



National Technical University of Athens  
School of Electrical and Computer Engineering  
Multimedia, Communications & Web Technologies



# Incremental Export of Relational Database Contents into RDF Graphs

Nikolaos Konstantinou, Dimitris Kouis, Nikolas Mitrou

By Dr. Nikolaos Konstantinou



European Union  
European Social Fund



MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS  
MANAGING AUTHORITY

Co-financed by Greece and the European Union



EUROPEAN SOCIAL FUND

# Outline

- Introduction
- Proposed Approach and Measurements
- Discussion and Conclusions

# Introduction

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

# Mapping Relational Data to RDF

- No one-size-fits-all approach
- Synchronous
  - Real-time
  - Real-time SPARQL-to-SQL
- Asynchronous RDF Views
  - Ad hoc RDF Views
  - 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, not all of the initial information that lies in the database should be transformed into RDF, but only the one that changed
- Incremental *storage*
  - Storing (persisting) to the destination RDF graph only the triples that were modified and not the whole graph
  - Only when the resulting RDF graph is stored in a relational database or using Jena TDB
  - Regardless to whether the transformation took place fully or incrementally

# R2RML and *Triples Maps*

- RDB to RDF Mapping Language
- A W3C Recommendation, as of 2012
- Triples Map: a reusable mapping definition
  - Specifies 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 a specific result set (logical table)

EMP			
EMPNO	ENAME	JOB	DEPTNO
INTEGER PRIMARY KEY	VARCHAR(100)	VARCHAR(20)	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".
```

# The R2RML Parser

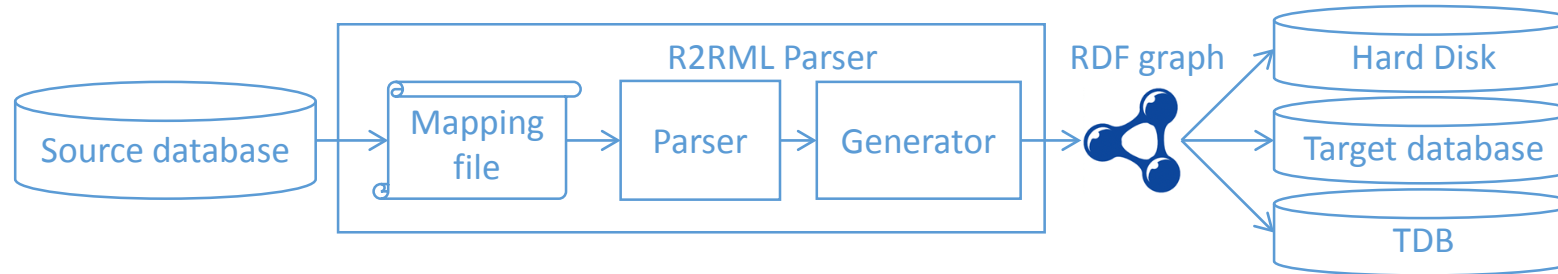
- An R2RML implementation
- Command-line tool that can export relational database contents as RDF graphs, based on an R2RML mapping document
- Open-source (CC BY-NC), written in Java
  - Publicly available at <https://github.com/nkons/r2rml-parser>
- Tested against MySQL and PostgreSQL
- Output can be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



# Outline

- Introduction
- **Proposed Approach and Measurements**
- Discussion and Conclusions

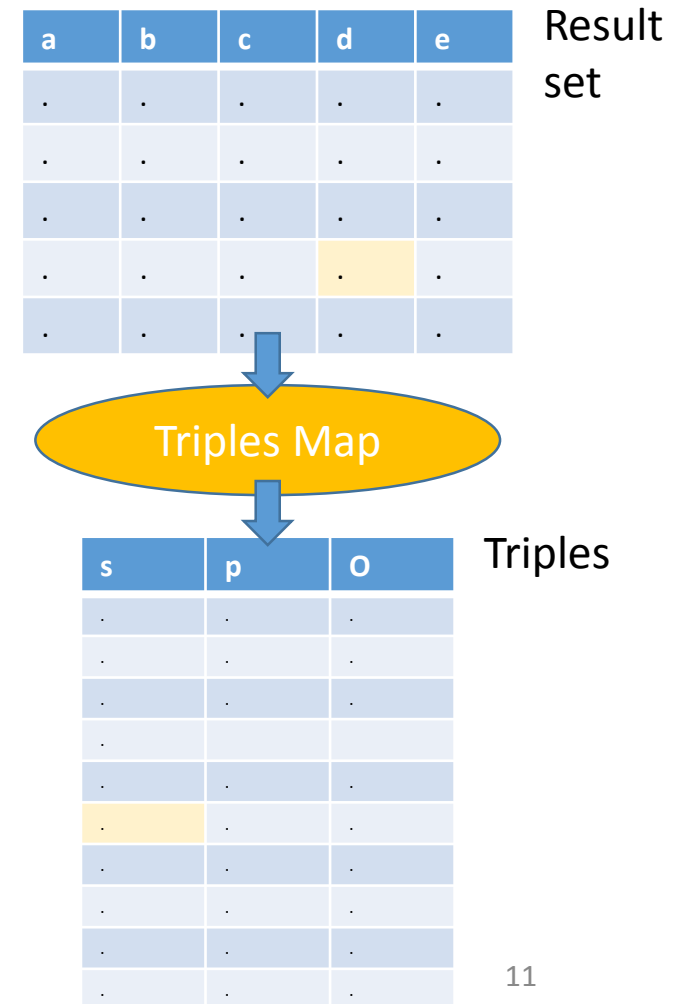
# Information Flow



- 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, or
  - In Jena's TDB (Tuple Data Base, a custom implementation Of B+ Trees)
- Log the results

# Incremental RDF Triple Generation

- 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*



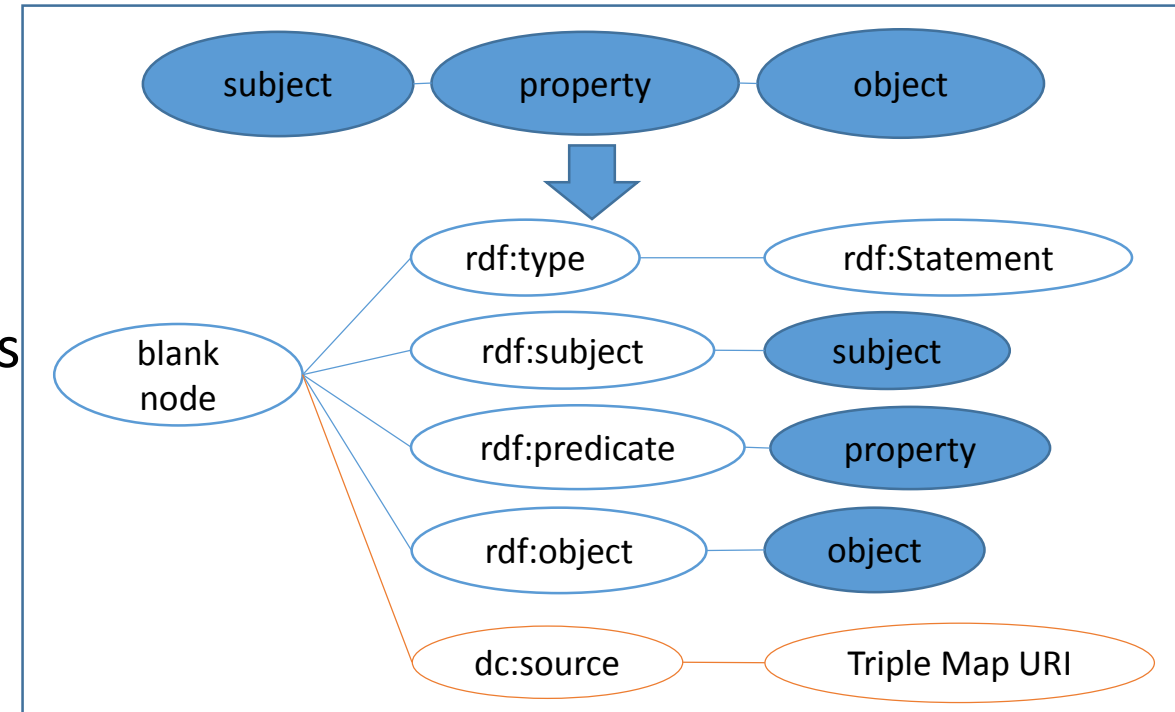
# Keeping Track

- Reification

- Allows assertions about RDF statements
- “Reified” model
  - A model that contains only reified statements
  - Stores the Triples Map URI that produced each triple

- Logging

- Store MD5 hashes of
  - Triples Maps, SELECT queries, Result sets
  - A change in any of the hashes triggers execution of the Triples Map



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



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

# 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 a number of RDF triples
  - RDF triples are logged as reified statements, followed by a provenance note
  - Incremental generation
    - In subsequent executions, modify the existing reified model, by reflecting only the changes in the source database
- Dumping to the Database or TDB
  - No log is needed, storage is incremental by default

# 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
- A set of complicated, a set of simplified, and a set of simple queries
  - 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

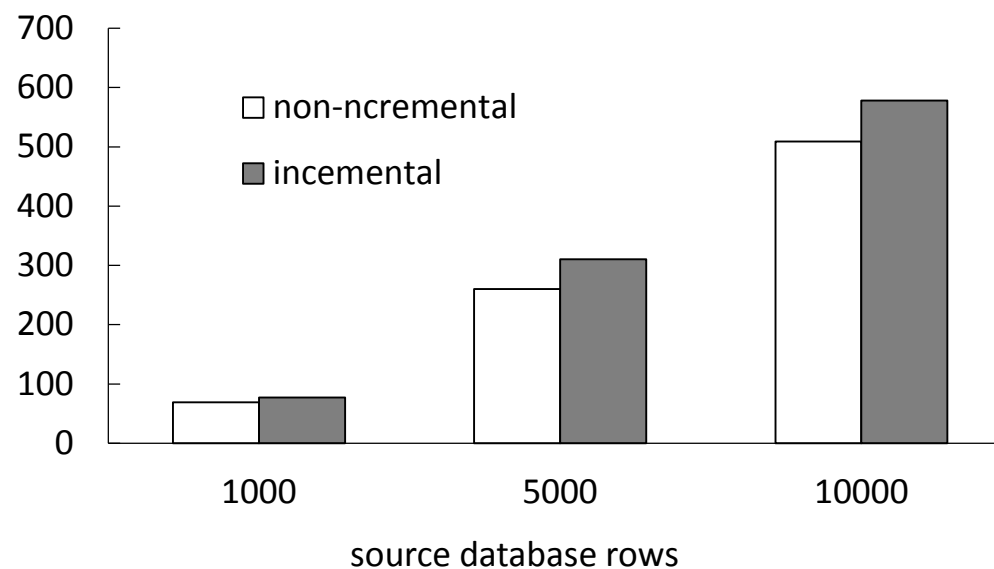
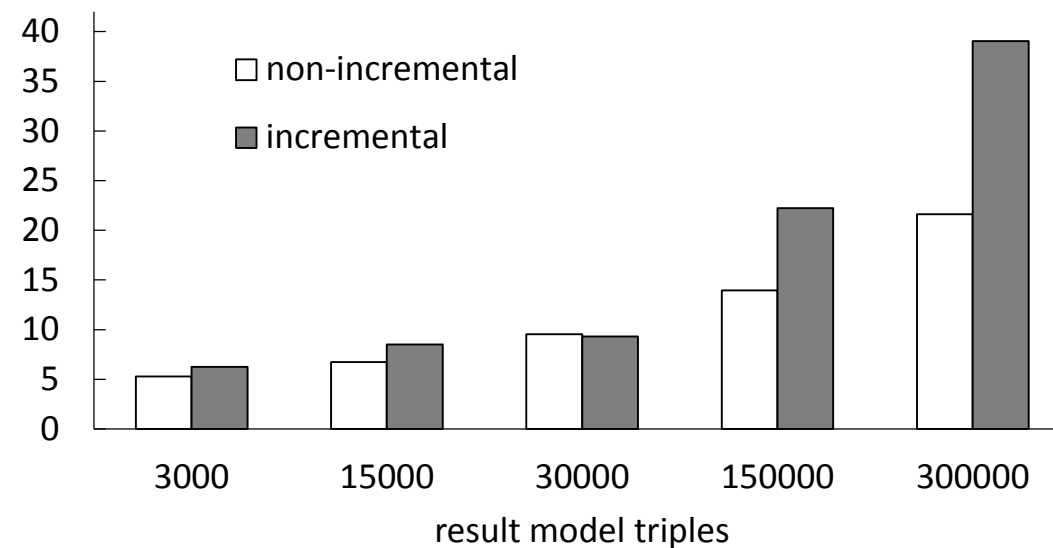
```
SELECT i.item_id AS item_id, mv.text_value AS text_value
FROM item AS i, metadatavalue AS mv,
metadataschemaregistry
AS msr, metadatafieldregistry AS mfr WHERE
msr.metadata_schema_id=mfr.metadata_schema_id AND
mfr.metadata_field_id=mv.metadata_field_id AND
mv.text_value is not null AND
i.item_id=mv.item_id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

```
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'
```

```
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"
```

# Measurements (1/3)

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





# Measurements (2/3)

- 12 Triples Maps

a. non-incremental mapping transformation

b. incremental, for the initial time

c. 0/12

d. 1/12

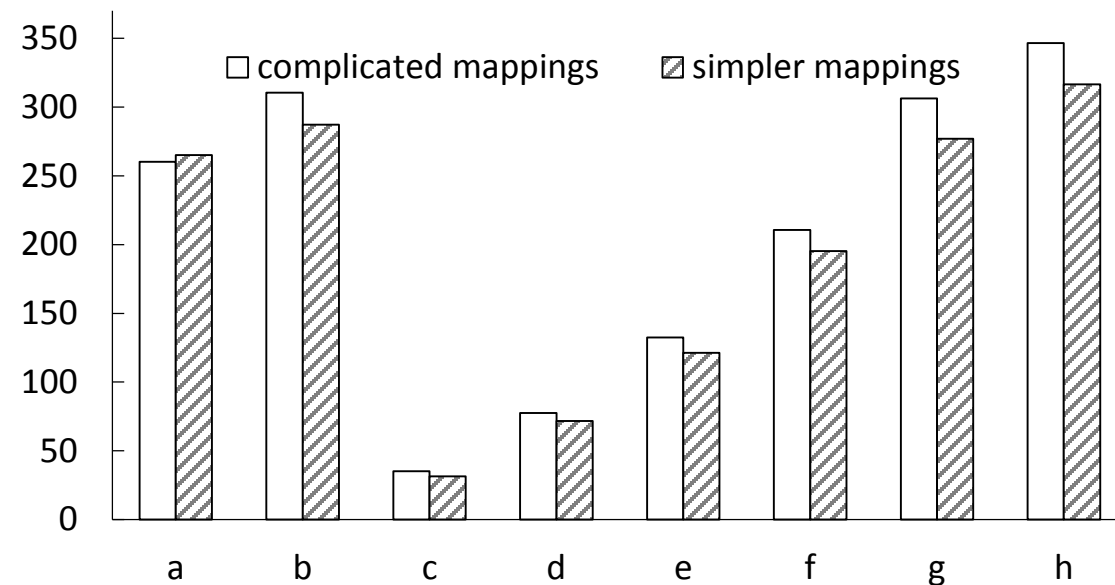
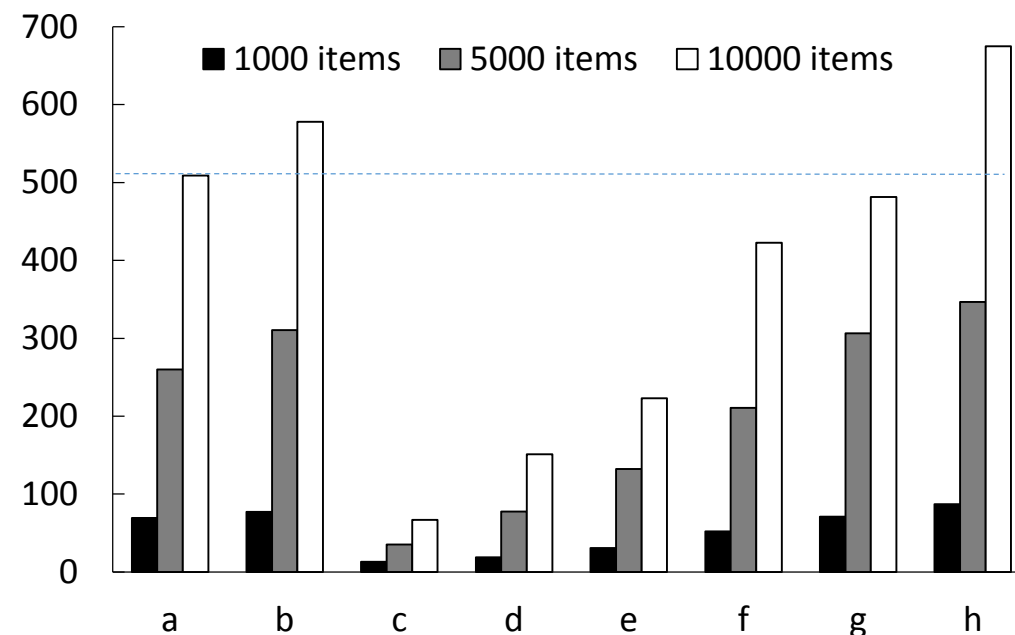
e. 3/12

f. 6/12

g. 9/12

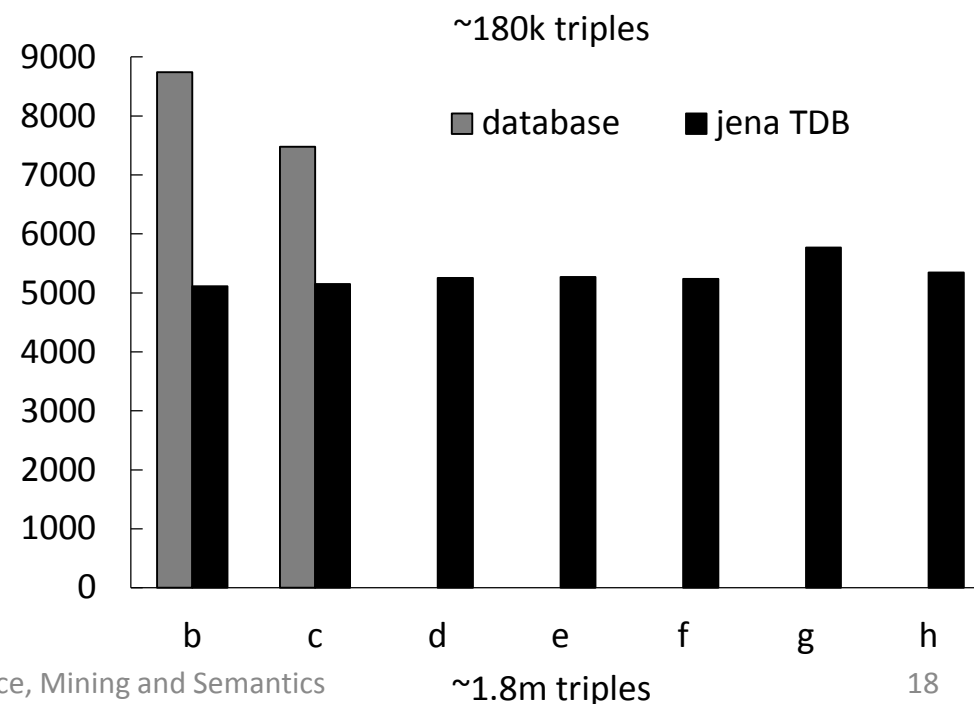
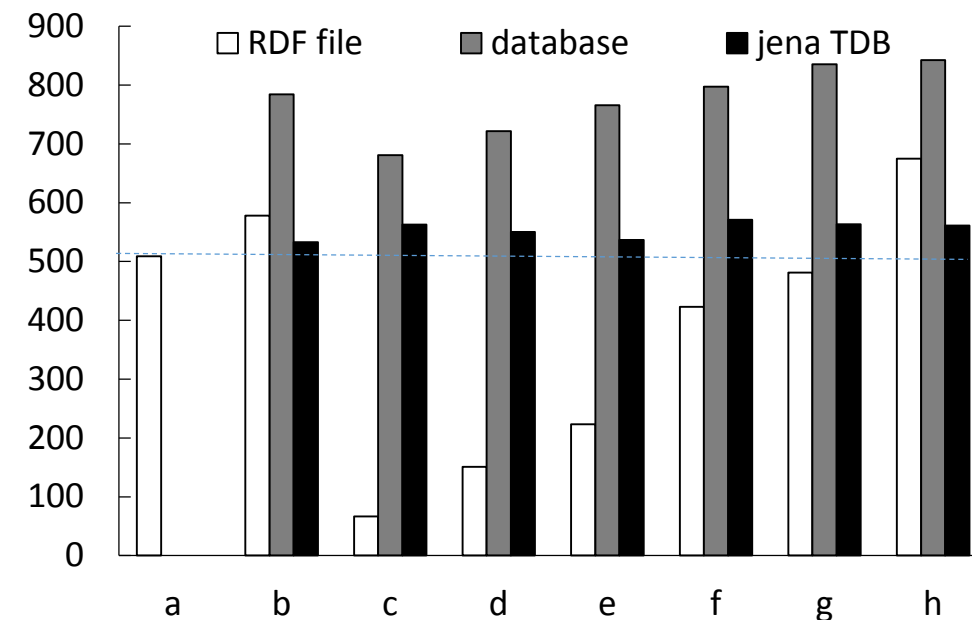
h. 12/12

Data change



# Measurements (3/3)

- Similar overall behavior in cases of up to 3 million triples
- Poor relational database performance
- Jena TDB is the optimal approach regarding scalability



# Outline

- Introduction
- Proposed Approach and Measurements
- Discussion and Conclusions

# Discussion (1/2)

- 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
- Updates can be pushed or pulled from the database
- TDB is the optimal solution regarding scaling
- Caution is still needed in producing de-referenceable URIs

# Discussion (2/2)

- On the efficiency of the approach for storing 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  $\frac{3}{4}$  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 reified model *did* affect performance
    - RDF/XML and TTL try to pretty-print the result, consuming extra resources, N-TRIPLES is the optimal

# Room for Improvement

- 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
- Reification is currently being reconsidered in RDF 1.1 semantics
  - Named graphs being the successor
- Investigation of two-way updates
  - Send changes from the triplestore back to the database

Thank you for your attention!

Questions?