Database Optimization Techniques for Semantic Queries

Ioana Manolescu

INRIA, France
ioana.manolescu@inria.fr,
http://pages.saclay.inria.fr/Ioana.Manolescu
https://team.inria.fr/oak

SEBD 2015, Gaeta
Part I

Motivation and outline
Motivation and context

Querying Semantic data

Most popular data model: **RDF** (W3C’s Resource Description Framework)
- Famous application: the Linked Open Data cloud
Querying Semantic data

Most popular data model: **RDF** (W3C’s Resource Description Framework)

- Famous application: the Linked Open Data cloud (2007)
Motivation and context

Querying Semantic data

Most popular data model: **RDF** (W3C’s Resource Description Framework)

- Famous application: the Linked Open Data cloud (2008)
Querying Semantic data

Most popular data model: **RDF** (W3C’s Resource Description Framework)

- Famous application: the Linked Open Data cloud (2009)
Querying Semantic data

Most popular data model: **RDF** (W3C’s Resource Description Framework)

- Famous application: the Linked Open Data cloud (2010)
Motivation and context

Linked Open Data cloud, 2014

900,129 documents describing 8,038,396 resources
(Schmachtenberg, Bizer, Paulheim, ISWC 2014)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Datasets</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government</td>
<td>183</td>
<td>18.05</td>
</tr>
<tr>
<td>Publications</td>
<td>96</td>
<td>9.47</td>
</tr>
<tr>
<td>Life sciences</td>
<td>83</td>
<td>8.19</td>
</tr>
<tr>
<td>User-generated</td>
<td>48</td>
<td>4.73</td>
</tr>
<tr>
<td>Cross-domain</td>
<td>41</td>
<td>4.04</td>
</tr>
<tr>
<td>Media</td>
<td>22</td>
<td>2.17</td>
</tr>
<tr>
<td>Geographic</td>
<td>21</td>
<td>2.07</td>
</tr>
<tr>
<td>Social web</td>
<td>520</td>
<td>51.28</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1014</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

There is more (Billion Triple Challenge etc.)
RDF: three-attribute relation (subject, property, object)

- The subject is the **resource** being **described**
- The resource has the property **property** whose value is **object**
- Resource **type** is a property, specified just like any other
Querying Semantic Data

1. **RDF**: three-attribute relation (subject, property, object)
   - The subject is the **resource** being **described**
   - The resource has the property **property** whose value is **object**
   - Resource **type** is a property, specified just like any other

2. **RDFS semantics** providing information about the properties and classes (types) of resources:
   - Any **undergraduateStudent** is a **Student**
   - Anyone having a **graduationDate** is a **Student** (but may also be of other types) ...
1. RDF: three-attribute relation (subject, property, object)
   - The subject is the resource being described
   - The resource has the property whose value is
   - Resource type is a property, specified just like any other

2. RDFS semantics providing information about the properties and classes (types) of resources:
   - Any undergraduateStudent is a Student
   - Anyone having a graduationDate is a Student (but may also be of other types) ...

Semantics leads to implicit data
Querying Semantic Data

Main RDFS constraints: rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range

Beyond RDFS

Many (richer) constraint (or ontology) languages

- W3C standards: OWL 2 profile family
- DL Lite family
  - Calvanese, De Giacomo, Lembo, Lenzerini, Rosati, JAR 2007
- Datalog$^\pm$
  - Cali, Gottlob Lukasiewick, Marnette, Pierris, LICS 2010
Motivation and context

Do we really need the semantics?

Yes. All the time.

Application knowledge / constraints:

- Every Senator is an ElectedOfficial which is a Person
- (On Wikipedia) being BornInAPlace means being a Person
- The source and destination of a tripFromTo are either a streetAddress, or a cityAddress or a countryAddress
Do we really need the semantics?

Yes. All the time.

Application knowledge / constraints:

- Every Senator is an ElectedOfficial which is a Person
- (On Wikipedia) being BornInAPlace means being a Person
- The source and destination of a tripFromTo are either a streetAddress, or a cityAddress or a countryAddress

1. Without the semantics, we may miss query answers
2. Semantic contraints are a compact way of encoding information ("every ElectedOfficial is a Person" stated only once)
Outline

1. Motivation
2. RDF data model and query language
3. Query answering techniques
4. Cover-based query reformulation for the database fragment of RDF
5. Cover-based query reformulation framework for FOL-reducible settings
6. Performance and concluding remarks
Part II

RDF data and queries
The Resource Description Framework (RDF)

**RDF graph** – set of triples

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Triple</th>
<th>Relational notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>s rdf:type o</td>
<td>o(s)</td>
</tr>
<tr>
<td>Property</td>
<td>s p o</td>
<td>p(s, o)</td>
</tr>
</tbody>
</table>

- Resource (URI): blank node
- Literal (string): property

Example:
- doi1
  - hasTitle: "El Aleph"
  - publishedIn: "1949"
  - writtenBy: "J. L. Borges"
- _:b1
  - hasName: "J. L. Borges"
RDF Schema (RDFS)

Declare **deductive constraints** between classes and properties

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Triple</th>
<th>OWA interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass</td>
<td>s rdfs:subClassOf o</td>
<td>s ⊆ o</td>
</tr>
<tr>
<td>Subproperty</td>
<td>s rdfs:subPropertyOf o</td>
<td>s ⊆ o</td>
</tr>
<tr>
<td>Domain typing</td>
<td>s rdfs:domain o</td>
<td>Π_domain(s) ⊆ o</td>
</tr>
<tr>
<td>Range typing</td>
<td>s rdfs:range o</td>
<td>Π_range(s) ⊆ o</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Publication**
  - rdfs:subClassOf
    - **Book**
      - rdfs:domain
        - **writtenBy**
          - rdfs:subPropertyOf
            - **Person**
      - rdfs:range
Open-world assumption and RDF entailment

RDF data model – based on the open-world assumption. → deductive constraints – implicitly propagate triples

Implicit triples: part of the graph – not explicitly present

Entailment – reasoning mechanism
Open-world assumption and RDF entailment

RDF data model – based on the open-world assumption.

→ deductive constraints – implicitly propagate triples

Implicit triples: part of the graph – not explicitly present

Entailment – reasoning mechanism

<table>
<thead>
<tr>
<th>set of explicit triples</th>
<th>+</th>
<th>derive implicit triples</th>
<th>some entailment rules</th>
</tr>
</thead>
</table>

Exhaustive application of entailment → saturation (closure)
Open-world assumption and RDF entailment

RDF data model – based on the open-world assumption. → deductive constraints – implicitly propagate triples

Implicit triples: part of the graph – not explicitly present

Entailment – reasoning mechanism

\[
\begin{array}{c}
\text{set of explicit triples} \\
+ \\
\text{some entailment rules}
\end{array} \rightarrow \text{derive implicit triples}
\]

Exhaustive application of entailment → saturation (closure)
The semantics of an RDF graph $G$ is its saturation $G^\infty$.

<table>
<thead>
<tr>
<th>Sample RDFS entailment rules</th>
<th>Instance entailment from combining schema and instance triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs9</td>
<td>$c_1 \ rdfs\text{:subClassOf} \ c_2 \land s \ rdf\text{:type} \ c_1 \vdash_{\text{RDF}} s \ rdf\text{:type} \ c_2$</td>
</tr>
<tr>
<td>rdfs7</td>
<td>$p_1 \ rdfs\text{:subPropertyOf} \ p_2 \land s \ p_1 \ o \vdash_{\text{RDF}} s \ p_2 \ o$</td>
</tr>
<tr>
<td>rdfs2</td>
<td>$p \ rdfs\text{:domain} \ c \land s \ p \ o \vdash_{\text{RDF}} s \ rdf\text{:type} \ c$</td>
</tr>
<tr>
<td>rdfs3</td>
<td>$p \ rdfs\text{:range} \ c \land s \ p \ o \vdash_{\text{RDF}} o \ rdf\text{:type} \ c$</td>
</tr>
</tbody>
</table>
SPARQL query language and SPARQL conjunctive queries

SPARQL is the W3C query language for RDF.

**SPARQL conjunctive queries** = Basic Graph Pattern (BGP) queries

Sample BGP query:

\[
q(a, t) \; :- \; (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")
\]
The evaluation of a query only uses the graph’s **explicit triples**

For the **(complete) answer set**, evaluate $q$ against the graph’s saturation
Query answering example

Given the query $q(x, y) :- x \text{ rdf:type } y$:
Given the query $q(x, y) : \neg x \text{ rdf:type } y$:

- $q(G) = \{(\text{doi}_1, \text{Book})\}$
Query answering example

Given the query \( q(x, y) :\quad x \text{ rdf:type } y \):

\[
q(G) = \{ (\text{doi}_1, \text{Book}) \}
\]

\[
q(G^\infty) = \{ (\text{doi}_1, \text{Book}), (\text{doi}_1, \text{Publication}), (\_:b_1, \text{Person}) \}
\]
Part III

Query answering techniques
The need for reasoning

Query answering needs explicit and implicit data!

- **Saturation**-based query answering (when the saturation result is finite)
- **Reformulation**-based query answering
- Hybrids of the above
  - J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal, “QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases”, ISWC 2011
Saturation-based query answering

Query answering techniques

Graph saturation

Schema

G∞

Query q

Answer

G

G∞ needs time to be computed and space to be stored. Not suitable for high update rate (data and/or schema triples).
Saturation-based query answering

- $q(G^\infty)$ can be computed using an RDBMS
- $G^\infty$ needs time to be computed and space to be stored
- Not suitable for high update rate (data and/or schema triples)
Saturation maintenance

Compute $\Delta$ for an update of an RDF graph $G$ s.t.

- $(\mu(G))^\infty = G^\infty \cup \Delta$ when $\mu$ an insertion
- $(\mu(G))^\infty = G^\infty \setminus \Delta$ when $\mu$ a deletion
Reformulation-based query answering

\[ q_{\text{ref}}(G) \] can be evaluated using an RDBMS.

Robust to updates.

Reformulated queries are complex, thus costly to evaluate.
Reformulation-based query answering

- $q^{ref}(G)$ can be evaluated using an RDBMS
- Robust to updates
- Reformulated queries are complex, thus costly to evaluate
Reformulation-based query answering
Reformulation-based query answering

- $q^{ref}(G)$ can be evaluated using an RDBMS
- Robust to updates
- Reformulated queries are complex, thus costly to evaluate
The logical storage of the data is related to the ontology by means of mappings.

A mapping states that a query on the database is included into a query on the ontology.

Reformulation-based query answering in the presence of (GAV) mappings:

- Lanti, Rezk, Xiao and Calvanese, EDBT 2015

1. Start-up phase (may perform some materialization)
2. Query rewriting phase (based on the schema)
3. Query translation phase (e.g., into SQL, based on the mappings)
4. Query execution phase (e.g., through an RDBMS)
Semantic query answering in the presence of mappings

- Reformulation-based query answering in the presence of (GAV) mappings:
  - Lanti, Rezk, Xiao and Calvanese, EDBT 2015
  1. Start-up phase (may perform some materialization)
  2. Query rewriting phase (based on the schema)
  3. Query translation phase (e.g., into SQL, based on the mappings)
  4. Query execution phase (e.g., through an RDBMS)
Semantic query answering in the presence of mappings

- Reformulation-based query answering in the presence of (GAV) mappings:
  - Lanti, Rezk, Xiao and Calvanese, EDBT 2015
  1. Start-up phase (may perform some materialization)
  2. Query rewriting phase (based on the schema)
  3. Query translation phase (e.g., into SQL, based on the mappings)
  4. Query execution phase (e.g., through an RDBMS)

**This talk:** Optimized reformulation technique based on considering translation and rewriting together

Can be composed with mappings (and further mapping-based optimizations)
 Semantic query answering vs. Ontology-Based Data Access (OBDA)

Fundamentally the same

\textbf{Input} data (stored somehow), ontology, query

\textbf{Output} query answer

In general, mappings may be GAV, LAV, GLAV etc.
LAV can be very practical (see: RDBMS indexes!)

\textbf{We optimize query rewriting and translation independently of the global $\leftrightarrow$ local (source) data mappings}

Integration/interactions are intriguing and current/future work
Target reformulation languages for conjunctive queries (CQs):

- **mainly unions of CQs (UCQs)**
  - F. Goasdoué, I. Manolescu, A. Roatiş: “Efficient query answering against dynamic RDF databases”, EDBT 2013

- **joins of single-atom CQs (SCQs)**

- **joins of UCQs (JUCQs)**
Reformulation-based query answering landscape

Target reformulation languages for conjunctive queries (CQs):

- mainly **unions of CQs (UCQs)**
  - F. Goasdoué, I. Manolescu, A. Roatiş: “Efficient query answering against dynamic RDF databases”, EDBT 2013

**Wait: is this about SQL syntax?!...**

- **joins of UCQs (JUCQs)**
Reformulation-based query answering landscape

Target reformulation languages for conjunctive queries (CQs):

Wait: is this about SQL syntax?!…

Yes. And it makes a big difference.

From failing to feasible, or 4 orders of magnitude speed-up on the 8 M triples DBLP dataset.
Part IV

Cover-based reformulation for the database fragment of RDF
The database fragment of RDF

Largest RDF fragment for which an FOL reformulation-based query answering technique is known

- Restrict RDF entailment to the rules dedicated to RDF Schema only (a.k.a. RDFS entailment).
- No restriction to graphs: any triple allowed by the RDF specification is also allowed in the DB fragment (e.g., blank nodes in data and schema)

Algorithm Reformulate\((q, db)\) for BGP queries over schema and data. Output size bound \(O((6 \times \#db^2)\#q)\)

▷ Goasdoué, Manolescu, Roatiş, EDBT 2013
CQ-to-UCQ query reformulation example

\[ q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \] leads to \( q^{\text{ref}} : \)

\begin{align*}
(0) \quad q(a, t) & :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")
\end{align*}
CQ-to-UCQ query reformulation example

$q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")$ leads to $q^{ref}$:

(0) $q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")$  
(1) $q(a, t) :- (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")$
CQ-to-UCQ query reformulation example

\[ q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \] leads to \( q^{\text{ref}} : \)

\[
\begin{align*}
(0) \quad q(a, t) & \quad : (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), \\
& \quad (b, \text{publishedIn}, "1949") \\
(1) \quad q(a, t) & \quad : (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), \\
& \quad (b, \text{publishedIn}, "1949") \\
(2) \quad q(a, t) & \quad : (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), \\
& \quad (b, \text{publishedIn}, "1949")
\end{align*}
\]
CQ-to-UCQ query reformulation example

$q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a),
(b, \text{publishedIn}, "1949")$ leads to $q^{ref}$:

(0) \[ q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a),
(b, \text{publishedIn}, "1949") \]

(1) \[ q(a, t) : - (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a),
(b, \text{publishedIn}, "1949") \]

(2) \[ q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a),
(b, \text{publishedIn}, "1949") \]

(3) \[ q(a, t) : - (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a),
(b, \text{publishedIn}, "1949") \]
CQ-to-UCQ query reformulation example

$q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")$ leads to $q^{\text{ref}}$:

(0) $q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \\
 U$

(1) $q(a, t) :- (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \\
 U$

(2) $q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), (b, \text{publishedIn}, "1949") \\
 U$

(3) $q(a, t) :- (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), (b, \text{publishedIn}, "1949")$
CQ-to-UCQ query reformulation example

\[ q(a, t) :- (b, \text{rdf\:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \] leads to \( q^{\text{ref}} \):

\[
\begin{align*}
(0) & \quad q(a, t) :- (b, \text{rdf\:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \\
(1) & \quad q(a, t) :- (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \\
(2) & \quad q(a, t) :- (b, \text{rdf\:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), (b, \text{publishedIn}, "1949") \\
(3) & \quad q(a, t) :- (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), (b, \text{publishedIn}, "1949")
\end{align*}
\]

\[ q(G^{\infty}) = q^{\text{ref}}(G) = \{(\_b_1, "El\ Aleph")\}. \]
CQ-to-SCQ query reformulation example

\[ q(a, t):- (b, \text{rdf:} \text{type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, } "1949") \] produces the \( q^{\text{ref}}: \)

(0) \quad q(b) :- (b, \text{rdf:} \text{type}, \text{Book}) \cup (b, \text{writtenBy}, x) \checkmark 

(1) \quad q(b, a) :- (b, \text{hasAuthor, } a) \cup (b, \text{writtenBy}, a) \checkmark 

(2) \quad q(b) :- (b, \text{publishedIn, } "1949")
Cover-based reformulation for the DB fragment of RDF

CQ-to-SCQ query reformulation example

$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, } "1949")$ produces the $q^{\text{ref}}$:

(0) $q(b) :- (b, \text{rdf:type, Book}) \cup (b, \text{writtenBy, } x)$  
(1) $q(b, a) :- (b, \text{hasAuthor, } a) \cup (b, \text{writtenBy, } a)$  
(2) $q(b) :- (b, \text{publishedIn, } "1949")$

$q(G^{\infty}) = q^{\text{ref}}(G) = \{(\_:_b_1, "El Aleph")\}$. 

"1949"  
"El Aleph"  
"J. L. Borges"
CQ-to-JUCQ query reformulation

1. Enlarges the query reformulation language w.r.t. UCQ/SCQ to have more than one reformulation alternative
CQ-to-JUCQ query reformulation

1. Enlarges the query reformulation language w.r.t. UCQ/SCQ to have more than one reformulation alternative
2. Uses a cost model for estimating the cost of evaluating $q^{ref}$ through an RDBMS
CQ-to-JUCQ query reformulation

1. Enlarges the **query reformulation language** w.r.t. UCQ/SCQ to have more than one reformulation alternative
2. **Uses a cost model** for estimating the cost of evaluating $q^{\text{ref}}$ through an RDBMS
3. **Chooses the cheapest alternative** from the search space.
Cover-based reformulation for the DB fragment of RDF

Optimized reformulation into JUCQs

Query $q$
CQ-to-UCQ ref. algo.
Graph $G$

Results

RDBMS

$\equiv$

Query $q$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$

$\equiv$
Given the query

\[
q_1(x, y) : \quad x \text{ rdf:type } y, \\
\quad x \text{ ub:degreeFrom } "\text{http://www.University532.edu}" , \\
\quad x \text{ ub:memberOf } "\text{http://www.Department1.University7.edu}" 
\]

\( (t_1) \) \hspace{1cm} \( (t_2) \) \hspace{1cm} \( (t_3) \)
CQ-to-JUCQ query reformulation

Given the query

\[ q_1(x, y) : - ~ x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom "http://www.University532.edu"}, \]
\[ x \text{ ub:memberOf "http://www.Department1.University7.edu"} \]

and a state-of-the-art CQ-to-UCQ reformulation algorithm \( ref \):

- the UCQ reformulation is: \((t_1, t_2, t_3)^{ref}\)
- the SCQ reformulation is: \((t_1)^{ref} \Join (t_2)^{ref} \Join (t_3)^{ref}\)
CQ-to-JUCQ query reformulation

Given the query

\[ q_1(x, y) : \quad x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom } "http://www.University532.edu", \]
\[ x \text{ ub:memberOf } "http://www.Department1.University7.edu" \]

and a state-of-the-art CQ-to-UCQ reformulation algorithm \( \text{ref} \), the space of JUCQs is:

1. \( (t_1, t_2, t_3)^{\text{ref}} \)
2. \( (t_1)^{\text{ref}} \bowtie (t_2)^{\text{ref}} \bowtie (t_3)^{\text{ref}} \)
3. \( (t_1, t_2)^{\text{ref}} \bowtie (t_3)^{\text{ref}} \)
4. \( (t_1)^{\text{ref}} \bowtie (t_2, t_3)^{\text{ref}} \)
5. \( (t_1, t_3)^{\text{ref}} \bowtie (t_2)^{\text{ref}} \)
6. \( (t_1, t_2)^{\text{ref}} \bowtie (t_1, t_3)^{\text{ref}} \)
7. \( (t_1, t_2)^{\text{ref}} \bowtie (t_2, t_3)^{\text{ref}} \)
8. \( (t_1, t_3)^{\text{ref}} \bowtie (t_2, t_3)^{\text{ref}} \)
CQ-to-JUCQ query reformulation algorithms

Exhaustive algorithm
Impractical since search space size is the number of minimal covers of a set of \( n \) elements

Greedy algorithm
Driven by cost; starting from 1-atom fragments and “growing” them

Given the query

\[
q_1(x, y) :\quad \times \text{ rdf:type } y, \\
\times \text{ ub:degreeFrom “http://www.University532.edu”),} \\
\times \text{ ub:memberOf “http://www.Department1.University7.edu”} \\
\]

a cost-based greedy exploration is:

\[
(t_1)^\text{ref} \Join (t_2)^\text{ref} \Join (t_3)^\text{ref}
\]
CQ-to-JUCQ query reformulation: greedy algorithm example

Given the query

$$q_1(x, y) : - \quad x \text{ rdf:type } y,$$
$$x \text{ ub:degreeFrom } "http://www.University532.edu" ,$$
$$x \text{ ub:memberOf } "http://www.Department1.University7.edu"$$

a cost-based greedy exploration is:

$$\langle t_1 \rangle^{ref} \sqcap \langle t_2 \rangle^{ref} \sqcap \langle t_3 \rangle^{ref}$$

$$\langle t_1, t_2 \rangle^{ref} \sqcap \langle t_3 \rangle^{ref}$$

$$\langle t_1, t_3 \rangle^{ref} \sqcap \langle t_2 \rangle^{ref}$$

$$\langle t_1 \rangle^{ref} \sqcap \langle t_2, t_3 \rangle^{ref}$$
CQ-to-JUCQ query reformulation algorithm

Given the query

\[ q_1(x, y) : - \ x \text{ rdf:type } y, \]
\[ \quad x \text{ ub:degreeFrom } \text{"http://www.University532.edu"}, \]
\[ \quad x \text{ ub:memberOf } \text{"http://www.Department1.University7.edu"} \]

(a cost-based greedy exploration is:

\[ (t_1)^{\text{ref}} \Box (t_2)^{\text{ref}} \Box (t_3)^{\text{ref}} \]
\[ (t_1, t_2)^{\text{ref}} \Box (t_3)^{\text{ref}} \]
\[ (t_1, t_3)^{\text{ref}} \Box (t_2)^{\text{ref}} \]
\[ (t_1)^{\text{ref}} \Box (t_2, t_3)^{\text{ref}} \]
\[ (t_1, t_2, t_3)^{\text{ref}} \]
\[ (t_1, t_3)^{\text{ref}} \Box (t_1, t_2)^{\text{ref}} \]
\[ (t_1, t_3)^{\text{ref}} \Box (t_2, t_3)^{\text{ref}} \]
Part V

Cover-based reformulation in FOL-reducible settings
For any setting where query answering is reducible to FOL query evaluation
The same overall approach:

\begin{align*}
q_{ref} &\equiv c(q_{ref}) \\
\vdots &\equiv \ \vdots \\
q^1 &\equiv c(q^1) \\
q^n &\equiv c(q^n) \\
q_{best} &\equiv c(q_{best})
\end{align*}
Given:

- $N_C$: set of **concept names** (unary predicates),
- $N_R$: set of **role names** (binary predicates),
- $N_I$: set of **individuals** (constants).
DL-Lite\(_R\) facts (a.k.a. ABox)

Given:

- \(N_C\): set of **concept names** (unary predicates),
- \(N_R\): set of **role names** (binary predicates),
- \(N_I\): set of **individuals** (constants).

Finite number of:

- **Concept assertions**: \(A(a)\) with \(A \in N_C\) and \(a \in N_I\)
- **Role assertions**: \(R(a, b)\) with \(R \in N_R\) and \(a, b \in N_I\)
Given:

- $N_C$: set of **concept names** (unary predicates),
- $N_R$: set of **role names** (binary predicates),
Given:

- $N_C$: set of concept names (unary predicates),
- $N_R$: set of role names (binary predicates),

Set of:

- **Concept inclusions**: $B \sqsubseteq C$
- **Role inclusions**: $Q \sqsubseteq S$

using the following grammar (where $A \in N_C$ and $R \in N_R$):

$$
B := A \mid \exists Q, \quad C := B \mid \neg B, \quad Q := R \mid R^-, \quad S := Q \mid \neg Q
$$
FOL Reformulation

Consider the query $q(x) :- \text{PhDStudent}(x) \land \text{worksWith}(y, x)$ and KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ with the TBox $\mathcal{T}$ and ABox $\mathcal{A}$ below:

- $(T1)$  \text{PhDStudent} \sqsubseteq \text{Researcher}
- $(T2)$  \exists \text{worksWith} \sqsubseteq \text{Researcher}
- $(T3)$  \exists \text{worksWith} \sqsubseteq \text{Researcher}
- $(T4)$  \text{worksWith} \sqsubseteq \text{worksWith}$
- $(T5)$  \text{supervisedBy} \sqsubseteq \text{worksWith}
- $(T6)$  \exists \text{supervisedBy} \sqsubseteq \text{PhDStudent}
- $(T7)$  \text{PhDStudent} \sqsubseteq \neg \exists \text{supervisedBy}$

- $(A1)$  \text{worksWith}(Ioana, Francois)
- $(A2)$  \text{supervisedBy}(Damian, Ioana)
- $(A3)$  \text{supervisedBy}(Damian, Francois)
The UCQ reformulation of q is:

\[ q^{UCQ}(x) :- q^1(x) :- \text{PhDStudent}(x) \land \text{worksWith}(y, x) \]
\[ \lor q^2(x) :- \text{PhDStudent}(x) \land \text{worksWith}(x, y) \]
\[ \lor q^3(x) :- \text{PhDStudent}(x) \land \text{supervisedBy}(y, x) \]
\[ \lor q^4(x) :- \text{PhDStudent}(x) \land \text{supervisedBy}(x, y) \]
\[ \lor q^5(x) :- \text{supervisedBy}(x, z) \land \text{worksWith}(y, x) \]
\[ \lor q^6(x) :- \text{supervisedBy}(x, z) \land \text{worksWith}(x, y) \]
\[ \lor q^7(x) :- \text{supervisedBy}(x, z) \land \text{supervisedBy}(y, x) \]
\[ \lor q^8(x) :- \text{supervisedBy}(x, z) \land \text{supervisedBy}(x, y) \]
\[ \lor q^9(x) :- \text{supervisedBy}(x, x) \]
\[ \lor q^{10}(x) :- \text{supervisedBy}(x, y) \]
The UCQ reformulation of $q$ is:

$$q^{UCQ}(x)\text{:- } q^1(x)\text{:- } \text{PhDStudent}(x) \land \text{worksWith}(y, x)
\lor q^2(x)\text{:- } \text{PhDStudent}(x) \land \text{worksWith}(x, y)
\lor q^3(x)\text{:- } \text{PhDStudent}(x) \land \text{supervisedBy}(y, x)
\lor q^4(x)\text{:- } \text{PhDStudent}(x) \land \text{supervisedBy}(x, y)
\lor q^5(x)\text{:- } \text{supervisedBy}(x, z) \land \text{worksWith}(y, x)
\lor q^6(x)\text{:- } \text{supervisedBy}(x, z) \land \text{worksWith}(x, y)
\lor q^7(x)\text{:- } \text{supervisedBy}(x, z) \land \text{supervisedBy}(y, x)
\lor q^8(x)\text{:- } \text{supervisedBy}(x, z) \land \text{supervisedBy}(x, y)
\lor q^9(x)\text{:- } \text{supervisedBy}(x, x)
\lor q^{10}(x)\text{:- } \text{supervisedBy}(x, y)$$

Naive exhaustive application of specialization steps leads, in general, to highly redundant reformulations.
Minimization of $q^{UCQ}$ by eliminating disjuncts contained in another leads to:

$$q_{min}^{UCQ}(x) : - q_1(x) : - \text{PhDStudent}(x) \land \text{worksWith}(y, x)$$
$$\lor q_2(x) : - \text{PhDStudent}(x) \land \text{worksWith}(x, y)$$
$$\lor q_3(x) : - \text{PhDStudent}(x) \land \text{supervisedBy}(y, x)$$
$$\lor q_9(x) : - \text{supervisedBy}(x, y)$$
Consider the query \( q(x) :\neg A(x) \land R(x, y) \land R'(z, y) \) and KB \( \mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle \) with the TBox \( \mathcal{T} \) and ABox \( \mathcal{A} \) below:

\[
\begin{align*}
&T1 \quad B \subseteq \exists R' \\
&T2 \quad R' \subseteq R
\end{align*}
\]

\[
\begin{align*}
&A1 \quad A(a) \\
&A2 \quad B(a)
\end{align*}
\]
Cover-based FOL reformulation

▷ Bursztyn, Goasdoué, Manolescu (DL 2015)
Consider the query \( q(x) :\neg A(x) \land R(x, y) \land R'(z, y) \) and KB \( \mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle \) with the TBox \( \mathcal{T} \) and ABox \( \mathcal{A} \) below:

\[
\begin{align*}
(T1) \quad & B \sqsubseteq \exists R' \\
(T2) \quad & R' \sqsubseteq R
\end{align*}
\]

\[
\begin{align*}
(A1) \quad & A(a) \\
(A2) \quad & B(a)
\end{align*}
\]

The only answer to \( q(x) \) is \( a \).
Cover-based FOL reformulation

\[ q(x) :\neg A(x) \land R(x, y) \land R'(z, y) \]
\[ K = \langle T = \{ B \sqsubseteq \exists R', R' \sqsubseteq R \}, A = \{ A(a), B(a) \} \rangle \]

Consider the cover \( C_1 = \{ \{ A(x), R(x, y) \}, \{ R'(z, y) \} \} \).
Cover-based FOL reformulation

\[ q(x) :\neg A(x) \land R(x, y) \land R'(z, y) \]

\[ \mathcal{K} = \langle T = \{B \sqsubseteq \exists R', R' \sqsubseteq R\}, A = \{A(a), B(a)\} \rangle \]

Consider the cover \( C_1 = \{\{A(x), R(x, y)\}, \{R'(z, y)\}\} \).

The cover-based FOL reformulation of \( q \) w.r.t. \( C_1 \) is:

\[ q^{JUCQ}(x):- \quad q_{1}^{UCQ}(x, y):-\quad (A(x) \land R(x, y)) \]

\[ \lor (A(x) \land R'(x, y)) \land \]

\[ q_{2}^{UCQ}(y):-\quad R'(z, y) \]
Cover-based FOL reformulation

\[ q(x) :- A(x) \land R(x, y) \land R'(z, y) \]

\[ \mathcal{K} = \langle \mathcal{T} = \{ B \sqsubseteq \exists R', R' \sqsubseteq R \}, \mathcal{A} = \{ A(a), B(a) \} \rangle \]

Consider the cover \( C_1 = \{ \{ A(x), R(x, y) \}, \{ R'(z, y) \} \} \).

The cover-based FOL reformulation of \( q \) w.r.t. \( C_1 \) is:

\[ q^{\text{JUCQ}}(x) :- q_1^{\text{UCQ}}(x, y) :- (A(x) \land R(x, y)) \lor (A(x) \land R'(x, y)) \land R'(z, y) \]

**Cover \( C_1 \) prevents unification** \( \Rightarrow \) \( q^{\text{JUCQ}}(x) \) is **not** a FOL reformulation of \( q \) w.r.t. \( \mathcal{T} \).
q(x):- A(x) ∧ R(x, y) ∧ R'(z, y)

\( \mathcal{K} = \langle T = \{B \sqsubseteq \exists R', R' \sqsubseteq R\}, \mathcal{A} = \{A(a), B(a)\}\rangle \)

Consider the cover \( C_2 = \{\{A(x)\}, \{R(x, y), R'(z, y)\}\} \)
Safe cover

\[ q(x) : \neg \ A(x) \land R(x, y) \land R'(z, y) \]

\[ \mathcal{K} = \langle T = \{ B \sqsubseteq \exists R', R' \sqsubseteq R \}, A = \{ A(a), B(a) \} \rangle \]

Consider the cover \( C_2 = \{ \{ A(x) \}, \{ R(x, y), R'(z, y) \} \} \)

The cover-based FOL reformulation of \( q \) w.r.t. \( C_1 \) is:

\[ q_{\text{JUCQ}}^1(x) : \neg \quad q_{\text{UCQ}}^1(x) : \neg \quad (A(x)) \]

\[ q_{\text{UCQ}}^2(x) : \neg \quad (R(x, y) \land R'(z, y)) \]

\[ \lor (R'(x, y) \land R'(z, y)) \]

\[ \lor (R'(x, y)) \lor (B(x)) \]
Safe cover

\[ q(x) :- A(x) \land R(x, y) \land R'(z, y) \]
\[ K = \langle T = \{ B \sqsubseteq \exists R', R' \sqsubseteq R \}, A = \{ A(a), B(a) \} \rangle \]

Consider the cover \( C_2 = \{ \{ A(x) \}, \{ R(x, y), R'(z, y) \} \} \)

The cover-based FOL reformulation of \( q \) w.r.t. \( C_1 \) is:

\[ q^{JUCQ}(x) :- \quad q_1^{UCQ}(x) :- \quad (A(x)) \]
\[ q_2^{UCQ}(x) :- \quad (R(x, y) \land R'(z, y)) \]
\[ \lor (R'(x, y) \land R'(z, y)) \]
\[ \lor (R'(x, y)) \lor (B(x)) \]

**Cover \( C_2 \) preserves unification \( \Rightarrow \) \( q^{JUCQ}(x) \) is a FOL reformulation of \( q \) w.r.t. \( T \) and \( C_2 \) a safe cover.**
We call root cover the minimal safe cover.
We call root cover the minimal safe cover.

\[ C_2 = \{ \{ A(x) \}, \{ R(x, y), R'(z, y) \} \} \] is a root cover for \( q \) w.r.t. \( T \).
Extended cover and extended fragment queries

\[ q(x) : - A(x) \land R(x, y) \land R'(z, y) \]

\[ \mathcal{K} = \langle T = \{ B \sqsubseteq \exists R', R' \sqsubseteq R \}, A = \{ A(a), B(a) \} \rangle \]

Consider the cover \( C_3 = \{ f_1 \parallel f_1, \quad f_2 \parallel f_0 \} \), with:

- \( f_0 = \{ A(x) \} \)
- \( f_1 = \{ R(x, y), R'(z, y) \} \)
- \( f_2 = \{ A(x), R(x, y) \} \)
Extended cover and extended fragment queries

$q(x) :\text{-} A(x) \land R(x, y) \land R'(z, y)$

$\mathcal{K} = \langle \mathcal{T} = \{B \sqsubseteq \exists R', R' \sqsubseteq R\}, \mathcal{A} = \{A(a), B(a)\} \rangle$

Consider the cover $C_3 = \{f_1 \parallel f_1, \ f_2 \parallel f_0\}$, with:

- $f_0 = \{A(x)\}$
- $f_1 = \{R(x, y), R'(z, y)\}$
- $f_2 = \{A(x), R(x, y)\}$

The extended cover-based FOL reformulation of $q$ w.r.t. $C_3$ is:

$q^e(x) :\text{-} q^{|f_1||f_1|}(x) :\text{-} \quad q^{|f_2||f_0|}(x) :\text{-}$

\[
\begin{align*}
q^{|f_1||f_1|}(x) & :\text{-} ((R(x, y) \land R'(z, y)) \\
& \quad \lor R'(x, y) \lor B(x))
\end{align*}
\]

\[
\begin{align*}
q^{|f_2||f_0|}(x) & :\text{-} ((A(x) \land R(x, y)) \\
& \quad \lor (A(x) \land R'(x, y)) \\
& \quad \lor (A(x) \land B(x)))
\end{align*}
\]
Search space

Covers

$\mathcal{E}_q$ (extended covers)

$\mathcal{L}_q$ (safe covers)

$\mathcal{C}_{\text{root}}$
Part VI

Performance
CQ-to-JUCQ query reformulation performance

Given the query

\[ q_1(x, y) : - x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom } "http://www.University532.edu", \]
\[ x \text{ ub:memberOf } "http://www.Department1.University7.edu" \]

and the LUBM 100M benchmark:

<table>
<thead>
<tr>
<th>JUCQ</th>
<th>#reformulations</th>
<th>exec. time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((t_1, t_2, t_3))ref</td>
<td>2,256</td>
<td>6,387</td>
</tr>
<tr>
<td>((t_1))ref (\land) ((t_2))ref (\land) ((t_3))ref</td>
<td>195</td>
<td>1,074,026</td>
</tr>
<tr>
<td>((t_1, t_2))ref (\land) ((t_3))ref</td>
<td>755</td>
<td>1,968</td>
</tr>
<tr>
<td>((t_1))ref (\land) ((t_2, t_3))ref</td>
<td>200</td>
<td>846,710</td>
</tr>
<tr>
<td>((t_1, t_3))ref (\land) ((t_2))ref</td>
<td>568</td>
<td>554</td>
</tr>
<tr>
<td>((t_1, t_2))ref (\land) ((t_1, t_3))ref</td>
<td>1,316</td>
<td>2,734</td>
</tr>
<tr>
<td>((t_1, t_2))ref (\land) ((t_2, t_3))ref</td>
<td>764</td>
<td>2,289</td>
</tr>
<tr>
<td>((t_1, t_3))ref (\land) ((t_2, t_3))ref</td>
<td>576</td>
<td>588</td>
</tr>
</tbody>
</table>
Datasets and RDBMS engines

DBLP (8 M) and LUBM (1 M and 100 M) millions triples

- PostgreSQL 9.3.2
- System A
- System B
Reformulation algorithms

Basic CQ-to-UCQ algorithm
Picked that of [Goasdoué et al., EDBT’13] since it handles the largest known fragment of RDF.

Comparison:
1. **UCQ** reformulation
2. **SCQ** reformulation
3. **Greedy JUCQ** reformulation
4. **Exhaustive JUCQ** reformulation
Performance

Optimized reformulation experiments on three RDBMSs

Query answering on LUBM 100 M using PostgreSQL

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations
Query answering on LUBM 100 M using System A

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations
Query answering on LUBM 100 M using System B

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations
Optimized reformulation: take-home message

1. Equivalent SQL syntaxes are not equal from the RDBMS optimizer perspective (inside or outside well-supported dialect)

2. Chosing the reformulation with the help of textbook cost model formulas makes queries feasible or efficient when they were not

3. This amounts to enlarging the optimizer’s “can-do” dialect, at a very modest performance overhead.
Experiments on DL-Lite$^R$ reformulation

1. CQ-to-UCQ reformulation
2. $C_{\text{root}}$ reformulation
3. Our cover-based greedy reformulation
4. Our cover-based exhaustive reformulation (as a yardstick for the quality of the solution our greedy finds)
Query answering on LUBM\textsuperscript{\exists}_{20} 15 millions facts dataset using PostgreSQL

13 queries; 2 to 10 atoms, 5.77 on average; 35 to 667 reformulations, 290.2 on average.
Algorithms running time on LUBM$_{20}^{20}$ 15 millions facts dataset

13 queries; 2 to 10 atoms, 5.77 on average; 35 to 667 reformulations, 290.2 on average.

Cost-based cover selection algorithm running time

- EC-DL
- GC-DL

Time (ms)
Part VII

Conclusion and perspectives
Where we stand

**Constraints** (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”
Where we stand

**Constraints** (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”. We considered semantic query answering through **FOL reduction** (i.e., SQL).
Constraints (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”
We considered semantic query answering through FOL reduction (i.e., SQL).
Not any SQL query resulting from reformulation is handled well by current RDBMSs!
Constraints (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”. We considered semantic query answering through FOL reduction (i.e., SQL).

Not any SQL query resulting from reformulation is handled well by current RDBMSs!

Vast performance differences between different reformulations of the same query; cost-based approach.
What is ahead

Continuous push on the expressivity - efficiency frontier; updates...
What is ahead

Continuous push on the **expressivity** - **efficiency** frontier; updates...

RDBMSs are highly efficient for some forms of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints
Continuous push on the **expressivity** - **efficiency** frontier; updates...

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.
What is ahead

Continuous push on the **expressivity - efficiency** frontier; updates...

RDBMSs are highly efficient for **some** forms of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them attractive back-ends for query answering.

“Novel” platforms (MapReduce, NoSQL...) will raise the same query evaluation performance problems anyway.
Continuous push on the **expressivity - efficiency** frontier; updates...

RDBMSs are highly efficient for some forms of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them attractive back-ends for query answering.

“Novel” platforms (MapReduce, NoSQL...) will raise the same query evaluation performance problems anyway.

Most attractive languages currently: **DL-Lite, Datalog\(^\pm\)**
Continuous push on the **expressivity - efficiency** frontier; updates...

RDBMSs are highly efficient for some forms of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.

“Novel” platforms (MapReduce, NoSQL...) will raise the same query evaluation performance problems anyway.

Most attractive languages currently: **DL-Lite, Datalog±**

**New relevance of query optimization literature and research!**