & \quad T_F(K_F, U_F, A_F) \\
T_R(K_{RE}, U_{RF}) & \quad \text{key}_{RF}(U_F)
\end{align*}
\]

Mapping assertion:

\[
\begin{align*}
&s : T_R \bowtie u_{RF} = u_F T_F \\
t & : E/\{K_{RE}\} \quad p_R : F/\{K_F\}
\end{align*}
\]

Ontology axioms:

\[
\begin{align*}
\exists p_R \sqsubseteq C_E \\
\exists p_R^- \sqsubseteq C_F
\end{align*}
\]

For the application of the mapping pattern, we observe the following:

- Such pattern is a variation of pattern MpR, in which the foreign key in \( T_R \) does not point to the primary key \( K_F \) of \( T_F \), but to an additional key \( U_F \).
- Since the instances of class \( C_F \) corresponding to \( T_F \) are constructed using the primary key \( K_F \) of \( T_F \) (cf. pattern MpE), also the pairs that populate \( p_R \) should refer in their object position to \( K_F \).
- Note that \( K_F \) can only be retrieved by a join between \( T_R \) and \( T_F \) on the additional key \( U_F \).
Mapping pattern: Rel. with Identifier Alignment (MpRa) – Example

The primary key of the table TLocationCoord is now not given by the city and street, which are used in the table TAddress that relates clients to their addresses, but is given by the latitude and longitude of locations.

```
TClient(ssn, name, dateOfBirth, hobbies)
TLocationCoord(latitude, longitude, city, street)  # key[TLocation]: city, street
TAddress(client, locCity, locStreet)
```

FK: TAddress[client] -> TClient[ssn]
FK: TAddress[locCity,locStreet] -> TLocationCoord[city,street]

**Mapping:** The Address table is mapped to an address object property, for which the ontology asserts that the domain is the class Person and the range an additional class Location, corresponding to the Location table.

```
mappingId  MAddressCoord
source      SELECT client, latitude, longitude
            FROM TAddress JOIN TLocationCoord ON locCity = city AND locStreet = street
target      C/{client} :address :LC/{latitude}/{longitude}.
```

Diego Calvanese (unibz + umu)
Design scenarios for VKG mapping patterns

Depending on what information is available, we can consider different design scenarios where the patterns can be applied:

1. **Debugging of a VKG specification** that is already in place.

2. **Conceptual schema reverse engineering** for a DB that represents the domain of interest by using a given full VKG specification.

3. **Mapping bootstrapping** for a given DB and ontology that miss the mappings relating them.

4. **Ontology + mapping bootstrapping** from a given DB with constraints, and possibly a conceptual schema.

5. **VKG bootstrapping**, where the goal is to set up a full VKG specification from a conceptual schema of the domain.
Analysis of VKG use cases

We have analyzed the concrete mapping strategies in a number of VKG use cases:

- Understand how patterns occur in practice.
- Understand with which frequency patterns are used.

We have considered 6 different scenarios:

- Berlin Sparql Benchmark [Bizer and Schultz 2009]
- NPD Benchmark [Lanti et al. 2015]
- University Ontology Benchmark [Zhou et al. 2013]
- Südtirol OpenData  https://github.com/dinglinfang/suedTirolOpenDataOBDA
- Open Data Hub VKG  https://sparql.opendatahub.bz.it/
- Cordis  https://www.sirisacademic.com/wb/

We have (manually) classified 1582 mapping assertions, falling in 367 pattern applications.
## Occurrences of mapping patterns over the scenarios

<table>
<thead>
<tr>
<th></th>
<th>BSBM</th>
<th>NPD</th>
<th>UOBM</th>
<th>ODH</th>
<th>ST-OD</th>
<th>Cordis</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>8</td>
<td>52</td>
<td>57</td>
<td>505</td>
<td></td>
<td></td>
<td>102</td>
</tr>
<tr>
<td>SR</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>SRm</td>
<td>8</td>
<td>8</td>
<td>45</td>
<td>45</td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>SRR</td>
<td>–</td>
<td>–</td>
<td>1</td>
<td>12</td>
<td></td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>SH</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>132</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>DE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>5</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>DRm</td>
<td>5</td>
<td>5</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>36</td>
<td>63</td>
</tr>
<tr>
<td>DH</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>5</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>DRR</td>
<td>2</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2</td>
</tr>
<tr>
<td>DR1Nm</td>
<td>4</td>
<td>4</td>
<td>19</td>
<td>54</td>
<td>–</td>
<td>9</td>
<td>33</td>
</tr>
<tr>
<td>D11Rm</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>6</td>
<td>78</td>
<td>11</td>
</tr>
<tr>
<td>DH0N</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>CE2C</td>
<td>–</td>
<td>–</td>
<td>11</td>
<td>82</td>
<td>6</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>CE2D</td>
<td>–</td>
<td>–</td>
<td>23</td>
<td>49</td>
<td>5</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>CE2O</td>
<td>–</td>
<td>–</td>
<td>13</td>
<td>148</td>
<td>–</td>
<td>–</td>
<td>13</td>
</tr>
<tr>
<td>CR2O</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>unknown</td>
<td>–</td>
<td>–</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Left number: number of applications of a pattern in a scenario  
Right number: number of mappings involved
Outline

1. Challenges in Data Access
2. Virtual Knowledge Graphs for Data Access
3. Query Answering in VKGs
4. Mapping Patterns
5. Conclusions
Conclusions

• VKGs are by now a mature technology to address the data access and integration problems.

• This paradigm has been well-investigated and applied in real-world scenarios (mostly for the case of relational data sources).

• Performance and scalability of query answering for data access w.r.t. large datasets (volume), large and complex ontologies (variety, veracity), and multiple heterogeneous data sources (variety, volume) are a challenge, but sophisticated techniques and tools are available that achieve a good performance.

• However, the design of the ontology and notably the mapping layer is still a largely manual, labor-intensive, and error-prone activity.

• This represents a major bottleneck for a wider adoption of the VKG approach in real-world scenarios.

• Principled solutions based on well-established conceptual modeling principles, e.g., realized through mapping patterns look promising to address this challenge.
Ongoing work

We are currently developing technologies to make the VKG paradigm easier to adopt:

- Mapping patterns are at the basis of techniques for (semi-)automatic extraction/learning of ontology axioms and mapping assertions.
- We are working on automated support for the efficient management of mappings and ontology axioms, to support design, maintenance, and evolution. Ontopic is developing specific tools for this.
- In addition, we are working on improving the support for multiple, heterogeneous data sources.

In parallel, we are working on foundational and applied issues to enrich the VKG setting:

- More expressive queries, supporting analytical tasks.
- Support for additional types of data sources, such as graph data, geospatial data, temporal data.
- Dealing with data provenance and explanation.
- Dealing with data inconsistency and incompleteness – Data quality!
- Ontology-based update.
Conclusions and ongoing work

• We have identified a catalog of mapping patterns for the VKG framework.

• We have validated the patterns on a set of real-world use cases.

• We are currently working on using our patterns for mapping (and ontology) bootstrapping and generation.
Thank you!

- E: calvanese@inf.unibz.it
- H: http://www.inf.unibz.it/~calvanese/

**Ontop** website: https://ontop-vkg.org/
- Github: http://github.com/ontop/ontop/
- Facebook: https://www.facebook.com/obdaontop/
- Twitter: @ontop4obda
- **Ontopic** website: https://ontopic.biz/
A great thank you to all my collaborators
References


