National Technical
University of Athens

HEALLINK
Hellenic Academic Libraries Link

An Approach for the Incremental Export of Relational Databases into RDF Graphs

N. Konstantinou, D.-E. Spanos, D. Kouis, N. Mitrou

13th Hellenic Data Management Symposium (HDMS'15)

European Union
European Social Fund
OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS
MANAGING AUTHORITY
Co-financed by Greece and the European Union
NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

About this work

- Started as a project in 2012 in the National Documentation Centre, to offer Linked Data views over its contents
- Evolved as a standards-compliant open-source tool
- First results presented in IC-ININFO’13 and MTSR’13
- Journal paper of the presentation of the software used was awarded an Outstanding Paper Award
- Latest results presented in WIMS’14
- Revised and extended version in a special issue in IJAIT (2015)

Outline

- Introduction
- Background
- Proposed Approach
- Measurements
- Conclusions

Introduction

- Information collection, maintenance and update is not always taking place directly at a triplestore, but at a RDBMS
- It can be difficult to change established methodologies and systems
  - Especially in less frequently changing environments, e.g. libraries
- Triplestores are often kept as an alternative content delivery channel
- Newer technologies need to operate side-by-side to existing ones before migration

Mapping Relational Data to RDF

- Synchronous or Asynchronous RDF Views
- Real-time SPARQL-to-SQL or Querying the RDF dump using SPARQL
- Queries on the RDF dump are faster in certain conditions, compared to round-trips to the database
- Difference in the performance more visible when SPARQL queries involve numerous triple patterns (which translate to expensive JOIN statements)
- In this paper, we focus on the asynchronous approach
  - Exporting (dumping) relational database contents into an RDF graph

Incremental Export into RDF (1/2)

- Problem
  - Avoid dumping the whole database contents every time
  - In cases when few data change in the source database, it is not necessary to dump the entire database
- Approach
  - Every time the RDF export is materialized
    - Detect the changes in the source database or the mapping definition
    - Insert/delete/update only the necessary triples, in order to reflect these changes in the resulting RDF graph

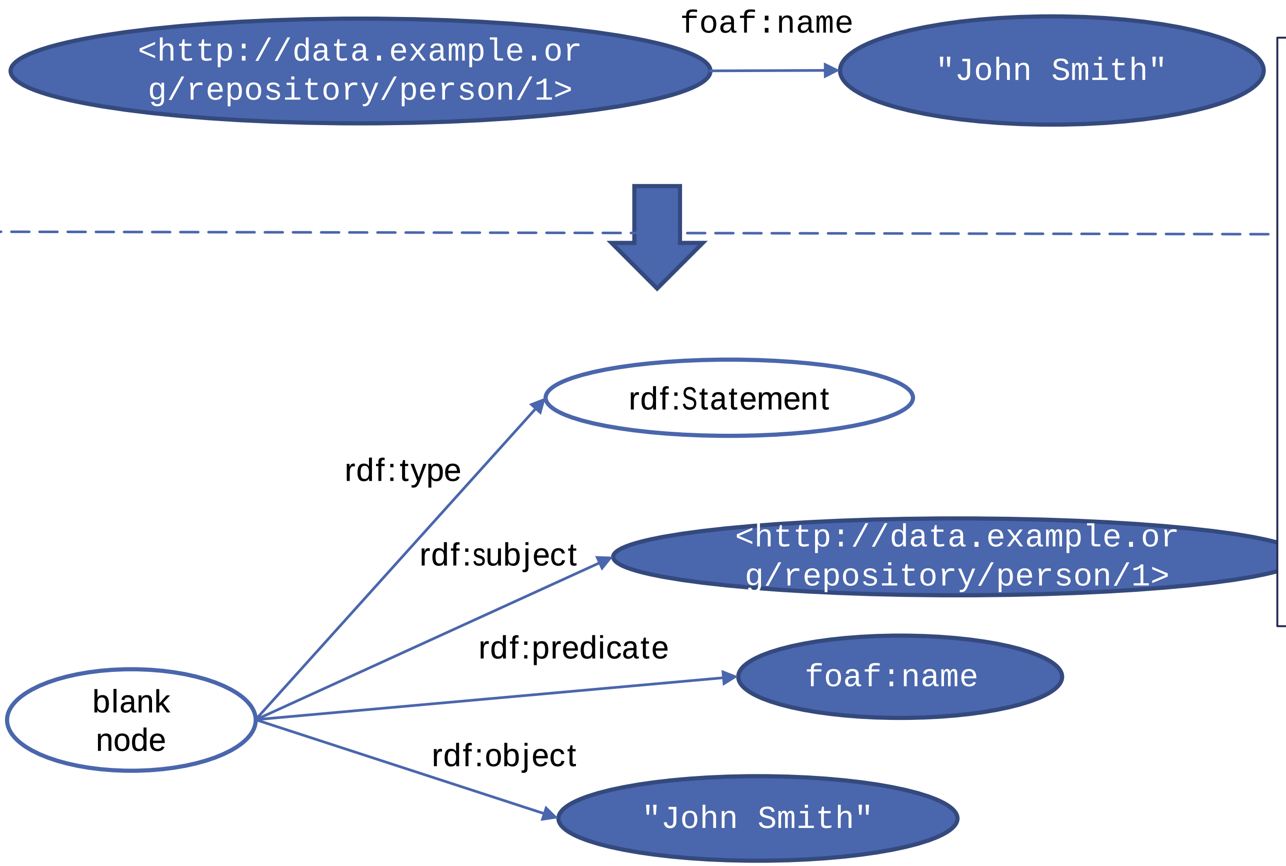
Incremental Export into RDF (2/2)

- Incremental *transformation*
  - Each time the transformation is executed, only the part in the database that changed should be transformed into RDF
- Incremental *storage*
  - Storing (persisting) to the destination RDF graph only the triples that were modified and not the whole graph
  - Possible only when the resulting RDF graph is stored in a relational database or using Jena TDB

Outline

- Introduction
- Background
- Proposed Approach
- Measurements
- Conclusions

Reification in RDF

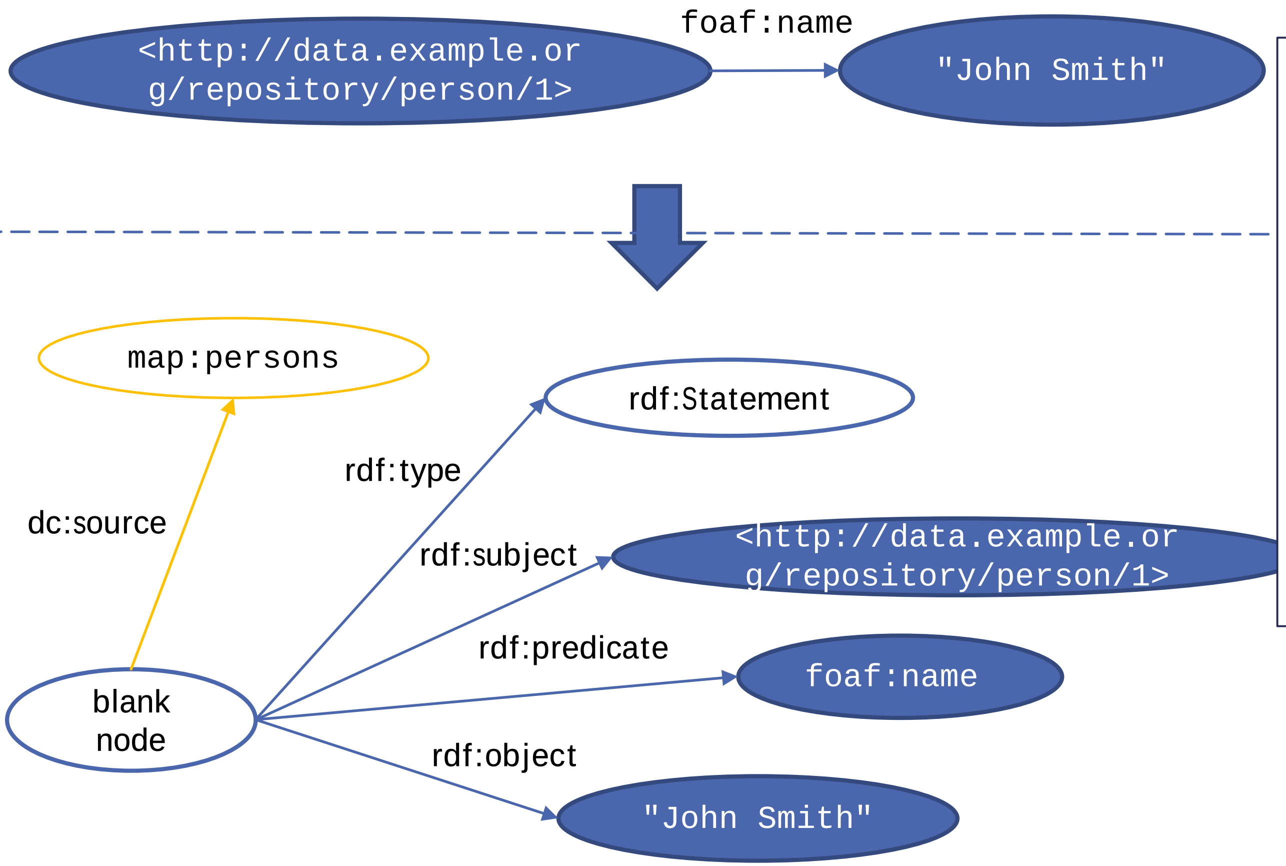



```
<http://data.example.org/repository/person/1>
foaf:name "John Smith" .
```

becomes

```
[] a rdf:Statement ;
   rdf:subject
<http://data.example.org/repository/person/1> ;
   rdf:predicate foaf:name ;
   rdf:object "John Smith" ;
   dc:source map:persons .
```

Reification in RDF

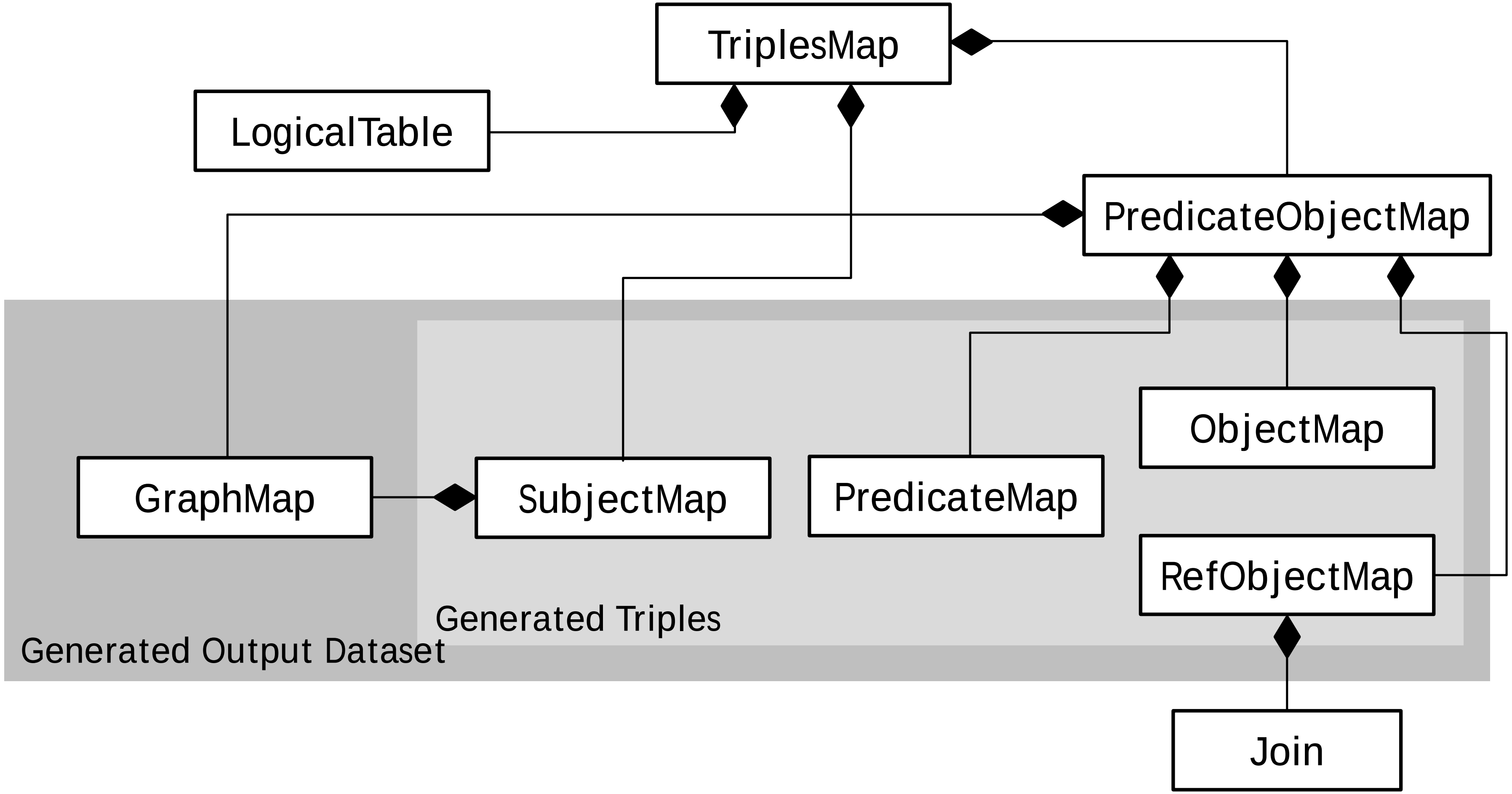



```
<http://data.example.org/repository/person/1>
foaf:name "John Smith" .

becomes

[] a rdf:Statement ;
   rdf:subject
<http://data.example.org/repository/person/1> ;
   rdf:predicate foaf:name ;
   rdf:object "John Smith" ;
   dc:source map:persons .
```

- Ability to annotate every triple
- E.g. the mapping definition that produced it

R2RML

- RDB to RDF Mapping Language
- A W3C Recommendation, as of 2012
- Mapping documents contain sets of Triples Maps



Triples Maps in R2RML (1)

- Reusable mapping definitions
  - Specify a rule for translating each row of a *logical table* to zero or more RDF triples
  - A *logical table* is a tabular SQL query result set that is to be mapped to RDF triples
  - *Execution* of a triples map generates the triples that originate from the specific result set

EMP

EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

Example R2RML mapping

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "EMP" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{EMPNO}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "ENAME" ];
    ].
```

Example output data

```
<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".
```

Triples Maps in R2RML (2)

- An example

```
map:persons
    rr:logicalTable [ rr:tableName '"eperson"'; ];
    rr:subjectMap [
        rr:template 'http://data.example.org/repository/person/{"eperson_id"}';
        rr:class foaf:Person; ];
    rr:predicateObjectMap [
      rr:predicate foaf:name;
      rr:objectMap [ rr:template '{"firstname"} {"lastname"}' ;
                                  rr:termType rr:Literal; ] ].
```

An R2RML Mapping Example

```
@prefix map: <#>.
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
map:persons-groups
  rr:logicalTable [ rr:tableName '"epersongroup2eperson"'; ];
  rr:subjectMap [
    rr:template 'http://data.example.org/repository/group/{"eperson_group_id"}';
    ];
  rr:predicateObjectMap [
    rr:predicate foaf:member;
    rr:objectMap [ rr:template 'http://data.example.org/repository/person/{"eperson_id"}';
    rr:termType rr:IRI; ] ].
```

Table epersongroup2eperson

	id [PK] integer	eperson_group_id integer	eperson_id integer
1	499501	1	1
2	499502	1	2
3	499503	1	3
4	499504	1	4
5	499505	1	5
6	499506	1	6

```
<http://data.example.org/repository/group/1> foaf:member
     <http://data.example.org/repository/person/1> ,
     <http://data.example.org/repository/person/2> ,
     <http://data.example.org/repository/person/3> ,
     <http://data.example.org/repository/person/4> ,
     <http://data.example.org/repository/person/5> ,
     <http://data.example.org/repository/person/6> .
```

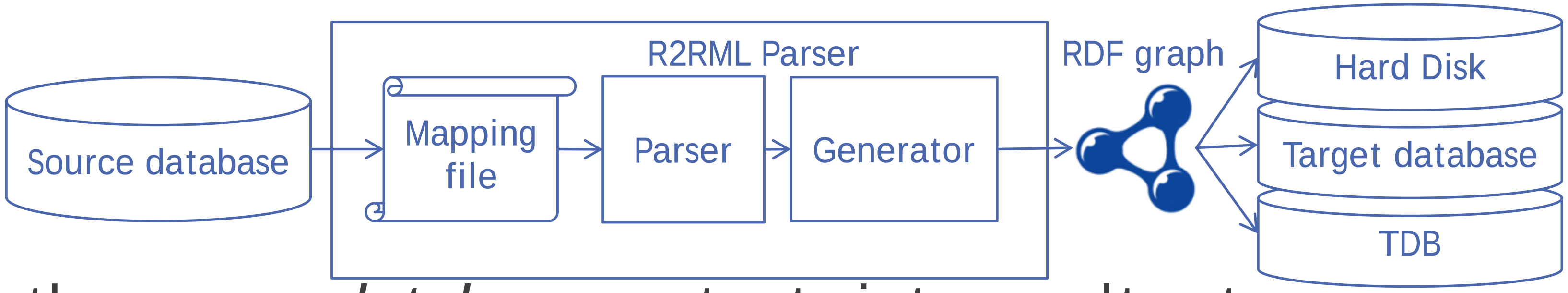
Outline

- Introduction
- Background
- Proposed Approach
- Measurements
- Conclusions

The R2RML Parser tool

- An R2RML implementation
- Command-line tool that can export relational database contents as RDF graphs, based on an R2RML mapping document
- Open-source (CC BY-NC), written in Java
  - Publicly available at https://github.com/nkons/r2rml-parser
  - Worldwide interest (Ontotext, Abbvie, Financial Times)
- Tested against MySQL, PostgreSQL, and Oracle
- Output can be written in RDF/OWL
  - N3, Turtle, N-Triple, TTL, RDF/XML(-ABBREV) notation, or Jena TDB backend
- Covers most (not all) of the R2RML constructs (see the wiki)
- Does not offer SPARQL-to-SQL translations

Information Flow (1)



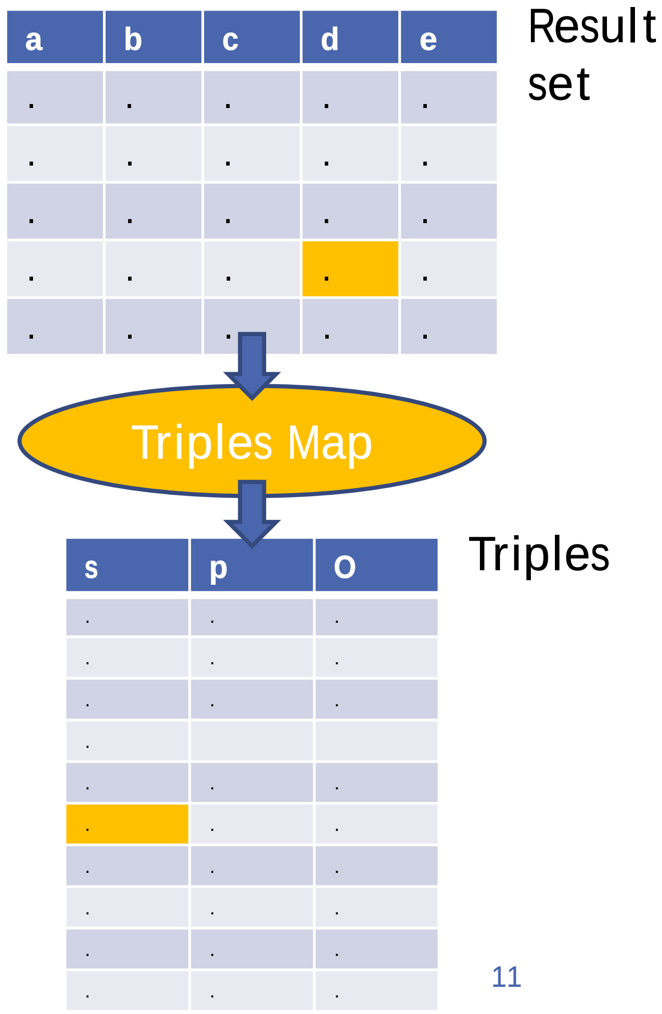
- Parse the *source database* contents into result sets
- According to the R2RML *Mapping File*, the *Parser* generates a set of instructions to the *Generator*
- The *Generator* instantiates in-memory the resulting RDF graph
- Persist the generated *RDF graph* into
  - An RDF file in the Hard Disk, or
  - In Jena's relational database (eventually rendered obsolete), or
  - In Jena's TDB (Tuple Data Base, a custom implementation of B+ trees)
- Log the results

Information Flow (2)

- Overall generation time is the sum of the following:
  - t1: Parse mapping document
  - t2: Generate Jena model in memory
  - t3: Dump model to the destination medium
  - t4: Log the results
    - In incremental transformation, the log file contains the *reified model*
      - A model that contains only reified statements
    - Statements are annotated with the Triples Map URI that produced them

Incremental RDF Triple Transformation

- Basic challenge
  - Discover, since the last time the incremental RDF generation took place
    - Which database tuples were modified
    - Which *Triples Maps* were modified
  - Then, perform the mapping only for this altered subset
- Ideally, we should detect the exact changed database cells and modify only the respectively generated elements in the RDF graph
  - However, using R2RML, *the atom* of the mapping definition becomes the *Triples Map*



Incremental *transformation*

- Possible when the resulting RDF graph is persisted on the hard disk
- The algorithm does not run the entire set of triples maps
  - Consult the log file with the output of the last run of the algorithm
    - MD5 hashes of triples maps definitions, SELECT queries, and respective query resultsets
  - Perform transformations only on the changed data subset
    - I.e. triples maps for which a change was detected
- The resulting RDF graph file is erased and rewritten on the hard disk
- Retrieve unchanged triples from the log file
  - Log file contains a set of reified statements, annotated as per source Triples Maps definition

Incremental *storage*

- Store changes without rewriting the whole graph
- Possible when the resulting graph is persisted in an RDF store
  - Jena's TDB in our case
  - The output medium must allow additions/deletions/modifications at the triples level

Proposed Approach

- For each *Triples Map* in the *Mapping Document*
  - Decide whether we have to produce the resulting triples, based on the logged MD5 hashes
- Dumping to the Hard Disk
  - Initially, generate the number of RDF triples that correspond to the source database
  - RDF triples are logged and annotated as reified statements
  - Incremental generation
    - In subsequent executions, modify the existing reified model, by reflecting only the changes in the source database
- Dumping to a database or to TDB
  - No log is needed, storage is incremental by default

Outline

- Introduction
- Background
- Proposed Approach
- Measurements
- Conclusions

Measurements Setup

- An Ubuntu server, 2GHz dual-core, 4GB RAM
- Oracle Java 1.7, Postgresql 9.1, Mysql 5.5.32
- 7 DSpace (dspace.org) repositories
  - 1k, 5k, 10k, 50k, 100k, 500k, 1m items, respectively
  - Random data text values (2-50 chars) populating a random number (5-30) of Dublin Core metadata fields
- A set of SQL queries: complicated, simplified, and simple
  - In order to deal with database caching effects, the queries were run several times, prior to performing the measurements

Query Sets

- Complicated
  - 3 expensive JOIN conditions among 4 tables
  - 4 WHERE clauses
- Simplified
  - 2 JOIN conditions among 3 tables
  - 2 WHERE clauses
- Simple
  - No JOIN or WHERE conditions

```
SELECT i.item_id AS item_id, mv.text_value AS text_value
FROM item AS i, metadatavalue AS mv,
metadataschemaregistry
AS msr, metadatafieldregistry AS mfr WHERE
msr.metadata_schema_id=mfr.metadata_schema_id AND
mfr.metadata_field_id=mv.metadata_field_id AND
mv.text_value is not null AND
i.item_id=mv.item_id AND
msr.namespace='http://dublincore.org/documents/dcmi-
terms/'
AND mfr.element='coverage'
AND mfr.qualifier='spatial'
```

Q1: 28.32 *

```
SELECT i.item_id AS item_id, mv.text_value AS text_value
FROM item AS i, metadatavalue AS mv,
metadatafieldregistry AS mfr WHERE
mfr.metadata_field_id=mv.metadata_field_id AND
i.item_id=mv.item_id AND
mfr.element='coverage' AND
mfr.qualifier='spatial'
```

Q2: 21.29 *

```
SELECT "language", "netid", "phone",
"sub_frequency","last_active", "self_registered",
"require_certificate", "can_log_in", "lastname",
"firstname", "digest_algorithm", "salt", "password",
"email", "eperson_id"
FROM "eperson" ORDER BY "language"
```

Q3: 12.52 *

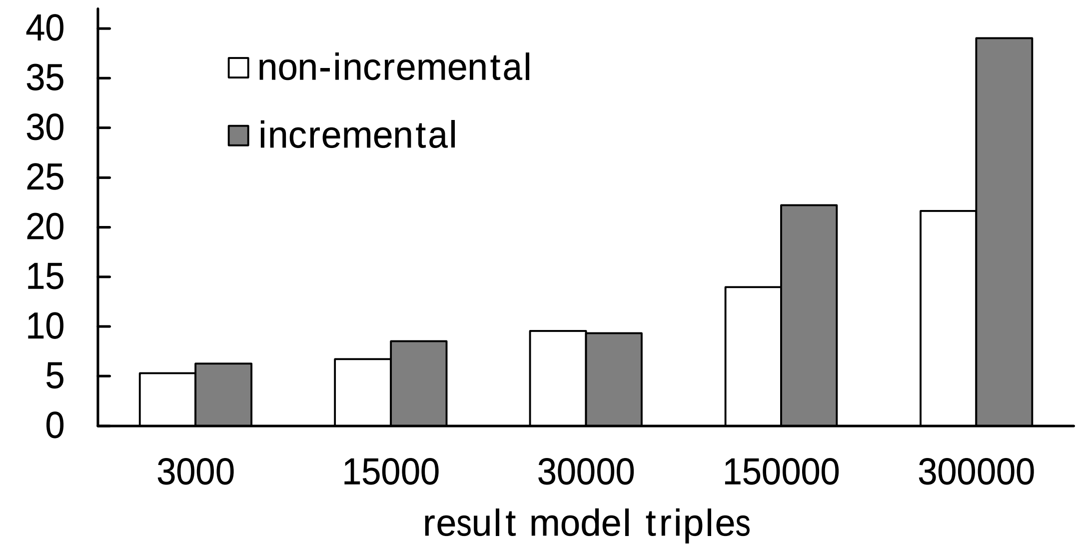
* Score obtained using PostgreSQL's EXPLAIN

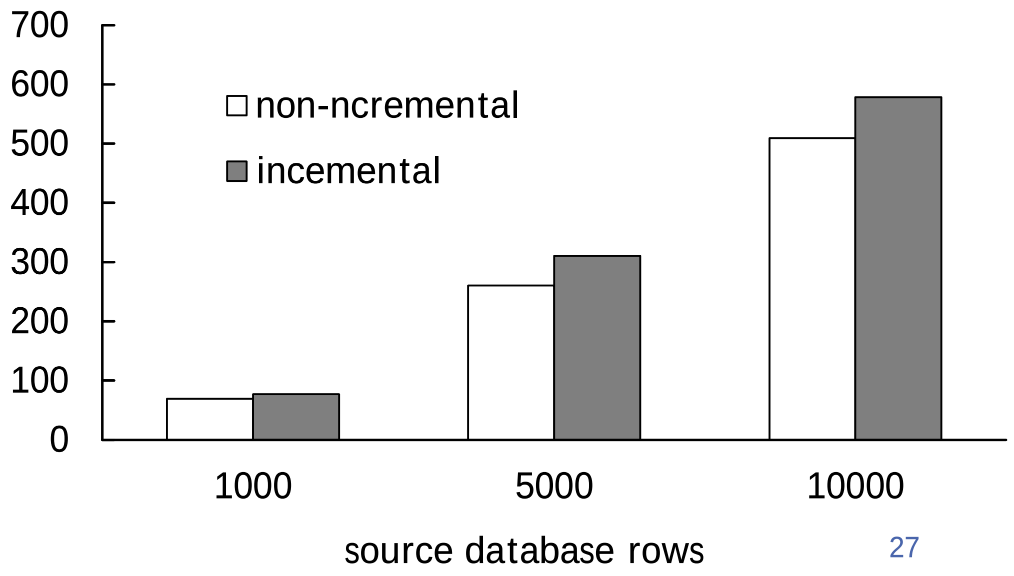
Measurements Results

- Exporting to an RDF File
- Exporting to a Relational Database
- Exporting to Jena TDB

Exporting to an RDF File (1)

- Export to an RDF file
- Simple and complicated queries, initial export
- Initial incremental dumps take more time than non-incremental, as the reified model also has to be created



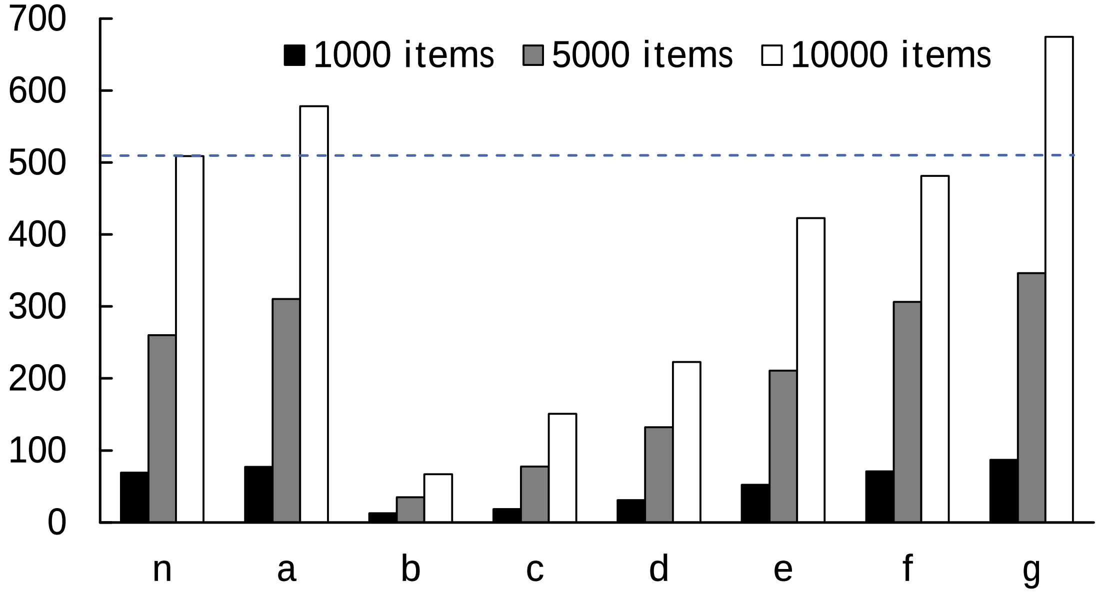


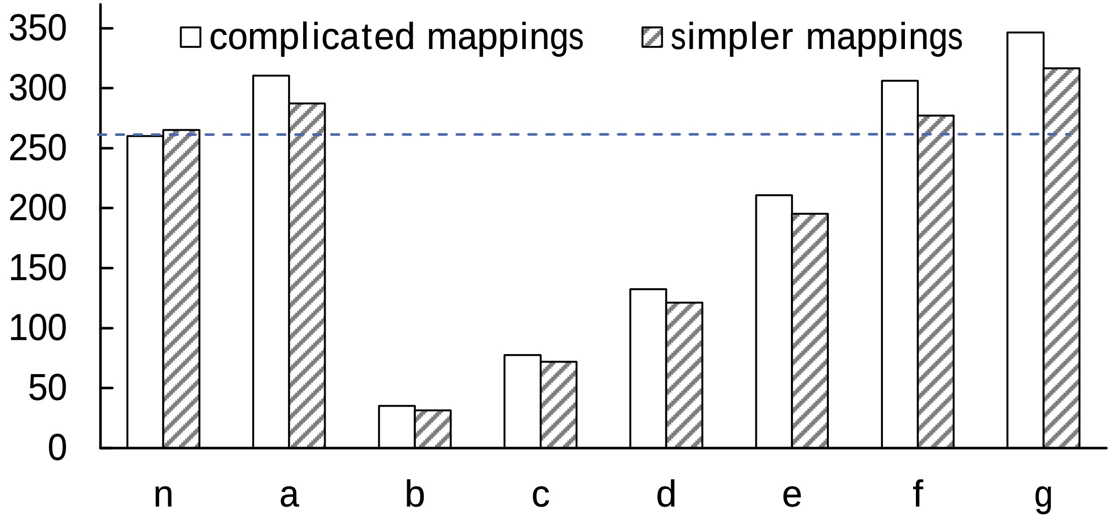
Exporting to an RDF File (2)

- 12 Triples Maps

n	non-incremental mapping transformation
a	incremental, for the initial time
b	0/12 (no changes)
c	1/12
d	3/12
e	6/12
f	9/12
g	12/12

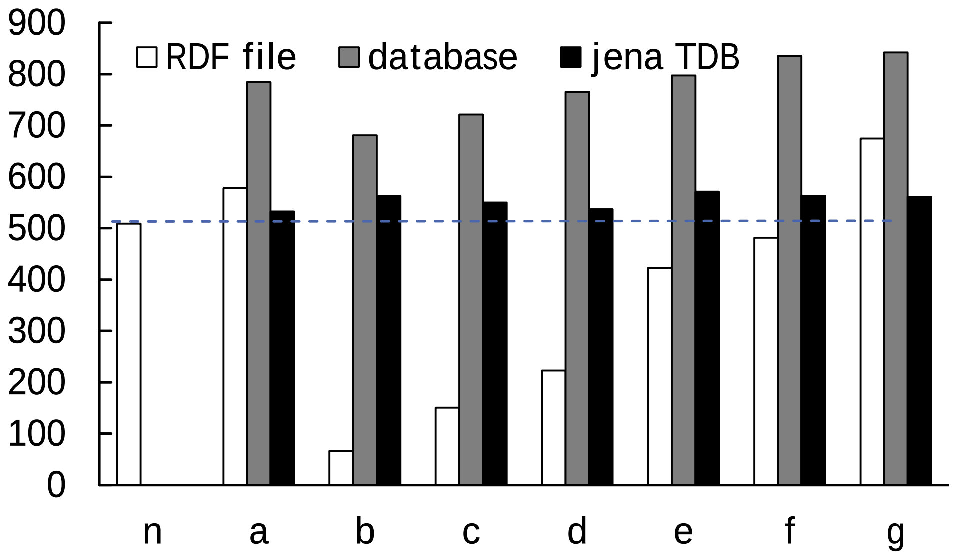
Data change (c–g)



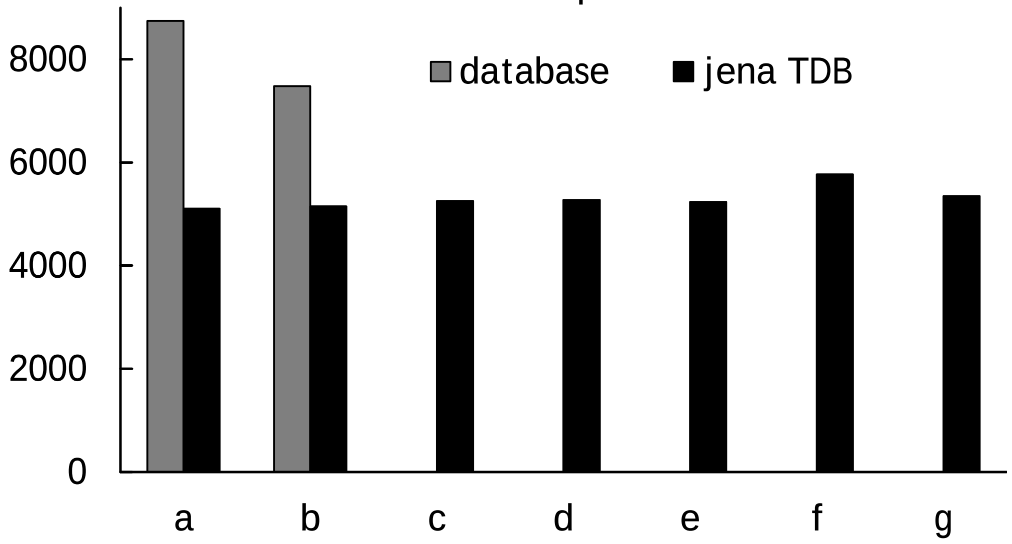


Exporting to a Database and to Jena TDB

- Jena TDB is the optimal approach regarding scalability



~180k triples



~1.8m triples

Outline

- Introduction
- Background
- Proposed Approach
- Measurements
- Conclusions

Conclusions (1)

- The approach is efficient when data freshness is not crucial and/or selection queries over the contents are more frequent than the updates
- The task of exposing database contents as RDF could be considered similar to the task of maintaining search indexes next to text content
- Third party software systems can operate completely based on the exported graph
  - E.g. using Fuseki, Sesame, Virtuoso
- TDB is the optimal solution regarding scalability
- Caution is still needed in producing de-referenceable URIs

Conclusions (2)

- On the efficiency of the approach for storing RDF on the Hard Disk
  - Good results for mappings (or queries) that include (or lead to) expensive SQL queries
    - E.g. with numerous `JOIN` statements
  - For changes that can affect as much as ¾ of the source data
  - Limitations
    - By physical memory
    - Scales up to several millions of triples, does not qualify as “Big Data”
  - Formatting of the logged model *did* affect performance
    - RDF/XML and TTL try to pretty-print the result, consuming extra resources
    - N-TRIPLES is optimal

Future Work

- Hashing Result sets is expensive
  - Requires re-run of the query, adds an "expensive" `ORDER BY` clause
- Further study the impact of SQL complexity on the performance
- Investigation of two-way updates
  - Send changes from the triplestore back to the database

Questions?