Expressive Querying of Semantic Databases with Incremental Query Rewriting

Alexandre Riazanov, UNB Saint John

joint work with Marcelo A. T. Aragão,
Manchester Univ. and Central Bank of Brazil

AWOSS 10.2, Moncton

Semantic Databases: Triplestores and Instance Stores

- RDF triplestores:

subject	***predicate***	***object***
http://www.freewebs.com/riazanov#this	rdf:type	foaf:Person
http://www.freewebs.com/riazanov#this	foaf:name	“Alexandre Riazanov”
http://www.freewebs.com/riazanov#this	foaf:mbox	“mailto:alexr@unb.ca”
	...	

- OWL instance stores: separately, class instances and property assertions.

Semantic Databases

- Semantic Databases don't have fixed schemas ⇒ greater flexibility and no DB design.
- Data in Semantic DBs carries its meaning directly, i.e., can be interpreted directly, even by end users ⇒ more straightforward drill-down and browsing, easier integration.
- **Likely replacement for Relational Databases**.
- Most likely basis for **Web-scale querying of semantic documents**: relevant RDF or OWL documents can be loaded in a transient Semantic DB before a query is computed on it.

Querying Semantic DB

- Typically, SPARQL or similar languages are used:

```
SELECT ?x ?y
WHERE { ?x rdf:type foaf:Person . ?x foaf:name ?y }
```

- Predicates and individuals can be defined externally in ontologies and other KBs **axiomatically**: *foaf:Person rdfs:subClassOf foaf:Agent*.
- **Querying should be done modulo the KBs** (deductive queries), but current implementations can only cope with very inexpressive KBs – little more than just taxonomies.
- Techniques for Semantic Querying of Relational Databases can be transfered to Semantic DB. Direct application is possible when Semantic DBs are implemented on top of RDBs.

Talk Outline

- Semantic Querying of Relational Databases in general.
- Expressive querying with Incremental Query Rewriting.
- Querying RDB-based Sesame triplestores.

Semantic Querying of RDB in General

Main scenario:

- One or more relational databases (RDB). Tables are conceptually treated as sets of ground logical assertions:

UNIV_DB_TAKES_COURSE	STUDENT	COURSE		
	s1	c1		UNIV_DB_TAKES_COURSE(s1,c1)
	s2	c2		UNIV_DB_TAKES_COURSE(s2,c2)
	...	...	...	

- KBs for the domains: e.g., ontologies (OWL) and rule sets (RuleML):

$$graduateStudent(X) :- takesCourse(X, C),\ graduateCourse(C)$$

- Semantic mapping for RDB schemas:

$$takesCourse(X, C)\ :-\ univ\_db\_takes\_course(X, C)$$

- User/client software formulates logical queries that have to be answered modulo the KB:

$$?-\quad graduateStudent(S),\ memberOf(S, D),\ suborganization(D, 'UNB').$$

$$S = \ 'JohnSmith',\ D = \ 'CSAS'\ ;$$

$$S = \ 'MaryTaylor',\ D = \ 'Math'\ ;$$

$$\ldots$$

Applications

Non-programmer interface to RDB:

- financial analysts, biomedical researchers, criminal investigators, patent examiners, etc., etc., don't (want to) know RDB programming!
  But they are usually able to formulate *complex queries in terms of their domains* or, at least, browse the virtual assertion space.

Applications by domain:

- Business Intelligence: *How many of our customers live within 2 km from a Future Shop?*
- especially Data Federation: before/instead of doing an 8 month/$1M ETL product, describe you RDBs semantically and play with queries.
- especially personalisation for e-marketing: advertise the right goods/services to the right customers by using rules. *A customer with a new laptop is unlikely to be interested in another laptop.*
- Health Care IT and Bioinformatics: cohort selection for trials, semantic access to biological data.

Existing Approaches

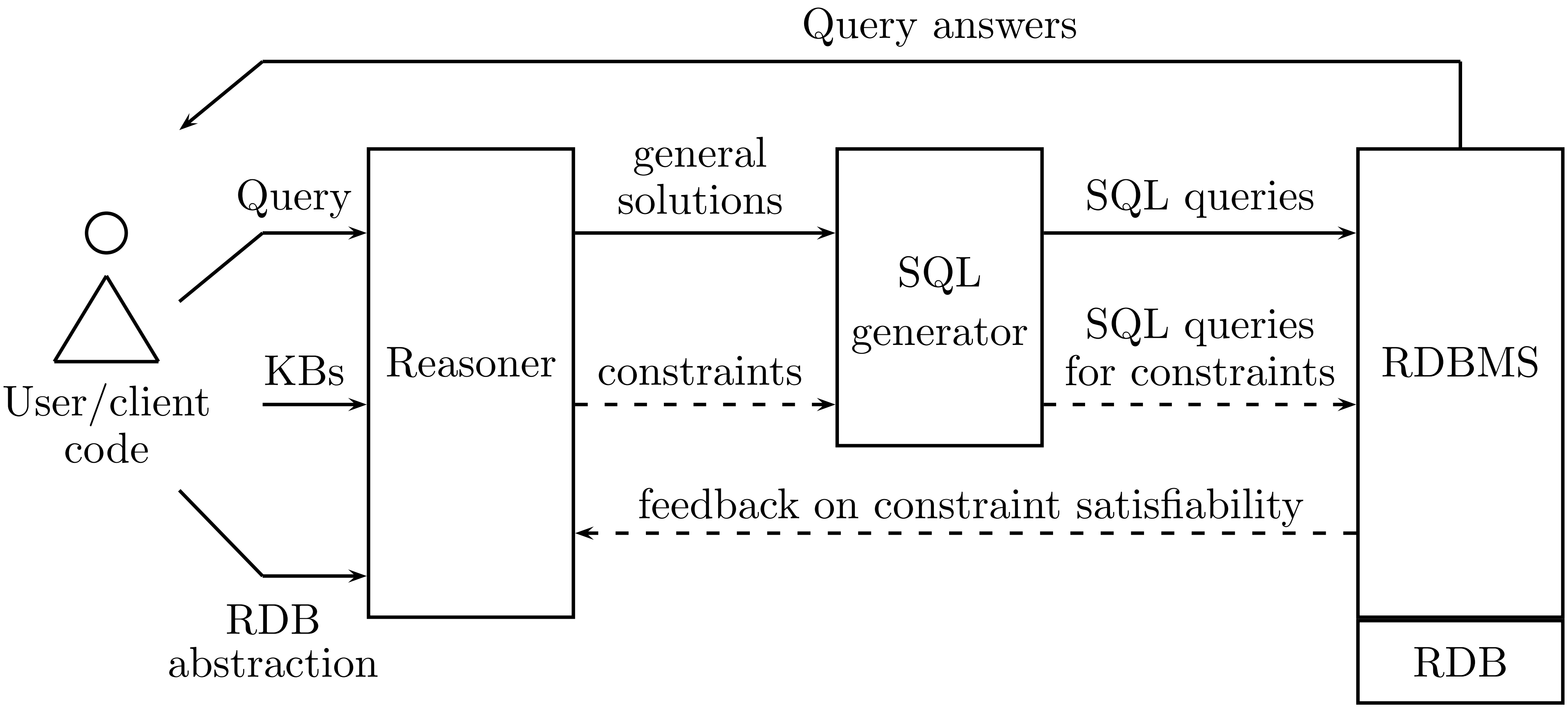
- **Direct reasoning** on RDB Tables (XSTONE): RDB table rows are loaded as facts into a reasoner.
  - Expressivity is practically unlimited. **Inherently inefficient**: some amount of reasoning is required to generate each answer.
- **Query rewriting:** use the KB axioms to rewrite a semantic query into something executable, e.g., SQL. The semantic query is rewritten into one or more SQL queries whose union is equivalent to the original query.
  - **Efficient**: answers are found in bulk. **Limited expressivity**: super simple DLs or rules. Popular: Tambis, Semantic SQL, QuOnto, OntoGrate.
- **Hybrid approach:** rewrite the original query with the inexpressive part of your KB, load the results and do the expressive reasoning on the pre-filtered data.
  - The RDB is usually preprocessed into an instance store. **Good expressivity**, but still may require **reasoning on a per-answer basis**. Several low-profile implementations (M.J. O'Connor, IBM).

Our Approach: Incremental Query Rewriting with Resolution

- **Query rewriting**, but with very expressive KBs (full First-Order Logic).
- A reasoner rewrites a semantic query into a possibly infinite sequence of **schematic answers**, using a *resolution*-based procedure. The schematic answers can be straightforwardly represented as **SQL queries**.
- **Completeness**: eventually the union of produced SQL queries covers all answers to the semantic query.
- Albeit **no general termination guarantee**: the reasoner may keep working indefinitely, producing or not producing new SQL queries.

Very expressive querying at reasonable cost.

Architecture



Example

Query: $?-\ stud(S),\ memberOf(S,''UNB'').$

Schematic answers:

$answer(S) :-$
$takes\_course(S, C),$
$gr\_course(C),$
$member\_of(S, D),$
$suborganization(D,''UNB'')$

$answer(S) :-$
$registered\_students(S, Deg),$
$Deg \simeq \text{``}MSc'',$
$member\_of(S, D),$
$suborganization(D,''UNB'')$

SQL query for the 1st schematic answer:

```
SELECT takes_course.subject AS S
FROM takes_course, gr_course, member_of, suborganization
WHERE takes_course.object = gr_course.instance
      AND takes_course.subject = member_of.subject
      AND member_of.object = suborganization.subject
      AND suborganization.object = “UNB”
```

Prototype Implementation

- Vampire was modified to do query rewriting. Accepts full FOL in the TPTP format, generates a stream of schematic answers in XML.
- A Java program translates the schematic answers to SQL. Optionally, can query a MySQL or Derby DB directly and print the results.
- User writes a semantic mapping as TPTP axioms –

  table predicates ↔ ontological primitives,

  and binds the *table predicates* to *SQL queries*, possibly complex.
- Components are glued with a shell script.
- Converters from OWL 2, SWRL and RIF, to TPTP.
- Successful experiment with a toy University DB queried modulo LUBM ontology.

Sesame Triplestores with RDBMS

- Sesame is an API. It allows different implementations of triplestores.
- RDB-based triplestores are conventions about how RDF triples are stored in RDB tables.
- Specify RDB layout, e.g., one table for all predicates or a separate binary table for each predicate.
- Specify how SPARQL queries are mapped to SQL.
- Address performance issues, e.g., sharing for URIs and/or literals.
- Can be viewed as semantically mapped RDB schemas: **RDF predicates are defined as SQL views** over the schemas ⇒ Incremental Query Rewriting and other semantic querying techniques for RDB are directly applicable.

MySQL RDF Store

- A separate table for each predicate:
  lubm:emailAddress → *emailAddress_25*(subj,obj), *rdf:type* → *type_3*(subj,obj)
- Surrogate keys for URIs: *URI_VALUES*(pk id: INTEGER, value: VARCHAR(255)). Improves join speed and saves memory.
- Surrogate keys for literals: *LABEL_VALUES*(pk id: INTEGER, value: VARCHAR(255)).

MySQL Triplestore Layout as a Semantic Mapping

- lubm:emailAddress(subject,object) :-

```
SELECT subj_uri.value AS subject, obj_val.value AS object
  FROM emailAddress_25 rel, URI_VALUES subj_uri,
       LABEL_VALUES obj_val
 WHERE rel.subj = subj_uri.id
   AND rel.obj = obj_val.id
```

- lubm:GraduateCourse(instance) :-

```
          SELECT instance_uri.value as instance
            FROM type_3 rel, URI_VALUES instance_uri, URI_VALUES class_uri
           WHERE instance_uri.id = rel.subj
             AND class_uri.id = rel.obj
             AND class_uri.value =
'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#GraduateCourse'
```

Experiment

- Lehigh University Benchmark – toy Semantic DB for universities.
- OWL ontology: departments, research groups, students, professors, etc. Quite expressive: DL features outside RDFS, e.g., inverse and transitive properties.
- 14 standard queries modulo the ontology.
- Data for 5 universities: 100K instance assertions, 500K property assertions.
- Severe performance problems in the first run: obvious need for optimisations.

Optimisations

- **Transitivity** of *lubm:subOrganizationOf* makes the reasoner generate infinitely many schematic answers: .. *lubm:subOrganizationOf*(X_0,X_1), .. , *lubm:subOrganizationOf*(X_{100},X_{101}), ..
  A simple limit on the chain length helps.
- MySQL 5.2 does not inline inline views ⇒ query planning is inefficient. Using named views helps.
- One promising optimisation is yet to be implemented: use the surrogate keys for URIs in joins instead of the URI strings. Temporary solution: index on *URI_VALUES.value*.
- After these optimisations, all queries except two are answered in real time. #6 and #8 take 9 min and 15 min.

Conclusions and Future Work

- Incremental Query Rewriting – possible new foundation for much more expressive querying in Semantic DB.
- Next step: SPARQL implementation. Can be triplestore-independent: SPARQL-to-SPARQL rewriting.