



**HAL**  
open science

# 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-04190926v1>**

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.

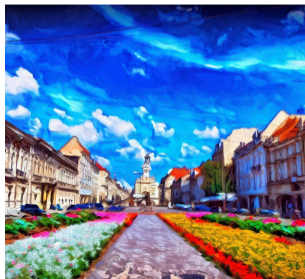
L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



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



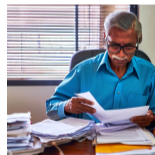
- 1 Motivation: Data Heterogeneity
- 2 ESTOCADA: Heterogeneous Polystore Performance Through Views
  - How **database geeks** conquer heterogeneity
- 3 ABSTRA: Understanding Data of Any Data Model
  - How **absolute newbies** conquer heterogeneity

- 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



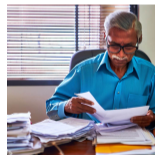
# Plan

- 1 Motivation: Data Heterogeneity
- 2 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



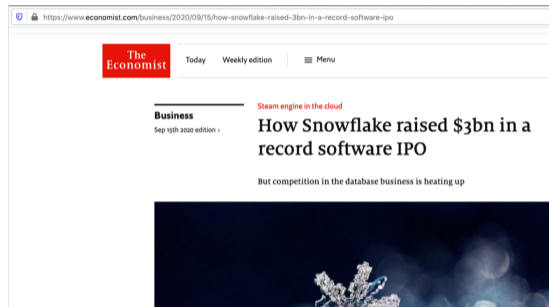
# Part I

## Motivation: Heterogeneous Data

# Motivation: large-scale data management

## Data management: a billion \$ market

- Industry, trade, telecom, health, government, social, ...
- Data publishing, sharing, reusing

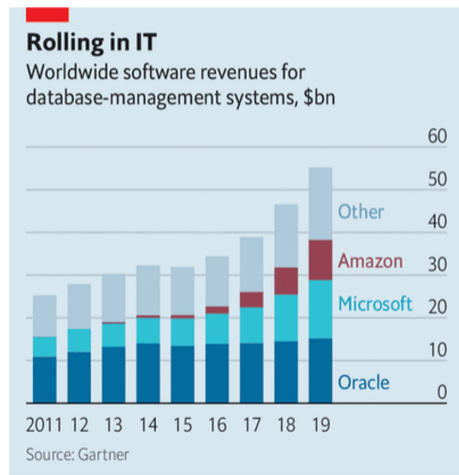




# Motivation: large-scale data management

## Data management: a billion \$ market

- Industry, trade, telecom, health, government, social, ...
- Data publishing, sharing, reusing



The Economist

# 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

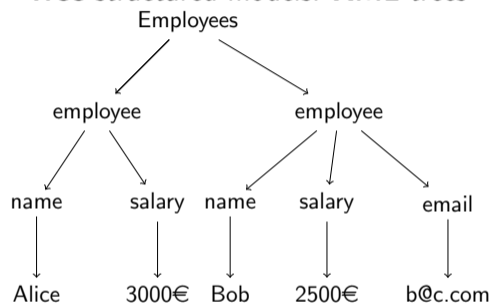
**Relational** model (also **CSV**): tables

Employees		
ID	name	salary
1	Alice	3000
2	Bob	2500

Query language: SQL (ISO Standard)

Most of the world's DB \$

**Tree-structured models: XML trees**



Query languages: XPath, XQuery, XSLT

Great for: Web (XHTML), structured documents:  
finance, chemistry... medieval (TEI)



# Data model heterogeneity: relational, XML

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

ID	name	salary
1	Alice	3000
2	Bob	2500

Query language: SQL (ISO Standard)

Most of the world's DB \$

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

```
<Employees>
  <employee> <name>Alice</name>
    <salary>3000€</salary>
  </employee>
  <employee> <name>Bob</name>
    <salary>2500€</salary>
    <email>b@c.com</email>
  </employee>
</Employees>
```

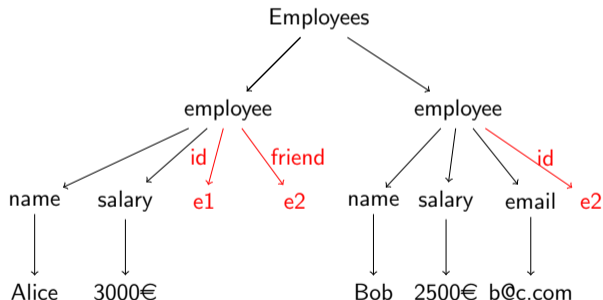
# XML documents: not always trees

## 1. XML elements may have **attributes**

```

<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>

```



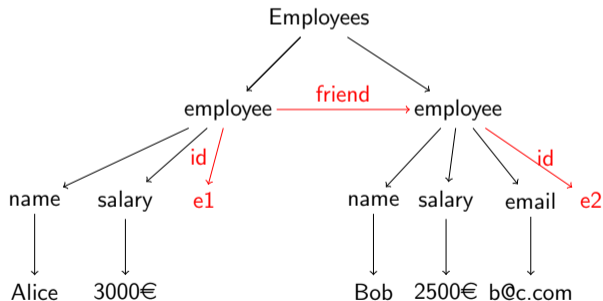
## XML documents: not always trees

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

```

<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>

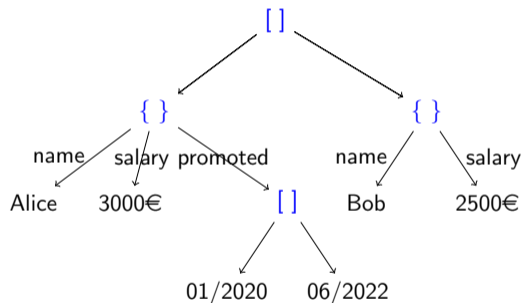
```



# Data model heterogeneity: JSON

## Tree-structured model

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



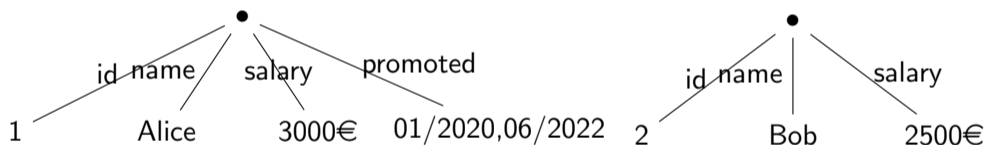
## JavaScript Object Notation

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.

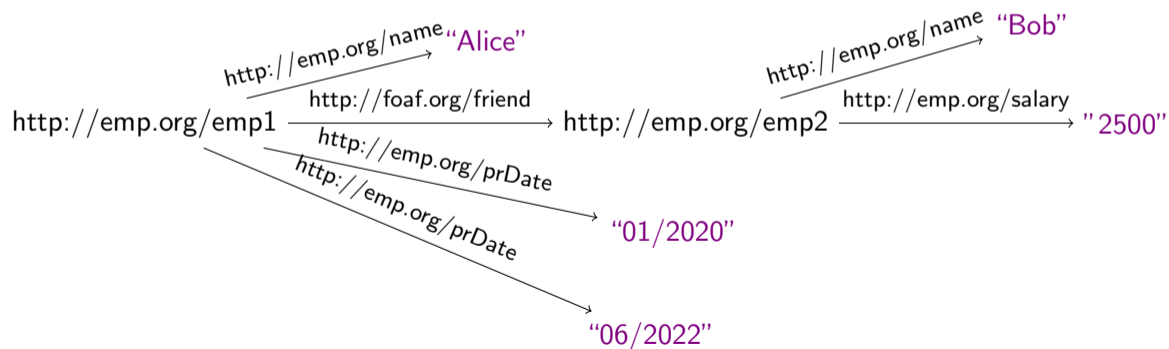


## Data model heterogeneity: RDF graphs

**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) w/ atomic values. A value can be an IRI or a **literal**.



# RDF graphs

**RDF**: **R**esource **D**escription **F**ramework, 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

### Legend

Cross Domain
Geography
Government
Life Sciences
Linguistics
Media
Publications
Social Networking
User Generated



# 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>

Item [Discussion](#)

## Cluj-Napoca (Q100188)

city and seat of Cluj County in northwestern Romania  
 Kolozsvár | Klausenburg | Cluj | Claudiopolis | Kolozsvar | Kluyzenburg | Napoca | Kluž | Kluzh | Kolozhvar | Kolozvar

[v](#) [In more languages](#)  
[Configure](#)

Language	Label	Description	Also known as
English	Cluj-Napoca	city and seat of Cluj County in northwestern Romania	Kolozsvár Klausenburg Cluj Claudiopolis Kolozsvar Kluyzenburg Napoca Kluž Kluzh Kolozhvar Kolozvar
French	Cluj-Napoca	commune roumaine	Kolozsvar Kolosvar Cluj Cluj Napoca Klausenburg Kolozsvár Klausenburg

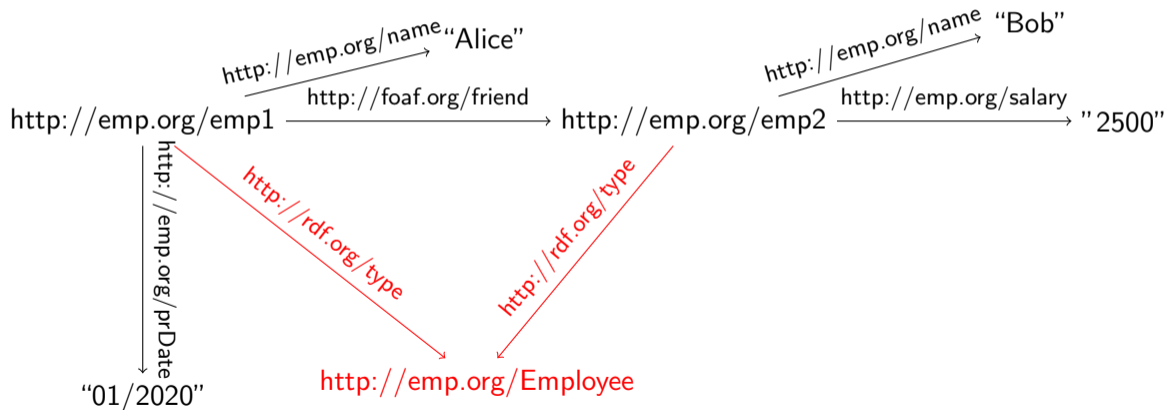
**Wikipedia** (100 entries)

- ace Cluj-Napoca
- ang Clüsenburg
- an Cluj-Napoca
- ar كروج نابوكا
- ary كروج نابوكا
- arz كروج نابوكا
- ast Cluj-Napoca
- azb کروج نابوکا
- az Kluj-Napoca
- ba Круг-Напока
- be\_x\_old Круг-Напока
- be Круг-Напока
- bg Круг-Напока
- bjn Cluj-Napoca
- bn ক্লুজ-নাপোকা
- br Cluj-Napoca
- ca Cluj-Napoca
- ceb Muncipiul Cluj-Napoca
- ...

## Adding meaning to RDF graph: types

They are **optional**

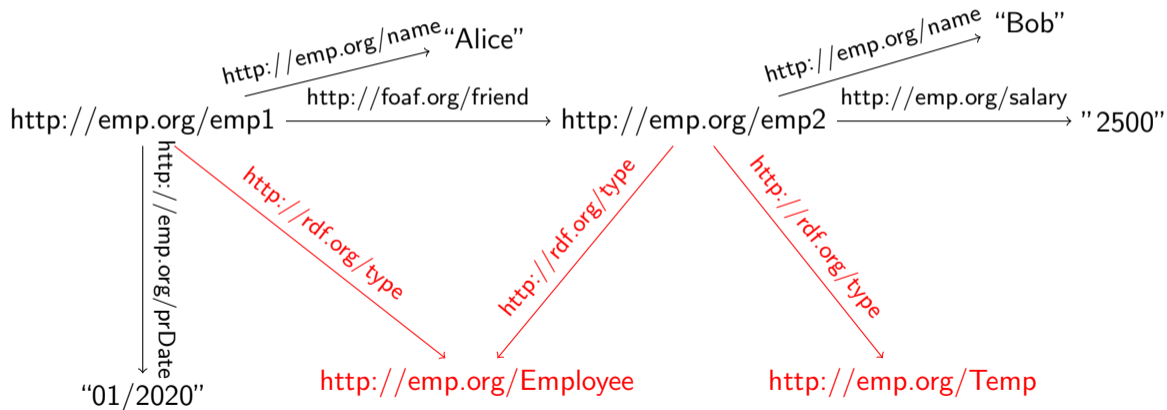
A resource can have **zero or more types** which can be related or not



## Adding meaning to RDF graphs: types

They are **optional**

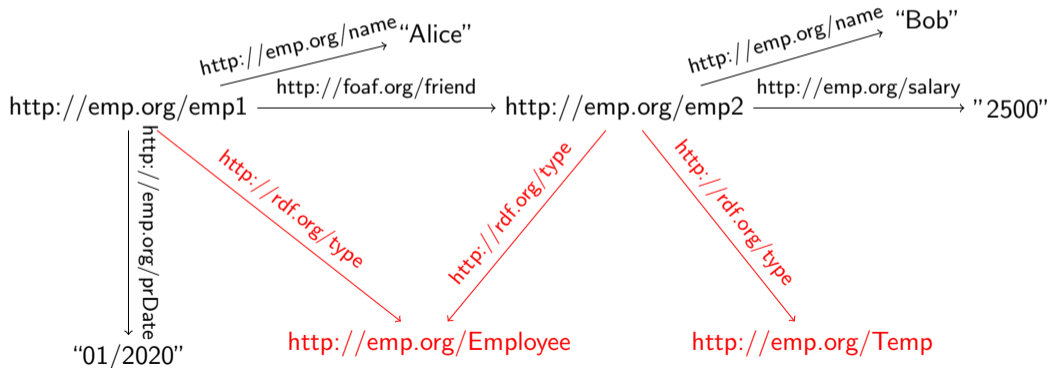
A resource can have **zero or more types** which can be related or not



## 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`

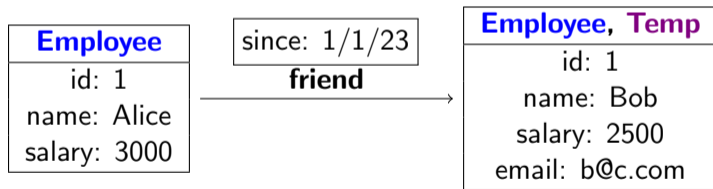


# Property graphs

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]



# 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  $\Rightarrow$  optimization)

**Trees** JSON, XML, K-V: Web content, structured documents, data exchange

**Graphs** RDF: Open Data; PGs: latest iteration of heterogeneous databases

### Support:

	Tables	XML	JSON	KV	RDF	PGs
Relational databases	✓	✓	✓	✓	✓	✓
XML databases		✓				
JSON databases			✓	✓		
KV stores				✓		
RDF databases					✓	✓
PG databases					✓	✓

# Data model heterogeneity: support (<https://db-engines.com>)

419 systems in ranking, July 2023

Rank			DBMS	Database Model	Score		
Jul 2023	Jun 2023	Jul 2022			Jul 2023	Jun 2023	Jul 2022
1.	1.	1.	Oracle	Relational, Multi-model	1256.01	+24.54	-24.28
2.	2.	2.	MySQL	Relational, Multi-model	1150.35	-13.59	-44.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	921.60	-8.47	-20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.83	+5.01	+1.96
5.	5.	5.	MongoDB	Document, Multi-model	435.49	+10.13	-37.49
6.	6.	6.	Redis	Key-value, Multi-model	163.76	-3.59	-9.86
7.	7.	7.	IBM Db2	Relational, Multi-model	139.81	-5.07	-21.40
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.59	-4.16	-14.74
9.	9.	9.	Microsoft Access	Relational	130.72	-3.73	-14.37
10.	10.	10.	SQLite	Relational	130.20	-1.02	-6.48
11.	11.	13.	Snowflake	Relational	117.69	+3.55	+18.53
12.	12.	11.	Cassandra	Wide column	106.53	-2.03	-7.88
13.	13.	12.	MariaDB	Relational, Multi-model	96.10	-1.21	-16.42
14.	14.	14.	Splunk	Search engine	87.12	-2.34	-11.09
15.	16.	15.	Microsoft Azure SQL Database	Relational, Multi-model	78.96	-0.01	-5.94
16.	15.	16.	Amazon DynamoDB	Multi-model	78.81	-1.10	-5.13
17.	17.	17.	Hive	Relational	72.87	-2.65	-6.61
18.	18.	22.	Databricks	Multi-model	68.47	+2.65	+17.25
19.	19.	18.	Teradata	Relational, Multi-model	60.25	-2.39	-10.67
20.	20.	24.	Google BigQuery	Relational	55.42	+0.78	+6.53
21.	21.	23.	FileMaker	Relational	53.32	-1.06	+2.12
22.	22.	19.	Neo4j	Graph	52.06	-0.71	-6.36

# Data model heterogeneity: can we eliminate it?

Short realist's answer:

It didn't happen so far!



# 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].



## 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 **data!**
- 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. **data!**
- Schema** A set of relations and/or views.

## Virtual views in SQL

Base table:

	city	street	name	descr	price
Hotel	Paris	Odéon	Flora	"Historic"	200
	Bastia	Corniche	Hippocampe	"Modern"	175

View definition:

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

Then:

`select name from ParisHotels` returns:

name
------

Flora
-------

# 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**

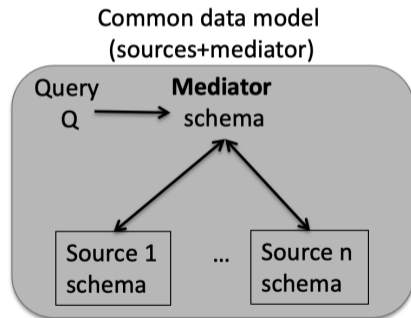


# Views for data integration

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

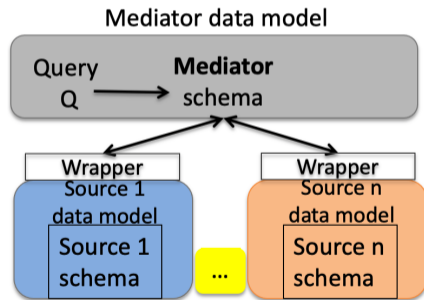


# Views for data integration

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

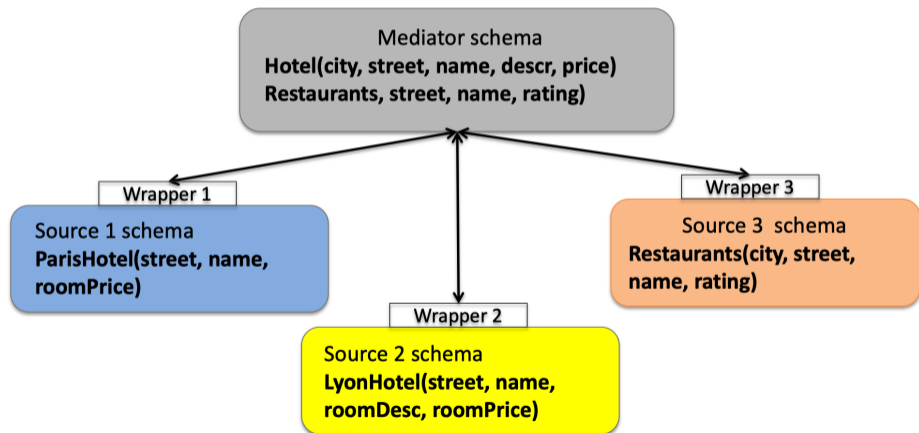
## Data integration system

- 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

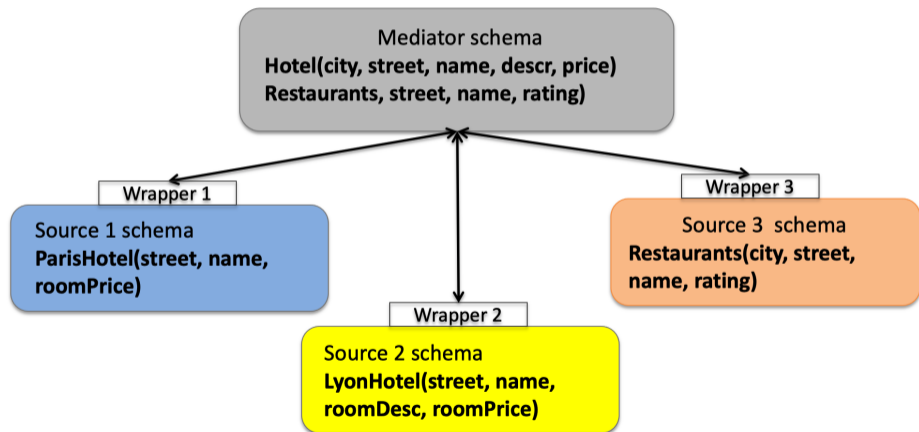
Sample scenario:



## Connecting the source schemas to the global schema

Data **only** exists in the sources.

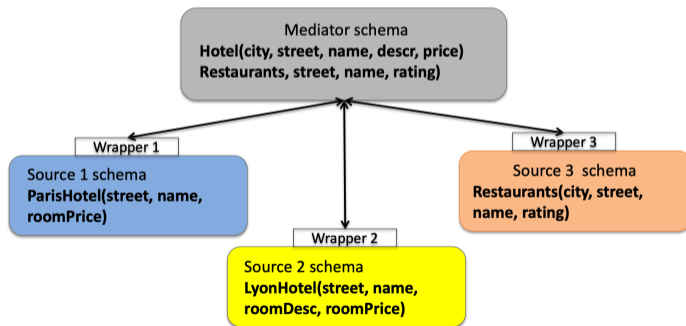
Applications can **only** query the global schema.



# 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)

Source 1	s1:ParisHotels(street, name, roomPrice)
Source 2	s2:LyonHotel(street, name, roomDesc, roomPrice)
Source 3	s3:Restaurant(city, street, name, rating)
Global	<b>Hotel</b> (city, street, name, descr, price), <b>Restaurant</b> (city, street, name, rating)

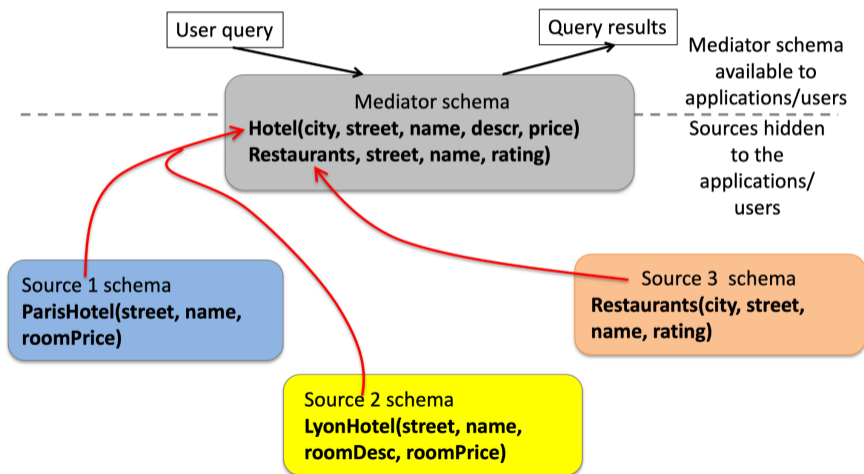
Defining Hotel as a view over the source schemas:

```
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:

```
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 <b>Hotel</b> 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	<b>Source-only!</b>



## 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:

---

```
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
```

---

## 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)

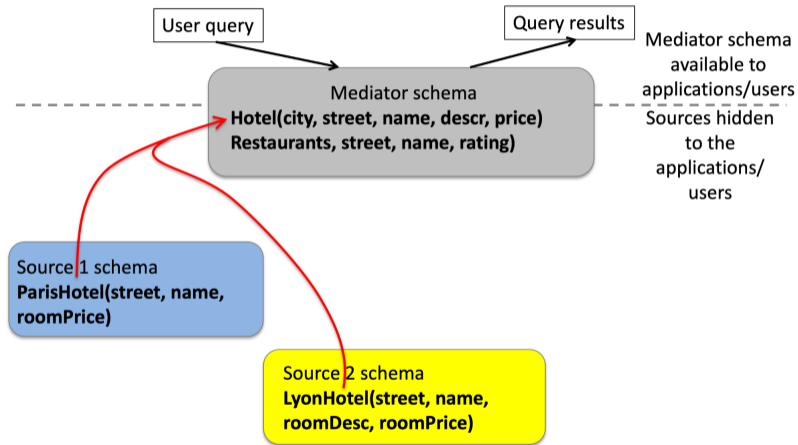
### Query processing = view unfolding

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

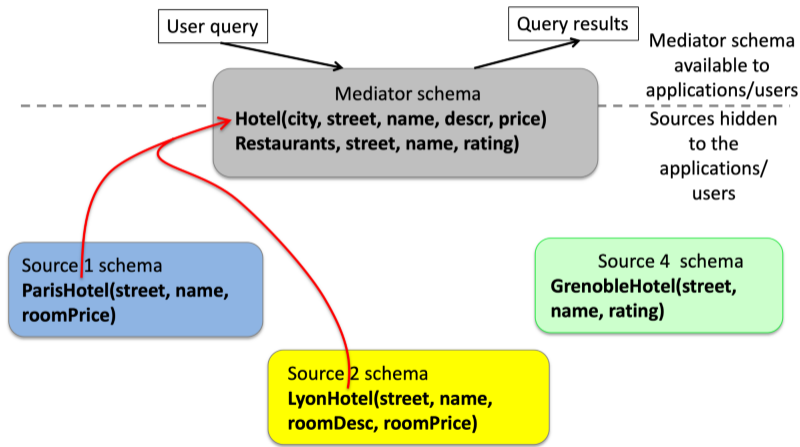
### Weakness: source changes impact the global schema

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

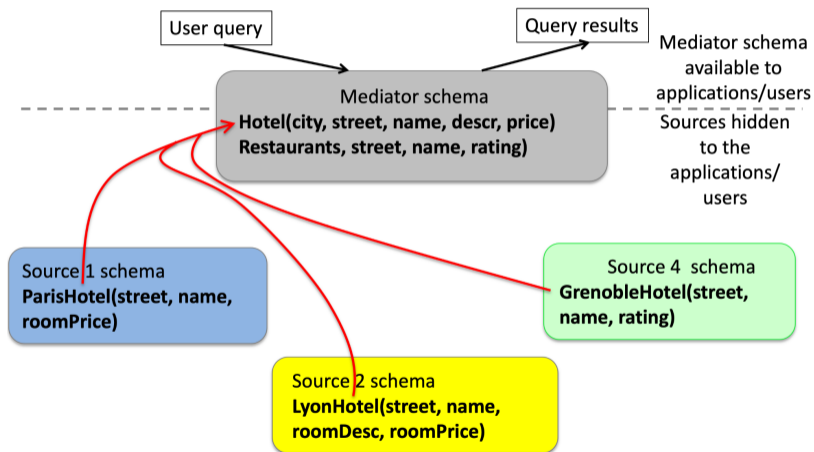
# Global-as-view: adding a new source



# Global-as-view: Adding a new source

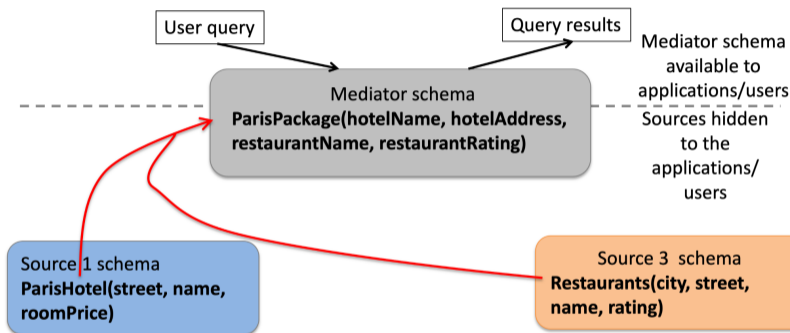


# Global-as-view: adding a new source



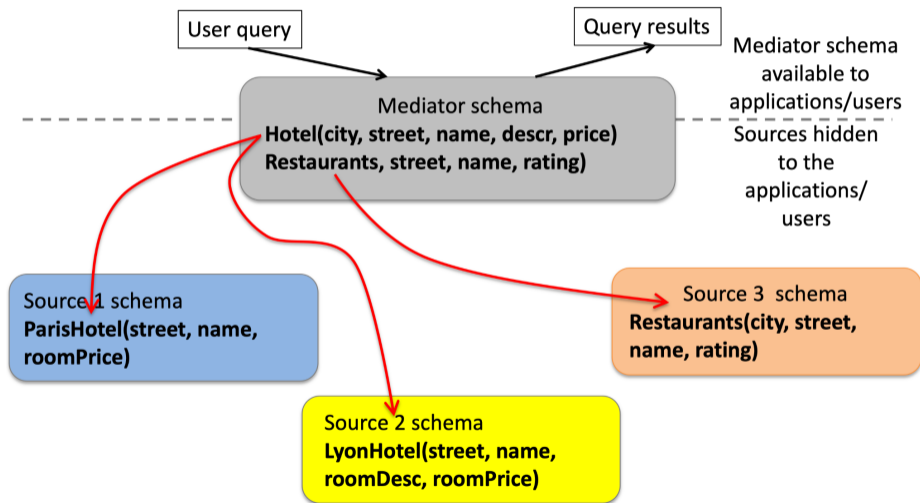


## Global-as-view: removing a source



- 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)



## 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

- s3:MHotel(city, street, name, price) only has data about Marseille hotels
- s4:WHotel(city, street, name price) has only data about Wien hotels
- **Hotel** is defined as:  

```
select * from MHotel union all select * from WHotel
```
- A query about hotels in Rome will also be sent to s3 and s4, although it will bring no results

## Query processing in Local-as-View (LAV)

```
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)

```
define view s1:ParisHotels as... from Hotel where city='Paris'
```

```
define view s2:LyonHotel as... from Hotel where city='Lyon'
```

```
define view 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 s2:LyonHotels h2, s3:Restaurant r3
where h2.city=h3.city and h2.street=r3.street
```

## Query processing in Local-as-View (LAV)

```
define view s1:ParisHotels as... from Hotel where city='Paris'
```

```
define view s2:LyonHotel as... from Hotel where city='Lyon'
```

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

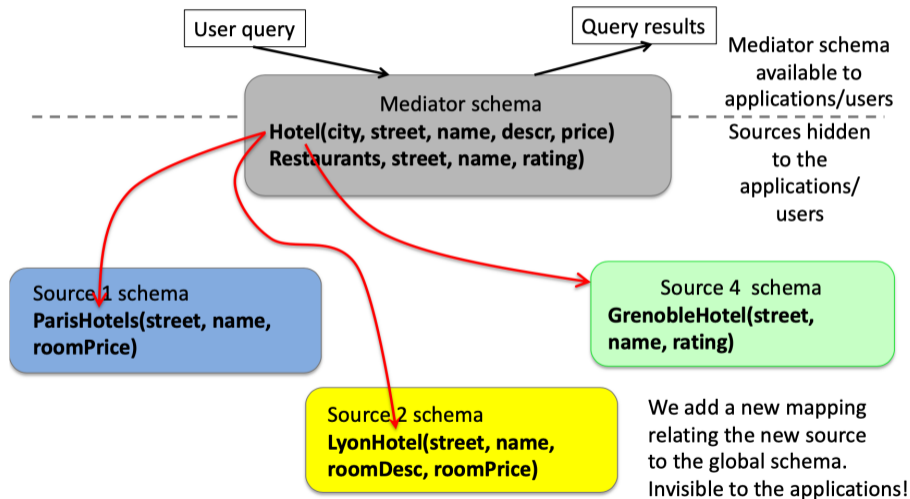
**Rewriting** of the query using the views:

```
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
```

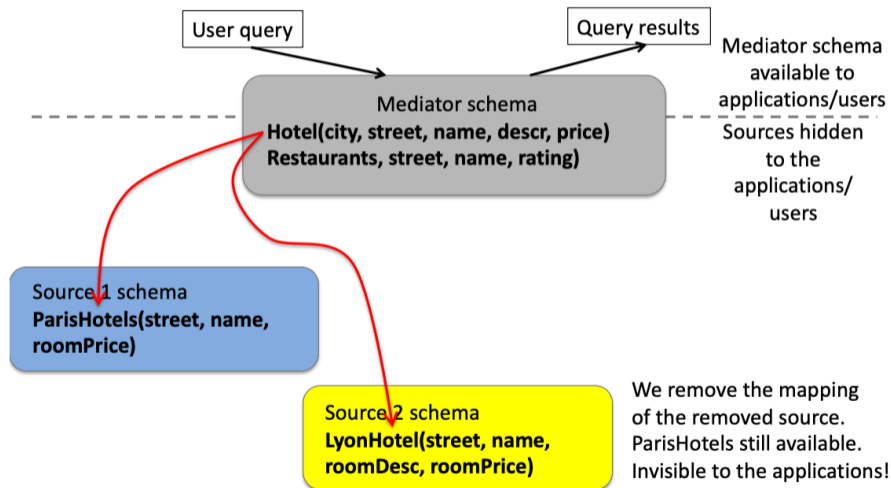
How do we find this?  
 It's more complicated  
 NP-hard problem [14]  
 in the size of views + query



# Local-as-View: adding a new source



# Local-as-View: removing a source



## 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.

# Applicability to (any) distributed data management scenario

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  $h(k) = i$

# Rewriting with Integrity Constraints

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

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

View	Query	Rewriting
“All restaurants”	“All restaurants with a license”	?

# Rewriting with Integrity Constraints

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

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

View	Query	Rewriting
“All restaurants”	“All restaurants with a license”	?

- 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” .

View	Query	Rewriting
“All restaurants”	“All restaurants with a license”	?

- 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.

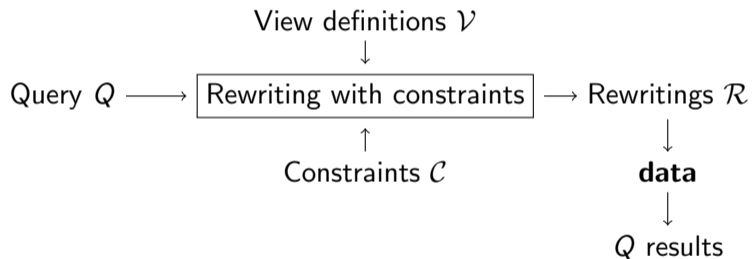
- *IC*: “Every restaurant has a sanitary license” .

View	Query	Rewriting
“All restaurants”	“All restaurants with a license”	?

- 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

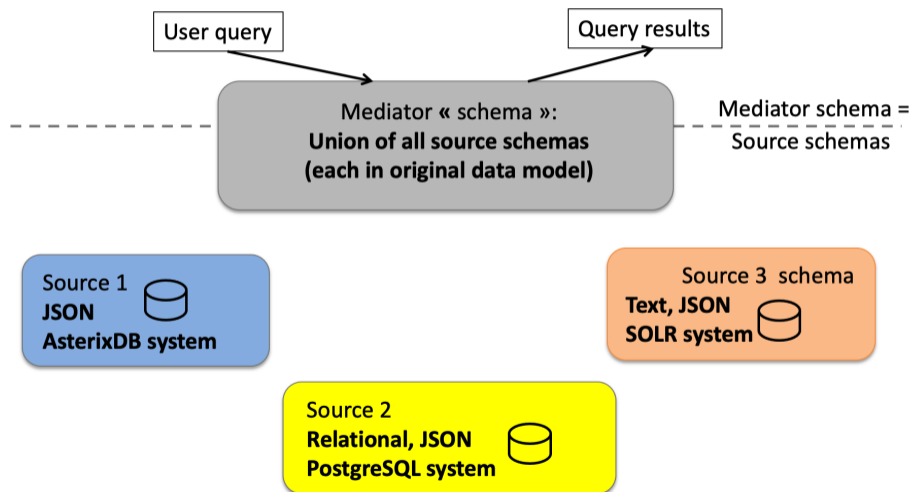
- A set of **data stores**, each supporting one or several **data models**
  - PostgreSQL supports relational, XML, JSON
- Each store can **evaluate queries** in some language / dialect
- Each store may have **better performance** on some operations
  - SOLR extremely good at full-text search; no  $\bowtie$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

ESTOCADA [2, 3, 4]: How to **leverage the best performance available in the system?**

# Connecting the source schemas to the global schema: ESTOCADA



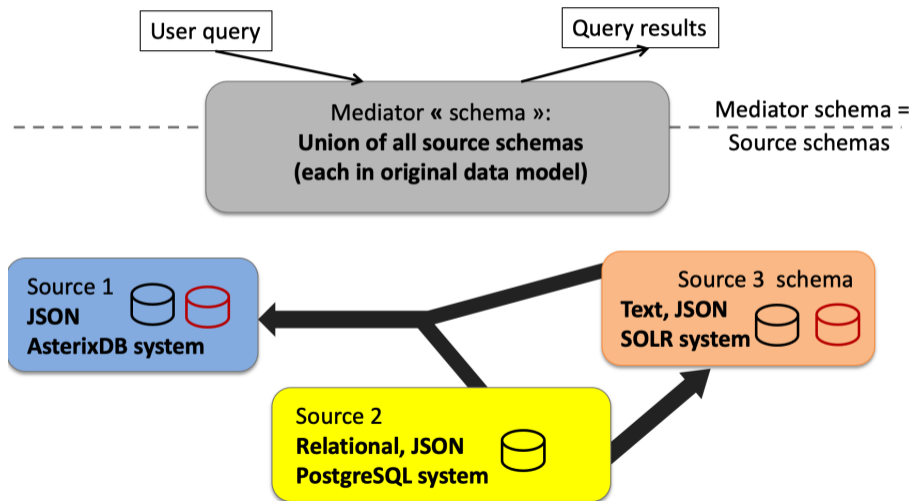
# Polystores: modern heterogeneous Big Data management systems

- 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

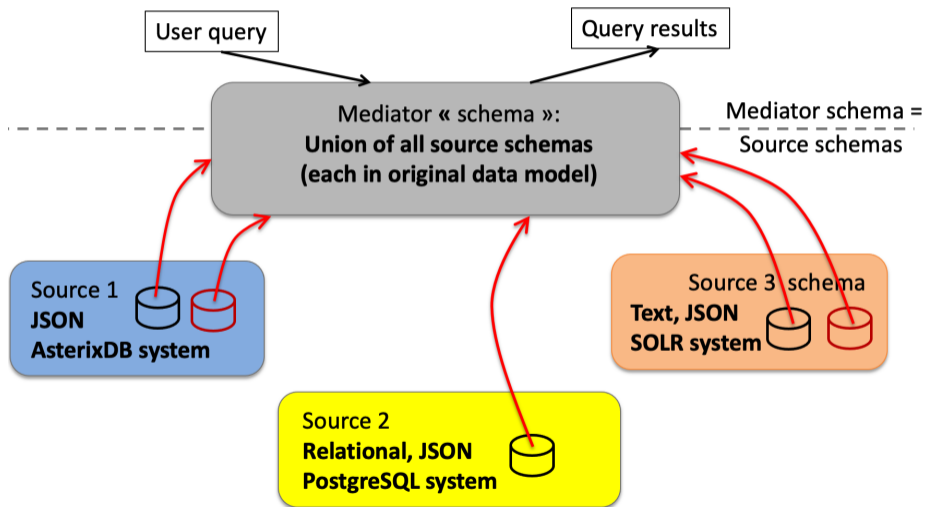
ESTOCADA [2, 3]: How to **leverage the best performance across all stores?**

- 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)

# Connecting the source schemas to the global schema: ESTOCADA



# Connecting the source schemas to the global schema: ESTOCADA (LAV)





# 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:{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}
```

- 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 ...

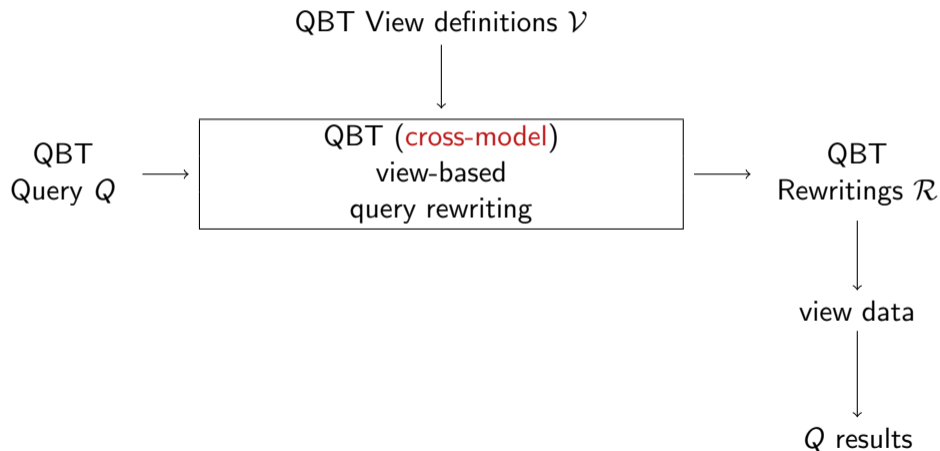
- One QBT block:  
FOR ... RETURN ...
- QBTs can be  
arbitrarily nested

## Sample query against JSON data originally stored in AsterixDB

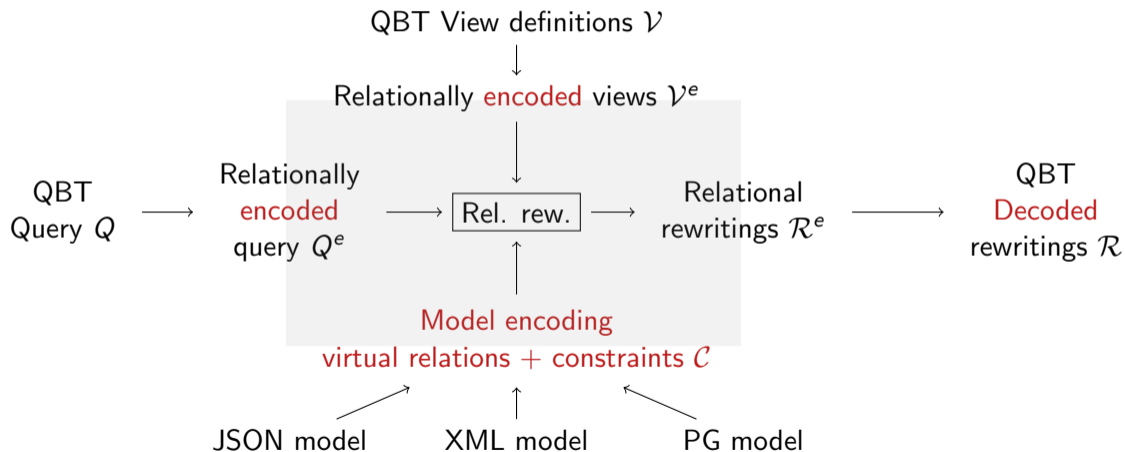
How to **rewrite** it using **PostgreSQL** and **SOLR** views?

```
FOR AJ:{SELECT M.patientID AS patientID,
            A.admissionLoc AS admissionID,
            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}
```

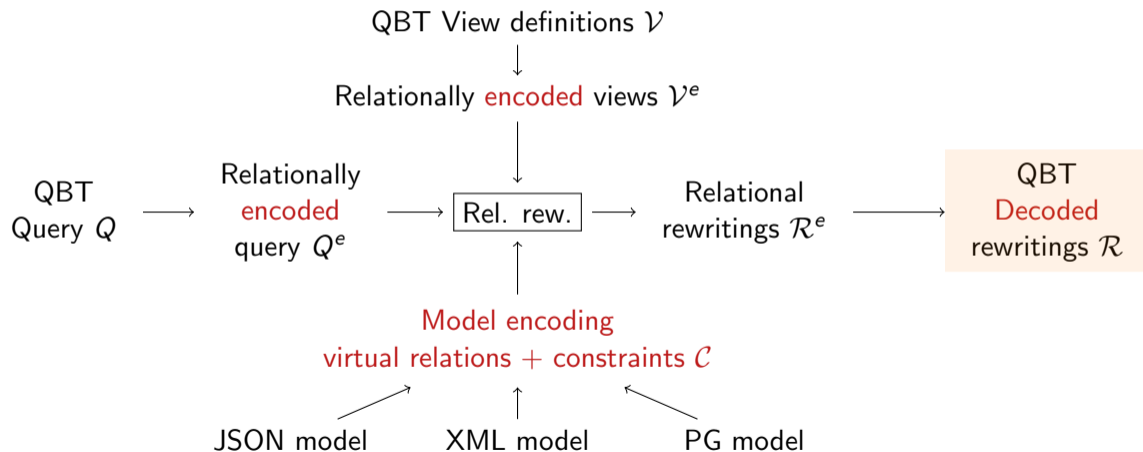
# ESTOCADA view-based query rewriting problem

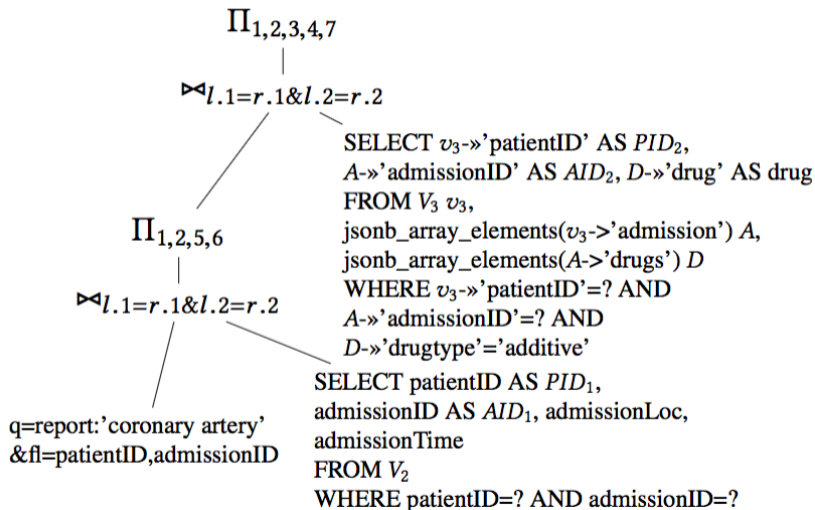


# ESTOCADA view-based rewriting approach



## ESTOCADA view-based rewriting approach

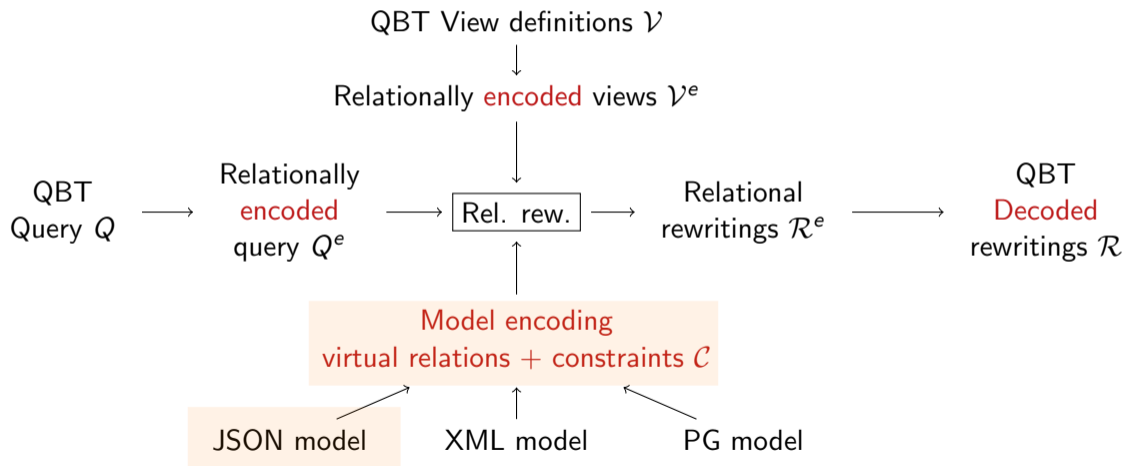


Our target: QBT decoded rewriting  $\mathcal{R}$ 

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



# ESTOCADA view-based rewriting approach



# Encoding the JSON data model in ESTOCADA

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

Constraints:

Describing the data:

$Collection_J(name, id)$
$Child_J(pld, cld, key, type)$

$$Collection_J(n, x) \wedge Collection_J(n, y) \rightarrow x = y \quad (1)$$

$$Child_J(p, c_1, k, t) \wedge Child_J(p, c_2, k, t) \rightarrow c_1 = c_2 \quad (2)$$

$$Eq_J(x, y) \rightarrow Eq_J(y, x) \quad (3)$$

$$Eq_J(x, y) \wedge Eq_J(y, z) \rightarrow Eq_J(x, z) \quad (4)$$

Needed for modelling:

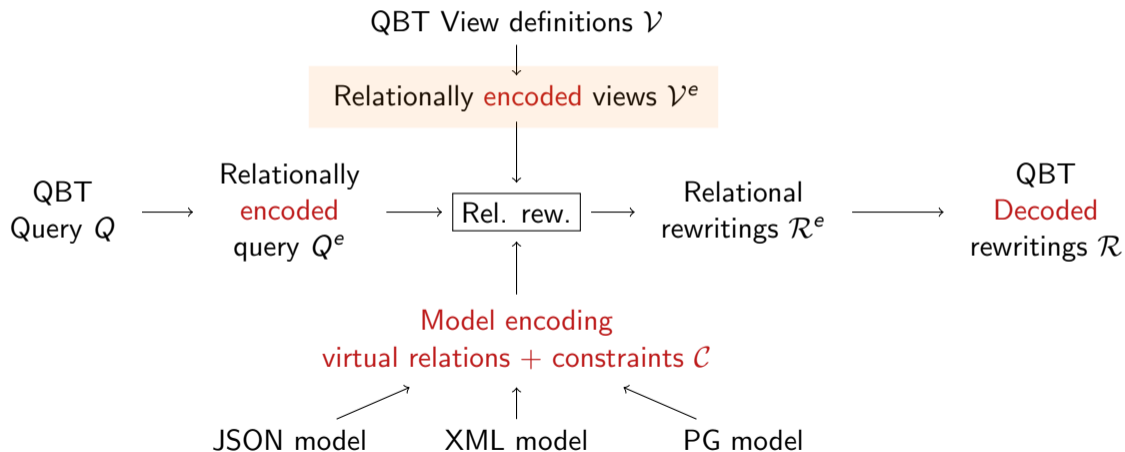
$Eq_J(x, y)$
$Value_J(x, y)$

$$Eq_J(p, p') \wedge Child_J(p, c, k, t) \rightarrow$$

$$\exists c' Eq_J(c, c') \wedge Child_J(p', c', k, t) \quad (5)$$

$$Value_J(i, v_1) \wedge Value_J(i, v_2) \rightarrow v_1 = v_2 \quad (6)$$

# ESTOCADA view-based rewriting approach



## Relational encoding of cross-model view $V_2$

```

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}

```

MIMIC(M),

$Child_J(M, patientID, "patientID", "o")$ ,

"o": object (or map) type

$Child_J(M, A, "admissions", "o")$ ,

$Child_J(A, admissionID, "admissionID", "o")$ ,

$Child_J(A, admissionLoc, "admissionLoc", "o")$ ,

$Child_J(A, admissionTime, "admissionTime", "o")$ ,

$Child_J(A, report, "reports", "o") \rightarrow$

$V_2(patientID, admissionID, admissionLoc, admissionTime)$

## 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

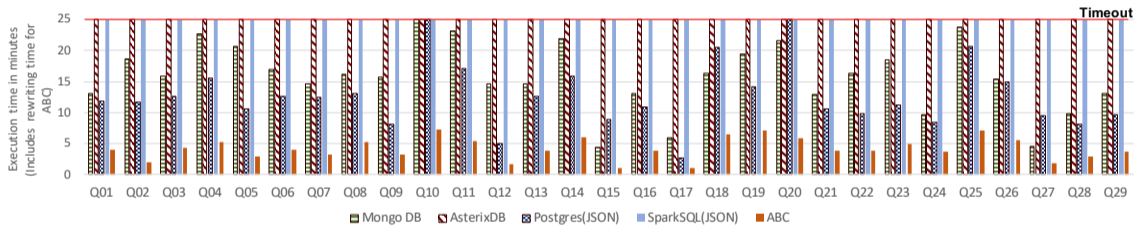
AsterixDB	SparkSQL	MongoDB	PostgreSQL JSON
25 min	> 60 min	16 min	12.6 min

### Materialized views:

- $V_1$  (SOLR): IDs of patients and their hospital admission data, caregivers' reports, notes on the patients' stay full-text search for “coronary artery”
- $V_2$  (Postgres): patients metadata, hospital admission information (admission time and patients' location) great join algos
- $V_3$  (Postgres JSON): drugs prescribed for each patient with “abnormal blood” test results

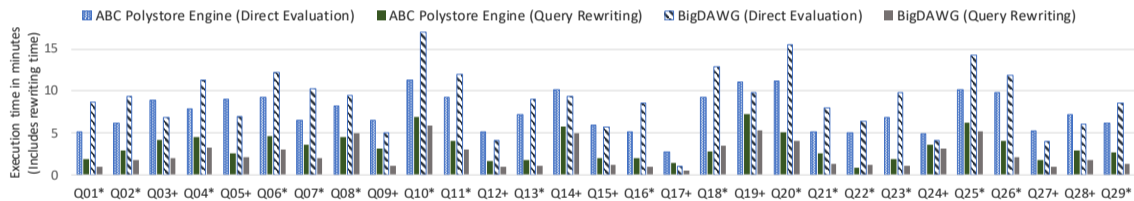
**Execution using mat. views** filter( $V_1$ )  $\bowtie$   $V_3$   $\bowtie$   $V_2$ : 5.7 min

# Query speed-up: comparison with single-store [2]



ESTOCADA: Relational and JSON views in Postgres, JSON views in SOLR

# 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 ( $\sigma, join, \pi$ ) - LA ( $+, \times, T, \dots$ ) 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

# ABSTRA: Abstracting Data of Any Model

Joint work with Nelly Barret (Inria), Prajna Upadhyay (Inria, now IIT Khanpur)

## 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](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](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](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](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**

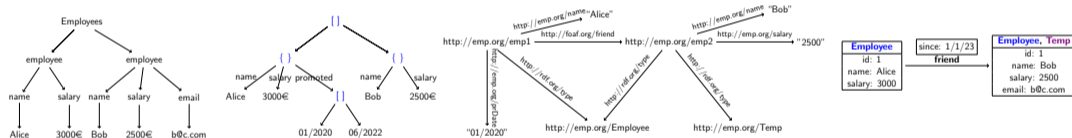
- Writers, not geeks
- Yet data skills needed more than ever

Research projects with *Le Monde*, *radiofrance* since 2015



# Data analysis needs for journalism

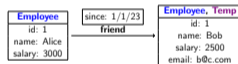
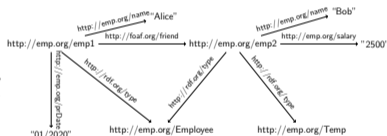
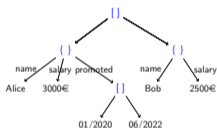
- The application domain **follows the topic of interest**: news cycle, or investigation topic.
- **Journalists use whatever data they can get their hands on**. Different data models! (All that we discussed + text, Office, etc.)



- Journalists are familiar with text and documents > spreadsheet ≫ anything else
- Data producers often **uncollaborative** ⇒ documentation, schema missing

# 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
- 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, ...

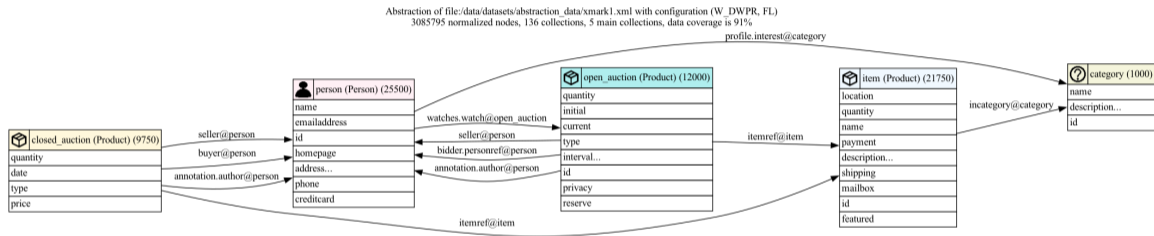
## Core principles

- 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

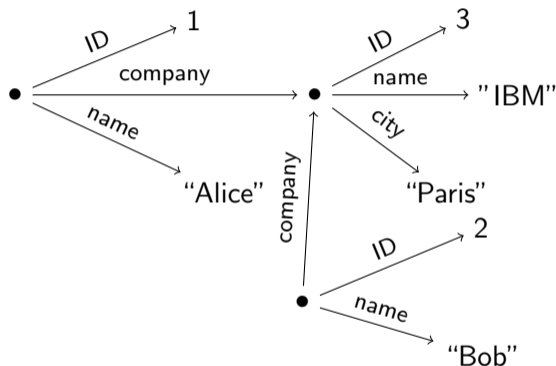
## Relational data conversion into graph

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

Employees		
ID	name	company
1	Alice	3
2	Bob	3

Companies		
ID	name	city
3	IBM	Paris



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

**Normalized** graph: make all edges unlabeled

## 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

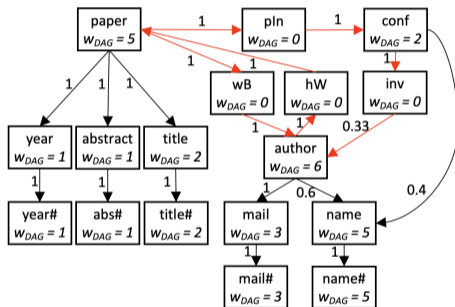
**RDF** (i) typed nodes are equivalent if they have the same set of types; (ii) untyped nodes are grouped according to a flexible notion of property cliques [13]

**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  $y_1 \equiv y_2$ .

# Graph summarization result: collection graph

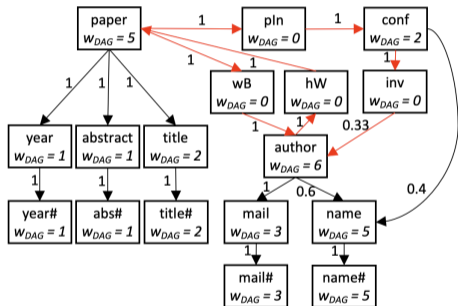
Each set of equivalent nodes is a **collection**; collections make up the **collections graph**



Red edges belong to cycles

year# is the collection of values of the nodes in the year collection

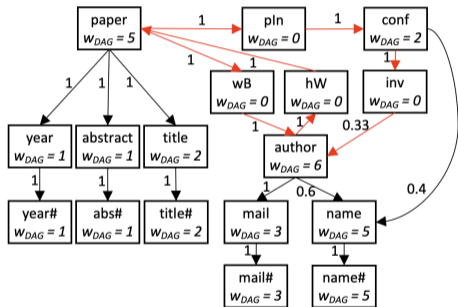
# Selecting entities and their boundaries



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.

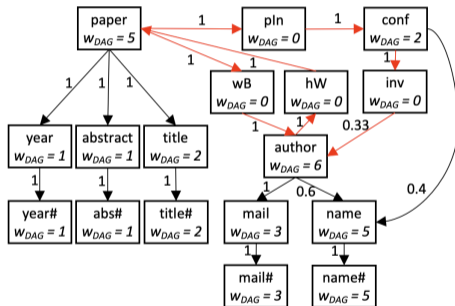
# Which collections make good entity roots?



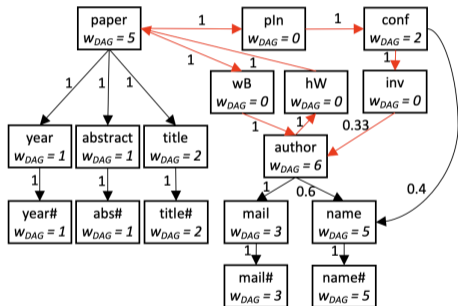


# Which collections make good entity roots?

- Must contain more than one node

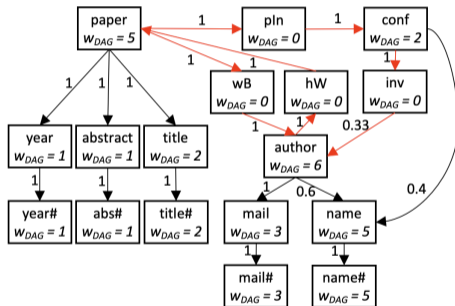


# Which collections make good entity roots?



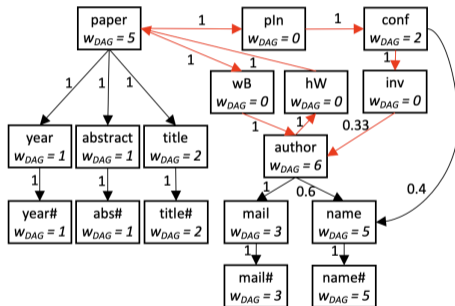
- Must contain more than one node
- The more attributes, the better

# Which collections make good entity roots?



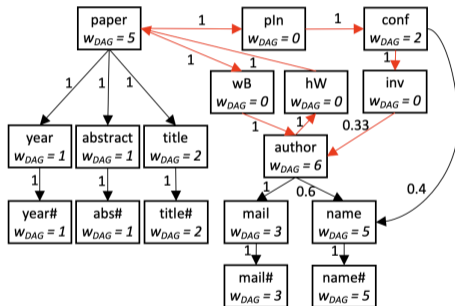
- Must contain more than one node
- The more attributes, the better
- The more descendant attributes, the better  
 ⇒ backward data weight propagation

# Which collections make good entity roots?



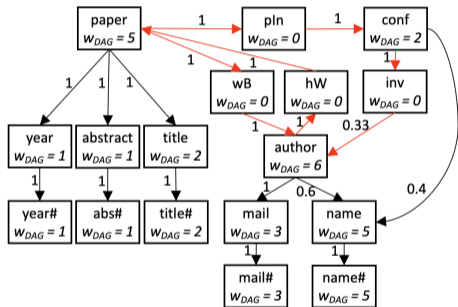
- Must contain more than one node
- The more attributes, the better
- The more descendant attributes, the better  
 $\Rightarrow$  backward data weight propagation
- More general: compute **PageRank** on the **inverse** collection graph: nodes with many descendant attributes are favored

## 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?

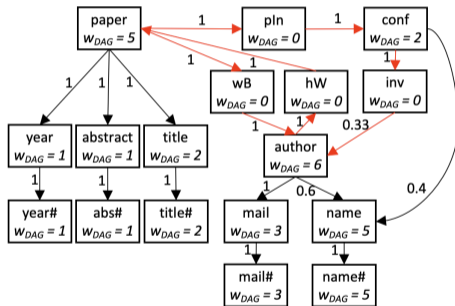


- Must contain more than one node
- The more attributes, the better
- The more descendant attributes, the better  
 $\Rightarrow$  backward data weight propagation
- More general: compute **PageRank** on the **inverse** collection graph: nodes with many descendant attributes are favored
- Pick the highest-rank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

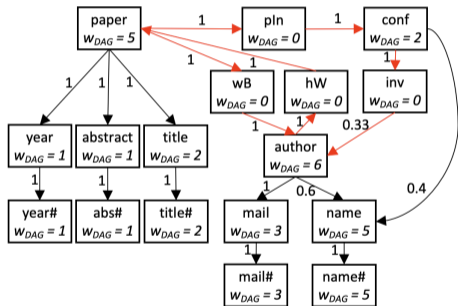
Assume **paper** is selected.

# What to include in an entity boundary?

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



# What to include in an entity boundary?

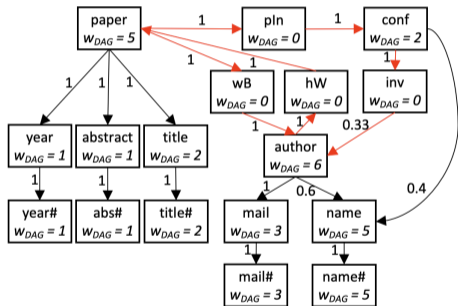


In the boundary of the entity rooted in  $c$ , we include any other collection  $c'$

- Such that there is a path  $c \rightsquigarrow c'$  with no in-cycle edges



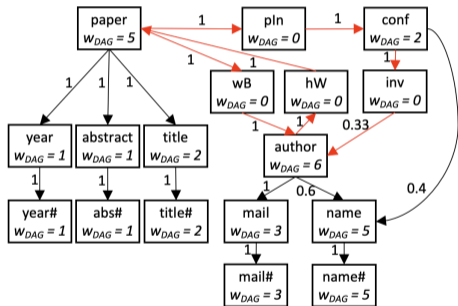
# What to include in an entity boundary?



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

- Such that there is a path  $c \rightsquigarrow c'$  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?

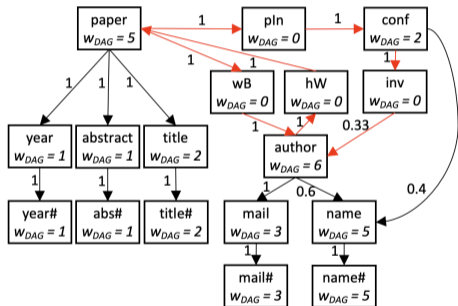


In the boundary of the entity rooted in  $c$ , we include any other collection  $c'$

- Such that there is a path  $c \rightsquigarrow c'$  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)

# Selecting the entities

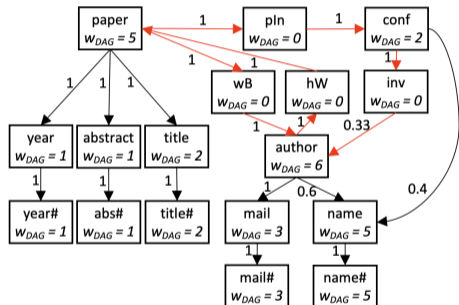


repeat

- 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

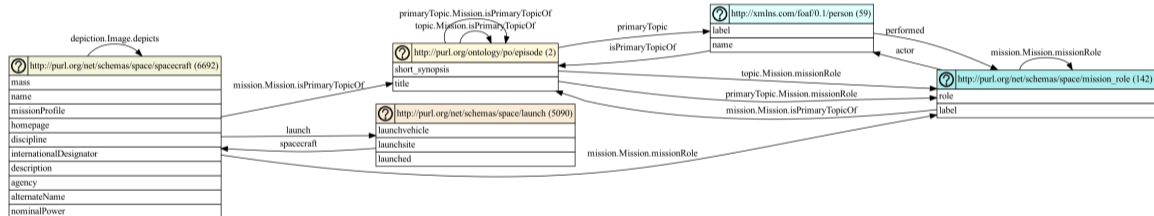
# Finding the relationships



- For each  $c_1$ , root of entity  $e_1$ 
  - For each path  $c_1 \rightsquigarrow c_2$  where  $c_2$  is the root of  $e_2$
  - Create a relationship  $e_1 \rightarrow e_2$  with the labels on the path

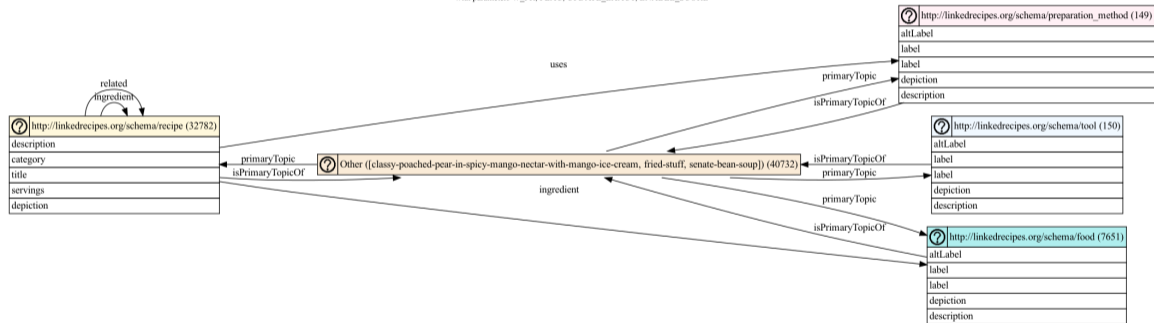
# Abstraction of NASA RDF graph

Abstraction of file:/Users/nelly/Documents/boulot/inria/abstraction-work/.data-CL/rd/nasa.nt (1156465 normalized nodes, 61 collections, 5 main collections, data coverage is 89%)  
with parameters W\_PR, FLAC, UPDATE\_EXACT, ENABLE\_SCORE



# Abstraction of Foodista RDF graph

Abstraction of file:/Users/nelly/Documents/boulot/inria/abstraction\_data/foodista.nt (1270301 normalized nodes, 49 collections, 5 main collections, data coverage is 50%)  
with parameters W\_PR, FLAC, UPDATE\_EXACT, ENABLE\_SCORE



# General Conclusion

- 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 I

- [1] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi.  
RHEEM: enabling cross-platform data processing - may the big data be with you!  
[PVLDB](#), 2018.
- [2] Rana Alotaibi, Damian Bursztyn, Alin Deutsch, Ioana Manolescu, and Stamatis Zampetakis.  
Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue.  
In [SIGMOD](#), June 2019.
- [3] Rana Alotaibi, Bogdan Cautis, Alin Deutsch, Moustafa Latrache, Ioana Manolescu, and Yifei Yang.  
ESTOCADA: towards scalable polystore systems.  
[Proc. VLDB Endow.](#), 13(12):2949–2952, 2020.
- [4] Rana Alotaibi, Bogdan Cautis, Alin Deutsch, and Ioana Manolescu.  
HADAD: A lightweight approach for optimizing hybrid complex analytics queries.  
In [SIGMOD](#), pages 23–35. ACM, 2021.
- [5] Angelos Anadiotis, Oana Balalau, Théo Bouganim, et al.  
Empowering investigative journalism with graph-based heterogeneous data management.  
[IEEE DEBull.](#), 2021.
- [6] Angelos Anadiotis, Oana Balalau, Catarina Conceicao, et al.  
Graph integration of structured, semistructured and unstructured data for data journalism.  
[Inf. Systems](#), 104, 2022.



# References II

- [7] Nelly Barret, Antoine Gauquier, Jia Jean Law, and Ioana Manolescu.  
Pathways: entity-focused exploration of heterogeneous data graphs (demonstration).  
In [ESWC](#), 2023.
- [8] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay.  
Abstra: toward generic abstractions for data of any model (demonstration).  
In [CIKM](#), 2022.
- [9] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay.  
Computing generic abstractions from application datasets.  
In [EDBT](#), 2024.
- [10] Raphaël Bonaque, Tien Duc Cao, Bogdan Cautis, François Goasdoué, Javier Letelier, Ioana Manolescu, Oscar Mendoza, Swen Ribeiro, Xavier Tannier, and Michaël Thomazo.  
Mixed-instance querying: a lightweight integration architecture for data journalism.  
In [VLDB](#), September 2016.
- [11] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, et al.  
Graph pattern matching in GQL and SQL/PGQ.  
In [SIGMOD](#), 2022.
- [12] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik.  
The BigDAWG polystore system.  
[SIGMOD](#), 2015.

# References III

- [13] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu.  
RDF graph summarization for first-sight structure discovery.  
[The VLDB Journal](#), 29(5), April 2020.
- [14] Alon Y. Halevy.  
Answering queries using views: A survey.  
[The VLDB Journal](#), 10(4), December 2001.
- [15] Ioana Ileana, Bogdan Cautis, Alin Deutsch, and Yannis Katsis.  
Complete yet practical search for minimal query reformulations under constraints.  
In [SIGMOD](#), 2014.
- [16] Martin L. Kersten and Lefteris Sidorouros.  
A database system with amnesia.  
In [CIDR](#). [www.cidrdb.org](http://www.cidrdb.org), 2017.
- [17] Boyan Kolev, Patrick Valduriez, Carlyna Bondiombouy, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira.  
Cloudmdsql: querying heterogeneous cloud data stores with a common language.  
[Distributed and parallel databases](#), 2016.