Data Virtual Machines: Enabling Data Virtualization

Damianos Chatziantoniou[1](✉) and Verena Kantere[2]

[1] Department of Management Science and Technology, Athens University of Economics and Business, Athens, Greece
damianos@aueb.gr

[2] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
verena@mail.ntua.gr

Abstract. Modern analytics environments are characterized by a data infrastructure that comprises a great variety of datasets, data formats, data management and processing systems. Such environments are dynamic and data analysis needs to be performed in a flexible and agile manner via data virtualization techniques. Towards this end, we have proposed the Data Virtual Machine (DVM), a graph-based conceptual model based on entities and attributes. The basic idea of the DVM is that the relations of entities and attributes are based and expressed as the output of data processing tasks. In this paper we discuss the notion of data virtualization and propose a set of goals for relevant techniques in terms of modeling capabilities, query formulation and schema flexibility. We also place DVMs with respect to these goals.

Keywords: Data virtualization · Data virtual machines

1 Introduction

Modern analytics environments in research and industry aim at the analysis of diverse datasets. The latter may be produced by a variety of applications or collected by autonomous agents and stored in dispersed locations. Therefore, such datasets are expected to be inherently heterogeneous in both the semantics that they encapsulate as well as the data structures that hold them and relevant systems that manage and process them. This heterogeneity extends from the data to the analysis itself, since various users of analytics environment – most probably data scientists, but also data engineers, simple end-users, data regulation officers etc. – want to deploy different analysis projects, related to traditional BI, data exploration, data mining, prediction etc.

Coping with this diversity in the data management landscape is a well-known research mandate [1,6]. Multiple big data systems and analysis platforms need to coexist, integrated and federated. While data warehousing is the usual approach, it is rigid for a rapidly changing data environment. In addition, [1] discusses the need for techniques that can create *late-bound schemas* on data that may be persisted but are processed seldomly, if ever. In the analytics environment as described above, this need is more prevalent, since the whole data infrastructure is dynamic and analysis purposes change agilely. For such environments, producing full-fledged classical integrated schemas on structured, semi-structured and unstructured data is not only extremely expensive in terms of time and resources, but it may also be impossible to achieve in the time-window of the analysis.

To tackle the above challenges, we have proposed the Data Virtual Machine (DVM), which was briefly introduced in [3], a graph-based 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 domains. These mappings are expressed by the output of data processing tasks (e.g. a query or a program.) This output must be provided as (v_1, v_2) value pairs. The argument is that these data processing tasks can be easily defined, in an on-demand and visual manner. The schema is *then* generated. Furthermore, this graph is appropriate for visual query formulation (emphasis is given on dataframes). We have developed a research prototype, called DataMingler [5] that demonstrates the DVM and its potential.

A DVM is a *data virtualization* technique, a significant enterprise trend [9]. The goal is to allow an application to retrieve and manipulate data without requiring technical details about the data, i.e. abstracting data out of its form and manipulating it regardless of the way it is stored or structured [11]. It also provides a single customer view (or single view of *any other entity*) of the overall data [14]. All major DB vendors offer products in this field, e.g. [7,10,12,13]. Data virtualization is closely related to mediators and federated databases [8], concepts that the research community has dealt with decades ago, when systems heterogeneity was also an issue. The focus of data virtualization however is on availing common database features, such as data modeling, querying and data extraction to DB-naive users [2,11].

In this paper we propose a set of goals for dynamic and efficient data virtualization environments (Sect. 2) in terms of modeling capabilities, query formulation and schema flexibility. We then discuss data virtual machines with respect to these challenges (Sects. 3 and 4).

2 Data Virtualization

The research question is how a db-literate person can create quickly, *agilely* a virtual data layer on top of an organization's data infrastructure that can be easily and *intuitively* used by db-illiterate people (usually data scientists) for data exploration and query formulation.

2.1 Analytics Environments

The term *data infrastructure* encompasses much more than data persistently stored in data management systems, SQL, NoSQL, or otherwise. It also involves flat files, spreadsheets and transient data handled by stream engines. Last but not least, it includes processes (e.g. queries, scripts, models, web services, programs in general) that produce output that could be perceived as data and exploited in the analysis phase, for instance a Python program that computes the social influence or the churn category of each customer. These are also attributes of the customer and it is important to easily and agilely represent them at the conceptual data layer (think: derived attributes in ER model).

We identify four distinct roles of data stakeholders in an analytics environment. *Data engineers* are technical people whose main duty is to extract, transform and integrate data from multiple sources. They would like to have a formalism where they can easily and quickly map data and programs' output onto it. They also create/prepare data structures appropriate for input to data analysis tasks, for example dataframes. They would like to delegate this task to data scientists. *Data scientists*, not necessarily DB-literate people, usually build a model for an entity, using the features (attributes) of that entity. They would like to have at their disposal a simple model that identifies entities and attributes, so they can easily select/experiment with those of interest. Ideally, they would like to create/manage themselves the data inputs to their algorithms. *Data subjects or contributors*, such as customers, suppliers, users, will soon be involved in the data supply chain due to the EU GDPR's data portability requirement – data markets have already emerged in various domains. They have to be provided with a high-level model, easy to understand and manage, and possibly link to data from other organizations. Some "self-service" data integration and data portfolios management will be necessary. In addition, data subjects should be enabled to export selected data to the logical model of their choice (e.g. XML, JSON, relational.) Finally, *data protection officers* whose primary role is to ensure that their organization processes personal data according to applicable data protection laws. They want to easily see data provenance and consent of data at a fine granular level.

2.2 Goals of Data Virtualization Systems

A data virtualization layer should serve appropriately all data stakeholders of an analytics environment. This layer should exhibit the properties mentioned below. As a running example, assume the following – very simple – collection of data sources.

Example 1. Assume a data setting with the following data sources:

- *S1*: A relational table `Customers(custID,Age,Gender,City)`,
- *S2*: An excel spreadsheet called Transactions having columns 1 to 4 labeled as: `transactionID, custID, Amount, Date`,
- *S3*: A text file keeping the customers' comments as `(custID, comment)` lines,
- *S4*: A java program implementing a churn prediction model which outputs when executed the customer's ID and her churn category for all customers. □

Model Simplicity, Plasticity and Agility. The data virtualization model should be simple, based on concepts that people understand well, for example entities and attributes. It should represent these in a graphical manner and should be amenable to visual manipulations, both in terms of schema management and query formulation. Data sources become rapidly available and unavailable in modern analytics environments. Data engineers should be able to easily and quickly, in an ad hoc manner, incorporate parts of these in a virtual schema without significant semantic effort. It should also be easy to modify the virtual schema with no major implications to the rest of it. In Example 1, the question is how can one *quickly* represent the `customer` entity along with his `transactions, age, gender, city, comments, churn category` attributes. At the same time, a `transaction` is an entity itself with its own attributes `customer, date, amount.`

Intuitive Data Exploration/Query Formulation. Data exploration and query formulation should be facilitated by non-DB experts (e.g. statisticians, data subjects) in a simple and intuitive, visual, manner. One should easily select attributes, express conditions and define transformations, using built-in or plug-in functions in some programming language of his/her choice to form a query. Query evaluation should be efficient and based on a theoretical framework, possibly similar to relational algebra. For example, one should easily ask for each `customer` his `age` and `gender`, the average sentiment of his `comments` containing the keyword "google", the actual list of her `comments` and the total amount of her `transactions` on May 2019.

Data Sharing: Data sharing has been identified as a key element of the big data era. Data sharing is related to many different research topics, such as ontologies, common vocabularies, vertical and horizontal sharing, etc. In real-life analytics environments, people need to share parts of spreadsheets, flat files, json documents or relations. In most cases this is done manually, in an ad hoc manner, by exporting data to a text file and moving this file around. There is no principled way to describe what someone shares in an intermediate representation, unless if imported to a data warehouse. A data virtualization model should make this process easy, quick and semantically clear. It should serve as the medium for data sharing in a standardized, collaborative, distributed manner.

Support for "Any Entity View": This is not a known term in conceptual modeling, yet it is an important, practical requirement in analytics applications of an organization. Traditionally refers to the *customer* entity ("single customer view"), which means structuring/representing attributes from several sources around the customer entity (e.g. think of a JSON document rooted on customer's id). However, data scientists want to study features of other entities as well, such as *suppliers, employees, products, transactions*, etc. Going one step further, they want to convert entities' attributes, such as the *age* of a customer, the *churn category* of a customer, or the *date* of a transaction, to entities and study them individually. For example, think of a JSON document rooted on *age* or *churn category* or *date*. To do so, all model's constructs should be conceived as an attribute and an entity at the same time, i.e. they have to be symmetric in a way. The ER model does not exhibit this property, since it represents entities and attributes with different constructs. It should be easy to "reorient" the schema around any entity.

Model Polymorphism: One size does not fit all and a virtual schema has to be easily concretized to different logical models. While in a traditional database design the data model is predefined and determines storage models, in a conceptual design one can create database instances in different logical models (e.g. relational, semi-structured, multi-dimensional). These instances can then be queried by data scientists via the native query language of the model (e.g. SQL, Mongo queries, MDX).

Linkability and Crawlability: Data exploration requires a "connected" model, where the schema can be navigated. In addition, feature selection is a well-known topic in statistics: the data scientist selects appropriate features for a model. For this, a model that supports crawling is needed, i.e. starting from an entity, an algorithm collects or defines relevant attributes. This property also plays an important role in GDPR's data portability requirement. Data subjects could build their own data models from multiple domains.

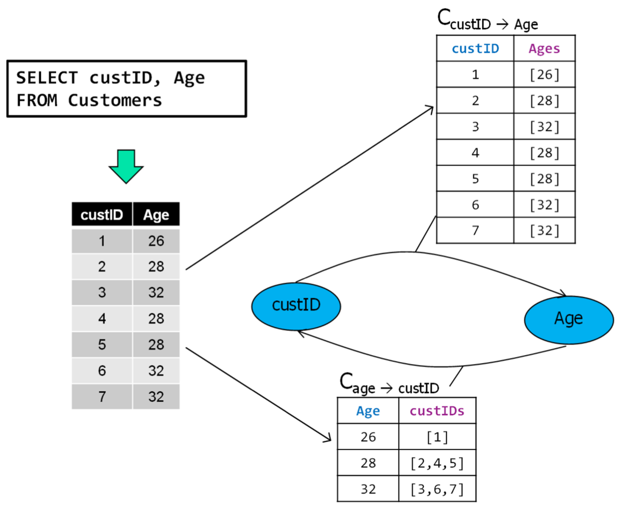
3 Data Virtual Machines

A Data Virtual Machine (DVM) is a graph-based representation of a collection of Data Manipulation Tasks (DMT) defined over the data infrastructure of an organization. For example, consider the SQL query: `''SELECT custID, Age from Customers''`. The *output* of this query provides two mappings between two attribute domains, `custID` and `Age`. These mappings can be represented by two key-list structures, as shown in Fig. 1. Attribute domains become nodes in the DVM graph and edges represent mappings between them, as manifested by the output of DMTs. The motivation of the main idea lies on multi-valued, derived attributes in Entity-Relationship theory: each node in a DVM is a derived, multi-valued attribute for any other attribute that is related to (which plays the role of the primary key of an entity) and vice versa. More details can be found in [4].

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 $list(k, K)$. If $k \notin keys(K)$, the value of $list(k, K)$ is null. The schema of a KL-structure K, denoted as $K(A, B)$ consists of two labels, A and B. A is the role of the key and B is the role of the list in the key-list pairs.* □

A key-list structure is essentially a multi-map data structure, where mapped values to a key are organized as a list. There are many key-value stores (e.g. Redis) that efficiently handle key-list structures.

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 manipulation 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:*



```
SELECT custID, Age
FROM Customers
```

custID	Age
1	26
2	28
3	32
4	28
5	28
6	32
7	32

$C_{custID \rightarrow Age}$

custID	Ages
1	[26]
2	[28]
3	[32]
4	[28]
5	[28]
6	[32]
7	[32]

$C_{age \rightarrow custID}$

Age	custIDs
26	[1]
28	[2,4,5]
32	[3,6,7]

Fig. 1. A data manipulation task as a mapping between attributes

K^S_{ij}: *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^S_{ij}.
K^S_{ji} *is similarly defined.*

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

– *each attribute becomes a node in* G
– *for each data manipulation task* S *we draw two edges* $A_i \rightarrow A_j$ *and* $A_j \rightarrow A_i$, *labeled with* K^S_{ij} *and* K^S_{ji} *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). □

The term "data manipulation task" refers to its output. The terms "attribute" and "node" are used interchangeably in the remaining of the paper.

Note that DVM modeling is somehow the inverse process of traditional data modeling. While in the latter one *first* designs the schema and *then* creates the DMTs to accommodate (fit) this schema, in DVM modeling one first defines the DMTs and a schema is produced according to the DMTs.

4 Data Virtualization and DVMs

In this section, we revisit the goals of data virtualization, as presented in Sect. 2.2, in the context of DVMs.

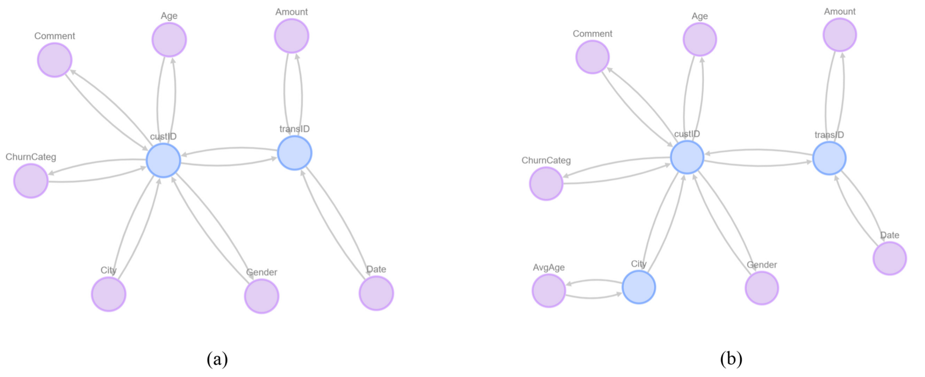


Fig. 2. A DVM generated according to a collection of DMTs

4.1 Agile Schema Modeling

Let us consider Example 1 and define the following data manipulation tasks over it – with the obvious semantics for each one:

- *DMT1:* `SELECT custID, Age From Customers` using `S1`
- *DMT2:* `SELECT custID, Gender From Customers` using `S1`
- *DMT3:* `SELECT custID, City From Customers` using `S1`
- *DMT4:* `SpreadsheetReader(S2,1,2,'transID','custID')`
- *DMT5:* `SpreadsheetReader(S2,1,3,'transID','Amount')`
- *DMT6:* `SpreadsheetReader(S2,1,4,'transID','Date')`
- *DMT7:* `TextReader(S3,1,2,'custID','Comment')`
- *DMT8:* `ExecEnv(S4,1,2,'custID','ChurnCateg')`

The generated DVM is shown in Fig. 2(a), as generated by the DataMingler tool [5]. Nodes having degree >1 are painted blue for visualization purposes and one could think of them as "entities", although all nodes have the same status in the graph. Assume now that one wants to relate each city with the average age of city's customers. s/he could do this by defining the following DMT.

- *DMT9:* `SELECT City, CAST(avg(Age) AS int) as AvgAge FROM Customers GROUP BY City` using `S1`

The generated DVM is shown in Fig. 2(b). Note that the `City` attribute is painted blue now. We reiterate here that *DMT9* provides not only an average age per city, but also a list of cities per age.

4.2 Simple and Efficient Dataframing

What kind of queries can we have on top of DVMs? Defining a complete query language over DVMs is work in progress. We focus on what data scientists/ statisticians usually do, since this is the primary target group of this work. They

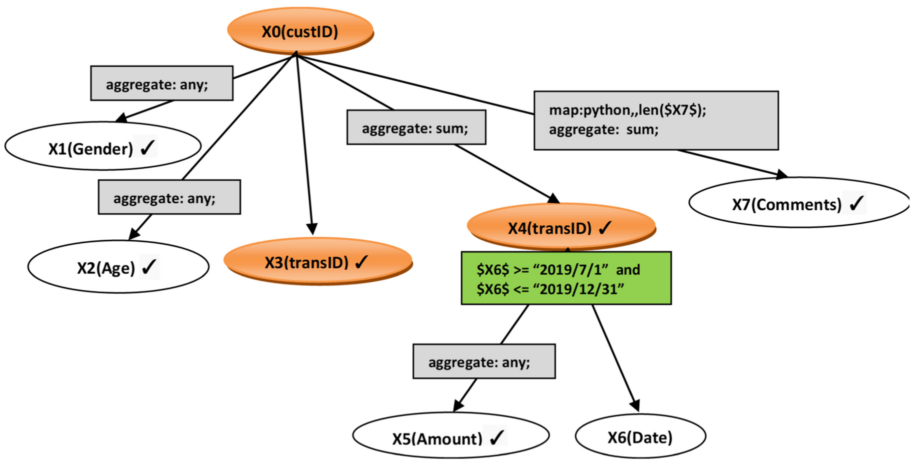


Fig. 3. A dataframe query example

usually form dataframes in Python, R or Spark. A dataframe is a table built incrementally, column-by-column, around an entity. The first column(s) is usually the key of the entity (e.g. customer ID) and the remaining ones are related attributes. An attribute can be processed (aggregated, filtered or transformed via a user-defined function in Python or R) before added as a column to the dataframe. The latter often serves as input to learning algorithms or for ad hoc reporting. It is important to facilitate this process in a simple and intuitive, visual manner. One should easily select (possibly along a path) attributes, express conditions and define transformations, using built-in methods or plug-in functions in a programming language of her choice (polyglotism) to form a dataframe.

A dataframe query over a DVM is defined as a tree, consisting of nodes and edges of the DVM [5] - *not necessarily* a subtree of the DVM. The same node/edge could appear multiple times in a dataframe query. For example, the query: "for each customer (custID), show his/her Gender and Age, the list of his/her transactions, the total amount of these transactions in the second semester of 2019 and the total number of characters of his/her comments" is shown in Fig. 3. To evaluate a dataframe query, the transformations on the edges have to be applied and the edges have to be combined, either along a path or across the same level. This evaluation (as well as optimization) take place within an algebraic framework equipped with operators that take as input one or more edges (i.e. key-list structures) and have as output an edge (another key-list structure).

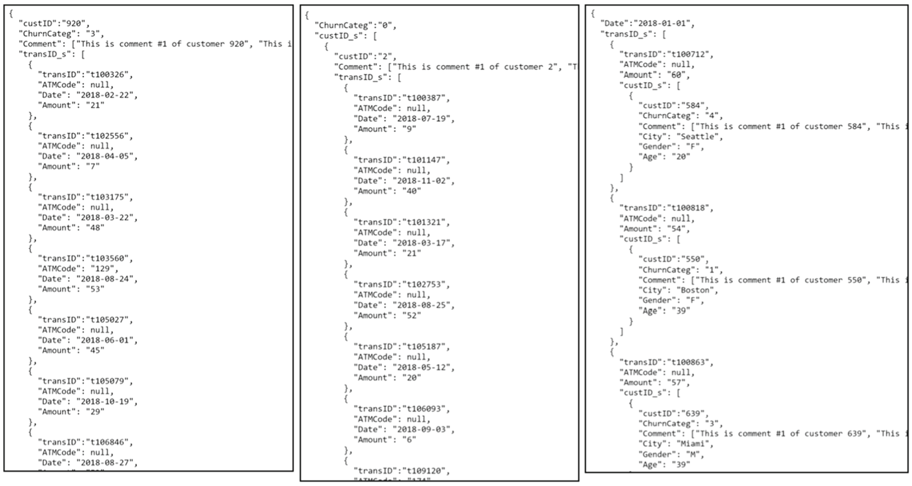
4.3 Any-Entity View, Model Polymorphism

One of the goals of data virtualization is to provide a single view of any entity of the overall data. Traditional logical models, such as the relational or semi-structured ones fall through this goal because they provide a single-angle view of a schema, built with a specific entity in mind. Moreover, traditional conceptual models, such as the ER or UML, build a diagram in which metadata about the data are encapsulated in schematic objects of the model in a absolute manner, e.g. as entity sets, as relationship sets (ER) or as attributes, with absolute relation to other such schematic objects (e.g. an attribute belongs to a specific entity set or two entity sets or more are connected semantically in specific way via a relationship set. As a result, it is difficult to reorient a schema (or diagram) adhering to traditional models, in a way that all metadata can be viewed with respect and in relation to a specific schematic object, essentially giving different conceptual angles of the encapsulated world. Such schema reorientation can be a very useful tool in the hands of data scientists, enabling them to define very easily and using always the same schema different views of the overall data with respect to different entities in the data.

As an example, let us consider the nodes `custID`, `transID`, `Date` and `ChurnCateg` in Fig. 2(b). `custID` and `transID` are blue nodes, i.e. are considered as entities with attributes. In a respective ER diagram, these would correspond to two entity sets (with their respective attributes) connected with a relationship set. The produced relational schema would encode only this specific conceptual view, in which, for example, `Date`s are referenced with respect to `transID`s and not to `custID`s. In order to view `Dates`s with respect to `custID`s (i.e. the dates in which customers have made transactions) one would need to query the data and produce new relations (not encapsulated in the original conceptual schema). Moreover, `Date`s, which correspond to an attribute in the respective ER and relational schema, can be viewed only in reference to `transID`s, whereas it may be useful to a user to view `transID`s and even, furthermore, `custID`s, with respect to `Dates`s (i.e. what are the transactions made on a date and which customers made these transactions). As in the previous case, this can be done only via querying the data to produce a new relation that does not adhere to the original conceptual schema, but, in this case, the query is also quite complex, as it attempts to turn around the relationship of entity-attribute between `transID` and `Dates`. The same holds if one wants to view the data of customers and their transactions with respect to `ChurnCateg`.

The DVM is a model that achieves the goal of 'any-entity view' by simplifying all conceptual objects to graph nodes and all conceptual relations to graph edges. The exact conceptual relations between two nodes are determined by the data itself via the corresponding data processing tasks. Therefore, the exact conceptual objects are also defined by the data itself, i.e. if a node is an entity or an attribute. In the DVM, we define the notion of *entity's view* with respect to some node N as a graph consisting of the node N and a collection of DVM's paths that originate from N and end on a node M different than N.

The simplest structure of the DVM together with the fact that it is the data and the data processing tasks that form the entities, attributes and relations between entities and entities as well as entities and attributes, make it possible for a DVM to represent multiple different conceptual and logical schemas. The latter certainly necessitate different representations in all traditional conceptual



```
{
  "custID":"920",
  "ChurnCateg": "3",
  "Comment": ["This is comment #1 of customer 920", "This i
  "transID_s": [
    {
      "transID":"t100326",
      "ATMCode": null,
      "Date": "2018-02-22",
      "Amount": "21"
    },
    {
      "transID":"t102556",
      "ATMCode": null,
      "Date": "2018-04-05",
      "Amount": "7"
    },
    {
      "transID":"t103175",
      "ATMCode": null,
      "Date": "2018-03-22",
      "Amount": "48"
    },
    {
      "transID":"t103560",
      "ATMCode": "129",
      "Date": "2018-08-24",
      "Amount": "53"
    },
    {
      "transID":"t105027",
      "ATMCode": null,
      "Date": "2018-06-01",
      "Amount": "45"
    },
    {
      "transID":"t105079",
      "ATMCode": null,
      "Date": "2018-10-19",
      "Amount": "29"
    },
    {
      "transID":"t106846",
      "ATMCode": null,
      "Date": "2018-08-27",
```

```
{
  "ChurnCateg":"0",
  "custID_s": [
    {
      "custID":"2",
      "Comment": ["This is comment #1 of customer 2", "T
      "transID_s": [
        {
          "transID":"t100387",
          "ATMCode": null,
          "Date": "2018-07-19",
          "Amount": "9"
        },
        {
          "transID":"t101147",
          "ATMCode": null,
          "Date": "2018-11-02",
          "Amount": "40"
        },
        {
          "transID":"t101321",
          "ATMCode": null,
          "Date": "2018-03-17",
          "Amount": "21"
        },
        {
          "transID":"t102753",
          "ATMCode": null,
          "Date": "2018-08-25",
          "Amount": "52"
        },
        {
          "transID":"t105187",
          "ATMCode": null,
          "Date": "2018-05-12",
          "Amount": "20"
        },
        {
          "transID":"t106093",
          "ATMCode": null,
          "Date": "2018-09-03",
          "Amount": "6"
        },
        {
          "transID":"t109120",
```

```
{
  "Date":"2018-01-01",
  "transID_s": [
    {
      "transID":"t100712",
      "ATMCode": null,
      "Amount": "60",
      "custID_s": [
        {
          "custID":"584",
          "ChurnCateg": "4",
          "Comment": ["This is comment #1 of customer 584", "This i
          "City": "Seattle",
          "Gender": "F",
          "Age": "20"
        }
      ]
    },
    {
      "transID":"t100818",
      "ATMCode": null,
      "Amount": "54",
      "custID_s": [
        {
          "custID":"550",
          "ChurnCateg": "1",
          "Comment": ["This is comment #1 of customer 550", "This i
          "City": "Boston",
          "Gender": "F",
          "Age": "39"
        }
      ]
    },
    {
      "transID":"t100863",
      "ATMCode": null,
      "Amount": "57",
      "custID_s": [
        {
          "custID":"639",
          "ChurnCateg": "3",
          "Comment": ["This is comment #1 of customer 639", "This i
          "City": "Miami",
          "Gender": "M",
          "Age": "39"
```

Fig. 4. Creating JSON documents for different DVM's nodes, conceived as entities

and logical models. For example, we would need two different ER schemas to represent `transID` as an entity set and `Date` as an attribute and the opposite. Furthermore, we would need schema mappings between logical schemas that are derived from the conceptual schemas. The DVM not only alleviates all this complexity, but enables the extraction of multiple schemas in traditional structured and semi-structured models, via the implementation of the notion 'entity view', achieving in this way, polyglotism. For example, based on a given graph for a specific 'entity view', a relational (work in progress) or semi-structured database can be materialized.

Using DataMingler [5], a user can selects a node in the DVM and a breadth-first-search tree rooted on that node is defined and the system generates a collection of JSON documents corresponding to the tree defined. It implements attributes as lists or strings, depending on the cardinality of lists (whether they contain multiple or single elements.) Fig. 4 shows the output when (a) `custID` is selected, (b) `ChurnCateg` is selected, and (c) `Date` is selected. This way one may define any node as an entity and analyze data based on that entity.

5 Conclusions

We discuss the notion of data virtualization as the enabler of agile, efficient and effective data processing in modern analytics environments. We focus on the fact that such environments are characterized on one hand by tremendous heterogeneity of data, formats, systems etc. and on the other by dynamicity and different analysis needs by users with multifarious roles that may not have data management expertise. We set goals of data virtualization and we briefly present DVM, a graph-based data virtualization model that we have proposed.

References

1. Abadi, D., et al.: The Beckman report on database research. Commun. ACM **59**, 692–699 (2016)
2. Alagiannis, I., Borovica-Gajic, R., Branco, M., Idreos, S., Ailamaki, A.: Nodb: efficient query execution on raw data files. Commun. ACM **58**(12), 112–121 (2015)
3. Chatziantoniou, D., Kantere, V.: Data virtual machines: data-driven conceptual modeling of big data infrastructures. In: Workshops of EDBT 2020 (2020)
4. Chatziantoniou, D., Kantere, V.: Data virtual machines: a novel approach to data virtualization (2021, submitted for publication)
5. Chatziantoniou, D., Kantere, V.: Datamingler: a novel approach to data virtualization. In: Li, G., Li, Z., Idreos, S., Srivastava, D. (eds.) SIGMOD 2021: International Conference on Management of Data, Virtual Event, China, 20–25 June 2021, pp. 2681–2685. ACM (2021). https://doi.org/10.1145/3448016.3452752, https://doi.org/10.1145/3448016.3452752
6. Chatziantoniou, D., Tselai, F.: Introducing data connectivity in a big data web. In: Proceedings of the Third Workshop on Data analytics in the Cloud, DanaC 2014, pp. 7:1–7:4 (2014). http://doi.acm.org/10.1145/2627770.2627773
7. Denodo: Data virtualization: the modern data integration solution (2019). https://www.denodo.com/en/document/whitepaper/data-virtualization-modern-data-integration-solution
8. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann, San Francisco (2012)
9. Gartner: Market Guide for Data Virtualization (2018). https://www.gartner.com/en/documents/3893219/market-guide-for-data-virtualization
10. IBM: IBM's data virtualization tool: Cloud Pak for data (2021). https://www.ibm.com/analytics/data-virtualization
11. Karpathiotakis, M., Alagiannis, I., Heinis, T., Branco, M., Ailamaki, A.: Just-in-time data virtualization: lightweight data management with ViDa. In: CIDR 2015 (2015)
12. Microsoft: Introducing data virtualization with polybase (2021). https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide?view=sql-server-ver15
13. Oracle Corp.: Oracle Data Service Integrator (2020). https://www.oracle.com/middleware/technologies/data-service-integrator.html
14. Data virtualization and data warehousing (2020). https://en.wikipedia.org/wiki/Data_virtualization