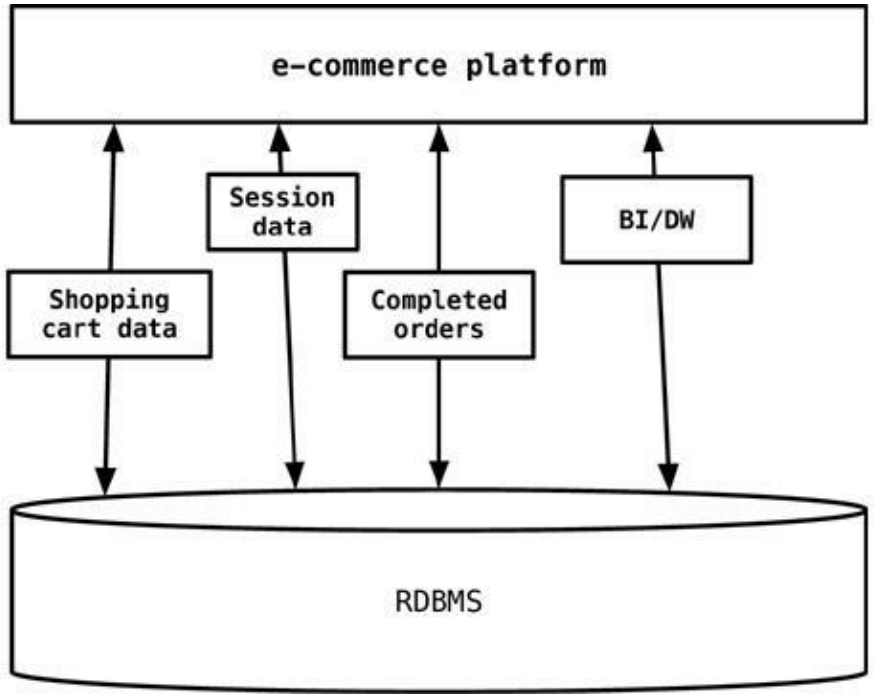
Polyglot persistence

What we are going to do

- Define polyglot persistence
- Define challenges
- Discuss current solutions and future directions

Polyglot persistence

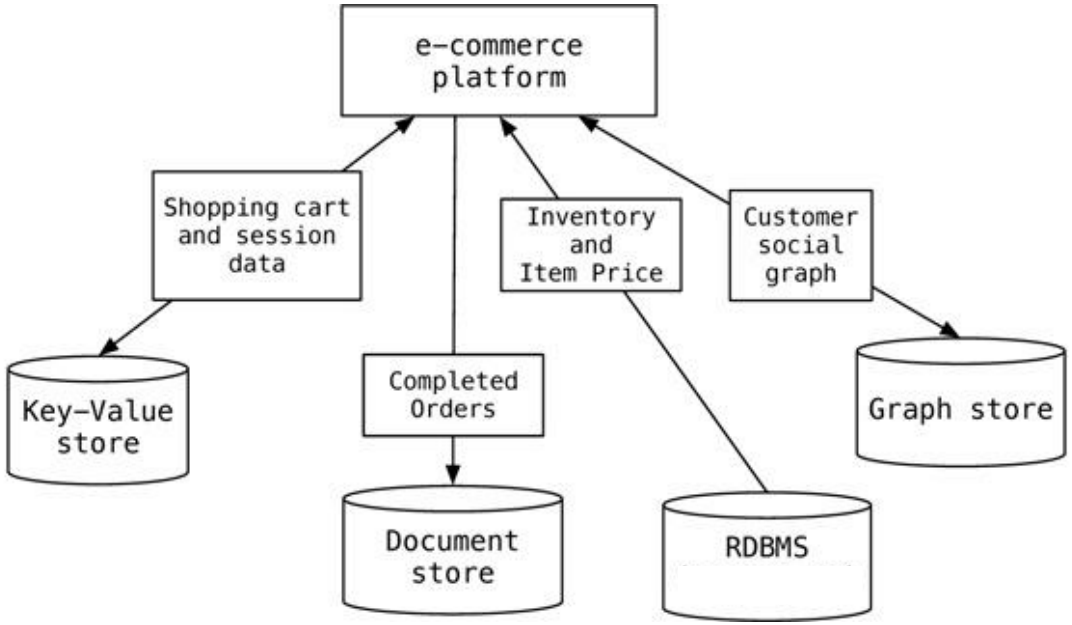
The *one-size-fits-all* solution



Polyglot persistence

~~The *one-size-fits-all* solution~~

Replaced by the *polyglot* solution



NoSQL databases

Emerged around the 2010's, NoSQL databases offer efficient solutions to specific problems

- Mainly distinguished by the supported data model

Model	Description	Use cases
Key-value	Associates any kind of value to a string	Dictionary, lookup table, cache, file and images storage
Document-based	Stores hierarchical data in a tree-like structure	Documents, anything that fits into a hierarchical structure
Wide-column	Stores sparse matrixes where a cell is identified by the row and column keys	Crawling, high-variability systems, sparse matrixes
Graph	Stores vertices and arches	Social network queries, inference, pattern matching

NoSQL databases: key-value

Each DB contains one or more collections (corresponding to tables)

Each collection contains a list of key-value pairs

- Key: a unique string
  - E.g.: ids, hashes, paths, queries, REST calls
- Value: a BLOB (binary large object)
  - E.g.: text, documents, web pages, multimedia files

Looks like a simple dictionary

- The collection is indexed by key
- The value may contain several information: one or more definitions, synonyms and antonyms, images, etc.

Key	Value
image-12345.jpg	Binary image file
http://www.example.com/my-web-page.html	HTML of a web page
N:/folder/subfolder/myfile.pdf	PDF document
9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

NoSQL databases: document-based

Each DB contains one or more collections (corresponding to tables)

Each collection contains a list of documents (usually JSON)

- Documents are hierarchically structured

Each document contains a set of fields

- The ID is mandatory

Each field corresponds to a key-value pair

- Key: unique string in the document
- Value: either simple (string, number, boolean) or complex (object, array, BLOB)
  - A complex field can contain other field

```
{
    "_id": 1234,
    "name": "Enrico",
    "age": 33,
    "address": {
        "city": "Ravenna",
        "postalCode": 48124
    },
    "contacts": [ {
        "type": "office",
        "contact": "0547-338835"
    }, {
        "type": "skype",
        "contact": "egallinucci"
    } ]
}
```

NoSQL databases: wide-column

Each DB contains one or more column families (corresponding to tables)

Each column family contains a list of rows in the form of a key-value pair

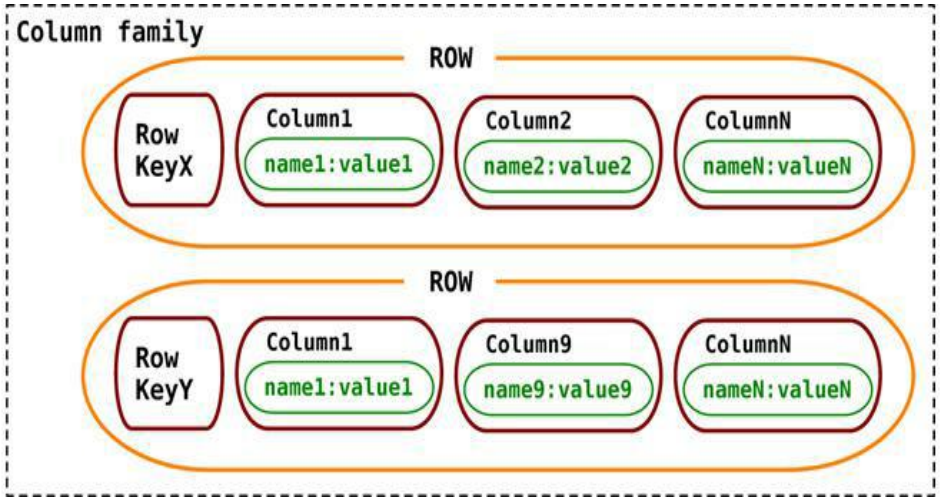
- Key: unique string in the column family
- Value: a set of columns

Each column is a key-value pair itself

- Key: unique string in the row
- Value: simple or complex (*supercolumn*)

With respect to the relational model:

- Rows specify only the columns for which a value exists
  - Particularly suited for sparse matrixes
- Timestamps can be used to defines *versions* of column values



NoSQL databases: graph

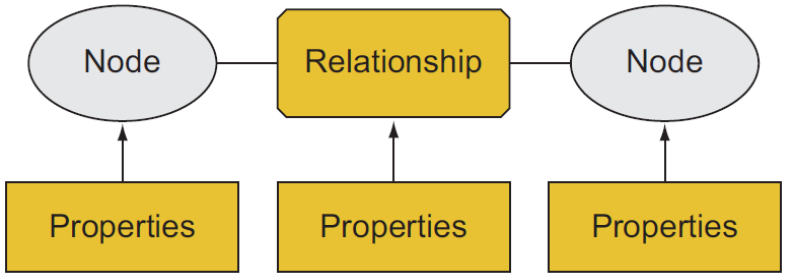
Each DB contains one or more graphs

Each graph contains vertices and arcs

- Vertices: usually represent real-world entities
  - E.g.: people, organizations, web pages, workstations, cells, books, etc.
- Arcs: represent directed relationships between the vertices
  - E.g.: friendship, work relationship, hyperlink, ethernet links, copyright, etc.
- Vertices and arcs are described by properties
- Arcs are stored as physical pointers

Most known specializations:

- Reticular data model
  - Parent-child or owner-member relationships
- Triplestore
  - Subject-predicate-object relationships (e.g., RDF)



Data modelling

Key-value, document and wide column are called aggregate-oriented

- Aggregate = key-value pair, document, row (respectively)
- The aggregate is the atomic block (no guarantees for multi-aggregate operations)

Based on the concept of encapsulation

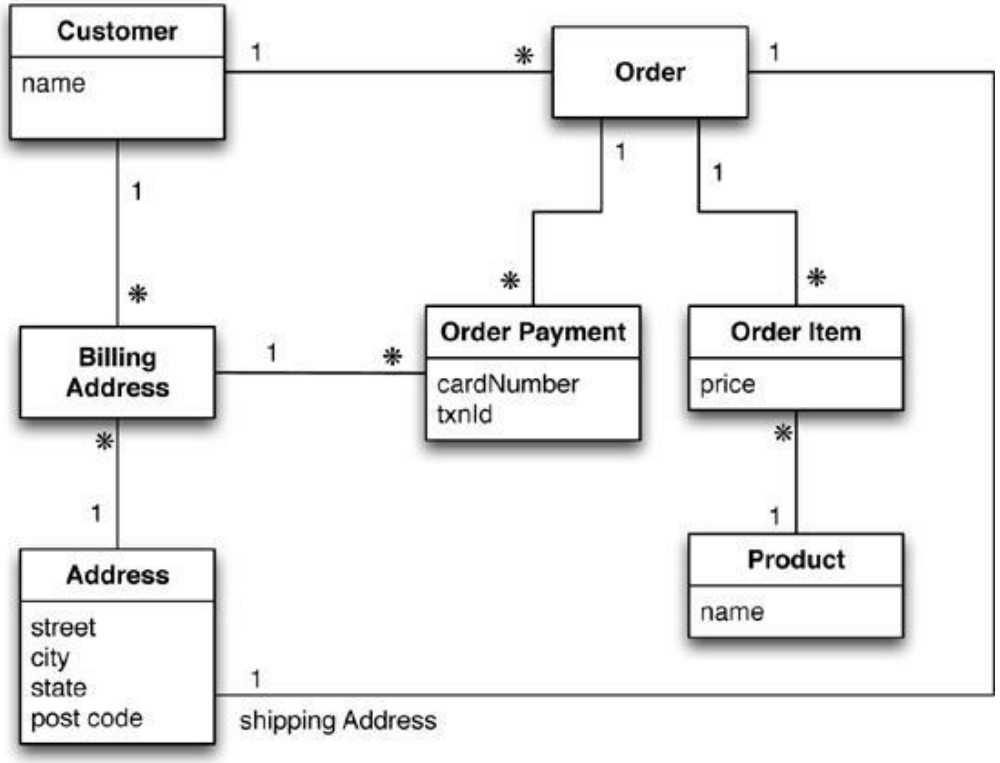
- Pro: avoid joins as much as possible → achieve **high scalability**
- Con: data denormalization → **potential inconsistencies in the data**
- Query-driven modeling

The graph data model is intrinsically different from the others

- Focused on the relationships rather than on the entities per-se
- **Limited scalability**: it is often impossible to shard a graph on several machines without "cutting" several arcs (i.e. having several cross-machine links)
  - Batch cross-machine queries: don't follow relationships one by one, but "group them" to make less requests
  - Limit the depth of cross-machine node searches
- Data-driven modeling

Data modelling

Typical use case: customers, orders and products



Relational data modelling

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

NoSQL data modelling (kv)

Customer collection

key	value
cust-1:name	Martin
cust-1:adrs	[{"street":"Adam", "city":"Chicago", "state":"Illinois", "code":60007}, {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}]

cust-1:ord-99	{
	"orderpayments": [
	{"card":477, "billadrs":
	{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007} },
	{"card":457, "billadrs":
	{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
	],
	"products": [
	{"id":1, "name":"Cola", "price":12.4},
	{"id":2, "name":"Fanta", "price":14.4}
	],
	"shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", code":10001}
	}

Product collection

key	value
p-1:name	Cola
p-2:name	Fanta

NoSQL data modelling (document)

Customer collection

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ],
  "orders": [ {
    "orderpayments":[
      {"card":477, "billadrs": {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
      {"card":457, "billadrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
    ],
    "products":[
      {"id":1, "name":"Cola", "price":12.4},
      {"id":2, "name":"Fanta", "price":14.4}
    ],
    "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  }]
}
```

Product collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

NoSQL data modelling (document)

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ]
}
```

Customer collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

Product collection

```
{
  "_id": 1,
  "customer":1,
  "orderpayments":[
    {"card":477, "billadrs":{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
    {"card":457, "billadrs":{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
  ],
  "products": [
    {"id":1, "name":"Cola", "price":12.4},
    {"id":2, "name":"Fanta", "price":14.4}
  ],
  "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
}
```

Order collection

NoSQL data modelling (wide-column)

Order table > Order details column family

Ord	CustName	Pepsi	Cola	Fanta	…
1	Martin		12.4	14.4	
2	…	…			…

Order table > Order payments column family

Ord	OrderPayments
1	Card / Steet / City / State / Code 477 / 9th / NewYork / NewYork / 10001 457 / Adam / Chicago / Illinois / 60007
2	…

NoSQL data modelling

The *aggregate* term comes from Domain-Driven Design

- An aggregate is a group of tightly coupled objects to be handled as a block
- Aggregates are the basic unit for data manipulation and consistency management

Advantages

- Can be distributed trivially
  - Data the should be used together (e.g., orders and details) are stored together
- Facilitate the developer's job
  - By surpassing the impedance mismatch problem

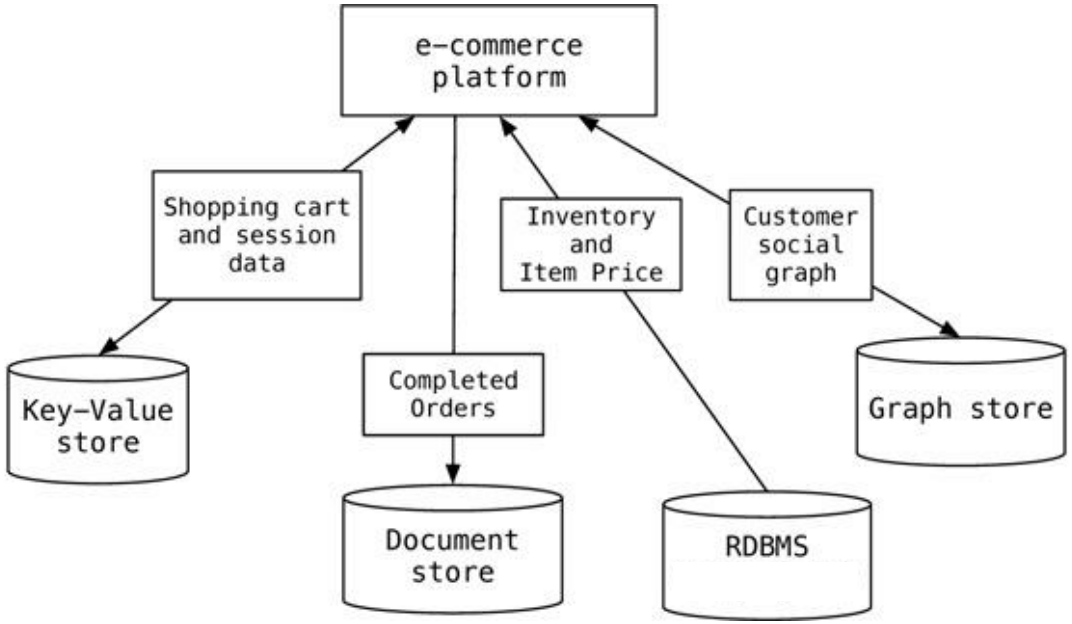
Disadvantages

- **No design strategy exists for aggregates**
  - It only depends on how they are meant to be used
- Can optimize only a limited set of queries
- Data denormalization → possible inconsistencies

RDBMSs are agnostic from this point of view

Polyglot persistence

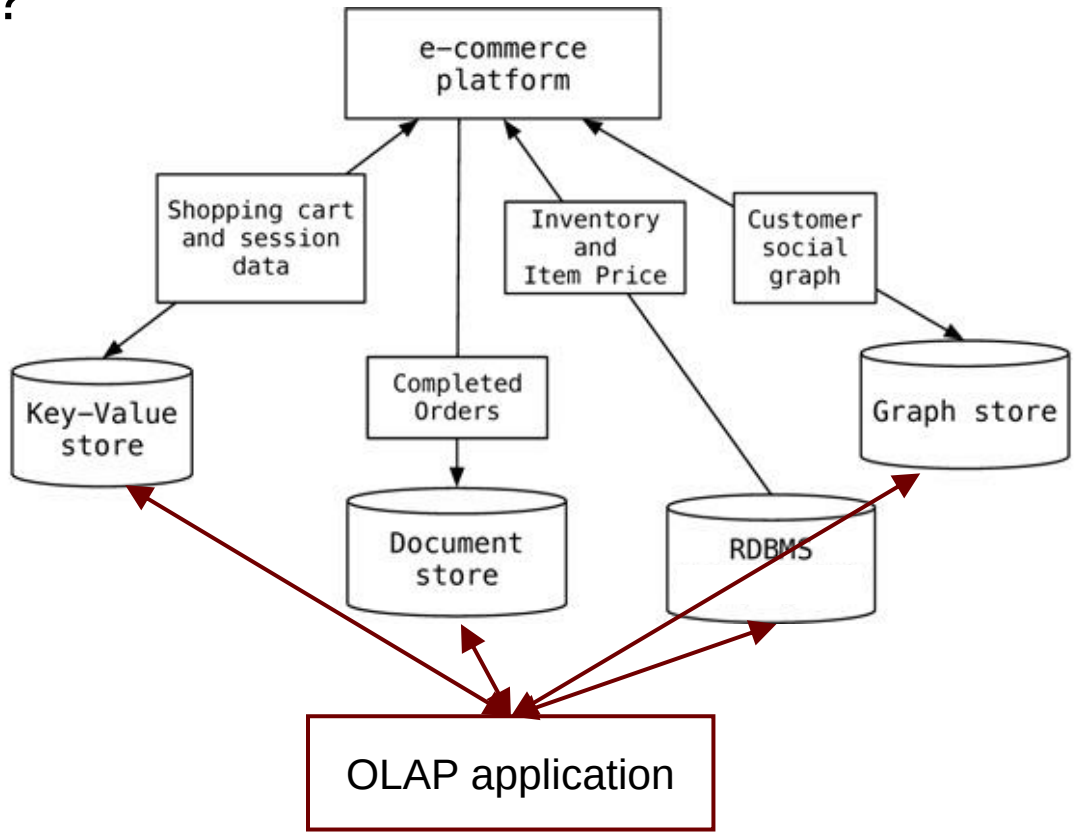
To each application the appropriate DBMS



Polyglot persistence

To each application the appropriate DBMS - works well for OLTP

What about OLAP?



Polyglot persistence: main challenges

Data model heterogeneity

- Support multiple models in the same database
- Or integrate data from different databases using different query languages

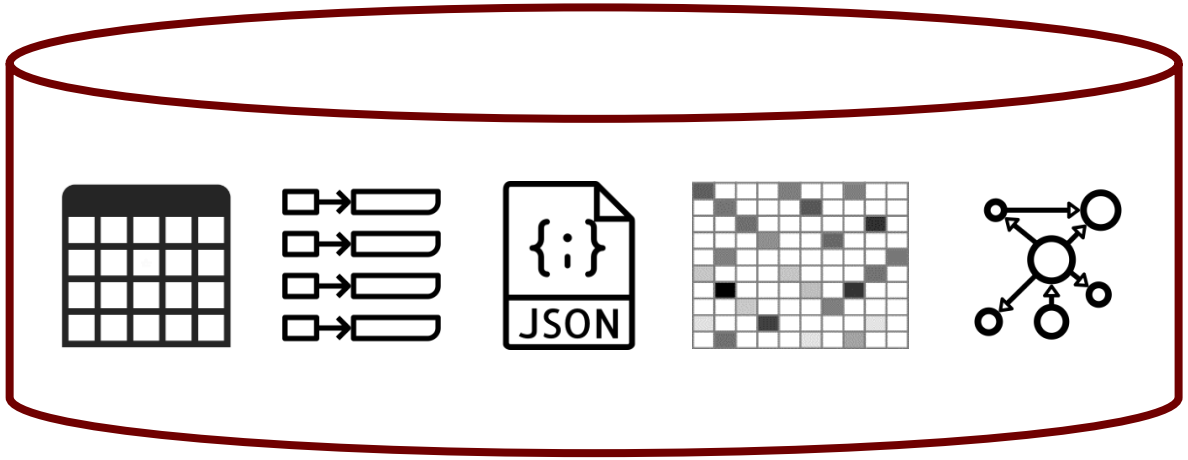
Schema heterogeneity

- Inter-collection: different records in **different** collections have different schemas
  - Not a new problem: think federated databases, corporate mergers, etc.
- Intra-collection: different records in **the same** collection have different schemas
  - Emerged with NoSQL databases

Data inconsistency

- Reconcile inconsistent versions of the same data (inter- or intra-collection)

Data model heterogeneity



Basic solutions

Some DBMSs offer multi-model support

- Extended RDBMSs
  - KV implementable as a table with two fields: a string key, and a blob value
  - Cypher query language on top of a relational implementation of a graph
  - Hstore data type in PostgreSQL for wide-column-like implementation
  - **Scalabilty issue remains**
- Multi-model NoSQL DBMSs
  - ArangoDB, OrientDB
  - **Support all NoSQL data models, but not the relational one**

Some approaches suggest strategies to model everything within RDBMSs

- DiScala, M., Abadi, D.J.: **Automatic generation of normalized relational schemas from nested key-value data**. In: *2016 ACM SIGMOD Int. Conf. on Management of Data*, pp. 295-310. ACM (2016)
- Tahara, D., Diamond, T., Abadi, D.J.: **Sinew: a SQL system for multi-structured data**. In: *2014 ACM SIGMOD Int. Conf. on Management of Data*, pp. 815-826. ACM (2014)

A taxonomy for distributed solutions

Federated database system

- **Homogeneous** data stores, exposes a **single** standard query interface
- Features a mediator-wrapper architecture, employs schema-mapping and entity-merging techniques for integration of relational data

Polyglot system

- **Homogeneous** data stores, exposes **multiple** query interfaces
- Takes advantage of the semantic expressiveness of multiple interfaces (e.g., declarative, procedural)

Multistore system

- **Heterogeneous** data stores, exposes a **single** query interface
- Provides a unified querying layer by adopting ontologies and applying schema-mapping and entity-resolution techniques

Polystore system

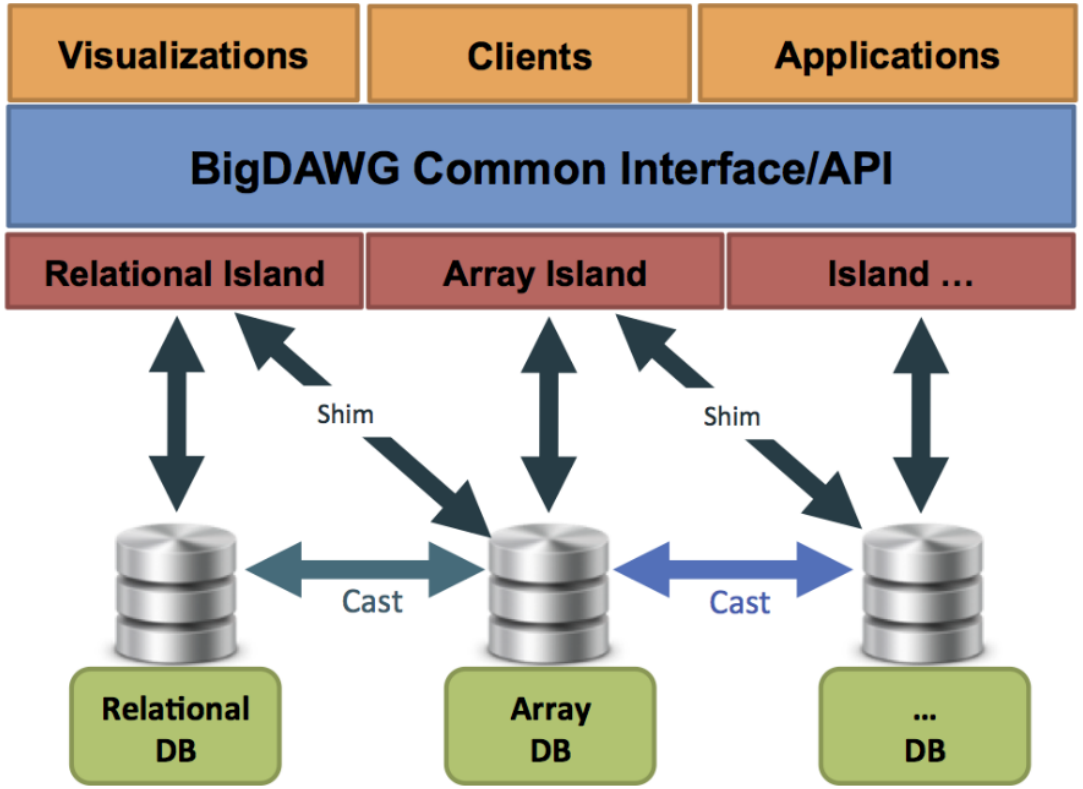
- **Heterogeneous** data stores, exposes **multiple** query interfaces
- Choose from a variety of query interfaces to seamlessly query data residing in multiple data stores

R. Tan, R. Chirkova, V. Gadepally and T. G. Mattson, "**Enabling query processing across heterogeneous data models: A survey,**" *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 3211-3220.

Advanced solutions

Example of a polystore

- Island = a middleware application to support a set of operations on a given data model
- Shim = a wrapper to convert from the island's query language to the target DB's query language



Vijay Gadepally, Kyle O'Brien, Adam Dziedzic, Aaron J. Elmore, Jeremy Kepner, Samuel Madden, Tim Mattson, Jennie Rogers, Zuohao She, Michael Stonebraker: **Version 0.1 of the BigDAWG Polystore System**. *CoRR abs/1707.00721* (2017)

Advanced solutions

Most notable multistore/polystore proposals

- BigDAWG
  - Focus on the ability to “move” data from one DB to another to improve query efficiency
    - V. Gadepally et al. **Version 0.1 of the BigDAWG Polystore System**. *CoRR abs/1707.00721* (2017)
- Estocada
  - Focus on taking advantage of possible (consistent) redundancy and previous query results
    - R. Alotaibi et al. **ESTOCADA: Towards Scalable Polystore Systems**. *Proc. VLDB Endow*. 13(12): 2949-2952 (2020)
- Awesome
  - Focus on supporting common analytical functions
    - S. Dasgupta. **Analytics-driven data ingestion and derivation in the AWESOME polystore**. *IEEE BigData 2016*: 2555-2564
- CloudMdsQL
  - Focus on taking advantage of local data store native functionalities
    - B. Kolev et al. **CloudMdsQL: querying heterogeneous cloud data stores with a common language**. *Distributed Parallel Databases* 34(4): 463-503 (2016)

Beyond data model heterogeneity

What else is there?

Entity resolution

- Every approach needs some kind of integrated knowledge
- Ample research from federated database systems
- Usually “out-of-scope”

Management of **schema heterogeneity** and **data inconsistency**

- Usually addressed as different problems in the literature

Schema heterogeneity

Heterogeneous data stored with variant schemata and structural forms

- Missing/additional attributes
- Different names/types of attributes
- Different nested structures

Two main problems

- Understand the data
- Query the data

Understanding the data

Early work on XML

- To deal with the widespread lack of DTDs and XSDs
- Extract regular expressions to described the content of elements in a set of XML documents

Recent work on JSON

- **Concise view**: a single representation for all schema variations
  - Union of all attributes
    - M. Klettke et al. **Schema extraction and structural outlier detection for JSON-based NoSQL data stores**., in: *Proc. BTW, volume 2105*, 2015, pp. 425-444.
  - A *skeleton* as the smallest set of core attributes according to a frequency-based formula
    - L. Wang et al. **Schema management for document stores**, *Proc. VLDB Endowment* 8 (2015) 922-933.
- **Comprehensive view**: multiple representations (a different schema for every document)
    - D. S. Ruiz, et al**. Inferring versioned schemas from NoSQL databases and its applications**, in: *Proc. ER*, *2015*, pp. 467-480.
- **Schema profile**: *explain why* there are different schemas
    - Enrico Gallinucci et al. **Schema profiling of document-oriented databases**. *Inf. Syst*. 75: 13-25 (2018)

Schema profiling

Schema profiles explain

- What are the differences between schemas
- When/why is one schema used instead of the other

SchemaID	User	Activity	Weight	Duration	Repetitions
S1	Jack	Run		108	
S2	John	Leg press	80	4	23
S1	Kate	Walk		42	
S3	John	Push-ups		8	40

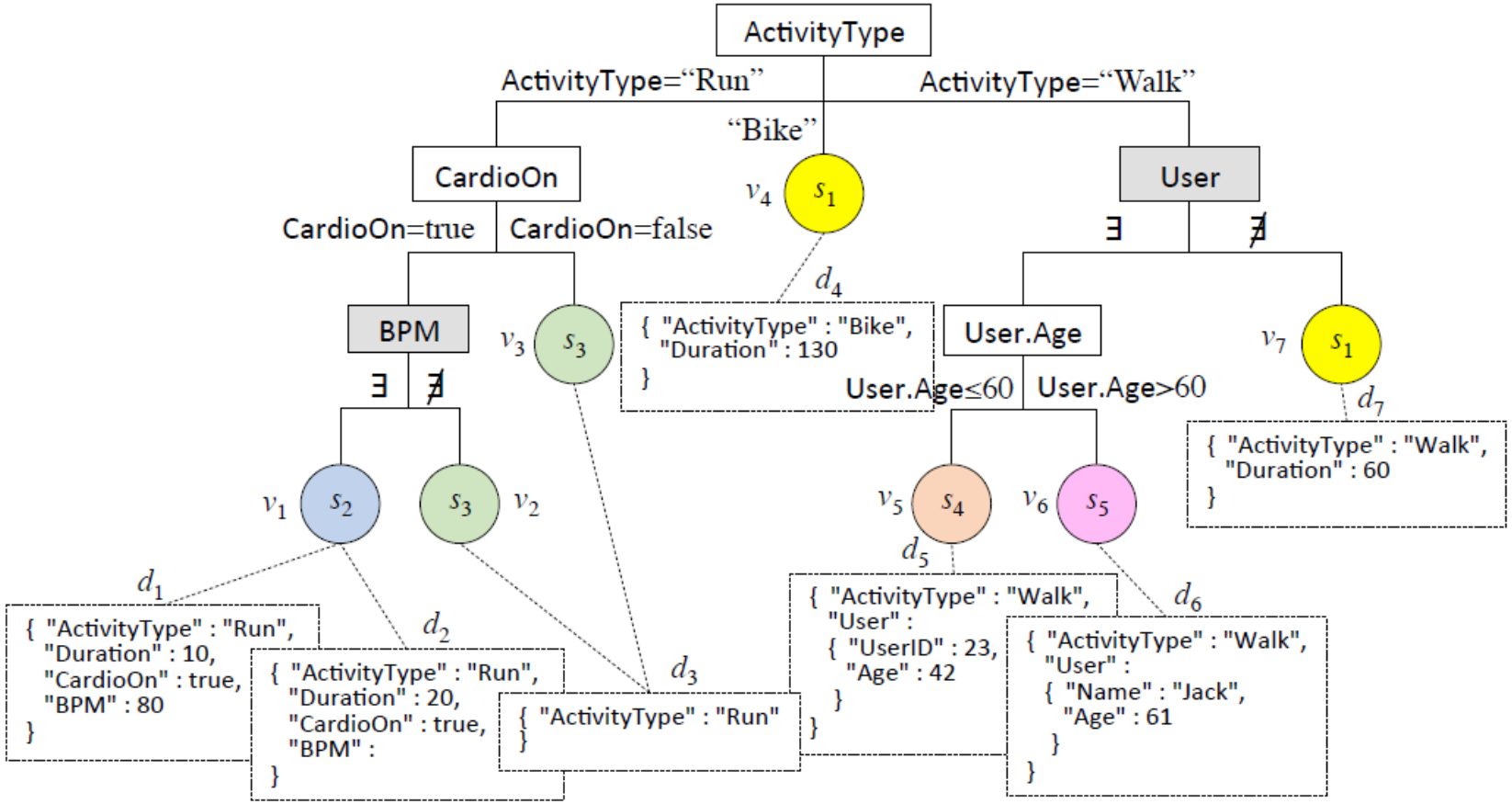
Documents / Observations

Schema / Class

The problem of schema profiling is quite similar to a classification problem

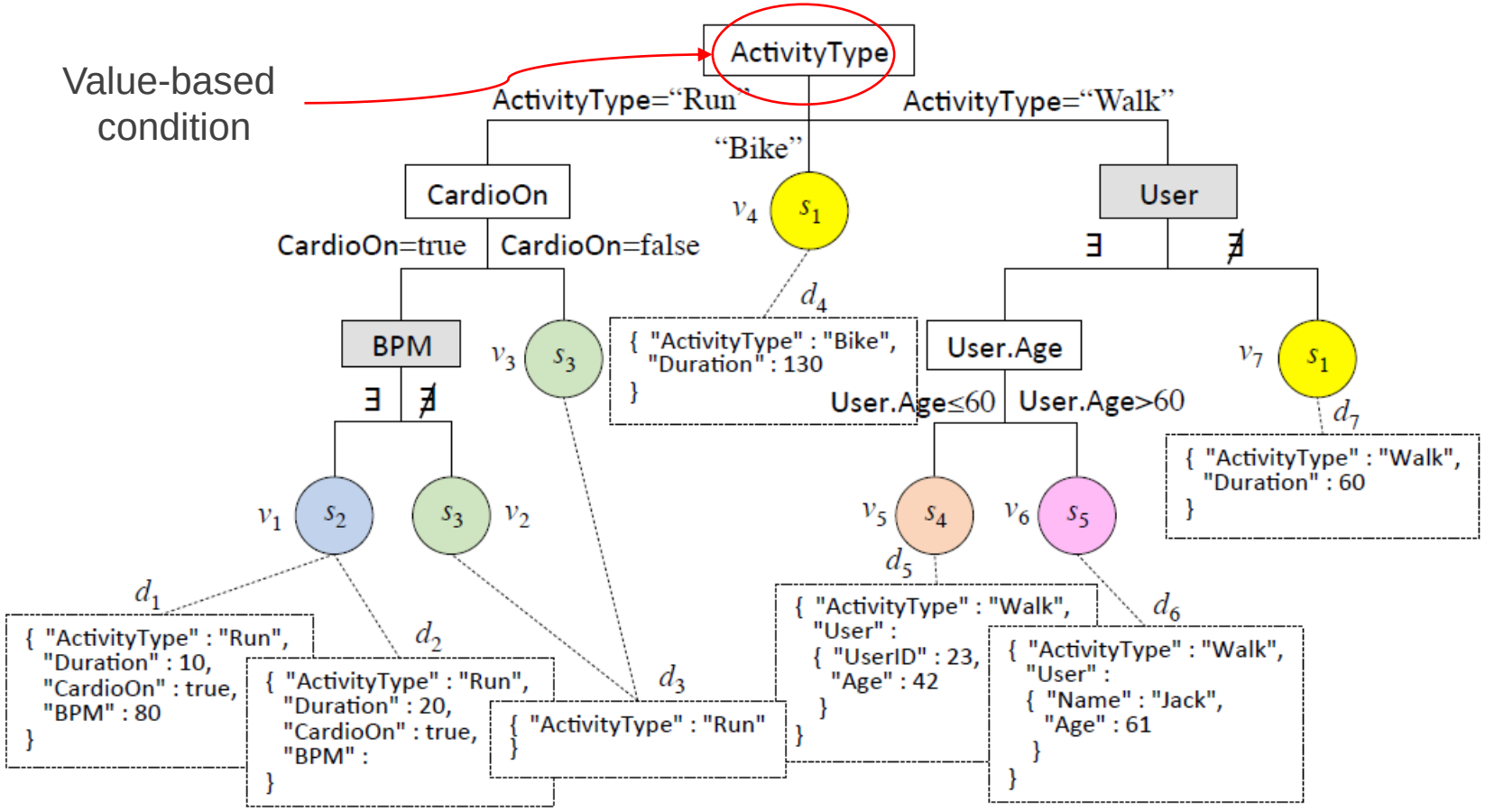
- Classifiers are also used to describe the rules for assigning a class to an observation based on the other observation features
- Based on the requirements collected from potential users, **decision trees** emerged as the most adequate

Schema profiling

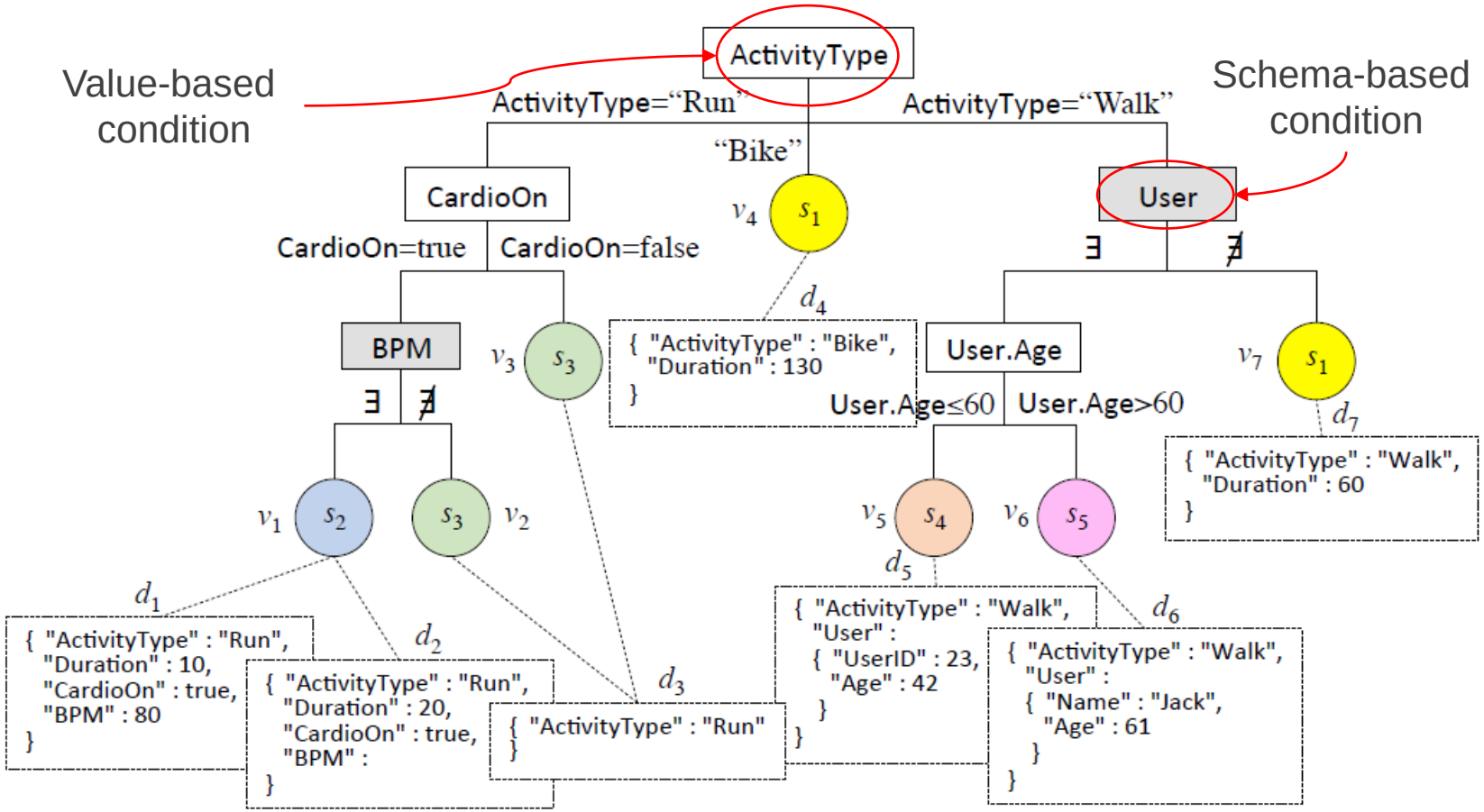


The **documents** are the observations
The **schemata** are the classes

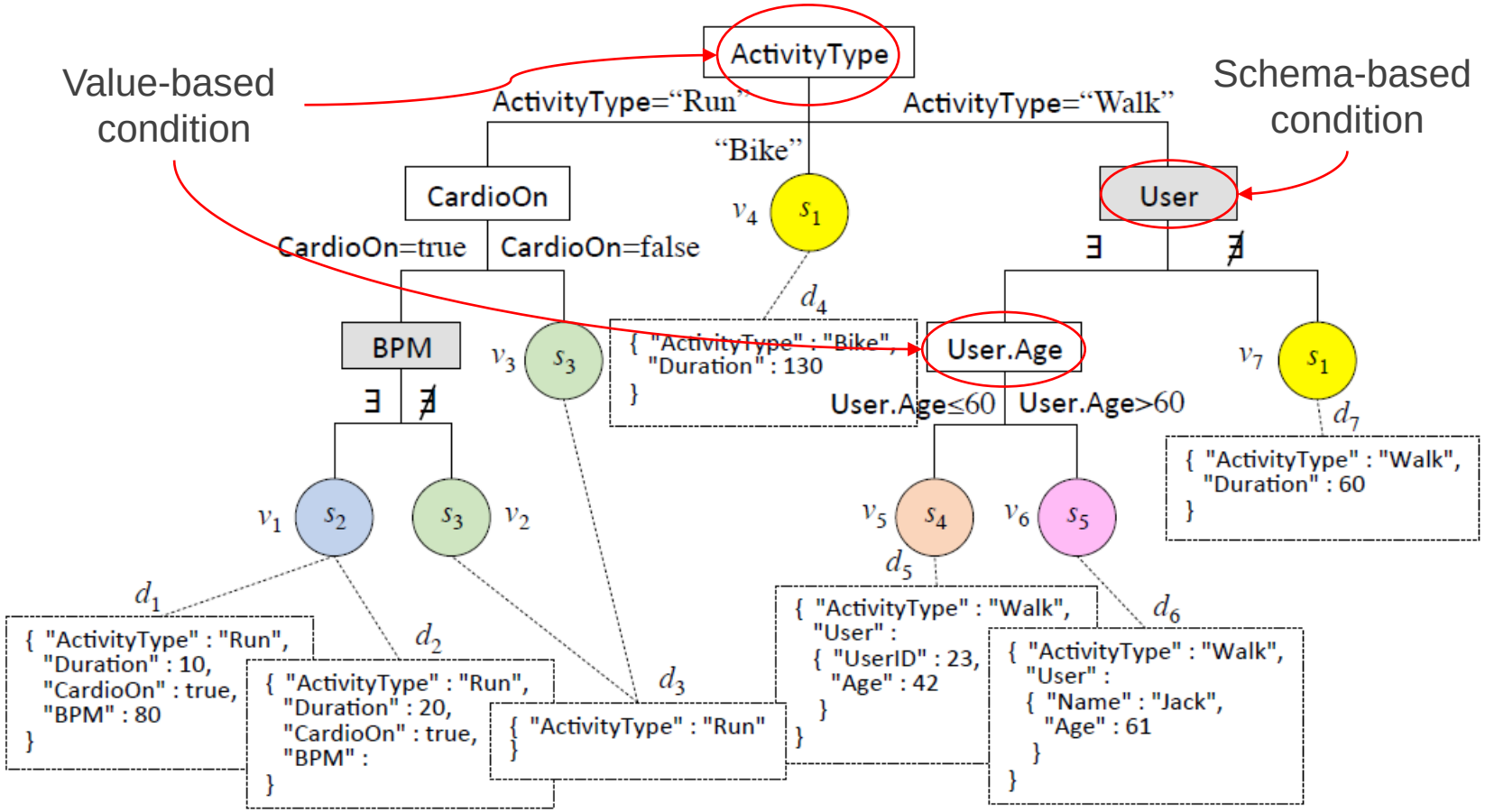
Schema profiling



Schema profiling



Schema profiling



Preliminary activities

Semi-structured interviews with 5 users

- Application domains: fitness equipment sales, software development
- Understand goals, requirements, visualization format
- Not one complete/correct dataset description

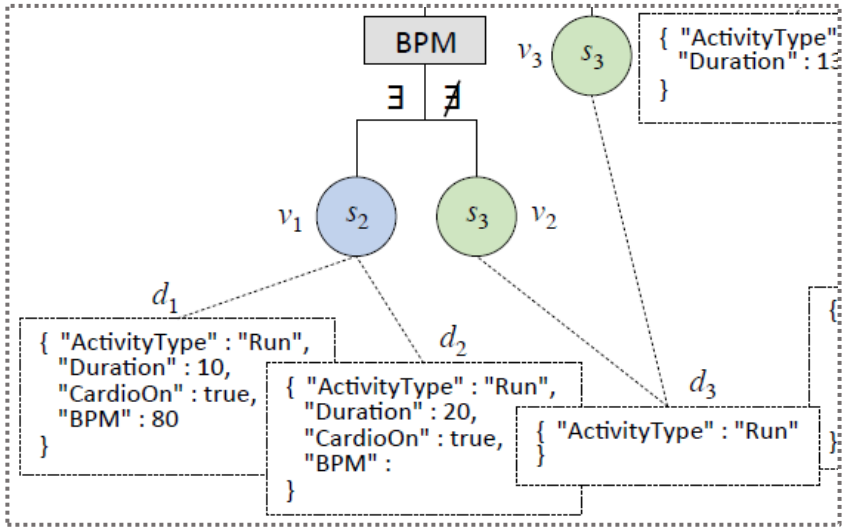
Definition of schema profile characteristics

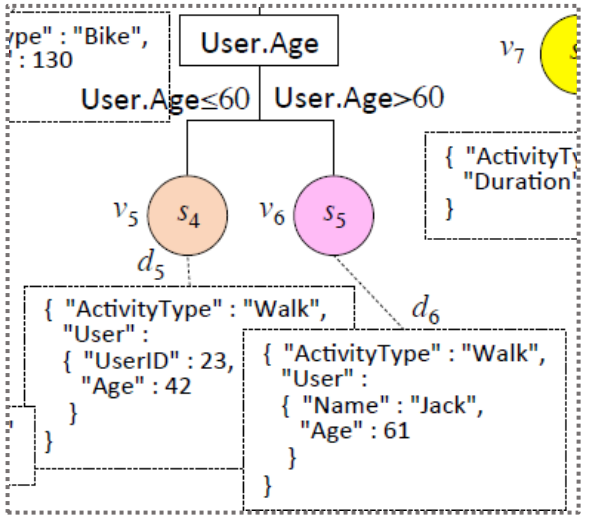
- Explicativeness
- Precision
- Conciseness

Explicativeness

Value-based (VB) conditions are preferred to schema-based (SB) ones

- SB: **acknowledge** a difference between schemata
- VB: **explain** it in terms of the values taken by an attribute





The less SB conditions, the more explicativeness

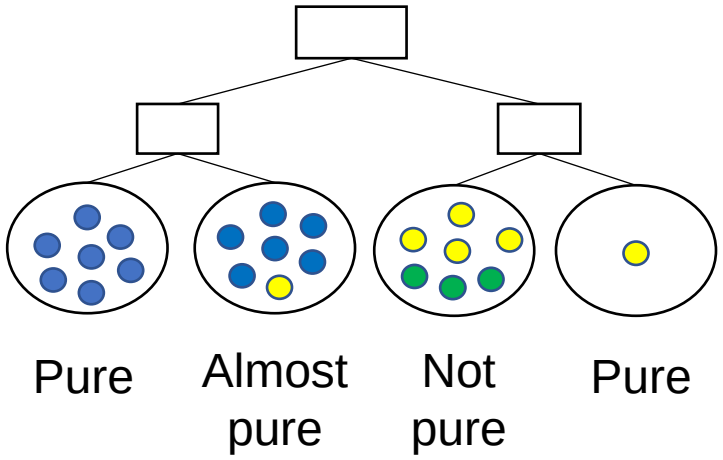
Precision

A decision tree is precise if all the leaves are pure

- A leaf is **pure** if all its observations belong to the **same class**
- Leaf v_j is pure if *entropy(v_j) = 0*

Entropy is strictly related to **precision**

- Divisive approaches typically stop only when the leaves are all pure



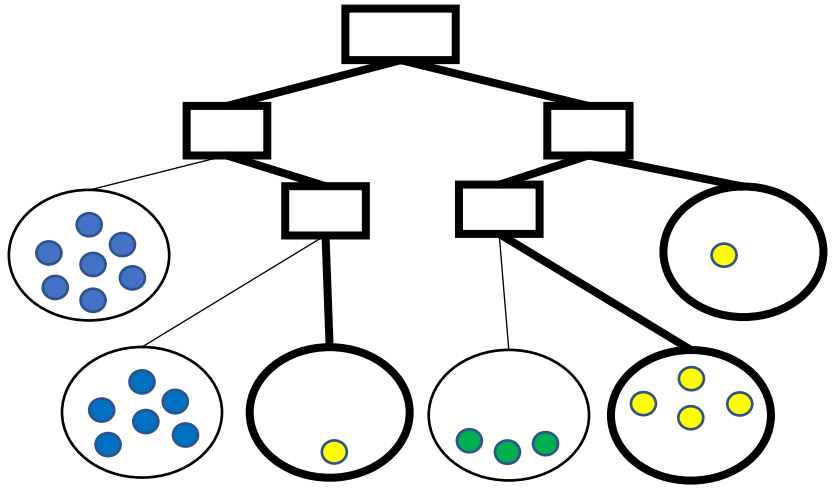
$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$$

probability of schema *s* within leaf v_j

Precision and conciseness

Minimization of entropy often leads to **splitting observations** of the same class **among several leaves**

- Entropy's sole focus is on node purity
- More frequent when the number of classes is high



Typically, precision is more important than readability

In schema profiling, this is a critical problem

- **It conflicts with the conciseness requirement**

Conciseness

A maximally concise schema profile is one where there is **a single rule for each schema**

Schema entropy: inverts the original definition of entropy, relating it to the **purity of the schemata** instead of the purity of the leaves

- Entropy:
  a leaf is pure if
  it contains only documents
  with the **same class**

$$entropy(v_j) = -\sum_{s \in S(D_v)} \frac{|D_{v_j}|_s}{|D_{v_j}|} \log \frac{|D_{v_j}|_s}{|D_{v_j}|}$$

- Schema entropy:
  a schema is pure if
  all its documents
  are in the **same leaf**

$$sEntropy(s) = \sum_{j=1}^{m} \frac{|D_{v_j}|_s}{|D|_s} \log \frac{|D_{v_j}|_s}{|D|_s}$$

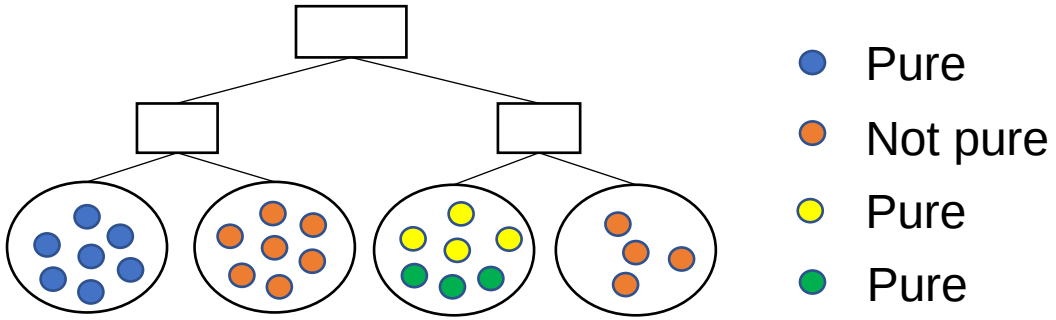
Conciseness

A maximally concise schema profile is one where there is
a single rule for each schema

Schema entropy: inverts the original definition of entropy, relating it to the **purity of the schemata** instead of the purity of the leaves

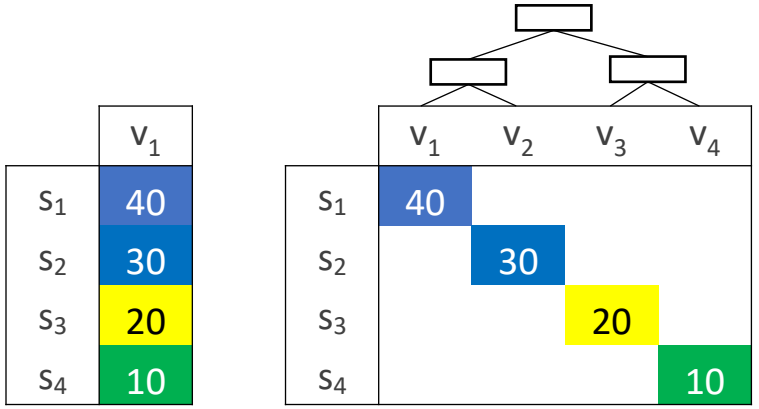
- Entropy:
  a leaf is pure if
  it contains only documents
  with the **same class**

- Schema entropy:
  a schema is pure if
  all its documents
  are in the **same leaf**



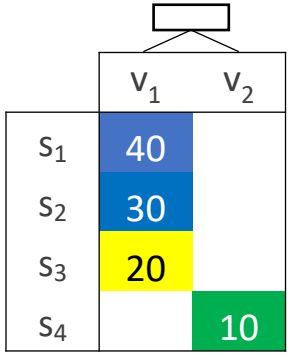
Schema profiling example

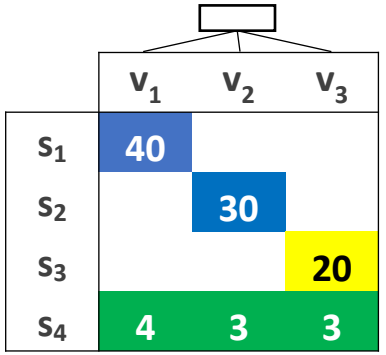
Starting situation
E = 1,85 (maximum)
SE = 0 (minimum)



Best outcome
E = 0
SE = 0

E = 1,38
SE = 0





E = 0,46
SE = 0,16

Schema profiling algorithm

Introduced the notion of *schema entropy loss*

$$loss(\sigma_a(v_j)) = sEntropy(T') - sEntropy(T)$$

Defined a criterion for comparing two splits in the decision tree

Definition 9. *Given two splits $\sigma(v)$ and $\sigma'(v)$ (possibly on different attributes), we say that $\sigma(v)$ is* better *than $\sigma'(v)$ (denoted $\sigma(v) \prec \sigma'(v)$) if either (i) $loss(\sigma(v)) < loss(\sigma'(v))$, or (ii) $loss(\sigma(v)) = loss(\sigma'(v))$ and $gain(\sigma(v)) > gain(\sigma'(v))$.*

Querying the data

One thing is understanding the data, another thing is enabling querying over heterogeneous data

What we need

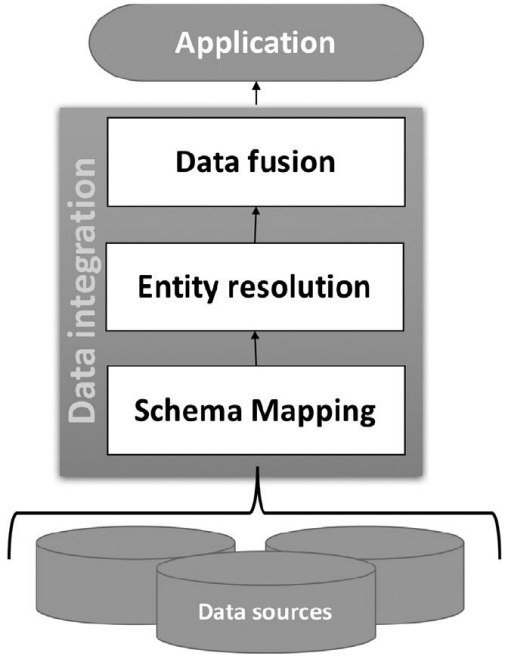
- Integration techniques to solve schema heterogeneity and produce a global knowledge
- Query rewriting techniques to translate queries on the global knowledge to queries on the actual schemas

(Focus on OLAP queries)

Integration techniques

Integration at the intensional level

- Schema matching and mapping
  - A match is a correspondence between attributes
  - A mapping is a function to explain the relationship between attributes
    - E.g., S1.FullName = CONCAT(S2.FirstName, S2.LastName)



Integration at the extensional level

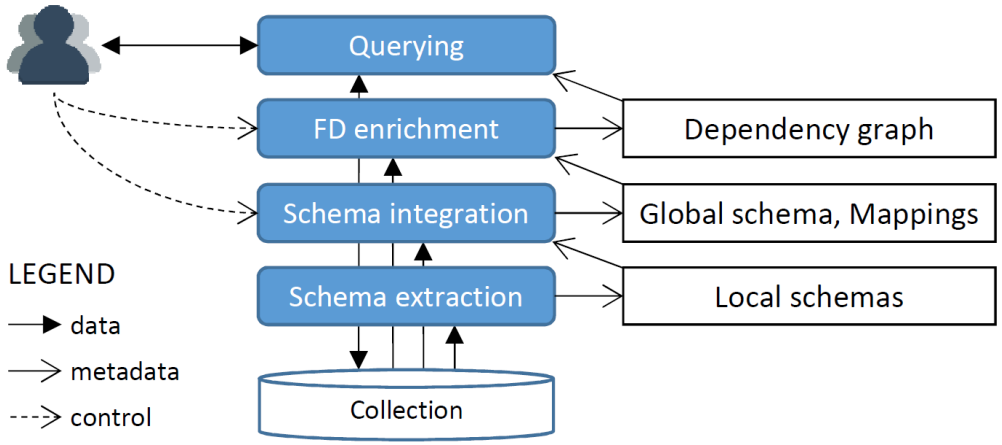
- Entity resolution (a.k.a. record linkage or duplicate detection)
  - Identifying (or linking, or grouping) different records referring to the same real-world entity
  - Aims at removing redundancy and increasing conciseness
- Data fusion
  - Fuse records on the same real-world entity into a single record and resolve possible conflicts
  - Aims at increasing correctness of data

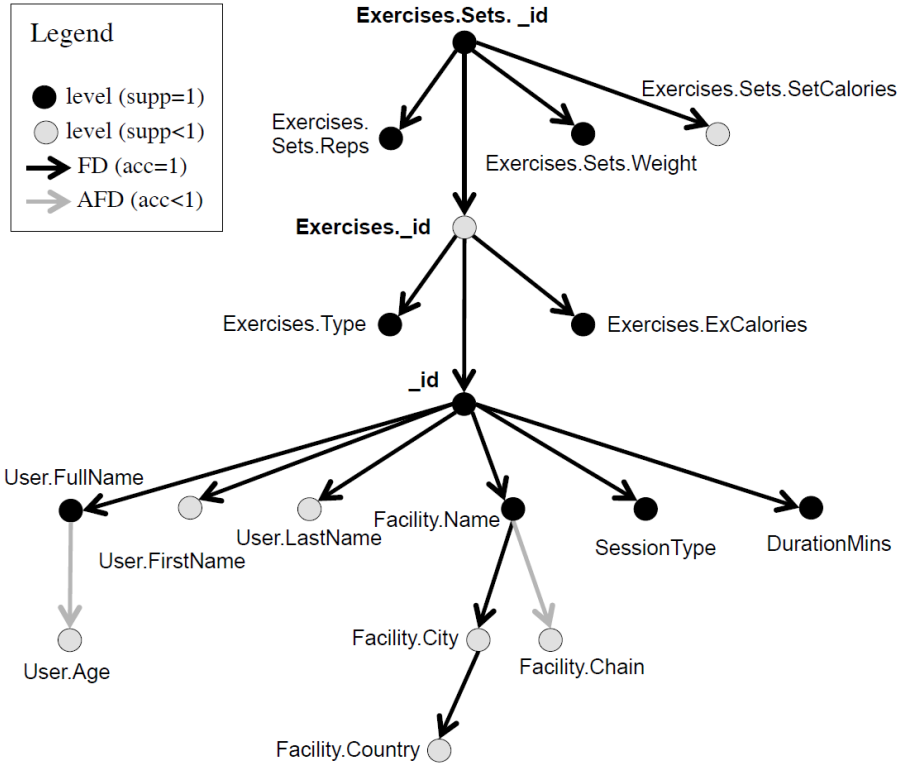
E. Rahm, P.A. Bernstein, **A survey of approaches to automatic schema matching**, *VLDB J.* 10 (4) (2001)

Mandreoli, F., & Montangero, M. (2019). **Dealing with data heterogeneity in a data fusion perspective: models, methodologies, and algorithms**. In *Data Handling in Science and Technology* (Vol. 31, pp. 235-270). Elsevier.

OLAP querying

A first approach to OLAP on heterogeneous data





Gallinucci, E., Golfarelli, M., & Rizzi, S. (2019). **Approximate OLAP of document-oriented databases: A variety-aware approach**. *Information Systems*, 85, 114-130.

OLAP querying

Some limitations

- Expensive querying
  - Does not scale well with the number of schemas
- Expensive integration
  - High levels of heterogeneity imply complex rewriting rules (requiring knowledge and time)
  - Assuming to be *always* able to obtain a global schema is a bit pretentious

OLAP querying

Some limitations

- Expensive querying
  - Does not scale well with the number of schemas
- Expensive integration
  - High levels of heterogeneity imply complex rewriting rules (requiring knowledge and time)
  - Assuming to be *always* able to obtain a global schema is a bit pretentious
    - *“One does not simply define a global schema”*

New integration techniques

Replace the global schema with a *dataspace*

- A dataspace is a lightweight integration approach providing basic query expressive power on a variety of data sources, bypassing the complexity of traditional integration approaches and possibly returning best-effort or approximate answers
  - Franklin, M., Halevy, A., & Maier, D. (2005). **From databases to dataspaces: a new abstraction for information management**. *ACM Sigmod Record*, *34*(4), 27-33.

Replace traditional integration with a *pay-as-you-go* approach

- The system incrementally understands and integrates the data over time by asking users to confirm matches as the system runs
  - Jeffery, S. R., Franklin, M. J., & Halevy, A. Y. (2008, June). **Pay-as-you-go user feedback for dataspace systems**. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 847-860).

New integration techniques

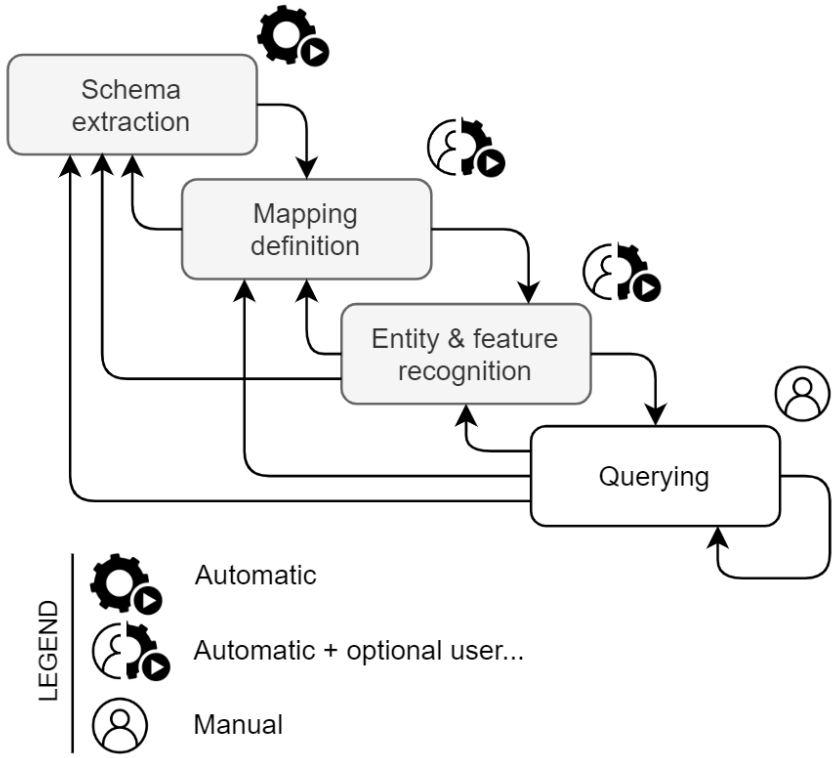
Introducing new concepts

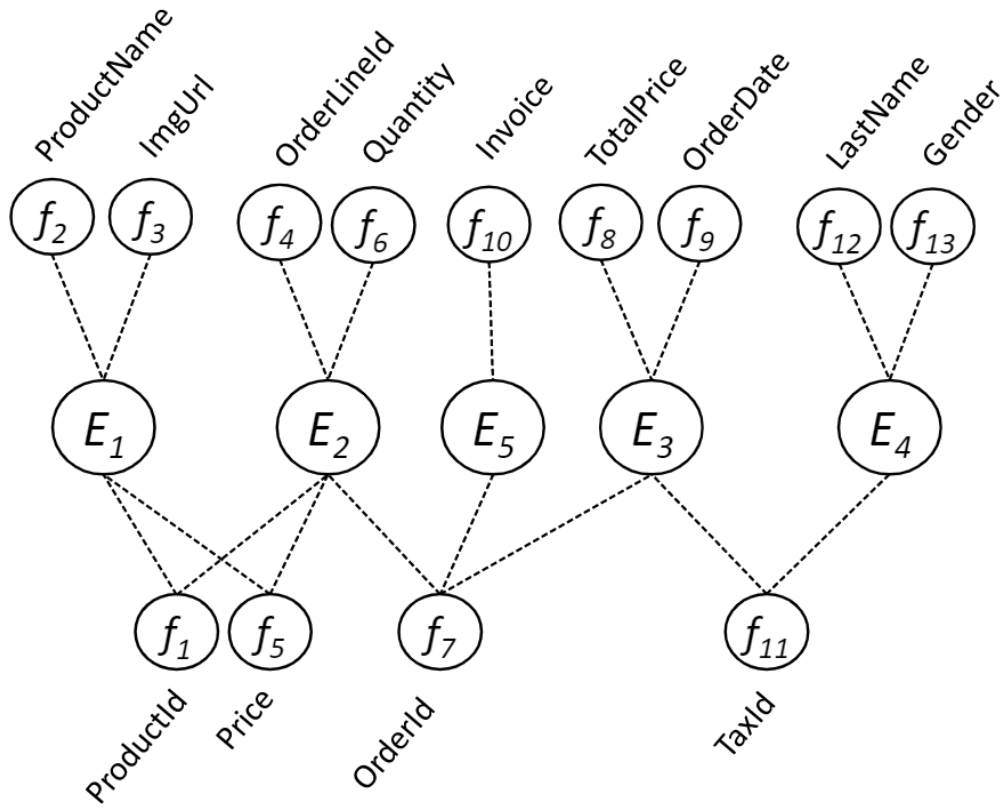
- Features: univocal representation of a group of semantically equivalent attributes
  - E.g., CustomerName = { S1.name, S2.fullname, S3.customer, S4.cName, … }
  - Mapping functions must be defined/definable between every couple
- Entities: representation of a real-world entity
  - E.g., customers, products, orders, etc.

The dataspace becomes an abstract view in terms of features and entities

New OLAP querying

What it looks like



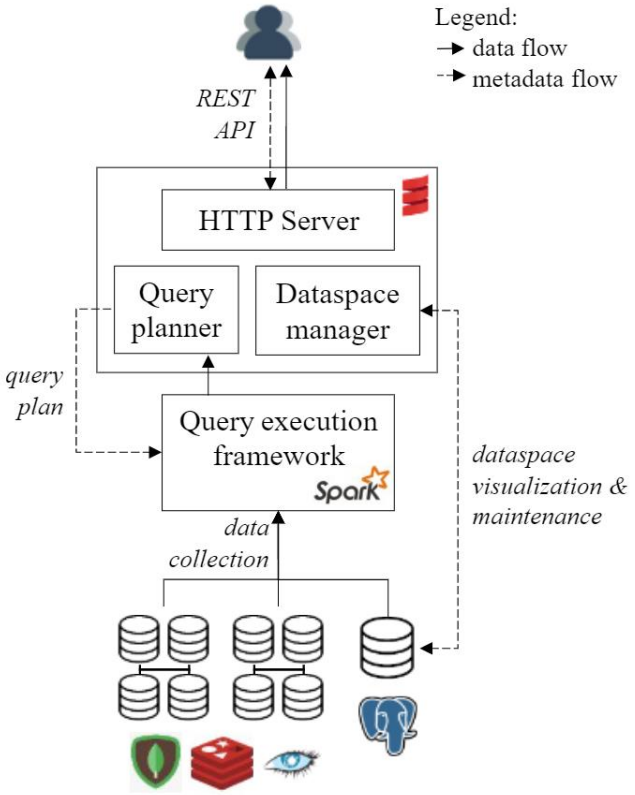


Forresi, C., Gallinucci, E., Golfarelli, M., & Hamadou, H. B. (2021). **A dataspace-based framework for OLAP analyses in a high-variety multistore**. *The VLDB Journal*, *30*(6), 1017-1040.

New OLAP querying

Previous issues

- ~~Expensive querying~~
  - Schema heterogeneity solved at query time
  - Requires complex - but feasible - algorithms
- ~~Expensive integration~~
  - Pay-as-you-go approach is quicker, iterative, and more flexible
  - Dataspace is conceptual, untied to logical data modeling



Now we have a multistore dealing with multiple data models and schema heterogeneity

Forresi, C., Gallinucci, E., Golfarelli, M., & Hamadou, H. B. (2021). **A dataspace-based framework for OLAP analyses in a high-variety multistore**. *The VLDB Journal*, *30*(6), 1017-1040.

Data inconsitency

Intra-collection

- Due to denormalized data modeling

Inter-collection

- Due to analytical data offloading
  - To reduce costs and optimize performance, the historical depth of databases is kept limited
  - After some years, data are offloaded to cheaper/bigger storages, e.g., cloud storages, data lakes
  - Offloading implies a change of data model, a change of schema, and obviously an overlapping of instances with the original data
- Due to multi-cloud architectures
  - Enables the exploitation of data spread across different providers and architectures, all the while overcoming data silos through data virtualization
  - Typical in presence of many company branches

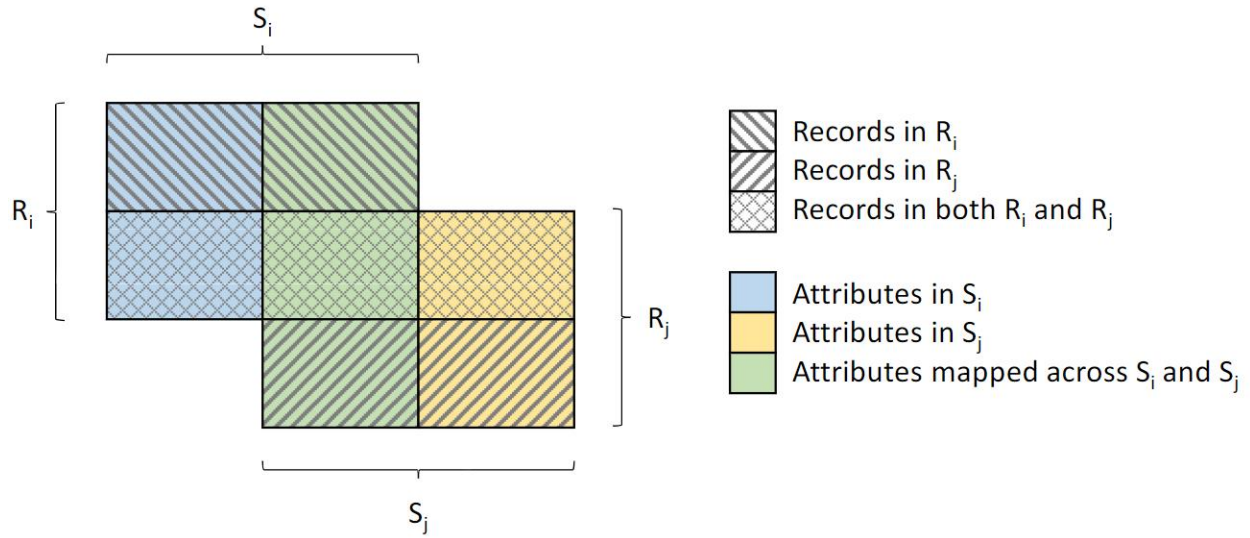
Solutions?

- Traditional ETL
- Solve inconsistencies on-the-fly

Data fusion

Merge operator

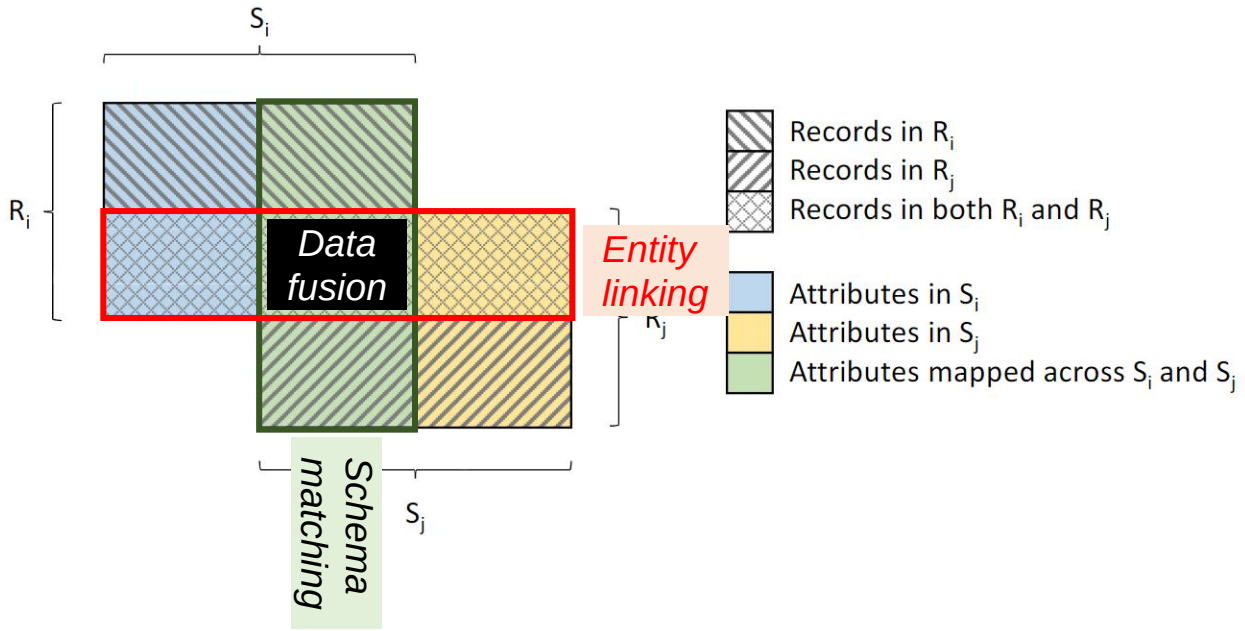
- Originally introduced as “full outer join merge”
  - Naumann, F., Freytag, J. C., & Leser, U. (2004). **Completeness of integrated information sources**. *Information Systems*, *29*(7), 583-615.
- Aims to keep as much information as possible when joining the records of two schemas
  - Avoid any loss of records
  - Resolve mappings by providing transcoded output
  - Resolving conflicts whenever necessary



Data fusion

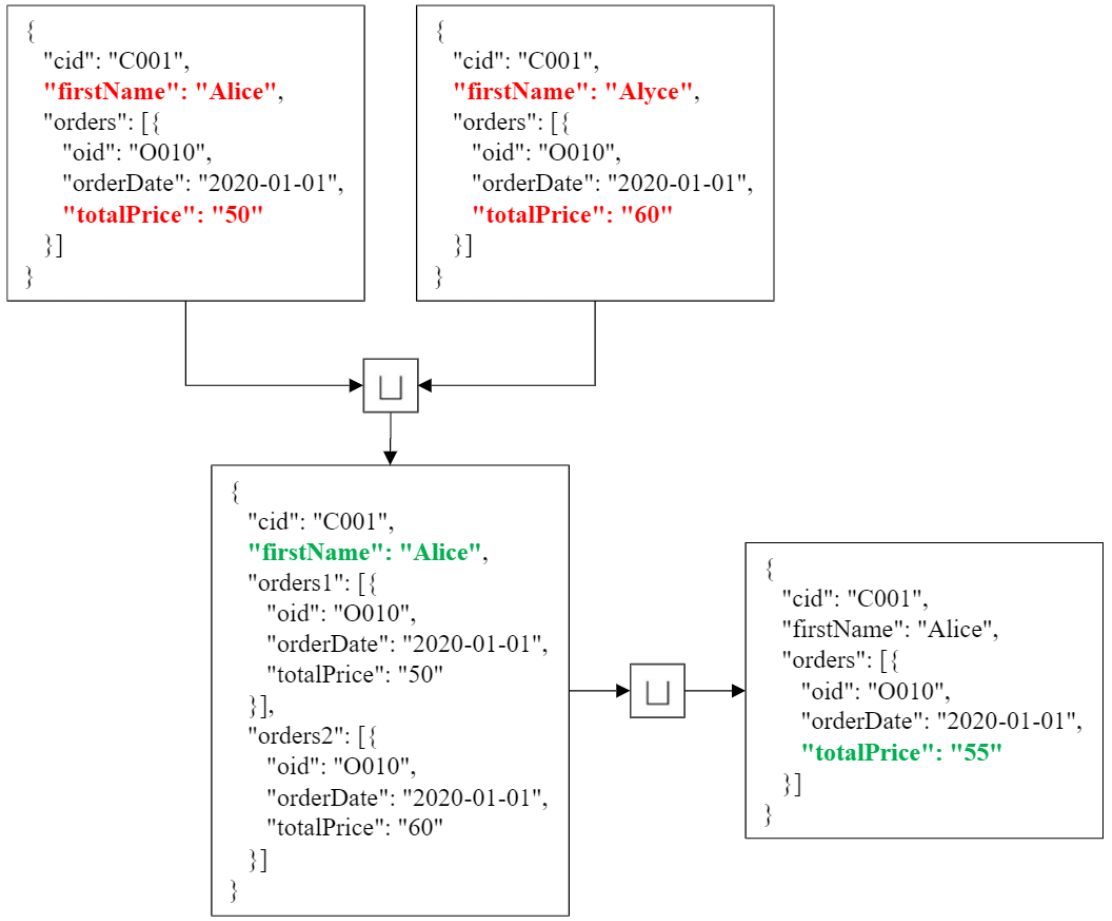
Merge operator

- Originally introduced as “full outer join merge”
  - Naumann, F., Freytag, J. C., & Leser, U. (2004). **Completeness of integrated information sources**. *Information Systems*, *29*(7), 583-615.
- Aims to keep as much information as possible when joining the records of two schemas
  - Avoid any loss of records
  - Resolve mappings by providing transcoded output
  - Resolving conflicts whenever necessary



Data fusion

Merge operator



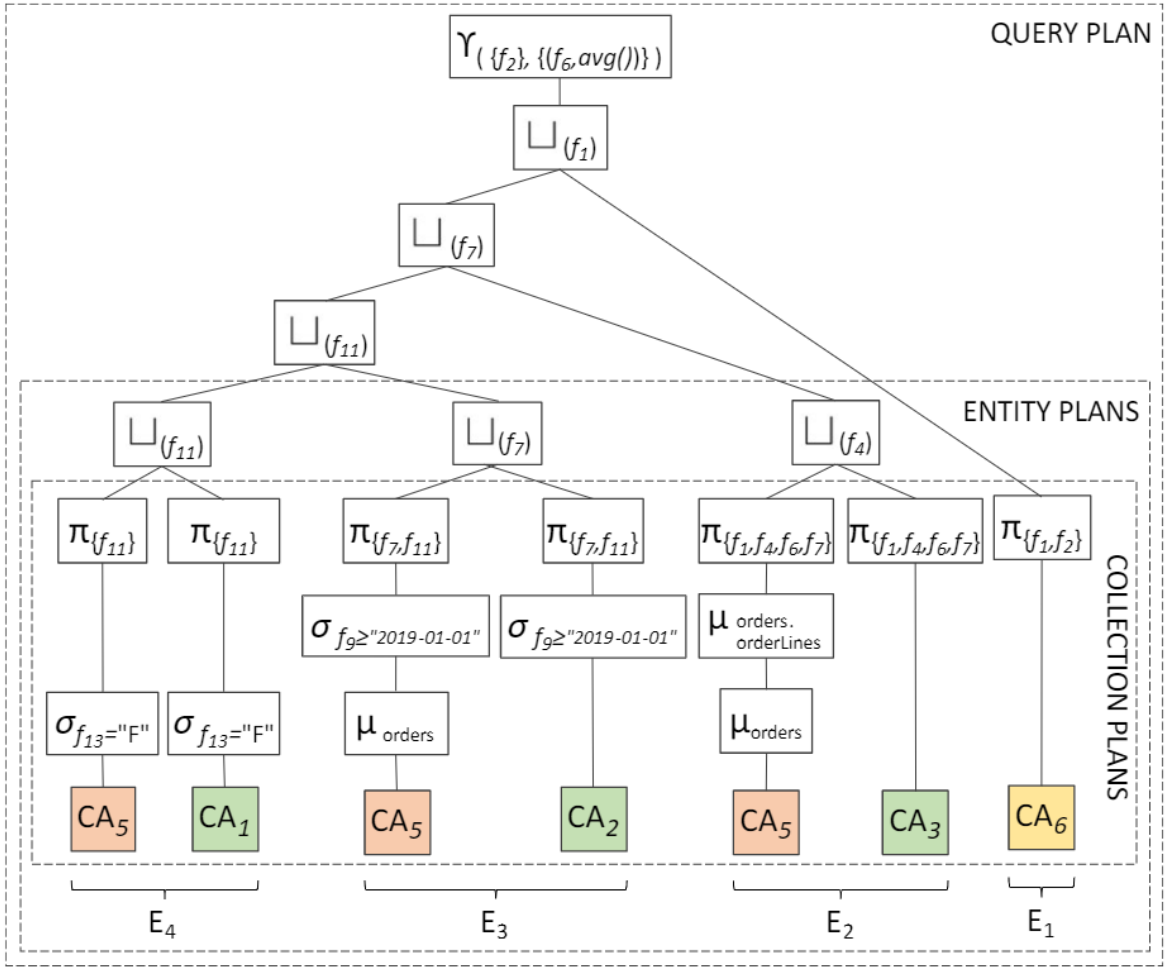
On-the-fly data fusion

Merge operator in a query plan

- Take the data from heterogeneous sources (in different colors)
- Extract records of the single entites (e.g., customer, products)
- Merge each entity
- Join and produce the final result

Now we have a multistore dealing with multiple data models, schema heterogeneity, and data inconsistency

- Are we done? Not yet!



On-the-fly data fusion

Main issue: performance

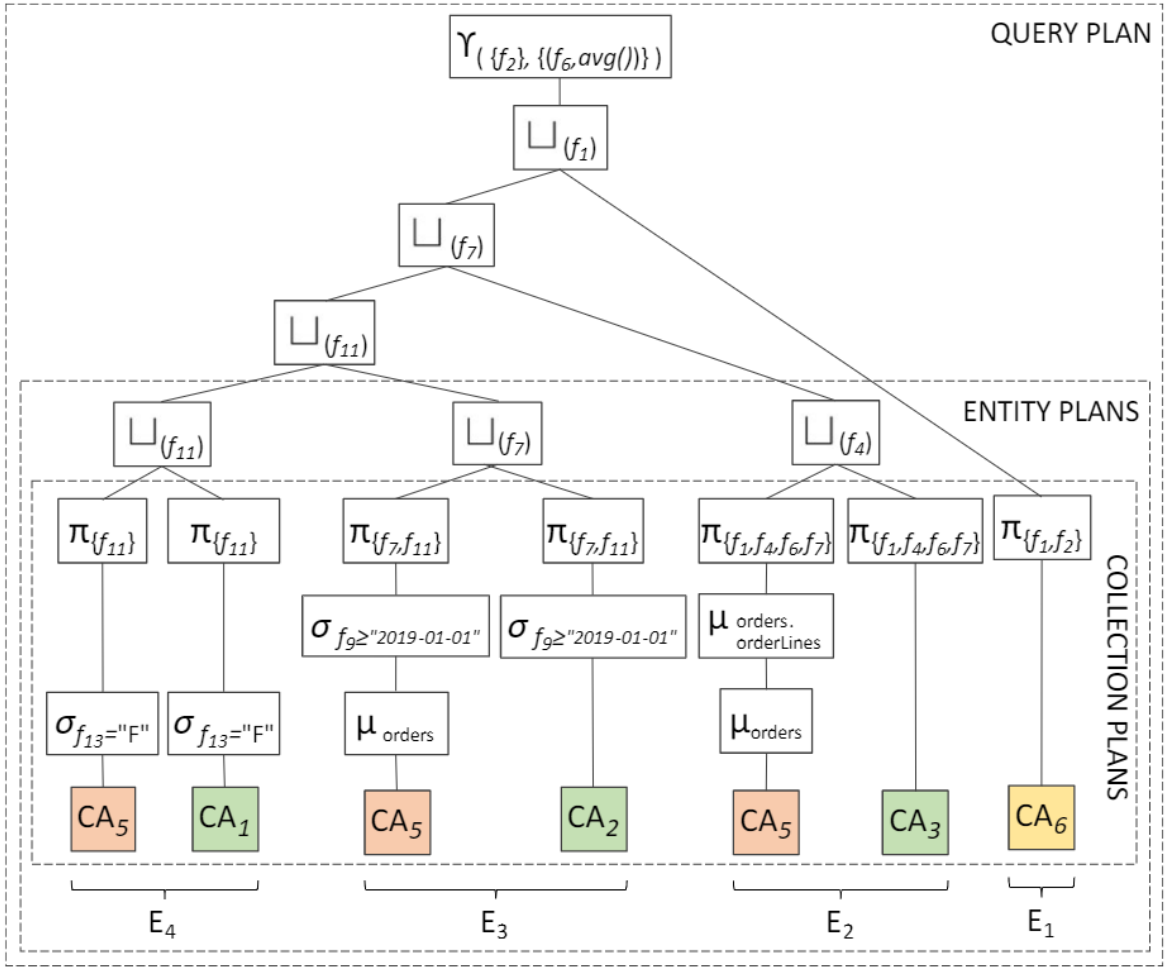
- Collections accessed more than once
- Most effort pulled to the middleware

What can we do about it?

- Exploit more the local DBMSs
- Exploit local data modelling
- Carry out multi-entity merges

Issues

- Several query plans could be devised
- Hard to find the most efficient one



Logical optimization

Logical rules to transform a query plan into a more efficent one

- Predicate push-down: applying selection predicates as close to the source as possible
  - Not always feasible (e.g., in presence of inconsistent data)
- Column pruning: extracting the only attributes relevant for the query
  - Not for granted when writing a custom query language
- Join sequence reordering: changing the order to do binary joins
  - Not so easy when merges are involved as well
  - Not so easy when data comes from different sources

Same query, several query plans

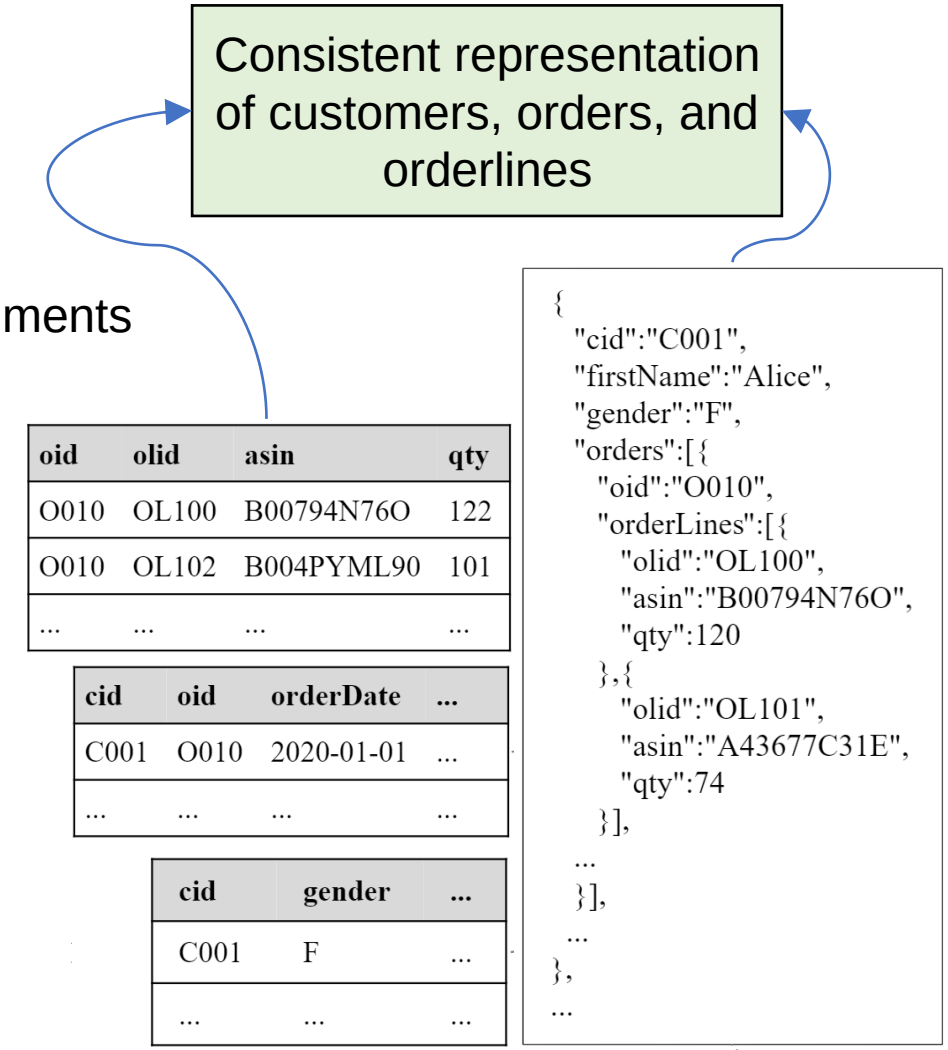
What is the most efficient solution?

- Single-entity merge and subsequent joins
- Nest relational data and multi-merge with documents
- Join relational data and multi-merge with flattened documents

Depends on several factors

- On the capabilities of each DBMS/middleware
- On the presence of indexes and statistics
- On the resources available to each DBMS/middleware
- On the number of records involved on each side

..which can change over time



Cost modelling

Cost-based evaluation of different plans

- White-box cost modelling
  - Associate theoretical formulas to each query operators, then build up the cost of a query by summing the cost of each operation
  - Cost can be determined in terms of disk I/O, CPU, network
  - Requires an enormous effort to effectively model the many factors that contribute to query costs in a complex and heterogeneous environment like a multistore
- Black-box cost modelling
  - Hide the behavior of an execution engine within a black-box, where the known information is mostly limited to the issued queries and the given response times
  - Cost is determined in terms of time
  - Easily adapts to evolving environments
  - Suffers from cold-start

Cost modelling

White-box cost modelling example

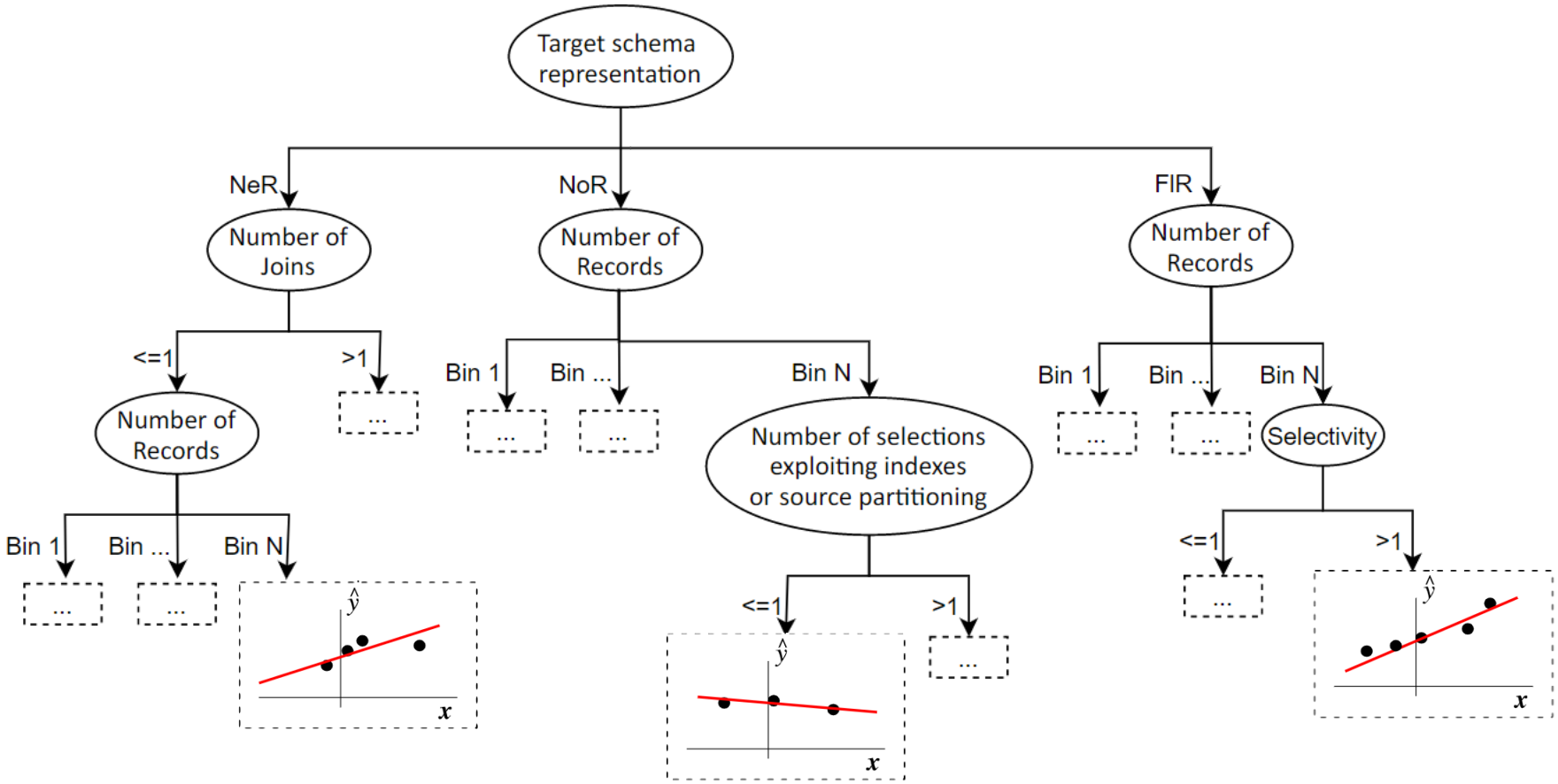
Parameter/Function	Description
$NR(C)$	Number of records in C
$NNR(C)$	If C is nested, number of nested records
$Len(C)$	Average byte length of a record in C
DPS	Size of a disk page
$PT(C)$	Number of partitions into which records in C are organized
NB	Number of memory buffers
$NP(C)$	Number of disk pages occupied by C: $\lceil \frac{NR(C) \cdot Len(C)}{DPS} \rceil$
$Part(C, nPart)$	Number of disk pages occupied by one of $nPart$ partitions of C

Operation	Description	Estimated cost	Sup.
$CA(C)$	Collection access	$NP(C)$	RMCS
$\pi(C)$	Projection	$NP(C)$	RMCS
$\gamma(C)$	Aggregation	$Sort(C) + NP(C)$	RM*S
$v(C)$	Nest	$Sort(C) + NP(C)$	RM-S
$\mu(C)$	Unnest	$NP(C)$	RM-S
$\overline{\mu}(C)$	Simult. unnest	$NR(C) \cdot SM(Part(C, \lceil \frac{NNR(C)}{NR(C)} \rceil))$	---S
$\overline{\cup}(C)$	Array union	$NP(C)$	RM-S
$(C_1) \bowtie (C_2)$	Join	$min(NLJ(C, C'),\ SMJ(C, C'),\ HJ(C, C'))$	RM-S
$(C_1) \sqcup (C_2)$	Merge	$min(NLJ(C, C'),\ SMJ(C, C'),\ HJ(C, C'))$	RM-S
$(C_1) \cup (C_2)$	Union	$NP(C) + NP(C')$	RM-S
$Shf(C)$	Data shuffle	$3 \cdot NP(C)$	-M-S
$SM(C)$	Sort-Merge	$2 \cdot NP \cdot (\lceil log_{NB-1} NP \rceil + 1)$	RM-S
$Sort(C)$	Data sort (central.)	$SM(C)$	RM--
	Data sort (distrib.)	$Shf(R) + PT(C) \cdot SM(Part(C, PT(C)))$	---S
$HJ(C, C')$	Hybrid hash Join	$3 \cdot (NP(C) + NP(C'))$	R---
$NLJ(C, C')$	Nested Loops Join	$NP(C) + NR(C) \cdot NP(C')$	R---
		$NP(C) + NR(C) \cdot NP(C') + Unnest(C'')$	-M--
$SMJ(C, C')$	Sort-Merge Join	$Sort(C) + Sort(C') + NP(C) + NP(C')$	R--S

Golfarelli, M. (2021, August). **Optimizing Execution Plans in a Multistore**. In *Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021*, Tartu, Estonia, August 24–26, 2021, Proceedings (Vol. 12843, p. 136). Springer Nature.

Cost modelling

Black-box cost modelling example



(work in progress)

Polyglot persistence - Conclusions

Two main issues

- Query performance
  - Improving caching/indexing techniques
  - Improving cost model effectiveness
- Data integration
  - Improving effectiveness (ever-lasting direction)
  - Improving efficiency on big data scales
    - Reduce the number of comparison, introduce approximation
    - Parallelize computation
    - Exploit additional information (e.g., temporal entity resolution models)
  - Human-in-the-loop (pay-as-you-go, crowdsourcing)
  - Frictionless integration within a big data platform
    - A metadata challenge that we explore tomorrow!

Mandreoli, F., & Montangero, M. (2019). Dealing with data heterogeneity in a data fusion perspective: models, methodologies, and algorithms. In *Data Handling in Science and Technology* (Vol. 31, pp. 235-270). Elsevier.