Efficient Handling of SPARQL OPTIONAL for OBDA

Guohui Xiao[1], Roman Kontchakov[2], *Benjamin Cogrel*[1],
Diego Calvanese[1], Elena Botoeva[1]

[1] KRDB Research Centre, Free University of Bozen-Bolzano, Italy
[2] Dept. of Computer Science and Inf. Syst., Birkbeck, University of London, UK

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen-Bolzano

ISWC'18
Monterey, California, 10 October 2018

NULL values are ubiquitous in relational databases

people

id	fullName	workEmail	homeEmail
1	Peter Smith	peter@company.com	peter@perso.org
2	John Lang	NULL	joe@perso.org
3	Susan Mayer	susan@company.com	NULL

NULL values are ubiquitous in relational databases

people

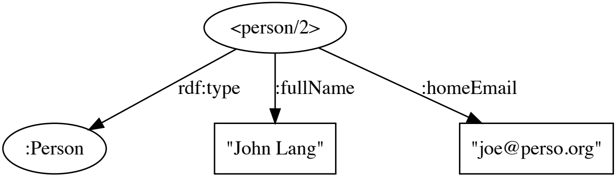
id	fullName	workEmail	homeEmail
1	Peter Smith	peter@company.com	peter@perso.org
2	John Lang	NULL	joe@perso.org
3	Susan Mayer	susan@company.com	NULL

SQL query: transparent handling of *NULL* values coming from the database

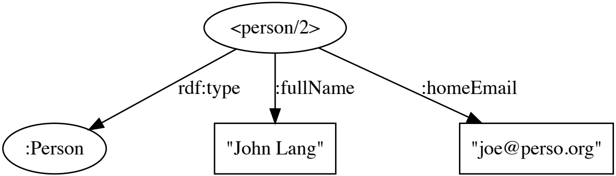
```
SELECT fullName, workEmail
FROM people
```

unibz

RDF graph: no *NULL* value



RDF graph: no *NULL* value



Corresponding SPARQL query: need for OPTIONAL

```
SELECT ?fullName ?workEmail {
    ?p :fullName ?fullName
     OPTIONAL {
         ?p :workEmail ?workEmail
      }
 }
```

unibz

LEFT JOIN: *NULL* values produced by the query

pet_ownership

ownerId	petId
2	100
2	101
3	102

LEFT JOIN: *NULL* values produced by the query

pet_ownership

ownerId	petId
2	100
2	101
3	102

SQL query with a LEFT JOIN

```
SELECT pp.fullName, pt.petId AS pet
FROM people pp LEFT JOIN pet_ownership pt
     ON pp.id = pt.ownerId
```

unibz

LEFT JOIN: *NULL* values produced by the query

pet_ownership

ownerId	petId
2	100
2	101
3	102

SQL query with a LEFT JOIN

```
SELECT pp.fullName, pt.petId AS pet
FROM people pp LEFT JOIN pet_ownership pt
     ON pp.id = pt.ownerId
```

"Corresponding" SPARQL query (if we omit IRI construction details)

```
SELECT ?fullName ?pet {
    ?p :fullName ?fullName
     OPTIONAL {
         ?p :pet ?pet
      }
 }
```

Ontology-Based Data Access (OBDA)

OBDA in a nutshell

- a.k.a. Virtual Knowledge Graphs
- SPARQL queries are reformulated into SQL queries
- Two main components:
  1. mapping (R2RML)
  2. ontology (OWL 2 QL TBox)

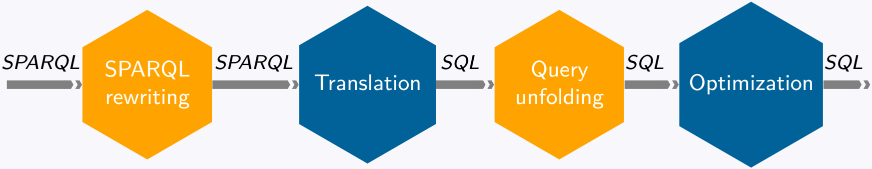
unibz

Ontology-Based Data Access (OBDA)

OBDA in a nutshell

- a.k.a. Virtual Knowledge Graphs
- SPARQL queries are reformulated into SQL queries
- Two main components:
  1. mapping (R2RML)
  2. ontology (OWL 2 QL TBox)

Query reformulation



NB: In our examples, the TBox is empty (orthogonal to OPTIONAL optimisation).

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

unibz

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

Previous State-of-the-Art reformulated query

```
SELECT v1.fullName, v2.workEmail
FROM people v1 LEFT JOIN people v2
   ON v1.id=v2.id AND v2.workEmail IS NOT NULL
```

unibz

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

Previous State-of-the-Art reformulated query

```
SELECT v1.fullName, v2.workEmail
FROM people v1 LEFT JOIN people v2
   ON v1.id=v2.id AND v2.workEmail IS NOT NULL
```

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

Ideal SQL reformulation

```
SELECT fullName, workEmail
FROM people
```

Previous State-of-the-Art reformulated query

```
SELECT v1.fullName, v2.workEmail
FROM people v1 LEFT JOIN people v2
   ON v1.id=v2.id AND v2.workEmail IS NOT NULL
```

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

Ideal SQL reformulation

```
SELECT fullName, workEmail
FROM people
```

Previous State-of-the-Art reformulated query

```
SELECT v1.fullName, v2.workEmail
FROM people v1 LEFT JOIN people v2
   ON v1.id=v2.id AND v2.workEmail IS NOT NULL
```

Padding effect

- Assignment of *NULL*s by the *LEFT JOIN*
- Used for reintroducing the *NULL*s eliminated by the mapping

unibz

Excessive use of the padding effect

SPARQL

```
SELECT ?fullName ?workEmail {
  ?p :fullName ?fullName
  OPTIONAL {
     ?p :workEmail ?workEmail
  }
}
```

Ideal SQL reformulation

```
SELECT fullName, workEmail
FROM people
```

Previous State-of-the-Art reformulated query

```
SELECT v1.fullName, v2.workEmail
FROM people v1 LEFT JOIN people v2
   ON v1.id=v2.id AND v2.workEmail IS NOT NULL
```

Padding effect

- Assignment of *NULL*s by the *LEFT JOIN*
- Used for reintroducing the *NULL*s eliminated by the mapping

Optimisation pattern

Reusing the *NULL*s coming for the database

unibz

Weakly well designed query (preferences)

SPARQL

```
SELECT ?n ?e {
  ?p :fullName ?n
  OPTIONAL {
    ?p :workEmail ?e
  }
  OPTIONAL {
    ?p :homeEmail ?e
  }
}
```

Weakly well designed query (preferences)

SPARQL

```
SELECT ?n ?e {
  ?p :fullName ?n
  OPTIONAL {
    ?p :workEmail ?e
  }
  OPTIONAL {
    ?p :homeEmail ?e
  }
}
```

Previous State-of-the-Art reformulated query

```
SELECT v3.fullName AS n,
  COALESCE(v3.workEmail,v4.homeEmail) AS e
FROM
  ( SELECT v1.fullName, v1.id, v2.workEmail
    FROM people v1 LEFT JOIN people v2
      ON v1.id=v2.id AND v2.workEmail IS NOT NULL ) v3
  LEFT JOIN people v4
  ON v3.id=v4.id AND v4.homeEmail IS NOT NULL
   AND (v3.workEmail=v4.homeEmail OR v3.workEmail IS NULL)
```

Weakly well designed query (preferences)

SPARQL

```
SELECT ?n ?e {
  ?p :fullName ?n
  OPTIONAL {
    ?p :workEmail ?e
  }
  OPTIONAL {
    ?p :homeEmail ?e
  }
}
```

Previous State-of-the-Art reformulated query

```
SELECT v3.fullName AS n,
  COALESCE(v3.workEmail,v4.homeEmail) AS e
FROM
  ( SELECT v1.fullName, v1.id, v2.workEmail
    FROM people v1 LEFT JOIN people v2
      ON v1.id=v2.id AND v2.workEmail IS NOT NULL ) v3
  LEFT JOIN people v4
  ON v3.id=v4.id AND v4.homeEmail IS NOT NULL
   AND (v3.workEmail=v4.homeEmail OR v3.workEmail IS NULL)
```

Weakly well designed query (preferences)

SPARQL

```
SELECT ?n ?e {
  ?p :fullName ?n
  OPTIONAL {
    ?p :workEmail ?e
  }
  OPTIONAL {
    ?p :homeEmail ?e
  }
}
```

Ideal SQL reformulation

```
SELECT fullName AS n,
  COALESCE(workEmail,homeEmail) AS e
FROM people
```

Previous State-of-the-Art reformulated query

```
SELECT v3.fullName AS n,
  COALESCE(v3.workEmail,v4.homeEmail) AS e
FROM
  ( SELECT v1.fullName, v1.id, v2.workEmail
    FROM people v1 LEFT JOIN people v2
      ON v1.id=v2.id AND v2.workEmail IS NOT NULL ) v3
  LEFT JOIN people v4
  ON v3.id=v4.id AND v4.homeEmail IS NOT NULL
   AND (v3.workEmail=v4.homeEmail OR v3.workEmail IS NULL)
```

Why are novel optimisations needed?

Database perspective [Galindo-Legaria and Rosenthal, 1997]

- Accidental LEFT JOINs in expert-written SQL queries: too rare
- Views can return *NULL*s and contain LEFT JOINs
  - After unfolding, a nullable column may be required
  - Good opportunity for optimisation (LEFT JOINs reduced to INNER JOINs)

unibz

Why are novel optimisations needed?

Database perspective [Galindo-Legaria and Rosenthal, 1997]

- Accidental LEFT JOINs in expert-written SQL queries: too rare
- Views can return *NULL*s and contain LEFT JOINs
  - After unfolding, a nullable column may be required
  - Good opportunity for optimisation (LEFT JOINs reduced to INNER JOINs)

Triplestore perspective [Chebotko *et al.*, 2009]

- No *NULL* stored in the triplestore, no opportunity for reuse
- Constraints (e.g. SHACL) not considered

unibz

Why are novel optimisations needed?

Database perspective [Galindo-Legaria and Rosenthal, 1997]

- Accidental LEFT JOINs in expert-written SQL queries: too rare
- Views can return *NULL*s and contain LEFT JOINs
  - After unfolding, a nullable column may be required
  - Good opportunity for optimisation (LEFT JOINs reduced to INNER JOINs)

Triplestore perspective [Chebotko *et al.*, 2009]

- No *NULL* stored in the triplestore, no opportunity for reuse
- Constraints (e.g. SHACL) not considered

OBDA: challenges and opportunities

- Many LEFT JOINs due to OPTIONALs
- Complex LEFT JOIN conditions
- Many *NULL*s in the database
- Integrity constraints (attribute nullability, uniqueness and foreign keys)

Contribution

1. SPARQL to SQL translation
2. Optimisations of translated queries
3. Evaluation based on the BSBM benchmark

unibz

SPARQL to SQL translation

- Fragment of SPARQL 1.1 (including OPTIONAL and MINUS)
- Succinctness due to the use of LEFT JOIN and COALESCE
- Bag semantics
- Three-valued logic for both SPARQL and SQL
- Formally proven correct

Optimisations of translated queries

1. Compatibility Filter Reduction (CFR)
   generalises [Chebotko *et al.*, 2009]
2. LEFT JOIN Naturalisation (LJN) to avoid padding
3. Natural LEFT JOIN Reduction (NLJR) into an inner join
4. JOIN Transfer (JT) to simplify the right operand of LEFT JOIN
5. LEFT JOIN Decomposition (LJD)
   complements [Galindo-Legaria and Rosenthal, 1997]
   by taking account of complex non-NULL-rejecting filters
6. For well-designed SPARQL, CFR+LJN $\approx$ [Rodriguez-Muro and Rezk, 2015]
   (even simpler)

unibz

Evaluation

Dataset and queries

- Dataset from BSBM (1M products and 10M reviews)
- 4 modified queries from BSBM (reduced selectivity)
- 7 new queries with preferences (weakly well-designed)

SQL query performance comparison

1. Only Previous State-of-the-Art optimisations (PSoA)
2. PSoA + our optimisations (O)

Systems

- PostgreSQL 9.6, MySQL 5.7 and the 3 main commercial databases
- t2.xlarge Amazon EC2 instance, 4 vCPUs, 16G memory, 500G SSD, Ubuntu 16.04 LTS

unibz

Evaluation results (in seconds)

	PostgreSQL		MySQL		X		Y		Z	
query	PSoA	O	PSoA	O	PSoA	O	PSoA	O	PSoA	O
Q1	1.79	1.77	0.43	0.38	0.90	0.80	0.56	0.52	29.06	25.09
Q2	18.75	2.07	19.95	0.36	40.00	16.07	0.44	0.37	27.99	5.97
$Q2_{BSBM}$		3.88		0.37		20.55		0.38		5.91
Q3	4.20	0.09	4.70	0.11	5.50	1.60	2.04	0.14	5.45	0.65
Q4	2.14	0.16	0.86	0.04	3.00	0.60	1.78	0.11	4.38	0.53
Q5	0.56	0.05	0.01	0.01	1.80	0.30	0.30	0.08	0.51	0.53
Q6	102.35	0.18	>600	0.04	1.90	0.40	4.50	0.14	0.82	0.54
Q7	102.00	0.17	>600	0.04	2.60	0.40	14.57	0.14	1.21	0.53
Q8	0.07	0.06	0.01	0.01	8.40	1.30	0.08	0.08	295.25	0.40
Q9	101.20	0.16	>600	0.04	>600	2.70	4.30	0.11	>600	0.43
Q10	103.30	0.15	>600	0.05	>600	4.20	5.20	0.14	>600	0.43
Q11	5.26	0.87	3.80	0.21	107.06	2.68	177.95	0.22	7.82	0.13

Observations

- Optimisations effective for ALL database engines
- Most queries can be evaluated in less than a second
- Improvement: up to 3 orders of magnitude

Conclusions

- Novel optimisations due to the opportunities offered by the OBDA setting
- Significant performance improvement, even for commercial databases
- Under implementation within the Ontop OBDA framework
- Could be of interest for storing constrained RDF graphs (e.g., using SHACL)

unibz

Why optimising LEFT JOINs?

Optimisation at the SQL level

1. Eliminating LEFT JOINs
2. Replacing them by INNER JOINs
3. Simplifying their joining conditions
4. Simplifying the right operand
   (by transferring parts to the left operand)

Either done by the query reformulator or by the database engine

Physical query planning

1. Less join operations
2. Join ordering
   - Known to be critical for performance
   - Hard because LEFT JOIN is not commutative nor associative!
   - Even more challenging with complex joining conditions

References I

[Chebotko *et al.*, 2009] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi.
Semantics preserving SPARQL-to-SQL translation.
DKE, 68(10):973–1000, 2009.

[Galindo-Legaria and Rosenthal, 1997] César Galindo-Legaria and Arnon Rosenthal.
Outerjoin simplification and reordering for query optimization.
ACM TODS, 22(1):43–74, 1997.

[Rodriguez-Muro and Rezk, 2015] Mariano Rodriguez-Muro and Martín Rezk.
Efficient SPARQL-to-SQL with R2RML mappings.
J. Web Sem., 33:141–169, 2015.

unibz