Information Discovery in Polystores: the Augmented Way (Discussion Paper)

Antonio Maccioni[1] and Riccardo Torlone[2]

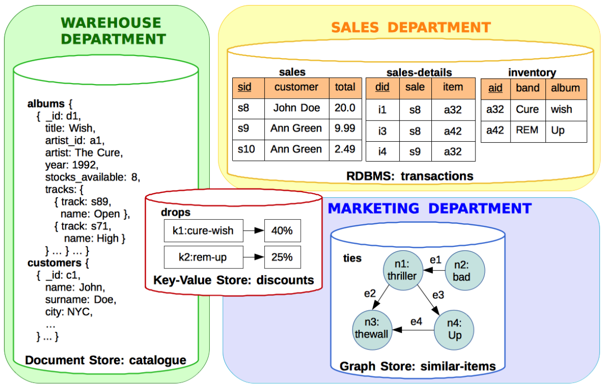
[1] Collective[i], New York City, USA
amaccioni@collectivei.com
[2] Roma Tre University, Rome, Italy
riccardo.torlone@uniroma3.it

Abstract. Polystores provide a loosely coupled integration of heterogeneous data sources based on the direct access, with the local language, to each storage engine for exploiting its distinctive features. In this framework, given the absence of a middleware exposing a global schema, it is hard to know if a query to one system can be satisfied by data stored elsewhere in the polystore. We address this problem by illustrating *query augmentation*, a data manipulation operator for polystores based on the automatic enrichment of the answer to a local query with related data in the rest of the polystore. Augmentation can be used to implement *augmented search* and *augmented exploration*: two effective methods for information discovery in polystores that avoid middleware layers, abstract query languages, and shared data models.

1 Introduction

The concept of polyglot persistence, which consists of using different database technologies to handle different data storage needs [9], is spreading within enterprises. Recent research has shown that, on average, each enterprise application relies on at least two or three different types of database engines.

Example 1. Let us consider a company called *Polyphony* selling music online. As shown in Fig. 1, each department uses a storage system that best fits its specific business objectives: (i) the sales department guarantees ACID properties for its transactions database with a relational system, (ii) the warehouse department supports its activities with a document store catalogue, where each item is represented by a JSON document, and (iii) the marketing department uses a graph database of similar-items supporting recommendations. In addition, a key-value store containing discounts on products is shared among the above departments.



```
albums {
  { _id: d1,
    title: Wish,
    artist_id: a1,
    artist: The Cure,
    year: 1992,
    stocks_available: 8,
    tracks: {
      { track: s89,
        name: Open },
      { track: s71,
        name: High }
    } ... } ... }
customers {
  { _id: c1,
    name: John,
    surname: Doe,
    city: NYC,
    ...
  } ... }
```

sales

sid	customer	total
s8	John Doe	20.0
s9	Ann Green	9.99
s10	Ann Green	2.49

sales-details

did	sale	item
i1	s8	a32
i3	s8	a42
i4	s9	a32

inventory

aid	band	album
a32	Cure	wish
a42	REM	Up

Fig. 1. A polyglot environment.

This approach, however, makes even harder the problem of data integration given the strong heterogeneity of the systems in play [4]. The traditional solution to this issue is based on a middleware layer involving a unified language, a common interface, or a universal data model [10]. However, the addition of this layer to the software stack adds computational overhead at runtime and, more importantly, hides the specificity and functionality for which these systems were adopted. In addition, it is hard to maintain, having an inherent complexity that increases significantly as new database systems take part to the environment.

Polystore systems (or simply, polystores) have been proposed recently as an alternative solution for this approach [12]. The basic idea is to provide a loosely coupled integration of data sources and allow the direct access, with the local language, to each specific storage engine to exploit its distinctive features. This solution meets the requirements of scenarios in which users are often aware of a single (or a few) database only but does not know anything about other databases (neither the content, nor the way to query them and, sometimes, not even their existence). On the other hand, it poses new challenges for accessing and combining all the available data. To recall a discussion about this approach, the issue is that *"if I knew what query to ask, I would ask it, but I don't"* [12].

In this paper, we present (*query*) *augmentation*, a new construct for data manipulation in polystores that, based on a simple notion of probabilistic relationship between objects in different data stores, allows the automatic enrichment of a dataset stored in a local database with data outside the database but available in the polystore. The implementation of this operator does not require the addition of an abstraction layer involving query translation and therefore has a minimal impact on the applications running on top of the data layer. The goal is to provide a soft mechanism for data integration in polystores that complements other approaches, such as those based on cross-db joins [3].

Two effective methods for data access in polystore can be defined with the augmentation construct: *augmented search* and *augmented exploration*.

Augmented search is a form of query relaxation [7] and consists of the automatic expansion of the result of a query over a local database with data that is relevant to the query but is stored elsewhere in the polystore. This is useful in scenarios where information is shared across the organization and the various databases complement or overlap each other. Assume for instance that Lucy, an employee of Polyphony working in the sales department, only knows SQL but needs all the information available on the album “Wish”. Then, she just needs to submit, in *augmented* mode, an SQL query over the database transactions in Fig. 1. By exploiting augmentation, the result of the query is the augmented object reported below, revealing details on the product that are not in the database of the sales department, including the fact that it is currently on a 40% discount.

$$\begin{tikzcd}
\text{< a32, Cure, Wish >} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \begin{array}{l}\text{(catalogue:\{ title: Wish,} \\ \quad\text{artist\_id: a1,} \\ \quad\text{artist: The Cure,} \\ \quad\text{year: 1992,} \\ \quad\text{... \})}\end{array} \\
\text{(discounts: 40\%)} &
\end{tikzcd}$$

In an augumented search, each retrieved element e is associated with the probability that e is related to the result of the query. Such probability is derived off-line from integrity constraints and data mining techniques. Colors (as in the example above) and rankings can be used in practice to represent, in a more intuitive way, external data and the degree of relevance.

Augmented exploration makes use of the augmentation operator to provide a more interactive and flexible way to access data, which consists of a guided expansion of the result of a local query. For example, if Lucy submits in *exploratory* mode another SQL query to the database transactions for retrieving all the sales whose total income is greater than 15, she obtains the tuple that follows, in which the links suggest that further information, related to the returned tuple, is available elsewhere and allows her to decide with a click (e.g., on the user name) how to deepen the result. As above, the result is probabilistic.

$$\begin{tikzcd}
\text{<\underline{s8}, \underline{John Doe}, \underline{20.0} >} \arrow[r, dashed, "on-click"] & \begin{array}{l}\text{(\{ \_id: c1,} \\ \quad\text{name: John,} \\ \quad\text{surname: Doe,} \\ \quad\text{city: NYC,} \\ \quad\text{... \})}\end{array}
\end{tikzcd}$$

This process is iterative and provides a method for database exploration [2], where the user can freely find her way through the polystore, by just clicking on the links as soon as they are made available.

We have implemented our approach in QUEPA[3], a polystore system equipped with the augmentation operator that provides the access methods discussed above and is compatible with most modern database engines. QUEPA operates in a plug-and-play mode and does not affect the access modalities of the various storage systems, thus reducing the need for ad-hoc configurations and for a middleware involving unified query languages or shared data models.

[3] A demo of QUEPA has been shown in [5].

In addition, we have developed an optimization technique that relies on machine learning methods to tune the execution of the augmentation operator.

In the rest of the paper we briefly describe the underlying data model (Section 2), the augmentation mechanism (Section 3), and its implementation (Section 4). Details and experimental results are in the full version of the paper [6].

2 A data model for polystores

In PDM (Polystore Data Model) a polystore $\mathcal{P}$ is made of a set of databases $\mathcal{P} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ stored in a variety of data management systems $\mathcal{S}_1, \ldots, \mathcal{S}_n$ respectively (relational, key-value, graph, etc.). A database $\mathcal{D} \in \mathcal{P}$ consists of a set of *data collections* each of which is a set of *(data) objects*. An object $o \in C$ is just a key-value pair: $o = \langle k, v \rangle$ where k identifies uniquely o in C and v is an atomic piece of data. A tuple and a JSON document are examples of data objects in a relational database and in a document store of a polystore, respectively.

By definition, given a database $\mathcal{D}$ of a polystore $\mathcal{P}$, a data collection C in $\mathcal{D}$ and a data object $o = \langle k, v \rangle$ in C, we can uniquely identify o in $\mathcal{P}$ by means of k, C and $\mathcal{D}$. We call $\hat{k} = \mathcal{D}.C.k$ the *global-key* of o in $\mathcal{P}$.

Example 2. In the polystore in Figure 1 $\langle \mathsf{d1}, \{\mathsf{\_id} : \mathsf{d1}, \mathsf{title} : \mathsf{Wish}, \ldots\} \rangle$ is an object of the albums collection in the catalogue database whereas $\langle \mathsf{s8}, (\mathsf{s8}, \mathsf{John\ Doe}, 20.0) \rangle$ is an object of the sales collection in the transactions database. The global-key of the latter is transactions.sales.s8.

This model captures polystores involving any database system satisfying the minimum requirement that every stored data object can be identified and accessed by means of a key. Note that the granularity of objects inevitably depends on the designer's choice [1].

The other main ingredient of PDM is the ability to correlate data objects of possibly different databases of the polystore by means of the following basic notion, which we call *p-relation* (for relation in a polystore).

Definition 1 (p-relation). *A p-relation on two objects o_1 and o_2, denoted by $o_1 R_p o_2$, represents the existence of a relation R between o_1 and o_2 with probability p ($0 < p \leq 1$), where R can be one of the following types:*

- *the* identity*, denoted by $\sim$: a reflexive, symmetric and transitive relation (i.e., an equivalence relation), representing the fact that o_1 and o_2 refer to the same real-world entity;*
- *the* matching*, denoted by $\rightleftharpoons$: a reflexive and symmetric relation (not necessarily transitive), representing the fact that o_1 and o_2 share some common information.*

We assume that in a polystore p-relations are "consistent" in the sense that they satisfy the following condition: for each triple of objects o_1, o_2 and o_3 such that $o_1 \rightleftharpoons o_2$ and $o_2 \sim o_3$ it is the case that $o_1 \rightleftharpoons o_3$. While this is a natural condition (two equivalent objects should match the same objects), it guarantees that the augmentation construct behaves consistently with equivalent objects.

Example 3. Consider the polystore in Fig. 1. By denoting the objects with their global keys we have for instance that: catalogue.albums.d1 $\sim_{0.8}$ discount.drop.k1:cure:wish, and transactions.inventory.a32$\rightleftharpoons_1$transactions.sales-details.i4.

Basically, while the identity relation serves to represent multiple occurrences of the same entity in the polystore, the matching relation models general relationships between data different from the identity (e.g., those typically captured by foreign keys in relational databases or by links in graph databases). On the practical side, p-relations are derived from the metadata associated with databases in the polystore (e.g., from integrity constraints) or are discovered using probabilistic mining techniques. For the latter task, we rely on the state-of-the-art techniques for probabilistic record linkage [8], that is, algorithms able to score the likelihood that a pair of objects in different databases match.

3 Augmentation in polystores

The augmentation construct. The basic operator of our approach takes as input an object o of a polystore and returns the augmented set $\alpha^n(o)$, which iteratively returns data objects in the polystore that are related to o with a certain probability. This probability is computed by combining the probabilities of the relationships that connect o with the retrieved objects.

Definition 2. *Let $\boldsymbol{o}$ be a set of objects in a polystore $\mathcal{P}$. The* augmentation α^n of level $n \geq 0$ *of $\boldsymbol{o}$ is a set $\boldsymbol{o}'$ of objects o^p, where $o \in \mathcal{P}$ and p is the probability of membership of o to $\boldsymbol{o}'$, defined as follows* $(m > 0)$*:*

- $\alpha^0(\boldsymbol{o}) = \boldsymbol{o} \cup \{o^p \mid o \sim_p o' \wedge o' \in \boldsymbol{o}\}$
- $\alpha^m(\boldsymbol{o}) = \alpha^{m-1}(\boldsymbol{o}) \cup \{o^{\hat{p}} \mid (o \rightleftharpoons_{p'} o') \wedge (o'^p \in \alpha^{m-1}(\boldsymbol{o})) \wedge (\hat{p} = p \cdot p')\}$

Example 4. Let us consider the polystore in Fig. 2 in which nodes represents objects, ovals represent different databases, solid lines represent identity p-relations, and dashed lines represent matching p-relations. Labels associated with p-relations denote their probability. Then, we have $\alpha^0(\{o_4\}) = \{o_4, o_1^{0.9}\}$, $\alpha^1(\{o_4\}) = \{o_4, o_1^{0.9}, o_7^{0.8}, o_8^{0.7}\}$, and $\alpha^3(\{o_4\}) = \{o_4, o_1^{0.9}, o_7^{0.8}, o_8^{0.7}, o_{10}^{0.7}\}$.

Note that, by construction, for every $k \geq 0$, if $o \in \alpha^k(\mathbf{o})$ then $o' \in \alpha^k(\mathbf{o})$ for each $o' \sim o$. Indeed, if o belongs to $\alpha^k(\mathbf{o})$ because of an object $o'' \in \alpha^{k-1}(\mathbf{o})$ such that $o'' \rightleftharpoons o$, the consistency condition above implies that if $o \sim o'$, we have also that $o'' \rightleftharpoons o'$ and so, by Definition 2, $o' \in \alpha^k(\mathbf{o})$.

The augmented construct can be at the basis of two alternative ways to access a polystore, as described in the following.

Augmented Search. An augmented search consists of the expansion of the result of a query over a local database with data that are relevant to the query but are stored elsewhere in the polystore. Consider again a polystore $\mathcal{P}$ composed of a set of databases stored in different data management systems each equipped with a specific query language. Now, let $Q^{\mathcal{S}}$ denote a query expressed in the query language of the storage system $\mathcal{S}$ and let $Q^{\mathcal{S}}(\mathcal{D})$ be the set of objects in the result of $Q^{\mathcal{S}}$ over a database $\mathcal{D}$ stored in $\mathcal{S}$.

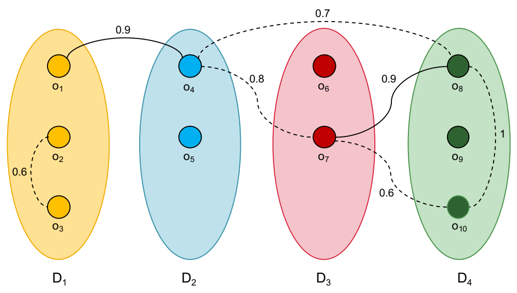


Fig. 2. A set of databases in a polystore

Definition 3. *The* augmentation of level $n \geq 0$ *of a query* $Q^{\mathcal{S}}$ *over a database* $\mathcal{D}$ *stored in* $\mathcal{S}$*, denoted by* $Q^{\mathcal{S}}(n)(\mathcal{D})$*, consists in the augmentation of level* $n \geq 0$ *of the result of* $Q^{\mathcal{S}}$ *over* $\mathcal{D}$ *ordered according to the probability of its elements.*

Example 5. Let Q be a query over the database D_1 in Fig. 2 that returns the objects o_1 and o_2. Then we have $Q(0)(D_1) = \{o_1, o_2, o_4^{0.9}\}$ and $Q(1)(D_1) = (o_1, o_2, o_4^{0.9}, o_7^{0.7}, o_8^{0.6}, o_3^{0.6})$.

Augmented Exploration Exploratory computing is a new trend in big data access that aims at helping users to make sense of very big data sets by means of step-by-step interaction with the system oriented to the progressive refinement of the data retrieval process [2]. The augmentation construct can provide effective support for exploratory computing in a polystore through a process that we call *augmented exploration*. Intuitively, augmented exploration consists of a guided expansion of the result of a query over a local database with related data stored elsewhere in the polystore. It works as follows: we start with a query $Q^{\mathcal{S}}$ expressed in the query language of the storage system $\mathcal{S}$ in the polystore and execute $Q^{\mathcal{S}}$ over a database $\mathcal{D}$ stored in $\mathcal{S}$. We then select, from the answer of the query an object o and apply to o the augmentation construct of level 0 (step 1) and order the result according to the probability of each element. Again, we select, from the result we obtain, an object o_1 and apply to it the augmentation construct of level 1 (step 2) and order the result. We then proceed similarly, by selecting an object o_i from the result of the previous step and apply the augmentation construct of level 1 to o_i, until the user is satisfied with her search. This can be formalized as follows.

Definition 4. *An* augmented exploration *of a polystore* $\mathcal{P}$ *starting from a query* $Q^{\mathcal{S}}$ *over a database* $\mathcal{D}$ *stored in* $\mathcal{S}$ *consists of a sequence of* k *steps:* $[(o_0 \rightarrow \boldsymbol{o}_0); (o_1 \rightarrow \boldsymbol{o}_1); \ldots; (o_k \rightarrow \boldsymbol{o}_k)]$ *where:*

- $o_0 \in Q^{\mathcal{S}}(\mathcal{D})$ *and* $\boldsymbol{o}_0 = \alpha^0(\{o_0\})$,
- $o_i \in \boldsymbol{o}_{i-1}$ *and* $\boldsymbol{o}_i = \alpha^1(\{o_i\})$ $(i > 0)$.

Example 6. Consider the query in Example 5 wich returns the set of objects $\{o_1, o_2\}$. Then, a possible augmented exploration involves the steps:

$$(o_1 \rightarrow \{o_4^{0.9}\}); (o_2 \rightarrow \{o_3^{0.6}\}); (o_4 \rightarrow \{o_7^{0.8}, o_8^{0.7}\})$$

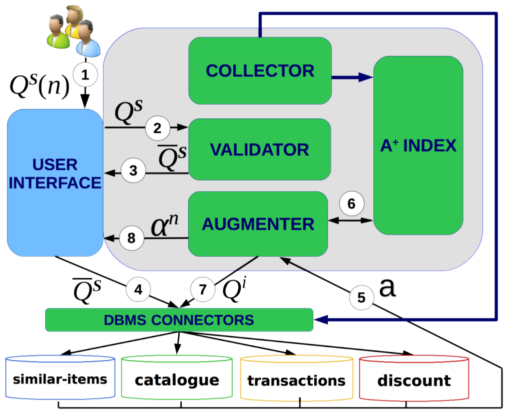


Fig. 3. Architecture of QUEPA.

4 A tool supporting augmentation

We have fully implemented our approach in a system called QUEPA [5]. Its architecture is illustrated in Fig. 3 and includes the main following components:

- *Augmenter:* it implements the augmentation operator and orchestrates augmented query answering. Various smart strategies have been adopted for its optimization aimed at being: (i) network-efficient, by retrieving all the objects from one database with a single query and thus reducing the number of queries to local databases and the communication between them, (ii) CPU-efficient, by assigning independent queries to parallel threads and leveraging the multi-core nature of modern CPU, and (iii) memory-efficient, by suitably caching the last accessed objects and thus reducing the I/O requests. In addition, since traditional cost-based optimizers are difficult to implement in a polystore, given the limited knowledge of each database in play, we have developed an adaptive, rule-based optimizer to dynamically predict the best combination of the above strategies for the the query under consideration.
- A$^+$ *index:* it is a global, graph-based index in which each global-key is represented in one node and there are two types of edges connecting nodes that represent identity and matching p-relations, respectively.
- *Collector:* it is in charge of discovering, gathering and storing p-relations in the A$^+$*index*. This is done by collecting fresh metadata, as soon as new datasets are added, and by leveraging existing techniques for record linkage [8]. Specifically, we used BLAST [11] for a non-supervised step of object blocking and DuKe[4] for the following step of pairwise matching. On the basis of the final matching score provided by DuKe, we decide whether a p-relation is an identity or a matching. In addition to the creation of the A$^+$*index* from scratch, we have developed a simple, yet effective, learning mechanism that

[4] `https://github.com/larsga/Duke`

adds matching p-relations depending on the navigation history of users when they operate in exploratory mode.

- *Connectors:* they are used to interact with the polystore. Each connector is able to communicate with a specific database system by sending queries in the local language and returning the result. Data objects are parsed into an internal representation.
- *Validator:* it is used to pre-evaluate a query and to assess rapidly whether it can be augmented or not.

The *validator* first checks if the query is correct (step ②) and possibly rewrites it (step ③) before its execution over the target database (step ④). The local answer **a** is returned to the *augmenter* which is now ready to compute the augmentation (step ⑤). It gets from the A$^+$*index* the global keys of data objects reachable from those in **a** with n applications of the augmentation primitive (step ⑥). These global keys are used to retrieve data objects from the polystore with local queries Q^i (step ⑦). Finally, the augmented answer is returned to the user (step ⑧). The interaction in the augmented exploration is similar but simplified since, at each step, only a single data object is augmented.

A comprehensive campaign of experiments done with QUEPA has shown that our approach is feasible, effective, and, unlike other approaches, scales nicely as the polystore grows in the number of stores and size of databases [6].

References

1. P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone. Data modeling in the NoSQL world. *Computer Standards and Interfaces*, 2016.
2. M. Buoncristiano et al. Database challenges for exploratory computing. *SIGMOD Record*, 44(2):17–22, 2015.
3. J. Duggan et al. The BigDAWG polystore system. *SIGMOD Record*, 44(2):11–16, 2015.
4. L. M. Haas. The power behind the throne: Information integration in the age of data-driven discovery. In *SIGMOD*, page 661, 2015.
5. A. Maccioni, E. Basili, and R. Torlone. QUEPA: QUerying and exploring a polystore by augmentation. In *SIGMOD*, pages 2133–2136, 2016.
6. A. Maccioni and R. Torlone. Augmented access for querying and exploring a polystore. In *ICDE*, pages 77–88, 2018.
7. D. Martinenghi and R. Torlone. Taxonomy-based relaxation of query answering in relational databases. *VLDB J.*, 23(5):747–769, 2014.
8. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 12 1969.
9. P. J. Sadalage and M. Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition, 2012.
10. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, Sep 1990.
11. G. Simonini, S. Bergamaschi, and H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12):1173–1184, 2016.
12. M. Stonebraker. The case for polystores. ACM SIGMOD Blog. `http://wp.sigmod.org/?p=1629`, July, 2015.