Ultrawrap: Using SQL Views for RDB2RDF

Juan F. Sequeda\textsuperscript{1}, Conor Cunningham\textsuperscript{2}, Rudy Depena\textsuperscript{1} and Daniel P. Miranker\textsuperscript{1}
\textsuperscript{1}Department of Computer Sciences - The University of Texas at Austin
\textsuperscript{2}Microsoft Corporation
Goals

✓ Automatically create SPARQL endpoint for legacy relational databases.
✓ Real-time consistency between the relational and RDF data
✓ Making maximal use of existing SQL infrastructure

➢ Research question: Do existing commercial SQL query engines already subsume all the algorithms needed to support effective SPARQL execution on relational data?
Ultrawrap Architecture

• Compile Time
  1. Create Putative Ontology (PO)
  2. Create Virtual Triple Store

• Run Time
  3. Naïve SPARQL to SQL translation
  4. SQL Optimizer is the rewriter

• Future
  5. Putative Ontology to Domain Ontology mapping
Step 1: Creating a Putative Ontology

1. Putative Ontology
2. TripleView
3. SPARQL to SQL
4. SQL Query Optimizer

- OWL
- RDF
- SPARQL
- SQL Query Optimizer
- Query Plan
- Data
- Schema
Ontology Quality? I.e. putative

• Putative: “commonly regarded as such”

• Putative Ontology (PO): automatic syntactic transformation from a data source schema to an ontology
  – data or information source ontology

• Evidence: SQL schema from E-R models make “interesting” ontologies
FOL rules transform SQL DDL to OWL

- Full mapping in Datalog
  - Stratified and safe
- Proof of total coverage of all key combinations

Example From the Transformation System

<table>
<thead>
<tr>
<th>SID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Juan</td>
</tr>
<tr>
<td>2</td>
<td>Hamid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS386</td>
</tr>
<tr>
<td>2</td>
<td>CS379H</td>
</tr>
</tbody>
</table>

Student

enrolled

Course
enrolledIn

Student

Course

domain

range
## Transformation System

### SQL Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>RDF Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>CREATE TABLE STUDENT</td>
<td>&lt;owl:Class rdf:ID=&quot;Student&quot;/&gt;</td>
</tr>
<tr>
<td>COURSE</td>
<td>CREATE TABLE COURSE</td>
<td>&lt;owl:Class rdf:ID=&quot;Course&quot;/&gt;</td>
</tr>
<tr>
<td>ENROLLED</td>
<td>CREATE TABLE ENROLLED</td>
<td></td>
</tr>
</tbody>
</table>

### RDF Classes and Properties

- **Student Class**
  - SQL: `create table STUDENT{SID integer primary key, NAME varchar not null }`
  - RDF: `<owl:Class rdf:ID="Student"/>`

- **Course Class**
  - SQL: `create table COURSE{CID integer primary key, NAME varchar not null }`
  - RDF: `<owl:Class rdf:ID="Course"/>`

- **enrolledIn Property**
  - SQL: `create table ENROLLED{Student integer foreign key references STUDENT(SID), Course integer foreign key references COURSE(CID), constraint REG_PK primary key {Student, Course}}`

### Relational Expressions

- **BinRel(r,s,t)**
  - SQL: `create table ENROLLED{Student integer foreign key references STUDENT(SID), Course integer foreign key references COURSE(CID), constraint REG_PK primary key {Student, Course}}`

- **Class(r)**
  - SQL: `CREATE TABLE STUDENT{SID integer primary key, NAME varchar not null }`
  - RDF: `<owl:Class rdf:ID="Student"/>`

- **ObjP(r,s,t)**
  - SQL: `CREATE TABLE ENROLLED{Student integer foreign key references STUDENT(SID), Course integer foreign key references COURSE(CID), constraint REG_PK primary key {Student, Course}}`

### Relational Algebra Expressions

- **BinRel(r,s,t)**
  - SQL: `CREATE TABLE ENROLLED{Student integer foreign key references STUDENT(SID), Course integer foreign key references COURSE(CID), constraint REG_PK primary key {Student, Course}}`

- **Class(r)**
  - SQL: `CREATE TABLE STUDENT{SID integer primary key, NAME varchar not null }`
  - RDF: `<owl:Class rdf:ID="Student"/>`

- **ObjP(r,s,t)**
  - SQL: `CREATE TABLE ENROLLED{Student integer foreign key references STUDENT(SID), Course integer foreign key references COURSE(CID), constraint REG_PK primary key {Student, Course}}`
Proof of total coverage of all key combinations

- **PK**: a relation only has a Primary Key
- **C-PK**: a relation only has a composite Primary Key
- **S-FK**: a relation only has one Foreign Key
- **N-FK**: a relation has at least two or more Foreign Keys

**Grammar**

- **E** $\Rightarrow$ **PK** + **T** | **C-PK** +**T**
- **E** $\Rightarrow$ **S-FK**
- **E** $\Rightarrow$ **N-FK**
- **T** $\Rightarrow$ **S-FK** | **N-FK**
LR(0) Item set construction represents all possibilities

The tree describes the complete space of relations when all possible combinations of primary and foreign keys are considered.
All Key Combinations Enumerated

- **PK + S-FK**: a relation has a Primary Key and only one Foreign Key
  - PK = S-FK: the Foreign Key is the Primary Key
  - PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
- **PK + N-FK**: a relation has a Primary Key and two (2) Foreign Keys
  - PK ∩ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - PK ⊆ N-FK: one of the Foreign Keys is also the Primary Key
- **PK + N-FK**: a relation has a Primary Key and more than two (>2) Foreign Keys
  - PK ∩ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - PK ⊆ N-FK: one of the Foreign Keys is also the Primary Key
- **C-PK + S-FK**: a relation has a Composite Primary Key and only one Foreign Key.
  - C-PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - S-FK ⊆ C-PK: the Foreign Key is part of the Primary Key
- **C-PK + N-FK**: a relation has a Composite Primary Key and two (2) Foreign Keys
  - C-PK ∩ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes
  - N-FK ⊆ C-PK: all the Foreign Keys are part of the Primary Key
  - C-PK ∩ N-FK ≠ 0, C-PK – N-FK ≠ 0, N-FK – C-PK ≠ 0: The Foreign Keys and Primary Key share common attributes
- **C-PK + N-FK**: a relation has a Composite Primary Key and more than two (>2) Foreign Keys
  - C-PK ∩ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes
  - N-FK ⊆ C-PK: all the Foreign Keys are part of the Primary Key
  - C-PK ∩ N-FK ≠ 0, C-PK – N-FK ≠ 0, N-FK – C-PK ≠ 0: The Foreign Keys and Primary Key share common attributes
Ultrawrap creates PO automatically

• Reads the Data Dictionary
  – Specific to each vendor
• Currently supporting Microsoft SQL Server and MySQL
Quality of a Putative Ontology

• Quality of the PO depends on the SQL DDL
  – Is it normalized?
  – Are all the constraints explicit?

• If the Quality is sufficient, all we need to do is rename

• Need to map the Putative Ontology to a Domain Ontology

• Evidence: SQL schema from E-R models make “interesting” ontologies

• SQL schemas made without any previous modeling make “poor” ontologies
Step 2. Create Virtual Triple Store

1. Putative Ontology
2. TripleView
3. SPARQL to SQL
4. SQL Query Optimizer

OWL
RDF

SPARQL

Schema
Data

Query Plan
Step 2. Create Virtual Triple Store

• Represent all relational data as triples using a view definition
  – Promise of avoiding self joins (optimizer will do this)

• Triple table approach: one table with three columns (s,p,o)
  – No symbol/lookup table. Strings are in the view

• Actually, the view is (s,spk, p, o, opk) where spk and opk are the index values
  – Optimizer needs to know the index values
Step 2. Create Virtual Triple Store

• Create SELECT statements that output triples

SELECT “Product”+id as s, id as spk, “product_label” as p, label as o, null as opk FROM Product

<table>
<thead>
<tr>
<th>S</th>
<th>SPK</th>
<th>P</th>
<th>O</th>
<th>OPK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product1</td>
<td>1</td>
<td>product_label</td>
<td>Label of Product 1</td>
<td>null</td>
</tr>
<tr>
<td>Product2</td>
<td>2</td>
<td>product_label</td>
<td>Label of Product 2</td>
<td>null</td>
</tr>
</tbody>
</table>

SELECT “Product”+ProductID as s, ProductID as spk, “product_productfeature” as p, “ProductFeature”+ProductFeatureID as o, ProductFeatureID as opk FROM ProductFeatureProduct

<table>
<thead>
<tr>
<th>S</th>
<th>SPK</th>
<th>P</th>
<th>O</th>
<th>OPK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product1</td>
<td>1</td>
<td>product_productfeature</td>
<td>ProductFeature45</td>
<td>45</td>
</tr>
<tr>
<td>Product1</td>
<td>1</td>
<td>product_productfeature</td>
<td>ProductFeature98</td>
<td>98</td>
</tr>
</tbody>
</table>

• Use the PO as basis to create all the SELECT statements
Step 2. Create Virtual Triple Store

- Triple View is a union of all the SELECT statementss

```sql
CREATE VIEW TripleView(s,spk, p, o, opk) as
SELECT "Product"+id as s, id as spk, "rdf:type" as p, "Product" as o, null as opk FROM Product
UNION
SELECT "Product"+id as s, id as spk, "label" as p, label as o, null as opk FROM Product
UNION
SELECT "Product"+ProductID as s, ProductID as spk, "product_productfeature" as p, "ProductFeature"+ProductFeatureID as o, ProductFeatureID as opk FROM ProductFeatureProduct
UNION ...
```

- BSBM generates ~80 select statements in order to represent all relational data as triples
Step 3: Naïve SPARQL to SQL Translation
SPARQL Query

```sparql
SELECT ?product ?label
WHERE{
  ?product producttype_product ProductType47.
  ?product product_productfeature ProductFeature76.
  FILTER (?v > 500)
}
```

SQL Query on the Triple View

```sql
SELECT t1.o as product, t2.o as label
FROM TripleView t1, t2, t3, t4, t5
WHERE
  t1.p = 'producttype_product'
  AND t1.opk = 47
  AND t2.p = 'product_label'
  AND t3.spk = t1.spk
  AND t3.p = 'product_productfeature'
  AND t3.opk = 76
  AND t4.spk = t1.spk
  AND t4.p = 'product_productfeature'
  AND t4.opk = 4242
  AND t5.spk = t1.spk
  AND t5.p = 'product_propertyNum1'
  AND t5.o > 500
```

Syntactic transformation from a SPARQL query to an equivalent SQL query on the Triple View
Step 4: SQL Query Optimizer is the Rewrite system
Step 4: SQL Query Optimizer is the Rewrite system

TripleView(1, label, ABC) :- Product(1,ABC, _, _)  
TripleView(1, propNum1, 1) :- Product(1,_, 1, _)  
TripleView(1, propNum1, 2) :- Product(1,_, _, 2)

SQL Query on the TripleView
Query(X, Y):-TripleView(X, label, Y),
TripleView(X, propNum1, 1),
TripleView(X, propNum2, 2)

SQL Query on the Relational Data
SELECT id, label FROM product
WHERE propNum1 = 1 and propNum2 = 2
Query(X, Y) :- Product(X, Y, 1, 2)

Evaluate SQL Query on the TripleView
Query(X,Y):-Product(X,Y,1,_),Product(X,Y,_, 2)
Query(X,Y):- Product(X, Y, 1, 2)
Step 4: SQL Query Optimizer is the Rewrite System

- **TripleView Plan**

  - Physical Op: Compute Scalar
    - Logical Op: Compute Scalar
    - Sub Cost: 0.0249674
    - Cost: 0.0%

  - Physical Op: Nested Loops
    - Logical Op: Inner Join
    - Sub Cost: 0.0000134 (0.05%)

  - Physical Op: Filter
    - Logical Op: Filter
    - Sub Cost: 0.000015 (0.01%)

- **Optimal Plan**

  - Physical Op: Nested Loops
    - Logical Op: Inner Join
    - Sub Cost: 0.0073275836
    - Cost: 0.000236 (0.09%)

  - Physical Op: Nested Loops
    - Logical Op: Inner Join
    - Sub Cost: 0.0073275836
    - Cost: 0.000236 (0.09%)

  - Physical Op: Index Seek
    - Logical Op: Index Seek
    - Sub Cost: 0.004015815 (0.03%)
Ultrawrap Architecture Summary

Compile Time

- Putative Ontology
- Generate
- Create View
- TripleView Schema
- Schema Data

Run Time

- SPARQL to SQL
- SQL Optimizer
- Query Plan
- Data

The process involves generating a putative ontology, creating views, and then optimizing SPARQL queries to SQL for runtime execution.
Current Implementation

- Running on Microsoft SQL Server
- Jess Rule Engine
- Initial test on BSBM on 1 million triples, execution time is close to running time of native SQL queries on relational data

<table>
<thead>
<tr>
<th>Query</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jena SDB</td>
<td>4.41</td>
<td>4.33</td>
<td>6.27</td>
<td>7.12</td>
<td>12.36</td>
<td>1.45</td>
<td>11.94</td>
<td>6.69</td>
<td>8.38</td>
<td>5.39</td>
<td>2.76</td>
<td>4.34</td>
</tr>
<tr>
<td>Sesame</td>
<td>2.49</td>
<td><strong>0.86</strong></td>
<td>3.52</td>
<td>3.78</td>
<td>7.31</td>
<td>1.76</td>
<td>15.51</td>
<td>3.02</td>
<td>1.17</td>
<td>3.63</td>
<td>1.49</td>
<td><strong>0.65</strong></td>
</tr>
<tr>
<td>Virtuoso RDF Views</td>
<td>8.29</td>
<td>2.77</td>
<td>9.79</td>
<td>16.13</td>
<td>1.89</td>
<td><strong>0.09</strong></td>
<td>16.59</td>
<td>6.83</td>
<td>2.14</td>
<td>6.96</td>
<td>9.50</td>
<td>3.44</td>
</tr>
<tr>
<td>D2R Server</td>
<td>5.03</td>
<td>5.28</td>
<td>7.93</td>
<td>7.63</td>
<td>222.73</td>
<td>0.94</td>
<td>10.96</td>
<td>12.46</td>
<td>13.37</td>
<td>7.16</td>
<td>30.61</td>
<td>2.55</td>
</tr>
<tr>
<td>Ultrawrap</td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
</tr>
<tr>
<td>Native SQL</td>
<td>0.94</td>
<td>0.67</td>
<td>0.90</td>
<td>0.80</td>
<td>1.09</td>
<td>0.62</td>
<td>0.67</td>
<td>0.72</td>
<td>1.03</td>
<td>1.02</td>
<td>0.94</td>
<td>0.30</td>
</tr>
</tbody>
</table>
Current Work

• Shifted problem to ontology-to-ontology mapping
  – Version 0: query only the Putative Ontology
  – Version 1: Manually mapping layer between Domain Ontology to Putative Ontology
  – Version 2: Automatic identify mappings

• Testing on Oracle, PostgreSQL, Virtuoso

• Road Map
  – Dec 2009: Version 0
  – Feb 2010: Version 1 running on other RDBMS
Thank You

Powered by The Miranker Lab

Research
In
Bioinformatics and the
Semantic Web