Data Models Matter Less Than You Think

Peter Boncz (CWI and VU University)
Keynote Graph-TA (March 4, 2016)
Keynote Statements

(1) Even if data models seem to be very different, the techniques to manage data are common among them (at least from a database architect perspective).

(2) Some datasets often assumed to belong to very different models are structurally very similar (RDF, graph, relational).
- 1970-
  Relational Data Model, SQL Query Language, Entity Relationship Modeling

- 1980-
  Object Oriented Data models, OQL query language, UML
    - Object-Relational

- 1990-
  XML, XPath & XQuery query languages, XML Schema
    - JSON

- 2000-
  RDF, SPARQL query Languages, Ontologies, OWL
    - Graph Data Models, Cypher query language
Tree Encoding: XPath Accellerator

Node-based relational encoding of XQuery's data model

<table>
<thead>
<tr>
<th>Pre</th>
<th>Size</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

pre + size - level = post

Peter Boncz
Pathfinder - MonetDB/XQuery
IBM Amaden 14-01-2005

11 years ago
a new data model does not imply a necessity for “everything new” (storage, compression, query optimization, execution)

• Virtuoso SPARQL ➔ a SQL system
• MonetDB XQuery ➔ XPath on top of relational algebra

“pointer based navigation is more efficient than relational join?” NO!
pointer swizzling de/serialization, join index, row-IDs

graph navigation = relational join = graph navigation = relational join = ...

proven techniques used and often invented in relational data management systems are not by themselves “relational”. They are data management techniques, widely applicable (not to be dismissed).

• relational hash-join? B-trees? Bloom-filters?
• relational dynamic-programming bottom-up enumeration?
• relational query algebra?
Are LOD Knowledge Graphs proper graphs?
Deriving an Emergent Relational Schema from RDF Data

Minh-Duc Pham\(^\Delta\)  Linnea Passing\(^\circ\)  Orri Erling\(^\circ\)  Peter Boncz\(^\circ\)
m.d.pham@vu.nl  passing@in.tum.de  oerling@openlinksw.com  boncz@wci.nl
\(^\Delta\)Vrije Universiteit Amsterdam, The Netherlands
\(^\circ\)Technische Universität München, Germany
\(^\circ\)OpenLink Software, UK

Abstract

We motivate and describe techniques that allow to detect an “emergent” relational schema from RDF data. We show that on a wide variety of datasets, the found structure explains well over 90% of the RDF triples. Further, we also describe technical solutions to the semantic challenge to give short names that humans find logical to these emergent tables, columns and relationships between tables. Our techniques can be exploited in many ways, e.g., to improve the efficiency of SPARQL systems, or to use existing SQL-based applications on top of any RDF dataset using a RDBMS.

RDF data can be stored in a tabular manner. Each triple 
\( (s, p, o) \) (subject, property, object) columns\(^1\). SQL systems tend to be more efficient than triple stores, because the latter need query plans with many self-joins — one per SPARQL triple pattern. Not only are these extra joins expensive, but because the complexity of query optimization is exponential in the amount of joins, SPARQL query optimization is much more complex than SQL query optimization. As a result, large SPARQL queries often execute with a suboptimal plan, to much performance detriment. RDBMS’s can further store data efficiently e.g. using advanced techniques such as column-wise compression, table partitioning, materialized views and multi-dimensional data clustering. These techniques require insight in the (tabular) structure of the
Main Problems in RDF Data Management

- Bad query plans
- Low storage locality
- Lack of user schema insight
### RDF Triple Indexing

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>book1</td>
<td>has_title</td>
<td>“Pride &amp; Prejudice”</td>
</tr>
<tr>
<td>book1</td>
<td>has_author</td>
<td>“Austen”</td>
</tr>
<tr>
<td>book1</td>
<td>isbn_no</td>
<td>“960-425-059-0”</td>
</tr>
<tr>
<td>book2</td>
<td>has_title</td>
<td></td>
</tr>
<tr>
<td>book2</td>
<td>has_author</td>
<td>“Pecker”</td>
</tr>
<tr>
<td>book2</td>
<td>isbn_no</td>
<td></td>
</tr>
</tbody>
</table>

**SPO Index**

<table>
<thead>
<tr>
<th>P</th>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_author</td>
<td>book0</td>
<td></td>
</tr>
<tr>
<td>has_author</td>
<td>book1</td>
<td>“Austen”</td>
</tr>
<tr>
<td>has_author</td>
<td>book2</td>
<td>“Pecker”</td>
</tr>
<tr>
<td>isbn_no</td>
<td>book0</td>
<td></td>
</tr>
<tr>
<td>isbn_no</td>
<td>book1</td>
<td>“960-425-059-0”</td>
</tr>
<tr>
<td>isbn_no</td>
<td>book2</td>
<td></td>
</tr>
</tbody>
</table>

**PSO Index**

- Most current RDF systems store data with triples sorted on various permutations
  - SPO, PSO, OPS, POS, OSP, SOP,
    - PSO – a bit like relational “column store”
    - SPO – a bit like relational “row store”
Bad Query Plans

- **Have unnecessary joins**
  - All subject having property `<isbn_no>` always has property `<has_author>`, but query plan still needs a join for these properties to construct the answer
  - Problems: **query optimization explosion** + **costly join operations**

- **Hard to find the optimal join order**
  - Being unaware of **structural correlations** makes it difficult to **estimate the join hit ratio** between triple patterns
  - SPARQL queries are very join-intensive
Main Problems in RDF Data Management

- Bad query plans
- Low storage locality
- Lack of user schema insight
Low Storage Locality

- Impossible to formulate **clustered index** or **partitioning scheme** without the notion of classes/tables (DBA would say “store all Book triples clustered by Year”)

- Exhaustive indexes for all permutations of S, P, O do not create real locality (contrary to common belief)

```sparql
SELECT ?a, ?n WHERE
{
  ?b <has_author>  ?a.
  ?b <in_year>  ?y.
  ?b <isbn_no>  ?n.
  FILTER (?y = 1997)
}
```

Using **POS index** for quick range selection (in_year,1997,?s)

Need **repeated lookups** into a **PSO index** for each attribute

⇒ **No locality**
Unclustered Index: Random Access Horrors

does NOT scale!!

yet...

all RDF stores rely on this
RDF “clustered index”: S-identifiers should follow some PO ordering ➔ S-identifiers now chosen at random 😞
Main Problems in RDF Data Management

- Bad query plans
- Low storage locality
- Lack of user schema insight
Lack of User Schema Insight

- RDF data does not have explicit schema
  - difficult to formulate SPARQL queries
  - would be good to get a schema (summary)

- Many more tools for relational data access, than for RDF
  - try to expose the regular part of RDF triple set as SQL
Recovering the Emergent Schema of RDF data

Emergent schema = “rough” schema to which the majority of triples conforms

Recognize:

- **Classes (CS)** – recognize “classes” of often co-occurring properties
- **Relationships (CS)** – recognize often-occurring references between such classes

+ give logical names to these

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rdfs.org/sioc/ns#num_replies>
<http://purl.org/dc/terms/title>
<http://rdfs.org/sioc/ns#has_creator>
<http://purl.org/dc/terms/date>
<http://purl.org/dc/terms/created>
<http://purl.org/rss/1.0/modules/content/encoded>
```

“Book”

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/name>
<http://xmlns.com/foaf/0.1/page>
```

“Author”
What does “schema” mean?

**Relational Schema**
- Describes the structure of the occurring data
- Concept mixing (for convenience)
- Designed for one database (=dataset)

**Semantic Web Schema**
- Purpose: knowledge representation
- Describing a concept universe (regardless data)
- Designed for interoperability in many contexts

Statement: it is useful to have both an (Emergent) Relational and Semantic Schema for RDF data
- useful for systems (higher efficiency)
- useful for humans (easier query formulation)
When is a **Emergent Schema** of RDF data useful?

- **Compact Schema**
  - as few tables as possible
  - homogeneous literal types (few NULLs in the tables)
- **Human-friendly “Labels”**
  - URIs + human-understandable table/column/relationship names
- **High “Coverage”**
  - the schema should match almost all triples in the dataset
- **Efficient to compute**
  - as fast as data import
Step 1: Basic CS discovery

(s1, offers, offer1)
(s1, region, region1)
(s2, offers, offer2)
(s2, offers, offer3)
(s2, region, region1)

... (offer1, availableDeliveryMethods, DHL)
(offer1, description, “Offer data”)
(offer1, hasBusinessFunction, “Sell”)
(offer1, hasEligibleQuantity, 1)
(offer1, hasInventoryLevel, 1)
(offer1, hasStockKeepingUnit, 112)
(offer2, availableDeliveryMethods, DHL)
(offer2, hasPriceSpec, price1)
(offer2, hasStockKeepingUnit, 112)
(offer2, type, Offering)

... (price1, hasCurrency, “EUR”)  
(price1, hasCurrencyValue, “35.99”) 
(price1, hasUnitOfMeasurement, “C62”) 
(price1, valueAddedTaxIncluded, “false”) 
(price1, eligibleTransactionVolume, 0) 
(price1, ... 

... <Example RDF triples>
Characteristic Sets in some well-known RDF datasets

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#triples*</th>
<th>#CS's</th>
<th>#CS’s to cover 90%</th>
<th>Avg. #prop.</th>
<th>#multi-type properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>100M</td>
<td>17</td>
<td>7</td>
<td>5.71</td>
<td>0</td>
</tr>
<tr>
<td>BSBM</td>
<td>100M</td>
<td>51</td>
<td>14</td>
<td>12.35</td>
<td>0</td>
</tr>
<tr>
<td>SP2Bench</td>
<td>100M</td>
<td>554</td>
<td>7</td>
<td>9.8</td>
<td>0</td>
</tr>
<tr>
<td>synthetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MusicBrainz</td>
<td>179M</td>
<td>27</td>
<td>10</td>
<td>4.7</td>
<td>0</td>
</tr>
<tr>
<td>EuroStat</td>
<td>70K</td>
<td>44</td>
<td>8</td>
<td>7.77</td>
<td>0</td>
</tr>
<tr>
<td>DBLP</td>
<td>56M</td>
<td>251</td>
<td>8</td>
<td>13.61</td>
<td>0</td>
</tr>
<tr>
<td>PubMed</td>
<td>1.82B</td>
<td>3340</td>
<td>35</td>
<td>19.27</td>
<td>0</td>
</tr>
<tr>
<td>relational</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WebData.</td>
<td>90M</td>
<td>13914</td>
<td>993</td>
<td>7.79</td>
<td>543</td>
</tr>
<tr>
<td>DBpedia</td>
<td>404M</td>
<td>472270</td>
<td>109831</td>
<td>24.02</td>
<td>1601</td>
</tr>
<tr>
<td>native</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* data created by benchmark data generator

* RDF data from a relational database dump

* real data originating as RDF
- frequency distribution
  - how many CS’s do I need to represent 90% of the triples?
Partial and Mixed Use of Ontologies

<table>
<thead>
<tr>
<th>dataset</th>
<th>partial</th>
<th>%ontology class properties used per CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>37%</td>
<td></td>
</tr>
<tr>
<td>BSBM</td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td>SP2Bench</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>MusicBrainz</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>EuroStat</td>
<td>84%</td>
<td></td>
</tr>
<tr>
<td>DBLP</td>
<td>8%</td>
<td></td>
</tr>
<tr>
<td>PubMed</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>WebData.</td>
<td>33%</td>
<td></td>
</tr>
<tr>
<td>DBpedia</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{cs}_4\)
- \text{dc:}description
- \text{gor:}description
- \text{gor:}validFrom
- \text{gor:}validThrough
- \text{gor:}hasCurrency
- \text{gor:}name
- \text{gor:}eligibleTransactionVolume
- \text{gor:}hasCurrencyValue
- \text{gor:}hasUnitOfMeasurement
- \text{gor:}valueAddedTaxIncluded
- \text{gor:}hasCurrency
- \text{gor:}hasCurrencyValue
- \text{gor:}hasUnitOfMeasurement
- \text{gor:}valueAddedTaxIncluded
- \text{gor:}hasMaxCurrencyValue
- \text{gor:}hasMinCurrencyValue

(prefix gor: [http://purl.org/goodrelations/v1#](http://purl.org/goodrelations/v1#)
prefix dc: [http://purl.org/dc/elements/1.1/](http://purl.org/dc/elements/1.1/)
Step 2: Labeling

Using ontologies to get class and property labels:
• exploit subclass hierarchy
• TF/IDF: how frequent is the label inside the CS divided by global frequency

<table>
<thead>
<tr>
<th>label of rdf:type</th>
<th>subjects CS</th>
<th>all %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thing</td>
<td>100</td>
<td>83</td>
</tr>
<tr>
<td>Organization</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>RadioStation</td>
<td>97</td>
<td>0.2</td>
</tr>
<tr>
<td>Company</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

We try to associate each CS with an ontology class (will not work always)
Step 2: Labeling

Labels assigned by
AAA: using ontologies
AAA: using **discriminative** properties
AAA: using CS’s relationships
Step 3: CS Merging

Semantic merging: based on ontology correspondences (found during labeling)
Step 3: CS Merging

Structural merging: based on class structure and **discriminative** properties
Step 4: Schema Filtering

Goal: make the schema more compact

• remove **infrequent CS**’s (small tables)
  • except “Dimension Tables”
    • CS that is small but is referred to very often
  • Run PageRank on the emergent schema
    • weight is initial frequencies

• remove **infrequent properties**
  • and infrequent relationships
Step 5: Instance Filtering

Reduce the amount of NULLs in relational table representation

- remove infrequent **literal types**
  - e.g. Person.name is a string, but sometimes a number (remove)
- remove infrequent **multi-valued properties**
  - e.g. Person usually has one birthdate (but a few have multiple)
- remove triples to improve **relationship cardinalities**
  - Car usually has 0 or at most one Brand
    - but some have multiple Brands (remove)
Results: compact schemas with high coverage

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Number of tables before merging</th>
<th>Number of tables after merging</th>
<th>Number of tables remove small tables</th>
<th>Coverage - Metric  C (%) remove small tables</th>
<th>Coverage - Metric  C (%) prune infreq. prop.</th>
<th>Coverage - Metric  C (%) final schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>17</td>
<td>13</td>
<td>12</td>
<td>100</td>
<td>100</td>
<td>100.00</td>
</tr>
<tr>
<td>BSBM</td>
<td>51</td>
<td>8</td>
<td>8</td>
<td>100</td>
<td>100</td>
<td>100.00</td>
</tr>
<tr>
<td>SP2B</td>
<td>554</td>
<td>13</td>
<td>10</td>
<td>99.99</td>
<td>99.65</td>
<td>99.65</td>
</tr>
<tr>
<td>MusicBrainz</td>
<td>27</td>
<td>12</td>
<td>12</td>
<td>100</td>
<td>99.9</td>
<td>99.60</td>
</tr>
<tr>
<td>EuroStat</td>
<td>44</td>
<td>10</td>
<td>5</td>
<td>99.73</td>
<td>99.53</td>
<td>99.53</td>
</tr>
<tr>
<td>DBLP</td>
<td>251</td>
<td>9</td>
<td>6</td>
<td>100</td>
<td>99.68</td>
<td>99.60</td>
</tr>
<tr>
<td>PubMed</td>
<td>3340</td>
<td>14</td>
<td>12</td>
<td>100</td>
<td>99.75</td>
<td>99.73</td>
</tr>
<tr>
<td>WebData.</td>
<td>13914</td>
<td>3000</td>
<td>253</td>
<td>98.17</td>
<td>94.37</td>
<td>92.79</td>
</tr>
<tr>
<td>DBpedia</td>
<td>472270</td>
<td>542</td>
<td>234</td>
<td>99.12</td>
<td>96.68</td>
<td>95.82</td>
</tr>
</tbody>
</table>
Are LOD Knowledge Graphs proper graphs?
Results: understandable labels & performance

<table>
<thead>
<tr>
<th>labels</th>
<th>WebData.</th>
<th>DBpedia</th>
</tr>
</thead>
<tbody>
<tr>
<td>top 3</td>
<td>3.6</td>
<td>3.8</td>
</tr>
<tr>
<td>final</td>
<td>4.1</td>
<td>4.6</td>
</tr>
</tbody>
</table>

Table 3: Human survey results on Likert scale

Likert Score: 1=bad ..... 5=excellent

<table>
<thead>
<tr>
<th>RDF Store</th>
<th>Query 3</th>
<th></th>
<th></th>
<th>Query 5</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cold</td>
<td>Hot</td>
<td>Opt. Time</td>
<td>Cold</td>
<td>Hot</td>
<td>Opt. Time</td>
</tr>
<tr>
<td>Virt-Quad</td>
<td>4210</td>
<td>53</td>
<td>40.2</td>
<td>3842</td>
<td>1350</td>
<td>18.6</td>
</tr>
<tr>
<td>Virt-CS</td>
<td>2965</td>
<td>9</td>
<td>5.4</td>
<td>2130</td>
<td>712</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Table 5: Query time (msecs) w/wo the recognized schema
(Cold: First query runtime after re-starting the server
Hot : Run the query 3 times after re-starting the server
Opt. Time: Query optimization time)
(>95%) Relational Storage + (a bit of) Triple Table Storage

Foreign Key Relationship

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Creator</th>
<th>Title</th>
<th>Issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>inproc1</td>
<td>inproceeding</td>
<td>{author3, author4}</td>
<td>&quot;AAA&quot;</td>
<td>conf1</td>
</tr>
<tr>
<td>inproc2</td>
<td>inproceeding</td>
<td>author2</td>
<td>&quot;BBB&quot;</td>
<td>conf1</td>
</tr>
<tr>
<td>inproc3</td>
<td>inproceeding</td>
<td>author3</td>
<td>&quot;CCC&quot;</td>
<td>conf2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Title</th>
<th>Issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf1</td>
<td>Conference</td>
<td>&quot;Conference1&quot;</td>
<td>2010</td>
</tr>
<tr>
<td>conf2</td>
<td>Proceedings</td>
<td>&quot;Conference2&quot;</td>
<td>2011</td>
</tr>
</tbody>
</table>

MonetDB Kernel

Front-Ends

SQL

SPARQL

MonetDB/RDF

MonetDB Kernel

Relational Storage

Triplet Table Storage

SQL

SPARQL
Conclusion

- identified main RDF Store problems
  - data locality, query optimization, query formulation

- Identified different notion of “schema” in relational vs semantic web
  - argument: we need both relational schema and semantic schema
  - can bring relational and semantic data management closer together

- Outlined an algorithm for Emergent Schema detection in RDF
  - compact, high coverage, understandable labels, efficient
Keynote Statements

(1) Even if data models seem to be very different, the techniques to manage data are common among them (at least from a database architect perspective)

(2) some datasets often assumed to belong to very different models are structurally very similar (RDF, graph, relational)