Translating SPARQL and SQL to XQuery

Peter M. Fischer
Dana Florescu
Martin Kaufmann
Donald Kossmann
Three Ways of Managing Data

Problem: Need for Integration of Information

Query Language
- Application 1: SQL
- Application 2: XQuery
- Application 3: SPARQL

Runtime
- Application 1: MySQL Server
- Application 2: XQuery Engine
- Application 3: ARQ SPARQL processor

Data Model
- Application 1: Relational
- Application 2: XML
- Application 3: RDF
Outline

- Motivation
- 2 Different Approaches for Integration
  - Data Integration
  - Common Runtime
- Translating SPARQL to XQuery
- Evaluation
- Conclusion
### Data Integration

<table>
<thead>
<tr>
<th></th>
<th>Relational</th>
<th>Semi Structured</th>
<th>Semantic Web</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query Language</strong></td>
<td>SQL</td>
<td>XQuery</td>
<td>SPARQL</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td>DB2, Postgres, Oracle, …</td>
<td>Saxon, Zorba, …</td>
<td>ARQ, Virtuozo, …</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Relational</td>
<td>XML/XDM</td>
<td>RDF</td>
</tr>
</tbody>
</table>

- Can read/write data in different format
Data Integration - Tradeoffs

**Advantage**
- Relatively easy to implement (yet another abstraction/layer)

**Disadvantage**
- Not flexible / bad to optimize
- Unable to exploit all information from different data models
### Common Runtime

<table>
<thead>
<tr>
<th>Query Language</th>
<th>Relational</th>
<th>Semi Structured</th>
<th>Semantic Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>XQuery</td>
<td>SPARQL</td>
<td></td>
</tr>
</tbody>
</table>

- Unified RT supports 3 query languages and 3 data formats.
Common Runtime - Tradeoffs

**Advantage**
- All information available
- Switch between different languages
- Perform optimization on a common abstraction layer
- More possibilities for optimization

**Disadvantage**
- Very complex
- Many contradictory goals
  - E.g. physical representation in tables vs. tree in RDF
## Challenges: Different Foundations

<table>
<thead>
<tr>
<th></th>
<th>SQL</th>
<th>XQuery</th>
<th>SPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fundamental Data Units</strong></td>
<td>Tuples</td>
<td>Atomic Values, Nodes =&gt; Trees</td>
<td>Triples =&gt; Graphs</td>
</tr>
<tr>
<td><strong>Data Model</strong></td>
<td>Unordered bag</td>
<td>Ordered sequence</td>
<td>Unordered bag</td>
</tr>
<tr>
<td><strong>Datatypes</strong></td>
<td>SQL</td>
<td>XML Schema (+)</td>
<td>XML Schema (+)</td>
</tr>
<tr>
<td><strong>Logic type</strong></td>
<td>3-valued</td>
<td>2-valued</td>
<td>3-valued</td>
</tr>
<tr>
<td><strong>Turing-Complete</strong></td>
<td>SQL 2008(?)</td>
<td>Yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Note: no reasoning!
Our approach: XQuery as a Common Runtime

Application 1
 SQL
 xQL to XQuery

Application 2
 XQuery

Application 3
 SPARQL
 xQL to XQuery

XQuery Engine

Common Runtime

XML Collections
Outline

- Motivation
- 2 Different Approaches for Integration
  - Data Integration
  - Common Runtime
- Translating SPARQL to XQuery
- Evaluation
- Conclusion
SPARQL: A Query Language for RDF

General Information

- Name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language.
- SPARQL is available as W3C Recommendation since 2008
- Queries may contain triple patterns, conjunctions, disjunctions, and optional patterns

Query Forms

- **SELECT**: return the value of variables which may be bound by a matching query pattern
- **ASK**: return true if a given query matches and false if not
- **CONSTRUCT**: return an RDF graph by substituting the values in given templates
- **DESCRIBE**: return an RDF graph which defines the matching resource
Translating SPARQL to XQuery (1)

General Translation Rules
- Generic translation to XQuery available for each pattern
- Patterns are composable

Example: Translation of a SPARQL SELECT query to XQuery

<table>
<thead>
<tr>
<th>SPARQL</th>
<th>XQuery</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultSPA := nsListSPA SELECT varListSPA</td>
<td>nsListXQu let $result := patternXQu</td>
</tr>
<tr>
<td>WHERE { patternSPA }</td>
<td>( order by orderListXQu )?</td>
</tr>
<tr>
<td>( ORDER BY orderListSPA )?</td>
<td>return $result([positionXQu])?</td>
</tr>
<tr>
<td>( limitOffsetSPA ) ?</td>
<td></td>
</tr>
</tbody>
</table>

SPARQL                                                                 XQuery
## Translating SPARQL to XQuery (2)

### Translation of Graph Patterns

<table>
<thead>
<tr>
<th>$result_{SPA}$ :=</th>
<th>$result_{XQu}$ :=</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nsList_{SPA}$ SELECT $varList_{SPA}$ WHERE { $pattern_{SPA}$ } ( ORDER BY $orderList_{SPA}$ )? ( $limitOffset_{SPA}$ )?</td>
<td>$nsList_{XQu}$ let $result := pattern_{XQu}$ ( order by $orderList_{XQu}$ )? return $result([position_{XQu}])$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$pattern_{SPA}$ :=</th>
<th>$pattern_{XQu}$ :=</th>
</tr>
</thead>
<tbody>
<tr>
<td>$patternL_{SPA}$ OPTIONAL { $patternR_{SPA}$ }</td>
<td>$xqllib:optional(patternL_{XQu}, patternR_{XQu})$</td>
</tr>
<tr>
<td>{ $patternL_{SPA}$ } UNION { $patternR_{SPA}$ }</td>
<td>(patternL_{XQu}, patternR_{XQu})</td>
</tr>
<tr>
<td>{ $patternL_{SPA}$ } { $patternR_{SPA}$ }</td>
<td>$xqllib:and(patternL_{XQu}, patternR_{XQu})$</td>
</tr>
</tbody>
</table>
Translating SPARQL to XQuery (3)

SPARQL Basic Pattern

Abstract translation rule for a SPARQL Basic Pattern to XQuery

∀ subjName (subjVars(patternSPA))
  for $subjName in xqllib:getSubj()
∀ predName (predVars(patternSPA))
  for $predName in xqllib:getPred($subjName)
∀ objName (objVars(patternSPA))
  for $objName in xqllib:getObj($predName)
( where
∀ constant (constants(patternSPA, subjName, predName, objName))
  $subjName = constant | $predName = constant | $objName = constant
∀ filterCondition filters(filterXqu)
  (and)? filterCondition
)!
return
  <result>
    ∀ varName (vars(patternSPA))
      <varName>{data($varName)}</varName>
  </result>
SPARQL: SELECT Queries

Example of an RDF tree: The Periodic System of Elements
**SPARQL: A Query Language for RDF**

**SPARQL Basic Pattern**

Get the ID and color of all elements which have the name “iron”

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>

SELECT ?element ?color
WHERE {
  ?element pse:name "iron".
}
ORDER BY ?color
```
SPARQL: A Query Language for RDF

SPARQL Basic Pattern

XQuery output

```xquery
let $doc := doc("chemistry.xml")
for $element in $doc/element
for $color in $element/color
where $element/name = "iron"
return <result>
<element>{$element/@ID}</element>
<color>{$element/color}</color>
</result>
```

- **load RDF data**
- **loop for each variable**
- **conditions**
- **output**
SPARQL: A Query Language for RDF

SPARQL Graph Pattern
Get the ID of all compounds for the element with the name “iron”

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>
PREFIX cmp: <http://www.daml.org/2003/01/compounds#>

SELECT ?compound
WHERE {
  ?element pse:name "iron" .
  ?compound cmp:has ?element .
}
```
**SPARQL: A Query Language for RDF**

**SPARQL Graph Pattern**

```
let $doc := doc("chemistry.xml")
for $elem in $doc/element
for $comp in $doc/compound
where $comp/has/@resource = $elem/@ID
and $elem/name = "iron"
return <result>
  <compound>{$comp/@ID}</compound>
</result>
```

**XQuery output**

```
load RDF data
loop for each variable
conditions
output
```
Translation Process

SQL Lexer
- generate tokens
  → token list

SQL Parser
- grammar check
- create object tree
  → object tree

Semantic Analysis
- variable scoping
- rewrites
  → object tree

Optimizer
- specific optimizer
  → object tree

XQuery Renderer
- generic code optimizer
  → object tree

SPARQL Lexer
- generate tokens
  → token list

SPARQL Parser
- grammar check
- create object tree
  → object tree

Semantic Analysis
- variable scoping
  → object tree

Optimizer
- specific optimizer
  → object tree

SPARQL

our contribution

current research

done

done

March 26th, 2011
ETH Zürich Systems Group
Outline

- Motivation
- 2 Different Approaches for Integration
  - Data Integration
  - Common Runtime
- Translating SPARQL to XQuery
- Evaluation
- Conclusion
Measurements: SQL to XQuery

Performance of a simple SQL query (BERLIN SQL 1)
Measurements: SPARQL to XQuery

Performance of a simple SPARQL query (BERLIN SPARQL 1)
Measurements: SQL to XQuery

Performance of a more complex SQL query (BERLIN SQL 8)
Measurements: SPARQL to XQuery

Performance of a more complex SPARQL query (BERLIN SPARQL 8)
Evaluation

- **Criteria**
  - Completeness, Correctness, Efficiency

- **Complete translation of SPARQL+SQL possible**
  (verification required)

- **Processors vs. processors, DBs vs. DBs**
  - Promising results
  - Can reach same order of magnitude than native engines

- **Problems:**
  - Wide range of optimizer quality
    - Differences in join detection
    - Difficult to write general code which can be optimized well by all engines
  - Different types, APIs of engines for indexes
  - Schema & indexing support not standardized and not available for all engines
  - Automatic verification difficult (trial and error)
Outline

- Motivation
- 2 Different Approaches for Integration
  - Data Integration
  - Common Runtime
- Translating SPARQL to XQuery
- Evaluation
- Conclusion
Lessons Learned

- Which language constructs can be optimized well
  - Translate everything into big FLOWR expressions
  - Bad: external function call
  - Effects of predicate push-down varies for different engines

- Write joins such that index can be used
Ongoing work in Evaluation

SPARQL
- Showing correctness by Running SPARQL Test Suite
- Testing effects of various optimizations for different engines
- Index Support

SQL
- Index Support
Summary

- Common runtime for all three languages desirable goal
- Investigated XQuery as basis
- SQL-92 and SPARQL expressible
- Initial performance encouraging, but still many open research issues
Demo

Project URL: http://www.xql2xquery.org

Username: ethz

Password: xquery