# Towards Data Discovery by Example

El Kindi Rezig[1], Allan Vanterpool[1,2], Vijay Gadepally[3], Benjamin Price[3], Michael Cafarella[1], and Michael Stonebraker[1]

[1] MIT
{elkindi, michjc, stonebraker}@csail.mit.edu
[2] United States Air Force
{allan.vanterpool@us.af.mil}
[3] MIT Lincoln Laboratory
{vijayg,ben.price}@ll.mit.edu

**Abstract.** Data scientists today have to query an avalanche of multi-source data (e.g., data lakes, company databases) for diverse analytical tasks. Data discovery is labor-intensive as users have to find the right tables, and the combination thereof to answer their queries. Data discovery systems automatically find and link (e.g., joins) tables across various sources to aid users in finding the data they need. In this paper, we outline our ongoing efforts to build a data discovery by example system, *DICE*, that iteratively searches for new tables guided by user-provided data examples. Additionally, *DICE* asks users to validate results to improve the discovery process over multiple iterations.

**Keywords:** Data Preparation · Data discovery · Data integration · Data cleaning

## 1 Introduction

Developing an end-to-end analytic pipeline consists of a number of different operations on the data. One component that is widely recognized as a bottleneck for data scientists, is the time spent discovering and preparing the data needed for their analyses [8, 9]. A key step in prepartion is finding and linking the relevant datasets one would need for the task at hand, which when performed manually tends to be both onerous and error-prone [2].

Due to the importance of data discovery, there have been several recent efforts to facilitate it [2, 3, 6]. Aurum[2] exposes a query API for users to query a graph representation of the data, which captures column similarity and PK-FK relationships. However, the burden is still on the user to write the code to discover and collect the relevant data. In another approach, DoD[4] asks users to provide a schema of the view they are looking for, and the system attempts to find the required joins necessary to produce it. Auto-Join [11] performs transformations on data columns to make them match other key columns and hence make them joinable. However, Auto-Join does not deal with querying the produced data.

Consider the tables in Figure 1. Suppose we have the following query $Q_1$: "Get me the list of papers co-authored by DB faculty at CSAIL and non-CSAIL

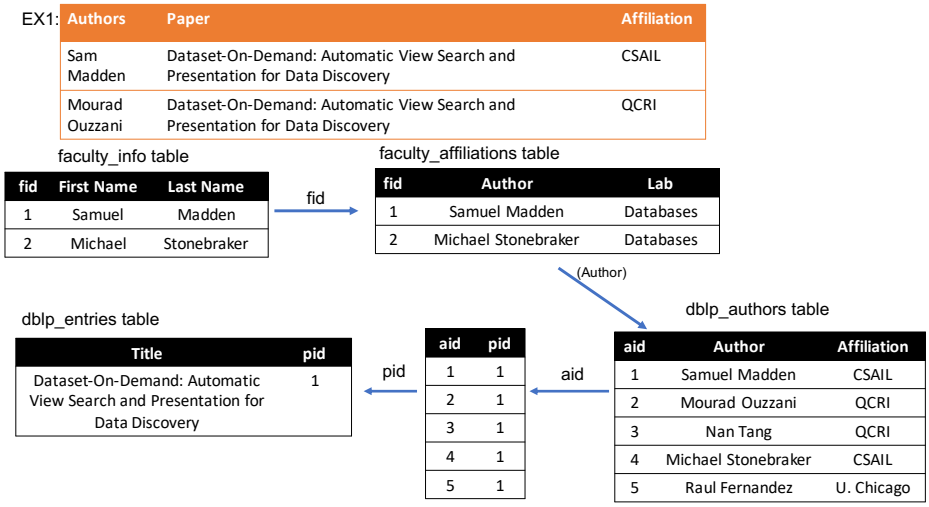


**Fig. 1.** Example PK-FK join path across different tables

authors". If the user knows every table in the data lake, then, it's straightforward to write a SQL query to answer the query. However, the number of tables in data lakes can be unbounded, which renders manual inspection impractical. A data discovery system that can aid in finding the PK-FK join paths (arrows in Figure 1) makes it easier for a user to know which tables join to what others, and then the user can write queries on top of that pre-processed data. However, the user still needs to write the queries, which in many cases can be time-consuming and cumbersome (i.e., the user does not know all the tables, and the relationships thereof), based on the level of depth and complexity of those relationships.

We are working on *DICE* (Data Discovery by Example), a system that enables data discovery by example, wherein the user provides a set of desired records as examples, and the system then automatically fetches the relevant tables/columns, which would require doing joins between columns of different tables. For instance, to express $Q_1$, the user could provide two example records of the desired output (EX1 in Figure 1) from which *DICE* extracts values and properties (e.g., *DICE* would notice that the values in affiliation can be different in EX1) to look up relevant tables in the data lake, and then construct join paths. The lookup function in DICE supports both exact and similarity matching.

From our ongoing engagement with one large organization, we have learned that being able to search data by example would be a highly desirable improvement for data analysts, as opposed to their current method of writing queries which requires explicit knowledge of the underlying data organization.

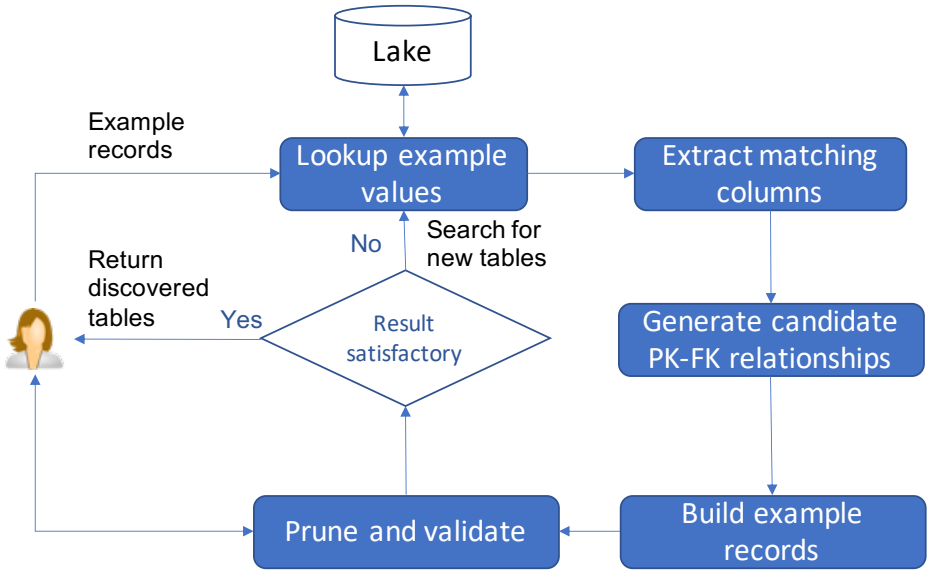


**Fig. 2.** General workflow of DICE

## 2 System Overview

Figure 2 outlines the overall workflow of *DICE*. In a nutshell, *DICE* does not perform data discovery in one shot, but instead is an interactive system. DICE interacts with the user in two ways: (1) by requesting example records; and (2) by asking them to validate constructed join paths. After each iteration the user can decide to either stop the search, or prompt *DICE* to look for more candidate tables and join paths.

As mentioned above, the first step is for the user to provide a table (see EX1 in Figure 1) as input containing a set of example records. After extracting the values and properties from the example, *DICE* fetches relevant tables from the data lake. *DICE* then attempts to construct PK-FK relationships from the columns of the extracted tables. *DICE* then constructs sample records from the selected join paths, presents them to the user and based on the user's (Yes/No) feedback on each record, fetches more tables or stops the search if the user is satisfied with the results. The user may also supply additional examples after each iteration. Given this design we note the following challenges:

– **Extracting the values/properties from the user's example records:** Interpreting example records can be ambiguous. For instance, in Figure 1, the example EX1 could mean: "get me all papers co-authored by DB CSAIL faculty and non-CSAIL authors" or "get me all papers written by anybody in Databases" or "get me all papers co-authored by Samuel Madden and

Mourad Ouzzani". Therefore, primitives are needed so the user can express their "intent" over the provided examples.
- **Initial search region:** Initially, *DICE* has to fetch a set of tables, and try to link them together. If too many tables match the user's examples, choosing the *correct* subset of tables for the user to consider first, can be challenging. The initial search region (first tables to consider in the lake) is key because *DICE* builds on it for its subsequent searches by expanding it to include more tables.
- **Minimizing the user's input:** *DICE* needs to actively involve the user by selecting "interesting" and relevant records for validation. Additionally, *DICE* will need to be able to ask the user for more examples if the current examples are too broad or limited for the search. It is crucial to minimize the user's interactions while attempting to maximize the value of the user's feedback, when needed.

### 2.1 Knowledge Graphs

In addition to the relational data model, we are also looking into how *DICE* could be used in other data models. For instance, a knowledge graph encodes entities (nodes) and the properties that connect them (edges). In this case, *DICE* is not looking for PK-FK relationships, but for any relationship between nodes in the graph (paths). This makes the problem more challenging because *DICE* has to select only relevant paths to consider (which would require a measure of *interestingness* attributed to different paths). Once determined, *DICE* can then build records from those paths to present samples to the user.

## 3 Benefits of *DICE*

Using a discovery tool such as *DICE*, can provide users with a number of benefits:

- **Simplifying data discovery:** By allowing users to specify examples of the data output they would like, *DICE* greatly reduces the number of iterations between the user and system. The user helps define the high-level data discovery task and the system provides viable alternative outputs for the user to choose between, allowing for a quick narrow-down to the most relevant alternative.
- **Improving data aggregate sensitivity:** In many computing environments, while individual records may not present concerns, aggregation of different types of data can lead to policy or security violations. For example, integration of different non-sensitive datasets may lead to sensitive outputs. Consider a medical scenario in which a user is trying to integrate data from *Table 1* and *Table 2*. Suppose *Table 1* consists of columns (PatientID, Patient Name, Patient Gender) and *Table 2* consists of columns (PatientID, Patient Date of Birth, Patient Age). If a researcher is interested in looking

at the distribution of gender and age within the datasets, they may attempt to do a "join" on Table 1 and Table 2 using the PatientID. Very often, combining Patient Name and Date of Birth can lead to sensitive Personally Identifiable Information (PII). Using a tool like *DICE*, the user could specify *a priori* that they are looking only for (Gender, Age) and *DICE* can filter data appropriately before aggregating.

- **Improved policy compliance:** In many environments, data is distributed across different systems with different owners. These data owners may not wish to provide unfettered access to traditional data discovery tools. In which case, they could expose only a subset of their data or metadata about their system. *DICE* could use this limited information to construct "safely" pruned datasets for the data scientist. If any of these are of interest, the data scientist can then reach out to data owners for greater access.
- **Working well within polystore or multistore environments:** Data distributed across heterogeneous systems, as seen in polystores [5] and multistore environments [10], pose new challenges for data discovery. A tool such as *DICE* would improve data integration across these heterogeneous systems. For example, the applications described in [7, 1], leverage data stored in disparate data management systems. For a data scientist to find data of interest, they must manually query each individual system (or write a polystore query that speaks to multiple, different backend systems). In this case as well, *DICE* could create these queries for the data scientist in order to help them find the specific data products of interest.

## 4 Conclusion

*DICE* is an interactive, user-in-the-loop, data discovery by example system. In this paper we presented a quick walkthrough of the workflow of *DICE* as well as the challenges associated with its implementation. We are currently working with one large-scale organization to implement *DICE* on their data lake, but believe that *DICE* may have widespread value and application to other organizations as data storage continues to grow in both complexity and distribution.

## Acknowledgement

Research was sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## References


1. Elmore, A.J., Duggan, J., Stonebraker, M., Balazinska, M., Cetintemel, U., Gadepally, V., Heer, J., Howe, B., Kepner, J., Kraska, T., et al.: A demonstration of the bigdawg polystore system. Proceedings of the VLDB Endowment **8**(12), 1908 (2015)
2. Fernandez, R.C., Abedjan, Z., Koko, F., Yuan, G., Madden, S., Stonebraker, M.: Aurum: A data discovery system. In: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018. pp. 1001–1012. IEEE Computer Society (2018). https://doi.org/10.1109/ICDE.2018.00094, https://doi.org/10.1109/ICDE.2018.00094
3. Fernandez, R.C., Mansour, E., Qahtan, A.A., Elmagarmid, A.K., Ilyas, I.F., Madden, S., Ouzzani, M., Stonebraker, M., Tang, N.: Seeping semantics: Linking datasets using word embeddings for data discovery. In: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018. pp. 989–1000. IEEE Computer Society (2018). https://doi.org/10.1109/ICDE.2018.00093, https://doi.org/10.1109/ICDE.2018.00093
4. Fernandez, R.C., Tang, N., Ouzzani, M., Stonebraker, M., Madden, S.: Dataset-on-demand: Automatic view search and presentation for data discovery. CoRR **abs/1911.11876** (2019), http://arxiv.org/abs/1911.11876
5. Gadepally, V., Chen, P., Duggan, J., Elmore, A., Haynes, B., Kepner, J., Madden, S., Mattson, T., Stonebraker, M.: The bigdawg polystore system and architecture. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2016)
6. Halevy, A.Y., Korn, F., Noy, N.F., Olston, C., Polyzotis, N., Roy, S., Whang, S.E.: Goods: Organizing google's datasets. In: Özcan, F., Koutrika, G., Madden, S. (eds.) Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. pp. 795–806. ACM (2016). https://doi.org/10.1145/2882903.2903730, https://doi.org/10.1145/2882903.2903730
7. Mattson, T., Gadepally, V., She, Z., Dziedzic, A., Parkhurst, J.: Demonstrating the bigdawg polystore system for ocean metagenomics analysis. In: CIDR (2017)
8. Rezig, E.K., Cao, L., Stonebraker, M., Simonini, G., Tao, W., Madden, S., Ouzzani, M., Tang, N., Elmagarmid, A.K.: Data civilizer 2.0: A holistic framework for data preparation and analytics. PVLDB **12**(12), 1954–1957 (2019). https://doi.org/10.14778/3352063.3352108, http://www.vldb.org/pvldb/vol12/p1954-rezig.pdf
9. Rezig, E., Cafarella, M., Gadepally, V.: Technical report: An overview of data integration and preparation (2020)
10. Tan, R., Chirkova, R., Gadepally, V., Mattson, T.G.: Enabling query processing across heterogeneous data models: A survey. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 3211–3220. IEEE (2017)
11. Zhu, E., He, Y., Chaudhuri, S.: Auto-join: Joining tables by leveraging transformations. Proc. VLDB Endow. **10**(10), 1034–1045 (2017). https://doi.org/10.14778/3115404.3115409, http://www.vldb.org/pvldb/vol10/p1034-he.pdf