Understanding and Querying Data Regardless of the Data Model
Ioana Manolescu

To cite this version:
Ioana Manolescu. Understanding and Querying Data Regardless of the Data Model. VLDB Summer School 2023, Jul 2023, Cluj-Napoca, Romania. 2023. hal-04190926

HAL Id: hal-04190926
https://inria.hal.science/hal-04190926
Submitted on 30 Aug 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Understanding and Querying Data Regardless of the Data Model

Ioana Manolescu, Inria and Institut Polytechnique de Paris, France
Motivation: Data Heterogeneity

ESTOCADA: Heterogeneous Polystore Performance Through Views
   - How database geeks conquer heterogeneity

ABSTRACT: Understanding Data of Any Data Model
   - How absolute newbies conquer heterogeneity
Motivation: Data Heterogeneity

**ESTOCADA:** Heterogeneous Polystore Performance Through Views
- How database geeks conquer heterogeneity

**ABSTRA:** Understanding Data of Any Data Model
- How absolute newbies conquer heterogeneity
1. Motivation: Data Heterogeneity
2. ESTOCADA: Heterogeneous Polystore Performance Through Views
   - How database geeks conquer heterogeneity
3. ABSTRA: Understanding Data of Any Data Model
   - How absolute newbies conquer heterogeneity
Plan

1. Motivation: Data Heterogeneity
2. View-Based Data Integration
3. ESTOCADA: Heterogeneous Polystore Performance Through Views
   - How database geeks conquer heterogeneity
4. ABSTRA: Understanding Data of Any Data Model
   - How absolute newbies conquer heterogeneity
Part I

Motivation: Heterogeneous Data
Motivation: large-scale data management

**Data management**: a billion $ market
- Industry, trade, telecom, health, government, social, ...
- Data publishing, sharing, reusing
Motivation: large-scale data management

**Data management**: a billion $ market
- Industry, trade, telecom, health, government, social, ...
- Data publishing, sharing, reusing
Motivation: large-scale data management

**Data management**: a billion $ market
- Industry, trade, telecom, health, government, social, ...
- Data publishing & sharing

**Data heterogeneity**
- Different **data formats**: relational, XML, JSON, RDF, property graphs, ...
- Replace but also frequently co-exist with other models
Data model heterogeneity: relational, XML

**Relational** model (also **CSV**): tables

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>3000</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>2500</td>
</tr>
</tbody>
</table>

Query language: **SQL** (ISO Standard)

Most of the world’s DB $
Data model heterogeneity: relational, XML

**Relational** model (also **CSV**): tables

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>3000</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>2500</td>
</tr>
</tbody>
</table>

Query language: **SQL** (ISO Standard)

Most of the world’s DB $

**Tree-structured models**: **XML** trees

```xml
<Employees>
  <employee>
    <name>Alice</name>
    <salary>3000€</salary>
  </employee>
  <employee>
    <name>Bob</name>
    <salary>2500€</salary>
    <email>b@c.com</email>
  </employee>
</Employees>
```
XML documents: not always trees

1. XML elements may have attributes

```xml
<Employees>
  <employee id="e1" friend="e2">
    <name>Alice</name>
    <salary>3000€</salary>
  </employee>
  <employee id="e2">
    <name>Bob</name>
    <salary>2500€</salary>
    <email>b@c.com</email>
  </employee>
</Employees>
```
XML documents: not always trees

2. An *schema* (Document Type Description or XML Schema) may state that *id* is a PK, and *id* is a FK

```xml
<Employees>
  <employee id="e1" friend="e2">
    <name>Alice</name>
    <salary>3000€</salary>
  </employee>
  <employee id="e2">
    <name>Bob</name>
    <salary>2500€</salary>
    <email>b@c.com</email>
  </employee>
</Employees>
```

```
Employees
  employee
    name
    salary
    id
      e1
      Alice
      3000€
  friend
  employee
    name
    salary
    id
      e2
      Bob
      2500€
      b@c.com
```
Data model heterogeneity: JSON

**Tree-structured model**

```
[
  {
    "id": "1", 
    "name": "Alice",
    "salary": "3000",
    "promoted": ["01/2020", "06/2022"]
  },
  {
    "id": "2", 
    "name": "Bob",
    "salary": "2500"
  }
]
```

**JavaScript Object Notation**

Compact format for heterogeneous tree-structured data

Some confusion in the use of [] vs. {}
Key-value pairs are simplified JSON (maps only)

Among the simplest imaginable data models. No order. No schema, no query language. Put/get API.

Very good for: (very) large, potentially heterogeneous, tables, e.g., GoogleTables.
Motivation

Data model heterogeneity: RDF graphs

**RDF**: Resource Description Framework, World Wide Web (W3C) standard (like XML, XPath, XQuery, etc.)
Resources identified by International Resource Identifiers (IRIs, previously URIs)
Resources described by properties (IRIs) with atomic values. A value can be an IRI or a literal.

```
http://emp.org/prDate → "01/2020"
http://emp.org/prDate → "06/2022"
http://emp.org/name → "Alice"
http://emp.org/salary → "2500"
```

Ioana Manolescu, Inria, France

Leveraging Heterogeneous Data

July 2023
RDF graphs

**RDF**: Resource Description Framework, World Wide Web (W3C) standard
Self-describing, unordered, semistructured
Best practice for sharing data, thus very present in the Open Data movement
Query language: SPARQL

*The Linked Open Data Cloud*
RDF graphs

Best practice for sharing data, thus very present in the Open Data movement
Used in knowledge graphs, e.g., DBPedia, Yago, WikiData
E.g., Cluj is https://www.wikidata.org/wiki/Q100188
Adding meaning to RDF graph: types

They are optional
A resource can have zero or more types which can be related or not


http://emp.org/name “Alice”  →  http://emp.org/Employee

“01/2020”

http://emp.org/prDate

http://emp.org/name “Bob”  →  ”2500”

http://emp.org/salary

http://rdf.org/type

http://emp.org/Employee

http://emp.org/name

http://foaf.org/friend

http://emp.org/prDate

http://rdf.org/type
Adding meaning to RDF graphs: types

They are **optional**
A resource can have **zero or more types** which can be related or not
Adding meaning to RDF graphs: ontologies

Optional, allow describing how types and properties are related. Simplest examples:

- http://emp.org/Temp is a subclass of http://emp.org/Employee
- any resource having a http://emp.org/salary is an http://emp.org/Employee

http://emp.org/emp1 "Alice" http://emp.org/emp2 "Bob"

http://emp.org/emp1 http://emp.org/emp2

http://emp.org/salary "2500"

http://emp.org/emp1 http://emp.org/emp2

http://emp.org/salary "2500"
Property graphs

Unlike RDF nodes, PG nodes contain their attributes. Nodes can have zero or more labels. Relationships can also have attributes.

Industry first (Neo4J), then research, then (painfully, now) standards [11]
Motivation

Data model heterogeneity: wrap-up

Models and strengths:

- **Tables**  CSV, relational (most mature; most regular $\Rightarrow$ optimization)
- **Trees**  JSON, XML, K-V: Web content, structured documents, data exchange
- **Graphs**  RDF: Open Data; PGs: latest iteration of heterogeneous databases
Data model heterogeneity: wrap-up

Models and strengths:

- **Tables**: CSV, relational (most mature; most regular ⇒ optimization)
- **Trees**: JSON, XML, K-V: Web content, structured documents, data exchange
- **Graphs**: RDF: Open Data; PGs: latest iteration of heterogeneous databases

Support:

<table>
<thead>
<tr>
<th></th>
<th>Tables</th>
<th>XML</th>
<th>JSON</th>
<th>KV</th>
<th>RDF</th>
<th>PGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XML databases</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSON databases</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KV stores</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF databases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PG databases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Data model heterogeneity: support (https://db-engines.com)

419 systems in ranking, July 2023

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score 2023</th>
<th>Score 2022</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Oracle</td>
<td>Relational, Multi-model</td>
<td>1256.01</td>
<td>+24.54</td>
</tr>
<tr>
<td>2.</td>
<td>MySQL</td>
<td>Relational, Multi-model</td>
<td>1150.35</td>
<td>-13.59</td>
</tr>
<tr>
<td>3.</td>
<td>Microsoft SQL Server</td>
<td>Relational, Multi-model</td>
<td>921.60</td>
<td>-8.47</td>
</tr>
<tr>
<td>4.</td>
<td>PostgreSQL</td>
<td>Relational, Multi-model</td>
<td>617.83</td>
<td>+5.01</td>
</tr>
<tr>
<td>6.</td>
<td>Redis</td>
<td>Key-value, Multi-model</td>
<td>163.76</td>
<td>-3.59</td>
</tr>
<tr>
<td>7.</td>
<td>IBM Db2</td>
<td>Relational, Multi-model</td>
<td>139.81</td>
<td>-5.07</td>
</tr>
<tr>
<td>8.</td>
<td>Elasticsearch</td>
<td>Search engine, Multi-model</td>
<td>139.59</td>
<td>-4.16</td>
</tr>
<tr>
<td>9.</td>
<td>Microsoft Access</td>
<td>Relational</td>
<td>130.72</td>
<td>-3.73</td>
</tr>
<tr>
<td>10.</td>
<td>SQLite</td>
<td>Relational</td>
<td>130.20</td>
<td>-1.02</td>
</tr>
<tr>
<td>11.</td>
<td>Snowflake</td>
<td>Relational</td>
<td>117.69</td>
<td>+3.55</td>
</tr>
<tr>
<td>12.</td>
<td>Cassandra</td>
<td>Wide column</td>
<td>106.53</td>
<td>-2.03</td>
</tr>
<tr>
<td>13.</td>
<td>MariaDB</td>
<td>Relational, Multi-model</td>
<td>96.10</td>
<td>-1.21</td>
</tr>
<tr>
<td>14.</td>
<td>Splunk</td>
<td>Search engine</td>
<td>87.12</td>
<td>-2.34</td>
</tr>
<tr>
<td>15.</td>
<td>Microsoft Azure SQL Database</td>
<td>Relational, Multi-model</td>
<td>78.96</td>
<td>-0.01</td>
</tr>
<tr>
<td>16.</td>
<td>Amazon DynamoDB</td>
<td>Multi-model</td>
<td>78.81</td>
<td>-1.10</td>
</tr>
<tr>
<td>17.</td>
<td>Hive</td>
<td>Relational</td>
<td>72.87</td>
<td>-2.65</td>
</tr>
<tr>
<td>18.</td>
<td>Databricks</td>
<td>Multi-model</td>
<td>68.47</td>
<td>+2.65</td>
</tr>
<tr>
<td>19.</td>
<td>Teradata</td>
<td>Relational, Multi-model</td>
<td>60.25</td>
<td>-2.39</td>
</tr>
<tr>
<td>20.</td>
<td>Google BigQuery</td>
<td>Relational</td>
<td>55.42</td>
<td>+0.78</td>
</tr>
<tr>
<td>21.</td>
<td>FileMaker</td>
<td>Relational</td>
<td>53.32</td>
<td>-1.06</td>
</tr>
<tr>
<td>22.</td>
<td>Neo4j</td>
<td>Graph</td>
<td>52.06</td>
<td>-0.71</td>
</tr>
</tbody>
</table>
Data model heterogeneity: can we eliminate it?

Short realist's answer:

It didn't happen so far!
Motivation

Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.
Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.

In more details:
Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.

In more details:
Existing system It is hard/expensive/cumbersome to change the back-end database on which an organization runs.
Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.

In more details:
Existing system It is hard/expensive/cumbersome to change the back-end database on which an organization runs.

Existing data Specific datasets that we need to work with come in specific formats due to industry or company standards.
Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.

In more details:
Existing system It is hard/expensive/cumbersome to change the back-end database on which an organization runs.
Existing data Specific datasets that we need to work with come in specific formats due to industry or company standards.
Data doesn’t die It is organizationally very hard to accept data deletion [16].
Motivation

Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far!  Reality has its own way of casting votes.

In more details:

Existing system  It is hard/expensive/cumbersome to change the back-end database on which an organization runs.

Existing data  Specific datasets that we need to work with come in specific formats due to industry or company standards.

Data doesn’t die  It is organizationally very hard to accept data deletion [16].

Migration is a pain  Model conversion may lose some features, needs to invent others, is error-prone, leads to spaghetti systems.
Data model heterogeneity: can we eliminate it?

Short realist’s answer:
It didn’t happen so far! Reality has its own way of casting votes.

In more details:
Existing system It is hard/expensive/cumbersome to change the back-end database on which an organization runs.

Existing data Specific datasets that we need to work with come in specific formats due to industry or company standards.

Data doesn’t die It is organizationally very hard to accept data deletion [16].

Migration is a pain Model conversion may lose some features, needs to invent others, is error-prone, leads to spaghetti systems.

Variety has merit It makes sense to experiment with new systems, especially if/when they have better performance for specific tasks. We’ll hear more about this.
Data heterogeneity: how to live with it?

**Data integration**: leverage data from several stores (sources)
- Sources may have different data models
- Sources may have different schemas (or no schema at all)
- Sources may have different query processing capabilities
- An individual source may be distributed

**Main questions to be addressed in data integration systems**
- What unified schema/interface do users get?
- How do we automatically process queries over this schema?
- How do we do so efficiently, taking best advantage of the available computation capabilities?
Part II

View-Based Data Integration
Views in Relational Databases

Relation  Named set (or bags, in SQL) of tuples

Query    First-order logic formula (or SQL query): computes some answers from the data.

View     Named query.
  - **Virtual view**: associates a name to a query. It can then be queried.
  - **Materialized view**: the query result is computed and stored on disk. It can be used like any other table.

Schema   A set of relations and/or views.
Virtual views in SQL

Base table:

<table>
<thead>
<tr>
<th>city</th>
<th>street</th>
<th>name</th>
<th>descr</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>Odéon</td>
<td>Flora</td>
<td>“Historic”</td>
<td>200</td>
</tr>
<tr>
<td>Bastia</td>
<td>Corniche</td>
<td>Hippocampe</td>
<td>“Modern”</td>
<td>175</td>
</tr>
</tbody>
</table>

View definition:

create view ParisHotels as select street, name, price from Hotel where city='Paris' and price < 210

Then:

select name from ParisHotels returns:

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flora</td>
</tr>
</tbody>
</table>
Views for data integration

Views for data integration

View allow defining another (possibly virtual) schema based on an existing schema.
Views for data integration

View allow defining another (possibly virtual) schema based on an existing schema. Very useful to integrate heterogeneous data sources.
Views for data integration

View allow defining another (possibly virtual) schema based on an existing schema. Very useful to integrate heterogeneous data sources

Data integration system

- A set of data sources, each with: data model, query language, and schema (also called source schemas).
- A mediator with its own DM, QL and global schema
- Queries are asked against the global schema
- Wrappers interface the sources to the mediator’s model
Views for data integration

*View* allow defining another (possibly virtual) schema based on an existing schema. Very useful to **integrate heterogeneous data sources**

**Data integration system**

- A set of **data sources**, each with: data model, query language, and schema (also called **source schemas**).
- A **mediator** with its own DM, QL and **global schema**
- Queries are asked against the global schema
- **Wrappers** interface the sources to the mediator’s model
Connecting the source schemas to the global schema

Sample scenario:

Mediator schema
- Hotel(city, street, name, descr, price)
- Restaurants, street, name, rating

Wrapper 1
- Source 1 schema
  - ParisHotel(street, name, roomPrice)

Wrapper 2
- Source 2 schema
  - LyonHotel(street, name, roomDesc, roomPrice)

Wrapper 3
- Source 3 schema
  - Restaurants(city, street, name, rating)
Connecting the source schemas to the global schema

Data only exists in the sources.

Applications can only query the global schema.

Mediator schema

Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Wrapper 1
Source 1 schema
ParisHotel(street, name, roomPrice)

Wrapper 2
Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)

Wrapper 3
Source 3 schema
Restaurants(city, street, name, rating)
Connecting the source schemas to the global schema

How to specify the relation between

- The global schema accessible to applications, and...
- the source schemas reflecting the data...
- so that a query over the global schema can be automatically translated into a query over the source schemas?
Connecting the source schemas to the global schema: Global-as-View (GAV)

<table>
<thead>
<tr>
<th>Source 1</th>
<th>s1:ParisHotels(street, name, roomPrice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source 2</td>
<td>s2:LyonHotel(street, name, roomDesc, roomPrice)</td>
</tr>
<tr>
<td>Source 3</td>
<td>s3:Restaurant(city, street, name, rating)</td>
</tr>
<tr>
<td>Global</td>
<td>Hotel(city, street, name, descr, price), Restaurant(city, street, name, rating)</td>
</tr>
</tbody>
</table>

Defining Hotel as a view over the source schemas:

```sql
define view Hotel as
  select 'Paris' as city, street, name, null as descr, roomPrice as price
  from s1:ParisHotels
union all
  select 'Lyon' as city, street, name, roomDesc as descr, price
  from s2:LyonHotel
```

Defining Restaurant as a view over the source schemas:

```sql
define view Restaurant as
  select * from s3:Restaurant
```
Connecting the source schemas to the global schema: Global-as-View (GAV)
Query processing in global-as-view (GAV) (1)

define view Hotel as
  select 'Paris' as city, street, name, null as descr, roomPrice as price
  from s1:ParisHotels
  union all
  select 'Lyon' as city, street, name, roomDesc as descr, price
  from s2:LyonHotel

Query:

| select * from Hotel where city='Paris' and price < 200 |
| becomes: |
| select * from (select 'Paris' as city... union... select 'Lyon' as city...) where city='Paris' and price < 200 which becomes: |
| select * from (select 'Paris' as city...) where city='Paris' and price < 200 which becomes: |
| select * from s1:ParisHotels where price < 200 Source-only! |
Global-as-View Data Integration

Query processing in global-as-view (GAV) (2)

define view **Hotel** as
select 'Paris' as city, street, name, null as descr, roomPrice as price
from s1:ParisHotels
union all
select 'Lyon' as city, street, name, roomDesc as descr, price
from s2:LyonHotel

define view **Restaurant** as
select * from s3:Restaurant

Query:

```
select h.street, r.rating from Hotel h, Restaurant r
where h.city=r.city and r.city='Lyon' and h.street=r.street and h.price<200
```

becomes:

```
select h.street, r.rating
from (select 'Paris' as city... from s1:ParisHotels union all select 'Lyon' as
city... from s2:LyonHotel) h, (select * from s3:Restaurant) r
where h.city=r.city and r.city='Lyon' and h.street=r.street and h.price<200
```

which becomes:
Query processing in global-as-view (GAV) (2)

**define view Hotel as**
```
select 'Paris' as city, street, name, null as descr, roomPrice as price
from s1:ParisHotels
union all
select 'Lyon' as city, street, name, roomDesc as descr, price
from s2:LyonHotel
```

**define view Restaurant as**
```
select * from s3:Restaurant
```

**Query:**
```
select h.street, r.rating
from (select ... from s2:LyonHotel) h, s3:Restaurant r
where r.city='Lyon' and h.street=r.street and h.price<200
```

which becomes:
```
select h.street, r.rating from s2:LyonHotel h, s3.Restaurant r
where r.city='Lyon' and h.price<200 and h.street=r.street
```
Query processing in global-as-view (GAV) (2)

Query:

\[
\begin{align*}
&\text{select } h.\text{street}, \ r.\text{rating} \text{ from } \text{Hotel } h, \ \text{Restaurant } r \\
&\text{where } h.\text{city}=r.\text{city} \text{ and } r.\text{city}=\text{’Lyon’} \text{ and } h.\text{street}=r.\text{street} \text{ and } h.\text{price}<200
\end{align*}
\]

has become:

\[
\begin{align*}
&\text{select } h.\text{street}, \ r.\text{rating} \text{ from } s2:\text{LyonHotel } h, \ s3:\text{Restaurant } r \\
&\text{where } r.\text{city}=\text{’Lyon’} \text{ and } h.\text{price}<200 \text{ and } h.\text{street}=r.\text{street}
\end{align*}
\]
Query processing in global-as-view (GAV) (2)

Query:

```
select h.street, r.rating from Hotel h, Restaurant r
where h.city=r.city and r.city='Lyon' and h.street=r.street and h.price<200
```

has become:

```
select h.street, r.rating from s2:LyonHotel h, s3.Restaurant r
where r.city='Lyon' and h.price<200 and h.street=r.street
```

How do we process this?

- s2 evaluates: select h.street from s2:LyonHotel h where h.price<200
- s3 evaluates: select r.rating, r.street from s3.Restaurant r where r.city='Lyon'
- Mediator joins the results, applies final projection
Concluding remarks on global-as-view (GAV)

Query processing = view unfolding

- Just like querying views in a database
- Heuristic: push as many operators (select, project, join; navigate...) on the sources as possible

Weakness: source changes impact the global schema

- The global schema may need to change as parts of it may become invalid (undefined)
- As a consequence, applications may need to be rewritten!
Global-as-view: adding a new source

User query

Mediator schema
Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users

Source 1 schema
ParisHotel(street, name, roomPrice)

Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)
Global-as-view: Adding a new source

Mediator schema:
- Hotel(city, street, name, descr, price)
- Restaurants, street, name, rating)

Sources:
- Source 1 schema: ParisHotel(street, name, roomPrice)
- Source 2 schema: LyonHotel(street, name, roomDesc, roomPrice)
- Source 4 schema: GrenobleHotel(street, name, rating)

User query

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users
Global-as-view: adding a new source

User query

Mediator schema

Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users

Source 1 schema
ParisHotel(street, name, roomPrice)

Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)

Source 4 schema
GrenobleHotel(street, name, rating)
Global-as-view: removing a source

- If Source3.Restaurant withdraws, the ParisPackage relation in the global schema becomes empty.
- Applications cannot even access Source1.ParisHotels, even though it is still available.
Connecting the source schemas to the global schema: Local-as-view (LAV)

User query

Mediator schema
Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users

Source 1 schema
ParisHotel(street, name, roomPrice)

Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)

Source 3 schema
Restaurants(city, street, name, rating)
Connecting the source schemas to the global schema: Local-as-view (LAV)

s1: ParisHotel(street, name, roomPrice)
s2: LyonHotel(street, name, roomDesc, roomPrice)
s3: Restaurant(city, street, name, rating)
Global: Hotel(city, street, name, descr, price), Restaurant(city, street, name, rating)

Defining the sources as views over the global schema:

- define view s1: ParisHotels as select street, name, price as roomPrice from Hotel where city='Paris'
- define view s2: LyonHotel as select street, name, descr as roomDesc, price as roomPrice from Hotel where city='Lyon'
- define view s3: Restaurant as select * from Restaurant
GAV and LAV have different expressive power

Some GAV scenarios cannot be expressed in LAV

create view ParisPackage as
select ph.name as hotelName, ph.street as hotelAddress,
    r.name as restaurantName, r.rating as restaurantRating
from s1:ParisHotel ph, s3:Restaurants r
where r.city='Paris' and r.street=ph.street

- Only (hotel, restaurant) pairs that are on the same street in Paris
- Not expressible in LAV which describes each source individually w.r.t. the global schema, not in correlation with data from other sources
GAV and LAV have different expressive power

Some LAV scenarios cannot be expressed in GAV

- $s_3: \text{MHotel}(\text{city}, \text{street}, \text{name}, \text{price})$ only has data about Marseille hotels
- $s_4: \text{WHotel}(\text{city}, \text{street}, \text{name}, \text{price})$ has only data about Wien hotels
- \text{Hotel} is defined as:
  \[
  \text{select * from MHotel union all select * from WHotel}
  \]
- A query about hotels in Rome will also be sent to $s_3$ and $s_4$, although it will bring no results
define view s1:ParisHotels as
    select street, name, price as roomPrice
from Hotel where city='Paris'

define view s2:LyonHotel as
    select street, name, descr as roomDesc, price as roomPrice
from Hotel where city='Lyon'

define view s3:Restaurant as select * from Restaurant

Query:

select h.street, h.price, r.rating
from Hotel h, Restaurant r
where r.city=h.city and h.street=r.street
Query processing in Local-as-View (LAV)

**Rewriting** of the query using the views:

```sql
select h1.street, h1.price, r3.rating
from s1:ParisHotels h1, s3:Restaurant r3
where h1.city = h3.city and h1.street = r3.street
union all
select h2.street, h2.price, r3.rating
from s2:LyonHotels h2, s3:Restaurant r3
where h2.city = h3.city and h2.street = r3.street
```
**Query processing in Local-as-View (LAV)**

Define views:

- `s1:ParisHotels` as... from `Hotel` where `city='Paris'`
- `s2:LyonHotel` as... from `Hotel` where `city='Lyon'`
- `s3:Restaurant` as select * from `Restaurant`

**Rewriting** of the query using the views:

```
select h1.street, h1.price, r3.rating
from s1:ParisHotels h1, s3:Restaurant r3
where h1.city=h3.city and h1.street=r3.street
union all
select h2.street, h2.price, r3.rating
from s1:ParisHotels h2, s3:Restaurant r3
where h2.city=h3.city and h2.street=r3.street
```

How do we find this? It’s more complicated NP-hard problem [14] in the size of views + query.
Local-as-View: adding a new source

User query

Mediator schema
Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users

Source 1 schema
ParisHotels(street, name, roomPrice)

Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)

Source 4 schema
GrenobleHotel(street, name, rating)

We add a new mapping relating the new source to the global schema. Invisible to the applications!
Local-as-View: removing a source

User query

Mediator schema
Hotel(city, street, name, descr, price)
Restaurants, street, name, rating)

Source 1 schema
ParisHotels(street, name, roomPrice)

Source 2 schema
LyonHotel(street, name, roomDesc, roomPrice)

Query results

Mediator schema available to applications/users
Sources hidden to the applications/users

We remove the mapping of the removed source.
ParisHotels still available. Invisible to the applications!
Concluding remarks on Local-as-View

Query processing: finding the query rewriting is significantly more complex
- NP-hard in the size of views + query, but...
- These are many orders of magnitude smaller than the data
- Practical algorithms have been proposed, in particular using chase and back-chase [15], improved since [2, 4], within Estocada (see next)

The connection between sources and global schema is much more robust
- Sources can be added and/or removed with no impact on global schema
- Thus, no impact on applications.
Views capture **logical relationship** between stored data collections and/or global schema (if any)

- Horizontal & vertical partitioning of a relation are just particular cases
  - Store FR sale in Paris, RO sale data in Cluj
  - Store current orders in Berlin, order history in Dublin
- Key-based distribution is just a particular case
  - Store on node $i$ the records such that $h(k) = i$
Integrity Constraints: express human knowledge about a domain.

- **IC**: “Every restaurant has a sanitary license”.

<table>
<thead>
<tr>
<th>View</th>
<th>Query</th>
<th>Rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All restaurants”</td>
<td>“All restaurants with a license”</td>
<td>?</td>
</tr>
</tbody>
</table>
**Integrity Constraints**: express human knowledge about a domain.

- *IC*: “Every restaurant has a sanitary license”.

<table>
<thead>
<tr>
<th>View</th>
<th>Query</th>
<th>Rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All restaurants”</td>
<td>“All restaurants with a license”</td>
<td>?</td>
</tr>
</tbody>
</table>

- Without *IC*, the view may return wrong answers (restaurants without a license)!
Rewriting with Integrity Constraints

**Integrity Constraints**: express human knowledge about a domain.

- **IC**: “Every restaurant has a sanitary license”.

<table>
<thead>
<tr>
<th>View</th>
<th>Query</th>
<th>Rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All restaurants”</td>
<td>“All restaurants with a license”</td>
<td>?</td>
</tr>
</tbody>
</table>

- Without **IC**, the view may return wrong answers (restaurants without a license)!
- Knowing **IC**, the view is a perfect rewriting.
Rewriting with Integrity Constraints

**Integrity Constraints**: express human knowledge about a domain.

- **IC**: “Every restaurant has a sanitary license”.

<table>
<thead>
<tr>
<th>View</th>
<th>Query</th>
<th>Rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All restaurants”</td>
<td>“All restaurants with a license”</td>
<td>?</td>
</tr>
</tbody>
</table>

- Without **IC**, the view may return wrong answers (restaurants without a license)!
- Knowing **IC**, the view is a perfect rewriting.
- How do we find this? Provenance-Aware Chase & Backchase [15], LAV style
Rewriting with integrity constraints using an algorithm such as PACB [15]
Part III

ESTOCADA: View-Based Rewriting for Polystores

Joint work with: Rana Al-Otaibi (UCSD, now Microsoft Research), Alin Deutsch (UCSD), Bogdan Cautis (U. Paris-Saclay), D. Bursztyn, S. Zampetakis (Inria)
Polystores: modern heterogeneous Big Data management systems

- A set of data stores, each supporting one or several data models
  - PostgreSQL supports relational, XML, JSON
- Each store can evaluate queries in some language / dialect
- Each store may have better performance on some operations
  - SOLR extremely good at full-text search; no $\sigma, \pi, \bowtie$
  - PostgreSQL good at $\sigma, \pi, \bowtie$, grouping; some JSON, XML querying
  - Redis good at key-value look-up; no $\bowtie$
Polystores: modern heterogeneous Big Data management systems

- A set of data stores, each supporting one or several data models
  - PostgreSQL supports relational, XML, JSON
- Each store can evaluate queries in some language / dialect
- Each store may have better performance on some operations
  - SOLR extremely good at full-text search; no \( \sigma \), \( \pi \), \( \bot \), grouping
  - PostgreSQL good at \( \sigma \), \( \pi \), \( \bot \), grouping; some JSON, XML querying
  - Redis good at key-value look-up; no \( \bot \)

Estocada [2, 3, 4]: How to leverage the best performance available in the system?
Connecting the source schemas to the global schema: Estocada

User query \rightarrow Mediator « schema »: Union of all source schemas (each in original data model) \rightarrow Query results

Source 1
- JSON
- AsterixDB system

Source 2
- Relational, JSON
- PostgreSQL system

Source 3
- Text, JSON
- SOLR system

Mediator schema = Source schemas
Polystores: modern heterogeneous Big Data management systems

- A set of data stores, each supporting one or several data models
  - PostgreSQL supports relational, XML, JSON
- Each store can evaluate queries in some language / dialect

**Estocada** [2, 3]: How to leverage the best performance across all stores?

- **Move (migrate) data from one store to another**, since they all cooperate
  - Materialize, in one store, view(s) over data from other store(s)
Connecting the source schemas to the global schema: **Estocada**

- **User query**
- **Query results**

**Mediator « schema »:**
Union of all source schemas (each in original data model)

**Mediator schema =**
Source schemas

**Source 1:**
- **JSON**
- AsterixDB system

**Source 2:**
- Relational, JSON
- PostgreSQL system

**Source 3:**
- Text, JSON
- SOLR system
Connecting the source schemas to the global schema: **ESTOCADA (LAV)**

**Mediator « schema »:**
Union of all source schemas (each in original data model)

**Source 1:** JSON, AsterixDB system

**Source 2:** Relational, JSON, PostgreSQL system

**Source 3:** Text, JSON, SOLR system
Polystores: modern heterogeneous Big Data management systems

- A set of **data stores**, each supporting one or several **data models**
  - PostgreSQL supports relational, XML, JSON
- Each store can evaluate queries in some language(s)/dialects

**Estocada** [2, 3]: How to **leverage the best performance across all stores?**

1. **Move (migrate) data from one store to another**, since they all cooperate
   - Materialize, in one store, view(s) over data from other store(s)
How to write views and queries across data models?

**Query Block Trees (QBT)** cross-model query language: select from some data model(s), return in another data model

View $V_1$:
```
FOR AJ:{
SELE
M.patientID AS patientID,
A.admissionID AS admissionID,
A.report AS report
FROM MIMIC M, M.admissions A
RETURN SJ:{
"patientID":patientID,
"admissionID":admissionID,
"report":report
}
```

View $V_2$:
```
FOR AJ:{
SELE
M.patientID AS patientID,
A.admissionID AS admissionID,
A.admissionLoc AS admissionLoc,
A.admissionTime AS admissionTime
FROM MIMIC M, M.admissions A
RETURN PR:{patientID, admissionID,
admissionLoc, admissionTime}
```

- AJ: AsterixDB’s JSON query language
- SJ: SOLR’s JSON query language
- PR: PostgreSQL SQL
- … add your own!
- QBT resembles CloudMDSQL [17] + nesting
How to write views and queries across data models?

Query Block Trees (QBT) cross-model query language: select from some data model(s), return in another data model

View $V_1$:
FOR AJ: {
SELECT M.patientID AS patientID,
A.admissionID AS admissionID,
A.report AS report
FROM MIMIC M, M.admissions A
RETURN SJ: {
"patientID": patientID,
"admissionID": admissionID,
"report": report
}

View $V_2$:
FOR AJ: {
SELECT M.patientID AS patientID,
A.admissionID AS admissionID,
A.admissionLoc AS admissionLoc
A.admissionTime AS admissionTime
FROM MIMIC M, M.admissions A
RETURN PR: {
patientID, admissionID,
admissionLoc, admissionTime
}

One QBT block: FOR ...
RETURN ...

- One QBT block: FOR ... RETURN ...
- QBTs can be arbitrarily nested
Sample query against JSON data originally stored in AsterixDB

How to rewrite it using PostgreSQL and SOLR views?

FOR AJ: {SELECT M.patientID AS patientID,
            A.admissionLoc AS admissionLoc,
            A.admissionTime AS admissionTime,
            P.drug AS drug
        FROM LABITEM L, MIMIC M, M.admissions A,
            A.labevents LE, A.prescriptions P
        WHERE L.itemID=LE.labItemID AND
            L.category='blood' AND L.flag='abnormal' AND
            P.drugtype='additive' AND
            contains(report,'coronary artery')}

RETURN AJ: {"patientID":patientID, "drug":drug,
             "admissionLoc":admissionLoc,
             "admissionTime":admissionTime}
Estocada view-based query rewriting problem

- QBT View definitions $\mathcal{V}$
- QBT (cross-model) view-based query rewriting
- QBT Rewritings $\mathcal{R}$
  - view data
  - $Q$ results

QBT Query $Q$
Estocada query rewriting

**Estocada view-based rewriting approach**

QBT View definitions $\mathcal{V}$

$\downarrow$

Relationally encoded views $\mathcal{V}^e$

$\downarrow$

Relational rewritings $\mathcal{R}^e$

$\downarrow$

QBT Decoded rewritings $\mathcal{R}$

QBT Query $Q$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ Rel. rew. $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ Model encoding $\quad$ virtual relations + constraints $\mathcal{C}$

$\downarrow$

Relationally encoded query $Q^e$

$\downarrow$

JSON model $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ XML model $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ PG model

Ioana Manolescu, Inria, France

Leveraging Heterogeneous Data

July 2023 67 / 97
Estocada query rewriting

Estocada view-based rewriting approach

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relationally encoded query $Q^e$

Rel. rew.

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

Relational rewritings $\mathcal{R}^e$
Our target: QBT decoded rewriting $\mathcal{R}$

- Each leaf is a source query
- Top $\bowtie$, $\Pi$ evaluated by a mediator [10]

```
π₁,₂,₃,₄,₇

π₁,₂,₅,₆

q=report:'coronary artery' &fl=patientID,admissionID

SELECT v₃->'patientID' AS PID₂,
A->'admissionID' AS AID₂, D->'drug' AS drug
FROM V₃ v₃,
jsonb_array_elements(v₃->'admission') A,
jsonb_array_elements(A->'drugs') D
WHERE v₃->'patientID'=? AND
A->'admissionID'=? AND
D->'drugtype'='additive'

SELECT patientID AS PID₁,
admissionID AS AID₁, admissionLoc,
admissionTime
FROM V₂
WHERE patientID=? AND admissionID=?
```
Estocada view-based rewriting approach

QBT View definitions $\mathcal{V}$

$\downarrow$

Relationally encoded views $\mathcal{V}^e$

$\downarrow$

Relationally encoded query $Q^e$

$\rightarrow$

Rel. rew.

$\rightarrow$

Relational rewritings $\mathcal{R}^e$

$\rightarrow$

QBT Decoded rewritings $\mathcal{R}$

Model encoding

virtual relations + constraints $\mathcal{C}$

$\uparrow$

JSON model

$\rightarrow$

XML model

$\rightarrow$

PG model

Ioana Manolescu, Inria, France

Leveraging Heterogeneous Data

July 2023
Encoding the JSON data model in \textsc{Estocada}

Virtual Relational Encoding of JSON (VREJ) schema and integrity constraints

Constraints:

Describing the data:

<table>
<thead>
<tr>
<th>Collection(_J)(name, id)</th>
<th>Collection(_J)(n, x) &amp; Collection(_J)(n, y) \rightarrow x = y \quad (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child(_J)(pld, cld, key, type)</td>
<td>Child(_J)(p, c_1, k, t) &amp; Child(_J)(p, c_2, k, t) \rightarrow c_1 = c_2 \quad (2)</td>
</tr>
</tbody>
</table>

Needed for modelling:

<table>
<thead>
<tr>
<th>Eq(_J)(x, y)</th>
<th>Eq(_J)(x, y) \rightarrow Eq(_J)(y, x) \quad (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eq(_J)(x, y) &amp; Eq(_J)(y, z)</td>
<td>Eq(_J)(x, z) \quad (4)</td>
</tr>
<tr>
<td>Eq(_J)(p, p') &amp; Child(_J)(p, c, k, t)</td>
<td>\rightarrow</td>
</tr>
<tr>
<td>\exists c' Eq(_J)(c, c') &amp; Child(_J)(p', c', k, t)</td>
<td>\quad (5)</td>
</tr>
<tr>
<td>Value(_J)(i, v_1) &amp; Value(_J)(i, v_2)</td>
<td>v_1 = v_2 \quad (6)</td>
</tr>
</tbody>
</table>
Estocada view-based rewriting approach

QBT View definitions $\mathcal{V}$

Relationally encoded views $\mathcal{V}^e$

Relational rewritings $\mathcal{R}^e$

QBT Decoded rewritings $\mathcal{R}$

QBT Query $Q$$\rightarrow$ Relationally encoded query $Q^e$$\rightarrow$ Rel. rew.$\rightarrow$ Model encoding virtual relations + constraints $\mathcal{C}$$\rightarrow$ JSON model XML model PG model
Relational encoding of cross-model view $V_2$

$$\text{View } V_2:$$
$$\text{FOR } \text{AJ:} \{ \text{SELECT}$$
$$\quad M.\text{patientID AS patientID,}$$
$$\quad A.\text{admissionID AS admissionID,}$$
$$\quad A.\text{admissionLoc AS admissionLoc}$$
$$\quad \text{A.admissionTime AS admissionTime}$$
$$\text{FROM MIMIC M, M.admissions A}$$
$$\text{RETURN PR:}\{\text{patientID, admissionID,}$$
$$\quad \text{admissionLoc, admissionTime}\}$$

$\text{MIMIC(M),}$
$\text{Child}_J(M, \text{patientID,"patientID","o"},)$
$\text{Child}_J(M, A,"\text{admissions","o"},)$
$\text{Child}_J(A, \text{admissionID,"admissionID","o"},)$
$\text{Child}_J(A, \text{admissionLoc,"admissionLoc","o"},)$
$\text{Child}_J(A, \text{admissionTime,"admissionTime","o"},)$
$\text{Child}_J(A, \text{report,"reports","o"}) \rightarrow$
$V_2(\text{patientID, admissionID, admissionLoc, admissionTime})$

“o”: object (or map) type
Sample experimental benefits [2]

**Dataset:** Medical Information Mart for Intensive Care (MIMIC-III), real-life, JSON (46 GB)

**Query:** for the patients with “coronary artery” issues, and abnormal blood test results, find the date/time of admission, their previous location (e.g., emergency room, clinic referral), and the drugs of type “additive” prescribed

**Single-system (JSON) evaluation**

<table>
<thead>
<tr>
<th></th>
<th>AsterixDB</th>
<th>SparkSQL</th>
<th>MongoDB</th>
<th>PostgreSQL</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>25 min</td>
<td>&gt; 60 min</td>
<td>16 min</td>
<td>12.6 min</td>
<td></td>
</tr>
</tbody>
</table>

**Materialized views:**

- $V_1$ (SOLR): IDs of patients and their hospital admission data, caregivers’ reports, notes on the patients’ stay
- $V_2$ (Postgres): patients metadata, hospital admission information (admission time and patients’ location)
- $V_3$ (Postgres JSON): drugs prescribed for each patient with ”abnormal blood” test results

**Execution using mat. views** $\text{filter}(V_1) \bowtie V_3 \bowtie V_2$: 5.7 min
**Estocada: Relational and JSON views in Postgres, JSON views in SOLR**

Query speed-up: comparison with single-store [2]
Query speed-up: comparison with BigDawg [12] polystore
**Estocada wrap-up**

- In any setting, **materialized views** have the biggest potential for performance improvement.
- In a polystore setting, **unequal source capabilities and performance** further motivate moving (or copying) parts of the data across sources.
- **Rewrite** a query into a QBT over one or more sources and/or view, distribute computation across sources.
- Efficient rewriting w/formal guarantees distinguishing from comparable systems, e.g., [1].
- The same benefits extend to mixed RA (\(\sigma, \text{join}, \pi\)) - LA (\(+, \times, T, \ldots\)) workloads DB/ML [3, 4].
- Demo in Docker instance at: https://gitlab.inria.fr/cedar/estocada-public/-/tree/master/estocada-MDD-tutorial
- Automatic recommendation of materialized views ongoing.
Part IV

**Abstract**: Abstracting Data of Any Model

Joint work with Nelly Barret (Inria), Prajna Upadhyay (Inria, now IIT Khanpur)
I was born in communist Romania, then a dictatorship. Estimated victims 1945-1989: 500,000 to 2M (https://ro.wikipedia.org/wiki/Rom%C3%A2nia_comunist%C4%83)
Data management for investigative journalists and fact-checking

I was born in communist Romania, then a dictatorship. Estimated victims 1945-1989: 500,000 to 2M (https://ro.wikipedia.org/wiki/Rom%C3%A2nia_comunist%C4%83) Fall of communism in Romania: 1142 dead, 3138 wounded. No one has been tried / convicted for that.
Data management for investigative journalists and fact-checking

I was born in communist Romania, then a dictatorship. Estimated victims 1945-1989: 500,000 to 2M (https://ro.wikipedia.org/wiki/Rom%C3%A2nia_comunist%C4%83)
Fall of communism in Romania: 1142 dead, 3138 wounded.
No one has been tried / convicted for that.

**Functional free press** is an important ingredient for democracy
- To debate and express dissent
- To analyze and expose society’s functioning
Data management for investigative journalists and fact-checking

I was born in communist Romania, then a dictatorship. Estimated victims 1945-1989: 500,000 to 2M (https://ro.wikipedia.org/wiki/Rom%C3%A2nia_comunist%C4%83)

Fall of communism in Romania: 1142 dead, 3138 wounded. No one has been tried / convicted for that.

**Functional free press** is an important ingredient for **democracy**
- To debate and express dissent
- To analyze and expose society’s functioning

**Overall, low IT use/literacy in newsrooms**
- Writers, not geeks
- Yet data skills needed more than ever

Research projects with **Le Monde, radiofrance** since 2015
Data analysis needs for journalism

- The application domain **follows the topic of interest**: news cycle, or investigation topic.
- Journalists use whatever data they can get their hands on. Different data models! (All that we discussed + text, Office, etc.)

- Journalists are familiar with text and documents > spreadsheet ≫ anything else
- Data producers often **uncollaborative** ⇒ documentation, schema missing
The application domain follows the topic of interest: news cycle, or investigation topic.

Journalists use whatever data they can get their hands on. Different data models! (All that we discussed + text, Office, etc.)

Journalists are familiar with text and documents > spreadsheet ≥ anything else

Data producers often uncollaborative ⇒ documentation, schema missing

Data understanding conditions even the earliest stages of journalistic work!
How to help journalists understand heterogeneous data

Focus on (semi)structured data formats: RDBs, CSV, JSON, XML, RDF, PGs, ...

Core principles

1. Do not expose data model details.
How to help journalists understand heterogeneous data

Focus on (semi)structured data formats: RDBs, CSV, JSON, XML, RDF, PGs, …

Core principles

1. Do not expose data model details.
2. Any application dataset (tabular/RDB, tree- or graph-oriented) contains some entities connected by some relationships. Find and (graphically) show these!
How to help journalists understand heterogeneous data

Focus on (semi)structured data formats: RDBs, CSV, JSON, XML, RDF, PGs, …

Core principles

1. Do not expose data model details.
2. Any application dataset (tabular/RDB, tree- or graph-oriented) contains some entities connected by some relationships. Find and (graphically) show these!
   - Entities may have nested structure
How to help journalists understand heterogeneous data

Focus on (semi)structured data formats: RDBs, CSV, JSON, XML, RDF, PGs, …

Core principles

1. Do not expose data model details.
2. Any application dataset (tabular/RDB, tree- or graph-oriented) contains some entities connected by some relationships. Find and (graphically) show these!
   - Entities may have nested structure
3. Let users control on how much information they want.

ABSTRA project [7, 8, 9]. Code and (many) examples:

https://team.inria.fr/cedar/projects/abstra/
Sample abstraction of an XMark XML document

Application benchmark: online auctions site

3M nodes, 80 different labels, on 124 labeled paths
Dataset abstraction stages

1. Turn any dataset into a graph
   - Based on ConnectionLens [5, 6]
   - Addition of extracted entity nodes

2. Summarize the graph: equivalent nodes grouped in collections

3. Identify (nested) entities:
   - Select some collections as entity roots
   - Determine each entity's boundaries

4. Classify entities: assign a human-understandable name

5. Identify relationships between entities
**Relational model (also CSV): tables**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>company</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>IBM</td>
<td>Paris</td>
</tr>
</tbody>
</table>

**XML, JSON, RDF:** direct translation; **PGs:** similar to relational + nodes for relationships

**Normalized graph:** make all edges unlabeled

---

**Employees**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>company</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

**Companies**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>IBM</td>
<td>Paris</td>
</tr>
</tbody>
</table>
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

- **Relations** all nodes representing tuples of the same table
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

- **Relations** all nodes representing tuples of the same table
- **CSV** all nodes representing rows
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

- **Relations** all nodes representing tuples of the same table
  - CSV all nodes representing rows
  - XML all XML elements w/ same name

...
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

**Relations** all nodes representing tuples of the same table
- **CSV** all nodes representing rows
- **XML** all XML elements w/ same name
- **JSON** all nodes on the same path from the root
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

**Relations**  all nodes representing tuples of the same table
- **CSV**  all nodes representing rows
- **XML**  all XML elements w/ same name
- **JSON**  all nodes on the same path from the root

**RDF**  
(i) typed nodes are equivalent if they have the same set of types; 
(ii) untyped nodes are grouped according to a flexible notion of property **cliques** [13]
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

- **Relations** all nodes representing tuples of the same table
  - CSV all nodes representing rows
  - XML all XML elements w/ same name
  - JSON all nodes on the same path from the root

- **RDF** (i) typed nodes are equivalent if they have the same set of types; (ii) untyped nodes are grouped according to a flexible notion of property cliques [13]

- **PGs** (i) nodes with the same label set are equivalent; (ii) relationship nodes corresponding to relationships of the same type are equivalent
Data graph summarization: which groups of nodes are equivalent?

Reverse each data model’s guidelines for representing “similar things”:

**Relations** all nodes representing tuples of the same table
- **CSV** all nodes representing rows
- **XML** all XML elements w/ same name
- **JSON** all nodes on the same path from the root

**RDF** (i) typed nodes are equivalent if they have the same set of types; (ii) untyped nodes are grouped according to a flexible notion of property cliques [13]

**PGs** (i) nodes with the same label set are equivalent; (ii) relationship nodes corresponding to relationships of the same type are equivalent

**And**: for all $x_1 \equiv x_2$ and label $a$, if $x_1 \xrightarrow{a} y_1$, $x_2 \xrightarrow{a} y_2$ and $y_1, y_2$ are leaves, then $y_1 \equiv y_2$. 

Ioana Manolescu, Inria, France 
Leveraging Heterogeneous Data 
July 2023
Graph summarization result: collection graph

Each set of equivalent nodes is a collection; collections make up the collections graph

Red edges belong to cycles
year# is the collection of values of the nodes in the year collection
Selecting entities and their boundaries

We need to:

- Select some collections as entity roots;
- For each selected entity root, determine which other collections are in its boundary;
- Until we identify $k$ entities and/or $f\%$ of the data is reflected in the returned entities.
Which collections make good entity roots?

- Must contain more than one node.
- The more attributes, the better.
- The more descendant attributes, the better.

⇒ backward data weight propagation

More general: compute PageRank on the inverse collection graph: nodes with many descendant attributes are favored.

Pick the highest-rank collection as the root of the next selected entity.
Which collections make good entity roots?

- Must contain more than one node
Which collections make good entity roots?

- **Must** contain more than one node
- The more attributes, the better
Which collections make good entity roots?

- **Must** contain more than one node
- The more attributes, the better
- The more descendant attributes, the better
  \[\Rightarrow\] backward data weight propagation
Which collections make good entity roots?

- **Must** contain more than one node
- The more attributes, the better
- The more descendant attributes, the better ⇒ **backward data weight propagation**
- More general: compute **PageRank** on the inverse collection graph: nodes with many descendant attributes are favored
Which collections make good entity roots?

- **Must** contain more than one node
- The more attributes, the better
- The more descendant attributes, the better $\Rightarrow$ backward data weight propagation
- More general: compute **PageRank** on the inverse collection graph: nodes with many descendant attributes are favored
- Pick the highest-rank collection as the root of the next selected entity
Which collections make good entity roots?

- **Must** contain more than one node
- The more attributes, the better
- The more descendant attributes, the better
  \[ \Rightarrow \text{backward data weight propagation} \]
- More general: compute **PageRank** on the inverse collection graph: nodes with many descendant attributes are favored
- Pick the highest-rank collection as the root of the next selected entity

Assume **paper** is selected.
What to include in an entity boundary?

In the boundary of the entity rooted in $c$, we include any other collection $c'$.
Understanding heterogeneous datasets

What to include in an entity boundary?

In the boundary of the entity rooted in $c$, we include any other collection $c'$

- Such that there is a path $c \sim c'$ with no in-cycle edges
What to include in an entity boundary?

In the boundary of the entity rooted in $c$, we include any other collection $c'$

- Such that there is a path $c \leadsto c'$ with no in-cycle edges
- And along this path, each edge $c_i \rightarrow c_j$ satisfies:
  - Each node in $c_i$ as at most one child in $c_j$; and/or
  - At least $x\%$ of the nodes in $c_j$ have a parent in $c_i$. 
What to include in an entity boundary?

In the boundary of the entity rooted in $c$, we include any other collection $c'$

- Such that there is a path $c \sim c'$ with no in-cycle edges
- And along this path, each edge $c_i \rightarrow c_j$ satisfies:
  - Each node in $c_i$ as at most one child in $c_j$; and/or
  - At least $x\%$ of the nodes in $c_j$ have a parent in $c_i$.

The **paper** entity includes: year, abstract, title (allowing in-cycle edges: all the collections)
Understanding heterogeneous datasets

Selecting the entities

repeat

- Compute PageRank on collection graph
- Highest-rank node $\rightarrow$ root of next entity $e$
- Assign other collections in $e$’s boundary
- Update the collection graph:
  - remove $e$’s root + outgoing edges
  - remove every collection in $e$’s boundary that contributed only to it
  - reduce the weights of the other collections in $e$’s boundary

until $k$ entities selected and/or $f\%$ of the data is reflected
Finding the relationships

- For each \( c_1 \), root of entity \( e_1 \)
  - For each path \( c_1 \sim c_2 \) where \( c_2 \) is the root of \( e_2 \)
  - Create a relationship \( e_1 \rightarrow e_2 \) with the labels on the path
Abstraction of NASA RDF graph

Abstraction of file: /Users/ih/in/Kent/ontology/-spacecraft.rdf

- 15665 normalized nodes, 61 collections, 5 main collections, data coverage is 89%
- with parameters W, PR, FLAC, UPDATE_EXACT, ENABLE_SCORE

- primaryTopic (Mission.isPrimaryTopicOf)
- topic (Mission.missionRole)
- name (http://xmlns.com/foaf/0.1/person)
- performed (actor)
- missionRole (role)
- spacecraft
- missionProfile
- homepage
- discipline
- internationalDesignator
- description
- agency
- alternateName
- nominalPower

Ioana Manolescu, Inria, France
Abstraction of Foodista RDF graph
Data heterogeneity is here to stay.
- Each model is particularly good for some things... and many applications need several of these
- Relational leads to most efficient evaluation but is rigid.
- Nested formats, e.g., JSON, much more flexible.

Many systems support several models, but not with the same performance
- Your full-text search predicate could kill you
- JSON index available at 1st but not at 2nd nesting level...
- Poor RDF reasoning support in IBM DB2
- “Export” Spark computations in Java streams, etc.

ESTOCADA: View-based rewriting allows to deploy data fragments where they are most efficiently processed

ABSTRA: Beyond the heterogeneity, find entities and relationships in application datasets.
References

RHEEM: enabling cross-platform data processing - may the big data be with you!
PVLDB, 2018.

Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue.
In SIGMOD, June 2019.

ESTOCADA: towards scalable polystore systems.

HADAD: A lightweight approach for optimizing hybrid complex analytics queries.

Empowering investigative journalism with graph-based heterogeneous data management.

Graph integration of structured, semistructured and unstructured data for data journalism.
Pathways: entity-focused exploration of heterogeneous data graphs (demonstration). 
In _ESWC_, 2023.

Abstra: toward generic abstractions for data of any model (demonstration). 
In _CIKM_, 2022.

Computing generic abstractions from application datasets. 
In _EDBT_, 2024.

Mixed-instance querying: a lightweight integration architecture for data journalism. 
In _VLDB_, September 2016.

Graph pattern matching in GQL and SQL/PGQ. 
In _SIGMOD_, 2022.

The BigDAWG polystore system. 
(SIGMOD, 2015.
References III


