Just-In-Time Modeling with DataMingler*

Damianos Chatziantoniou and Verena Kantere

[1] Dept. of Management Science and Technology
Athens University of Economics and Business
damianos@aueb.gr
[2] School of Electrical and Computer Engineering
National Technical University of Athens
verena@mail.ntua.gr

Abstract. DataMingler is a prototype tool that implements a novel conceptual model, the Data Virtual Machine (DVM) and can be used for agile just-in-time modeling of data from diverse sources. The DVM provides easy-to-understand semantics and fast and flexible schema manipulations. An important and useful class of queries in analytics environments, *dataframes*, is defined in the context of DVMs. These queries can be expressed either visually or through a novel query language, DVM-QL. We demonstrate DataMingler's capabilities map relational sources and queries on the latter in a DVM schema and augment it with information from semi-structured and unstructured sources. We also show how to express on the DVM easily complex relational queries or queries on structured, semi-structured and unstructured sources combined.

Keywords: Data virtualization · just-in-time modeling.

1 Introduction

In a recent analytics project for a telecom provider, a customers' feature store had to be built and used by data scientists to develop churn prediction models. The features had to be rapidly defined and presented to the analysts in an easy to understand and use manner. Rapid development usually suggests the use of Python or R to build dataframes. This approach, however, leads to implementations that lack data semantics, limited data exploration capabilities and involvement of the data engineers end-to-end. A classic alternative is the deployment of an application-specific data warehouse, which is quite expensive both in terms of resources and schema changes. However, this is the approach most consulting firms (e.g. Accenture, EY, PwC) follow for analytics project. The question was whether we can combine the semantics offered by data modeling and the agility of programming languages. The challenge was how to, quickly, in a just-in-time manner, bind an attribute to a customer entity. To tackle the above, we propose the Data Virtual Machine (DVM) (briefly introduced in [3]),

* Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

a conceptual model based on entities and attributes – concepts that users understand well. A DVM is a graph where nodes represent attribute domains and edges represent mappings between these. The mappings are expressed by data processing tasks (e.g. a query) that provide their output as pairs of values. It is a form of data virtualization, a rapidly emerging trend in the business world [9], the main function of which is to hide technical details from users and provide a single view of *any entity.* Since nodes of a DVM can be attributes and entities at the same time, the "single view of any entity" can be trivially served by a DVM. The DVM provides intuitive semantics and fast and flexible schema manipulations. Dataframes, an important class of queries in analytics environments, can be expressed either through a novel query language, DVM-QL, or visually, and are evaluated within a formal algebraic framework.

DataMingler is a prototype tool [3] that implements DVM design and querying. It enables (a) efficient DVM management, (b) visual and textual specification of dataframe queries, (c) optimization and evaluation of dataframe queries, and (d) materialization of a DVM into different logical models (model polymorphism.) In this paper we demonstrate how information in relational sources and queries on the latter can be mapped to a DVM, and, further, combined with information in semi-structured and unstructured and sources, like JSON, Excel and csv files. We also show how we can very easily express on a DVM queries that are either very complex to declare on the relational sources or impossible to declare on the ensemble of structured, semi-structured and unstructured sources.

2 Concepts and Current Landscape

Modern organizations collect, store and analyze a wealth of data from different sources and applications, used in a variety of analysis projects to provide a competitive advantage to business. This data has to be integrated to provide data scientists with a "holistic" view of the enterprise's *data infrastructure*, a term that encompasses much more than data persistently stored in a DBMS, SQL, NoSQL, or otherwise. It also involves flat files, spreadsheets and transient data handled by stream engines. Furthermore, it includes *processes* (i.e. programs) that produce output useful in the analysis phase, e.g. a Python program that computes the social influence or the churn category of a customer. It is important to easily and agilely represent all these in a conceptual data layer, e.g. by using data virtualization, a new business trend and is closely related to mediators and virtual databases (a form of data integration), if not a reinvention of these: It allows data manipulation without requiring technical details about the data, such as how it is formatted at source or where it is physically located by people who own, contribute, manage and analyse the data and assume several roles.

Current Situation in Industry The industrial scene is prevailed by companies that offer practical data virtualization. Oracle [10], Denodo [7] and other database technology companies [9] already offer data virtualization products. Most implement a virtual relational model, defining views over data sources having a relational interface. While the relational model and its query language, SQL, are well-known and well-understood by the IT community, they present shortcomings when used as a data virtualization technique – more or less the same shortcomings existing in data warehousing environments. The relational model is not a high-level conceptual model and is cumbersome for many data analysts and contributors. Schematic modifications and extensions are very rigid and may depend on complex constraints. As the relational model is flat, focusing on a conceptual entity and collecting all relevant data may be tedious and require the expression and execution of complex SQL queries. Last, visualization of schema manipulation is not intuitive due to the tabular format.

Current Situation in Research Classical data integration deals with the problem of defining a global schema on heterogeneous relational data sources using a mapping between the global and the local schemata [8]. The issue of query containment and equivalence is of great importance, which is investigated with respect to the open and closed world assumption. Oppositely, in the DVM approach our attention is not on the original queries on the data, neither with respect to definition, nor evaluation. Furthermore, there are works that create Virtual Knowledge Graphs as means of integrating and accessing autonomous and heterogeneous sources. The focus has been on the definition of mappings [1] with a well-balanced trade-off of expressiveness and complexity: the mappings need to be adequately expressive in order to allow for complex querying, but queries should be tractable especially in the size of data. Oppositely to OWL 2.0 and semantic reasoners, DVM is not meant for the expression of complex conceptual relations or for reasoning. In the DVM approach we are not interested in defining sophisticated mappings and balancing expressiveness with complexity.

Enabling query processing across heterogeneous data models by federating specialized data stores, an interesting research topic, is discussed in [11]. A system that supports this can be classified as a federated database, a polyglot, a multistore or a polystore system according to a taxonomy presented in [12]. Our work is motivated by similar concerns and could be considered a multistore.

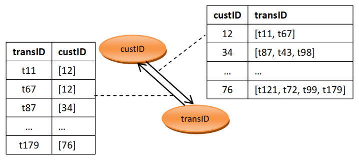
3 Data Virtual Machines

In this section we briefly present the definition of the Data Virtual Machine (DVM). For a more extended discussion the reader is referred to [2,3].

In simple terms, a DVM represents a collection of *mappings* (edges) between *attribute domains* (nodes), where mappings are manifested as data processes with a 2-dimensional output (the attribute domains) over existing data.

Definition 1. ***[Key-list Structure]*** *A key-list structure (KL-structure) K is a set of (key, list) pairs, $K = \{(k, L_k)\}$, where L_k is a list of elements or the empty list and $\forall\ (k_1, L_{k_1}), (k_2, L_{k_2}) \in K$, $k_1 \neq k_2$. Both keys and elements of the lists are strings. The set of keys of KL-structure K is denoted as $keys(K)$; the list of key k of KL-structure K is denoted as $list(k, K)$.* □

Definition 2. ***[Data Virtual Machines]*** *Assume a collection $\mathcal{A}$ of n domains $A_1, A_2, \ldots, A_n$, called attributes. Assume a collection $\mathcal{S}$ of m multisets, $S_1, S_2, \ldots, S_m$, where each multiset S has the form: $S = \{(u, v) : u \in A_i, v \in A_j,\ i, j \in \{1, 2, \ldots, n\}\}$, called data processing tasks. For each such $S \in \{S_1, S_2, \ldots, S_m\}$ we define two key-list structures, K^S_{ij} and K^S_{ji} as:*



transID	custID
t11	[12]
t67	[12]
t87	[34]
...	...
t179	[76]

custID	transID
12	[t11, t67]
34	[t87, t43, t98]
...	...
76	[t121, t72, t99, t179]

Fig. 1. Key-list structures to represent edges of DVMs

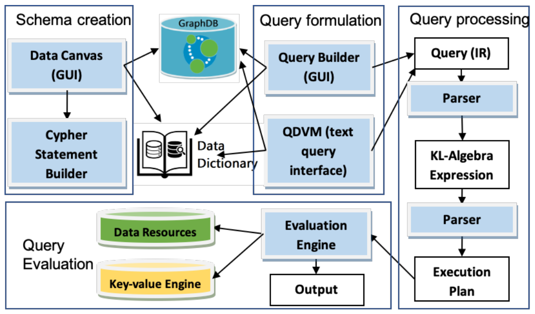


Fig. 2. DataMingler's Architecture

K_{ij}^S: *for each* u *in the set* $\{u : (u, v) \in S\}$ *we define the list* $L_u = \{v : (u, v) \in S\}$ *and* (u, L_u) *is appended to* K_{ij}^S.

K_{ji}^S *is similarly defined.*

The data virtual machine corresponding to these attributes and data processing tasks is a multi-graph $G = \{\mathcal{A}, \mathcal{S}\}$ *constructed as follows:*

– *each attribute becomes a node in* G

– *for each data processing task* S *we draw two edges* $A_i \rightarrow A_j$ *and* $A_j \rightarrow A_i$, *labeled with* K_{ij}^S *and* K_{ji}^S *respectively.*

The key-list structure that corresponds to an edge e :$A_i \rightarrow A_j$ *is denoted as* $KL(e)$, *with schema* (A_i, A_j). $\square$

Example 1. Assume the SQL query "`SELECT custID, transID FROM Customers` that maps transactions to customers and vice versa. The attributes, edges and the respective key-list structures are shown in Figure 1. $\square$

4 DataMingler: An Overview

The architecture of DataMingler is shown in Figure 2. Data Canvas is the module that enables the creation and manipulation of a DVM graph by mapping data and processes onto the graph and extending it with new nodes and edges. The resource types that DataMingler currently handles are: relational databases, csv files, excel and stand-alone programs (Java and Python). Description of resources (connection strings, database names, file paths and names, etc.) can be found in the *data dictionary* in XML. A DVM is kept in a Neo4j graph database. Nodes have as properties the *name* and the *description* of the attribute. Edges have as properties the *data source name*, the *query string* (if any), and the *positions* in the output that correspond to the head and tail nodes of the edge.

The user can formulate dataframe queries either textually, or visually using Query Builder. In both cases, queries are represented in an XML-based intermediate representation, and parsed and transformed to a key-list algebraic expression, which is given to the optimizer and an execution plan is generated. Redis is currently used as they key-value engine for loading and manipulating key-list structures. *Transformations()*, *rollupJoin()* and *thetaCombine()* operators have been implemented both in Python, Java and R. The output at the top-level is a dataframe in Python/R, i.e. columns may contain atomic values or lists.

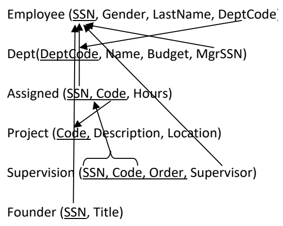
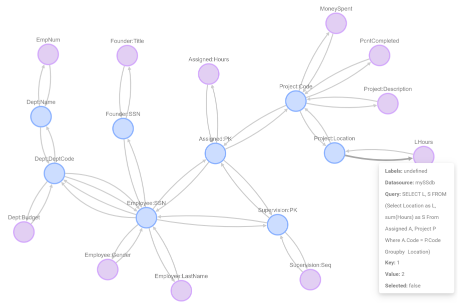

Employee (SSN, Gender, LastName, DeptCode)
Dept(DeptCode, Name, Budget, MgrSSN)
Assigned (SSN, Code, Hours)
Project (Code, Description, Location)
Supervision (SSN, Code, Order, Supervisor)
Founder (SSN, Title)

Fig. 3. A relational database schema to be mapped to a DVM



Labels: undefined
Datasource: mySSdb
Query: SELECT L, S FROM (Select Location as L, sum(Hours) as S From Assigned A, Project P Where A.Code = P.Code Groupby Location)
Key: 1
Value: 2
Selected: false

Fig. 4. The DVM schema from a relational source augmented with attributes from queries and an excel file

An association between nodes is defined via the use of a data source: the user specifies (a) the column/output position in the source that corresponds to the head node of the edge, the *root* (e.g. `custID`) and, (b) the column/output position in the source that corresponds to the tail node of the edge, the *child* (e.g. `Age`.) If these nodes do not exist in the DVM, they are created. The required data processing tasks (e.g. "SELECT custID, Age FROM Customers", "SELECT Age, custID FROM Customers") are derived automatically and attached to the edges between the nodes. The creation of a DVM using a relational database, an excel, a csv and a program is depicted in video [4].

Using Query Builder over a DVM, complex, conceptually difficult queries become simple and intuitive to express. Video [6] provides a step-by-step construction of two dataframe queries. Furthermore, DataMingler can be used to instantiate model-specific databases. The user selects a node and a breadth-first-search tree rooted on this node is defined. The system generates a collection of JSON documents corresponding to this tree. It implements attributes as lists or strings, depending on the cardinality of lists (whether they contain multiple or single values.), demonstrated in video [5].

5 Demonstration Description: Just-In-Time Modeling

In this demonstration we show how to use DataMingler in a real use-case scenario to create a schema that coalesces information from multiple sources and how to form visually queries that are otherwise very complex to declare in SQL or impossible to declare on multiple sources. We show in detail how we map relational data sources and existing queries on them to a DVM, as well as how we augment the latter with nodes from semi-structured sources line JSON and unstructured sources, like collections of csv or excel files. Let us assume a company with many departments that runs projects in multiple locations. Employees are managers, supervisors or founders of the company. The schema of a respective relational database is shown in Figure 3. There is also an Excel file that stores information on KPIs for some projects (e.g. the projects of a specific department). The DVM can be enriched with columns from the Excel file, concerning e.g. funding already spent ('MoneySpent' and percentage of the project that is completed ('Prct-

Completed'). Figure 4 shows the produced DVM. The screenshot also shows the query that produced the node 'LHours'. Note that the nodes 'ProjectLocation' and 'ProjectCode' which correspond to attributes in the relational schema, become entities in the DVM after the addition of further information from queries (entities shown in blue and attributes in pink). Also, attributes comprised in a primary key, such as 'SSN', 'Code' and 'Order' in relation 'Supervision', do not appear as nodes in the DVM, but are represented by the node 'Supervision:PK'. The latter is connected with 'Assigned:PK', the node representing the primary key of 'Assigned' (i.e. the combination of attributes 'SSN' and 'Code'), with a pair of edges that represents the respective foreign key constraint. Finally, the nodes 'Dept:Code' and 'Employee:SSN' are connected with two pairs of edges, representing the two foreign key constraints between the respective relations.

The demonstration delves in the details of expressing on a DVM in a simple, intuitive and seamless manner queries complex and non-intuitive to structure in SQL, and combinations of complex SQL queries and processing tasks on unstructured data. One such query returns the SSNs of employees that participate in at least one project with their manager. This is a complex query to express on the relational database, but a very easy one to express on the DVM, in which the user needs just to select the nodes and processing tasks for the query and build a tree. Another such query asks for the supervisors of projects together with the number of employees they supervise in these and the sum of their hours for all projects the employees work for. This can be expressed by a combination of SQL queries. Due to lack of space, we do not show the screenshots of the queries.

References

1. Calvanese, D., Gal, A., Lanti, D., Montali, M., Mosca, A., Shraga, R.: Mapping patterns for virtual knowledge graphs (12 2020)
2. Chatziantoniou, D., Kantere, V.: Data Virtual Machines: Data-Driven Conceptual Modeling of Big Data Infrastructures. In: Workshops of EDBT 2020 (2020)
3. Chatziantoniou, D., Kantere, V.: Datamingler: A novel approach to data virtualization. In: ACM SIGMOD. pp. 2681–2685 (2021)
4. DataMingler: Data canvas. https://drive.google.com/file/d/1hArnqc9HjSWobDWFbdjQZ1L_e9WU-eQQ/view?usp=sharing (2020)
5. DataMingler: Json export. https://drive.google.com/file/d/1Ruv356HQxtoZyRunBPuf-aA-NHkhGIlF/view?usp=sharing (2020)
6. DataMingler: Query builder. https://drive.google.com/file/d/1FQyuXQ8BDzcoDnHy0aQcW0FEKR2FapUZ/view?usp=sharing (2020)
7. Denodo: Data Virtualization: The Modern Data Integration Solution (2019)
8. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann (2012)
9. Gartner: Market Guide for Data Virtualization (2018)
10. Oracle Corp.: Oracle Data Service Integrator (2020)
11. Stonebraker, M.: The Case for Polystores. ACM SIGMOD Blog (July 13 2015), http://wp.sigmod.org/?p=1629
12. Tan, R., Chirkova, R., Gadepally, V., Mattson, T.G.: Enabling Query Processing Across Heterogeneous Data Models: A Survey. In: IEEE BigData. pp. 3211–3220 (2017)