Kyrix-J: Visual Discovery of Connected Datasets in a Data Lake

Wenbo Tao
wenbo@mit.edu
MIT CSAIL, Cambridge, MA

Adam Sah
asah@gmail.com
Independent

Leilani Battle
leibatt@cs.washington.edu
University of Washington, Seattle, WA

Remco Chang
remco@cs.tufts.edu
Tufts University, Boston, MA

Michael Stonebraker
stonebraker@csail.mit.edu
MIT CSAIL, Cambridge, MA

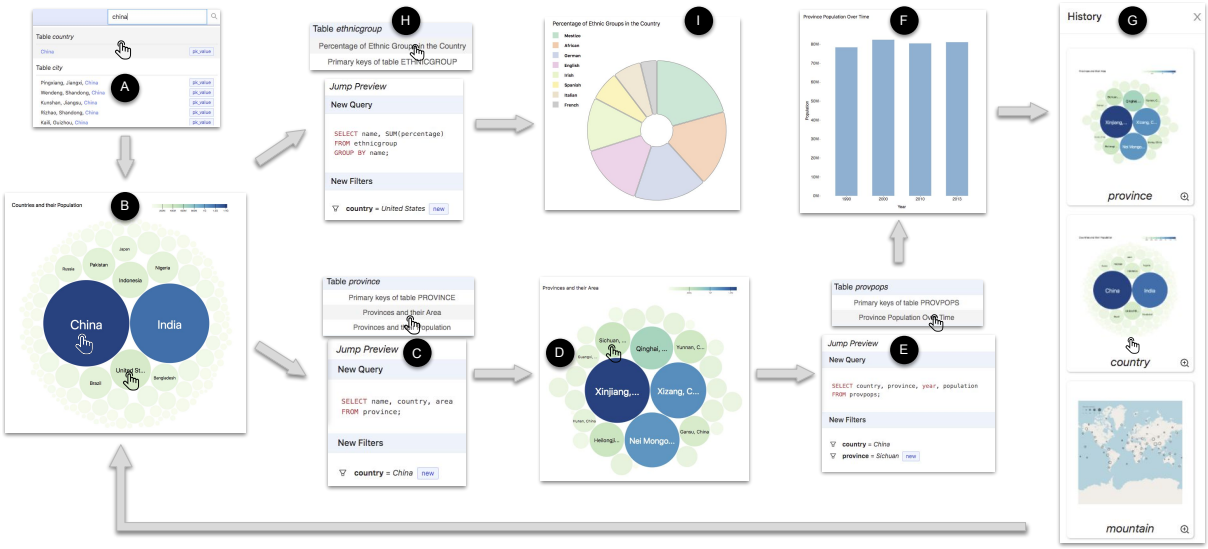

Figure 1: Kyrix-J facilitates the discovery of connected datasets in a data lake through interactive visualizations. Here we show a typical usage flow of Kyrix-J deployed on the MONDIAL database[13] which involves identifying a table of interest using the keyword search (a), inspecting data visualizations (b, d, f and i), performing "jumps" between visualizations (b→c→d, d→e→f or b→h→i) and using the history panel to go back to a visited visualization (g).

ABSTRACT

Understanding data in large data lakes is becoming increasingly challenging. While some existing systems help with data discovery in data lakes, they are limited in surfacing connections between datasets and helping users comprehend them, which is crucial for many applications. To this end, we have built a system called Kyrix-J. Kyrix-J uses interactive visualizations to enable rapid discovery of connected datasets in data lakes. We allow a user to "jump" from one visualization to another following connections between the underlying data. Kyrix-J automatically generates these jumps so that the user can start using the system without a manual app-authoring process. We also contribute a novel user interface with Kyrix-J which facilitates a variety of database exploration tasks. Finally, we conduct user study which shows that Kyrix-J is easy to use and allows the users to effortlessly explore connected datasets in a data lake.

ACM Reference Format:
Wenbo Tao, Adam Sah, Leilani Battle, Remco Chang, and Michael Stonebraker. 2021. Kyrix-J: Visual Discovery of Connected Datasets in a Data Lake. In *Proceedings of CIDR '22: Conference on Innovative Data Systems Research.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

CIDR '22: Conference on Innovative Data Systems Research, Jan 10–13, 2022, Chaminade, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We are witnessing an explosion in the amounts of data being generated across domains. Within an organization, there are usually lots of datasets that come from a variety of sources and are of different quality and formats. Collectively, these datasets are often termed "data lakes" [9, 11, 14, 19], which implies that the users of these datasets are "drowned" in them, i.e., not being able to discover and make sense of the datasets easily.

Datasets in a data lake are typically highly interconnected, which poses a major challenge to the users. Relational datasets are a ubiquitous example where data tables represent different entities in the business logic, and primary key-foreign key (PK-FK) links capture the relationships between these entities[25]. There can be hundreds to thousands of tables (sometimes without even a defined schema), which makes it hard to comprehend what is in the data. For example, the MIT data warehouse stores all information about entities at MIT. It is a data lake with over 2,000 tables and even more PK-FK connections, most of which the DBAs do not understand. Another classic example of data connections involves derived datasets in a data warehouse, where different teams within an organization make derivations of the fact tables for different analytic purposes. The derived tables commonly have a large overlap, which forms data lineages[12]. Understanding those lineages is challenging yet crucial for many use cases such as protecting user privacy[6] and determining data staleness[10].

Given the need to understand datasets and their connections in data lakes, organizations need effective data discovery systems. Several existing systems[2, 9, 11] enable text-search-based data discovery. Yet their ability to support exploration of connections between datasets is limited. Multiple lines of code or actions often need to be taken to see one connected dataset. We posit that in order to enable the user to fully understand data in data lakes, we need to make those connections first-class citizens, i.e., to make traversing along data connections *effortless* and to make it obvious what datasets are connected.

In this paper, we present the design of Kyrix-J, a system that enables visual data discovery for connected datasets in a data lake. Kyrix-J employs data visualizations to present data and their relationships in a visual way, which is a key distinction compared to prior data discovery systems. In particular, we make use of a "jump" interaction which allows the user to quickly navigate between visualizations of connected datasets.

We illustrate jumps in Kyrix-J through an example usage scenario (Figure 1) based on a public relational database called MONDIAL[13][1], which has 40 tables that correspond to real world entities (e.g. `country`, `lake`, `island`) and their connections (e.g. `lakeonisland`).

1.1 A Usage Scenario

To start, a user types the search term `China`, a keyword of interest, into a search box, sees a list of tables matching this keyword and then clicks on the table `country` (Figure 1a). The visualization view in the UI then switches to the default visualization for the table `country`, which is a circle pack visualization of countries where the color and size of circles encode the population (Figure 1b).

The user then clicks on the circle with the label `China` and sees a menu showing a list of possible "jumps" from the current visualization (Figure 1c), which are automatically generated by Kyrix-J. The user chooses to perform a jump to the visualization called `Provinces and their Area` for the table `province`. Prior to the jump, another popover window appears upon hovering over the jump option, allowing the user to examine the SQL query and filters of the new visualization (Figure 1c).

[1] A demo can be accessed here: http://104.197.126.112:4000/

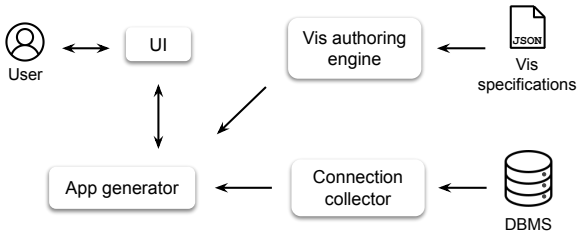

Figure 2: The Kyrix-J system architecture.

The target visualization (Figure 1d) is again a circle pack visualization showing provinces in China and their area, in which the user is interested in the `Sichuan` province. After clicking on `Sichuan`, the user decides to jump to a new visualization `Province Population Over Time` for the `provpops` table (Figure 1e) to see the population of `Sichuan` over time in a bar chart (Figure 1f).

Next, the user decides to see something about `United States`. Recalling seeing `United States` in the visualization in Figure 1b, the user opens up the history panel (Figure 1g) and clicks on the item containing that visualization to go back to Figure 1b. From there, the user clicks on `United States` and goes to the visualization `Percentage of Ethnic Groups in the Country` for the `ethnicgroup` table. The new visualization shows what ethnic groups there are in the `United States` and what their percentages are.

In this example, the user has performed two chains of jumps (b → c → d → e → f and b → h → i in Figure 1), each of which allows drilling into details of the data tables and seeing different facets of the visualizations.

1.2 Key Contributions

A key contribution of Kyrix-J is an *automatic* mechanism to connect visualizations through jumps (Section 5). No manual intervention is required to start using the system for a new data lake. The jumps are presented in a novel user interface (Section 6) which supports the user in performing common database exploration tasks.

We have built a prototype system and conducted a user study (Section 7). The results indicate that Kyrix-J is easy to use and learn and helps the participants effortlessly understand the relationship between datasets.

2 SYSTEM ARCHITECTURE

From the user's point of view, Kyrix-J should offer an easy-to-use interface which enables rapid discovery of connected data. To support this user interface, Kyrix-J has a few components which perform pre-processing steps such as collecting data connections, facilitating visualization authoring and building jump paths. Figure 2 shows the system architecture.

First and foremost, we assume that the data resides in a DBMS. As shown in Figure 2, there are three main components in addition to the UI, which provides the pre-processing and services needed to render the UI.

Connection profiler. Although some built-in connections between datasets may exist (e.g. PK-FK links), in the real-world data connections are often not given by default. The connection profiler

identifies data connections that are useful to show to the users. See Section 3 for details.

Vis authoring engine. We provide an engine for users to author visualizations of data in the DBMS if desired. This engine is based on a prior authoring system called Kyrix[24] we have built. We detail it in Section 4.

App generator. To enable the users to perform jumps along data connections, the app generator automatically generates meaningful jump paths between data visualizations. The app generator also serves as a server which sends necessary data to the client. In Section 5 we detail the auto-jump solution.

Last but not least, we describe our UI in greater details in Section 6. A user study of this UI is presented in Section 7.

3 PROFILING DATA CONNECTIONS

We use a simple model to characterize connected datasets in a data lake, where the data is organized into tables (such as those in an RDBMS), and each connection links two columns from two tables that are semantically similar/equivalent.[2] This type of connections often do not exist beforehand, although in some cases they are available (e.g. PK-FK links in an RDBMS schema).

There is an extensive literature on identifying similar columns between data tables. Many commercial and open source data integration tools also offer such capability (e.g. Aurum[9], Tamr[21] and Google's Goods[11]). These systems typically capture column similarities by building *column profiles*, which summarize the content of columns using set similarity metrics or ML-based embeddings. A similarity search/join process is then performed to identify top-k similar column pairs.

Kyrix-J treats the connection profiler as a black box that can be implemented differently depending on the use case. By default, the Aurum system[9] can be used as a generic method to identify similar or equivalent columns. Aurum uses a two-step process which first summarizes each column into a space-efficient signature, and then employs sketching techniques to identify column matches in linear time. When a custom-method is used, the general guideline is that the module needs to be scalable, and should not have a time complexity that is quadratic in the number of columns.

4 SUPPORTING VISUALIZATION AUTHORING

In Kyrix-J, we use data visualizations to present useful information about tables. The jumps between data visualizations bring the user from one visualization to another, enabling a visual data discovery experience. To support quick cold starts, we provide default visualizations for each table in the form of a word cloud or a tabular visualization.

To enable the creation of custom visualizations, we also provide a declarative language which the user can use to author a new visualization in tens of lines of JSON code. In our prototype, we support five custom visualizations: pie charts, bar charts, circle packs, treemaps and scatterplots.[3] This visualization language is implemented on top of the Kyrix system[23, 24], which is a more low-level visualization authoring system we have built. To support more types of visualizations, we plan to integrate with established visualization languages such as Vega-lite[17] in the future.

Currently, we offer a JSON-based interface where the user compiles a custom visualization into the application using the command line. It is our future work to support authoring visualizations directly in the UI with direct manipulation (e.g. similar to how Tableau[20] works).

5 AUTOMATIC GENERATION OF JUMPS

Given a set of visualizations, Kyrix-J aims to automatically identify a set of jumps, each of which connects a pair of visualizations. As illustrated in Section 1.1, a jump essentially allows the user to select one object in one visualization, use the selected object to filter the jump-to visualization, and repeatedly do so to perform drill downs. For each jump, there is an implicit join between two columns from the tables that the visualizations are based on. Naturally, we can construct the jumps based on the similar columns identified by the connection profiler. In this section, we formally define the problem and present a simple solution based on this idea.

5.1 Problem Definition

Here, we use the *auto jump problem* to refer to the problem of automatically identifying jumps between visualizations. To formally define the auto jump problem, we first introduce a few definitions.

Definition 1 (Visualizations). *A visualization V is a tuple (V_Q, V_D, V_T, V_{PK}). V_Q is the SQL query used to fetch the data items for V from the DBMS. V_D represents the query results of V_Q. V_T denotes the table used in V_Q. The primary key of a visualization V, denoted as V_{PK}, is the set of data fields in V_D which uniquely identify each object in V.*

To get V_{PK}, i.e., the primary key for each visualization V, we use V_Q, i.e., the SQL query of V. If V_Q is a SQL `GROUP BY` aggregation query, we use the `GROUP BY` columns in V_Q as V_{PK}; otherwise we use the primary key of V_T as V_{PK}.[4]

We illustrate Definition 1 with Figure 3b. V_2 is a bar chart showing countries in South America and their area. V_{2_T} is a table called `encompasses` where each data item denotes one continent encompassing a country and has a corresponding attribute `area` indicating the area of the country. V_{2_Q} is a SQL `GROUP BY` query which groups all data items by country and also computes the area of the country using the `MAX` aggregate.[5] Thus $V_{2_{PK}}$ is the `GROUP BY` columns of V_{2_Q}, i.e., the `country` column in table `encompasses`. This suggests that each object (bar) in V_2 is a country.

Next, we define a filter function.

[2] Connections on the table level or record-level may also be of interest, but we focus on column-level connections in this paper.

[3] See demos here: https://github.com/tracyhenry/Kyrix/wiki/Template-API-Reference#static-aggregations

[4] We make two assumptions to simplify the presentation. First, each DBMS table has exactly one primary key. Our techniques easily extend to the case where a table has multiple primary keys. Second, the SQL query for a visualization is either a `GROUP BY` query, or simply a `SELECT` query which selects some fields from a table. If a SQL query is too complex for this assumption, we assume that an expert (e.g. a database administrator) materializes the query result into a table and rewrites the SQL query to fit this assumption.

[5] Data items in the same group have the same area. The `MAX` aggregate is only used to get that number, and could be replaced by `MIN` or `AVG`.

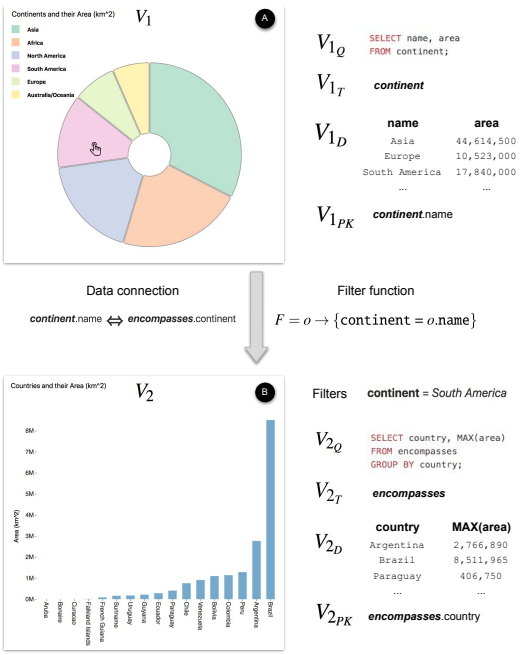

Figure 3: An example jump with annotations of the SQL queries, data items, primary keys of the visualizations, the filter function and the corresponding data connection.

Definition 2 (Filter Functions). *A filter function F takes a visual object o as input, and outputs a set of filters. Each filter is in the form $b = o.a$ where b is one column and $o.a$ is the value of the a field of o.*

A jump is then defined as follows.

Definition 3 (Jump). *A jump from V_1 to V_2 is a tuple (V_1, V_2, F) that satisfies the following: 1) V_1 and V_2 are visualizations; 2) F is a filter function; and 3) the output of F when applied on an object in V_1 satisfies the following: a) there are $|V_{1_{PK}}|$ filters in total; b) each column in $V_{1_{PK}}$ appears on the right-hand side of exactly one filter; and c) the left-hand side of each filter is a column in V_{2_T}.*

In other words, requirement 3) says that the resulting filters of F applied on an object in V_1 form a one-to-one mapping from columns in $V_{1_{PK}}$ to a subset of columns in V_{2_T}.

Consider the example jump in Figure 3 where a user selects `South America` in the pie chart V_1 and jumps to V_2 to see countries in `South America` and their area. The filter function returns `continent` = o.`name` when applied on an object o in V_1, which forms a one-to-one mapping from the only column in $V_{1_{PK}}$ (the `name` column in table `continent`) to one column in V_{2_T} (the `continent` column in table `encompasses`). When the user selects the object `South America` to start the jump, the filter function returns the filter `continent` = `South America`.

While Definition 3 is useful in characterizing a jump, note that there could be many possible filter functions that map columns in $V_{1_{PK}}$ to columns in V_{2_T}. Not every mapping is useful. Therefore, we need to define what constitutes a meaningful jump.

Definition 4 (Meaningful Jumps). *We say that a jump (V_1, V_2, F) is meaningful, if and only if any filter generated by F connects two columns from V_{1_D} and V_{2_D} that are semantically equivalent, i.e., representing the same real-world entity.*

For example, the filter `continent` = o.`name` makes the jump in Figure 3 meaningful because both columns represent continents. Yet if the filter function is changed to return `country` = o.`name`, the jump would not be meaningful.

With Definitions 1-4, we can now define the auto jump problem.

Definition 5 (The Auto Jump Problem). *Given a set of visualizations $\mathbb{V}$, identify all meaningful jumps (V_1, V_2, F) such that $V_1 \in \mathbb{V}$ and $V_2 \in \mathbb{V}$.*

5.2 The Auto-Jump Solution

The auto-jump solution makes use of data connections between columns identified by the connection profiler. The high-level strategy is to search for meaningful jumps between *all pair of visualizations*. For each pair V_1 and V_2, we attempt to find one-to-one mappings from $V_{1_{PK}}$ to V_{2_T} so that each pair of columns being mapped are semantically equivalent according to the connection profiler. We can then get a meaningful jump with each such mapping by constructing a filter function (Definition 4).

When we apply this solution to the example MONDIAL database and get over 1,000 meaningful jumps between 70 visualizations for 40 tables.

6 USER INTERFACE

The high-level goal of the UI is to present the jumps generated by the algorithm in Section 5 in an accessible way that facilitates rapid discovery of data connections. Specifically, we should make it easy to browse a large number of such jump paths and also help users stay oriented during their exploration.

The Kyrix-J UI features multiple coordinated views (Figure 4). The keyword search box (Figure 4a) allows a user to identify a table to start their exploration. The visualization view (Figure 4b) presents one visualization at a time. The graph view (Figure 4c) is a simplified Entity-Relationship diagram showing each table as a graph node and data connections between tables as graph edges. When hovering over a node/edge in the graph view, more information shows up in a popover (Figures 4i-j). Informational views (Figures 4d-f) show the current SQL query, filters and mappings from visual properties to data attributes. Figure 4g is a popover appearing after the user clicks on an object, which contains a list of jumps automatically generated by Kyrix-J. Figure 4h shows a list of bookmarked visualizations. In Figure 4k, data items show up in a tabular format after the user clicks on the "raw data" button.

7 USER STUDY

We conducted an observational first-use study to evaluate the usability of Kyrix-J with eight participants (5 females, 3 males, age range 21-55, diverse experiences in DBMS and visualization technologies). We asked the participants to complete five search tasks (Table 1) using Kyrix-J and recorded the time taken and the accuracy. After the completion of the tasks, we collected free-form feedback from the participant in a semi-structured interview. The study concluded with the participants filling out a post-study questionnaire where they rated how much they agree with statements about the

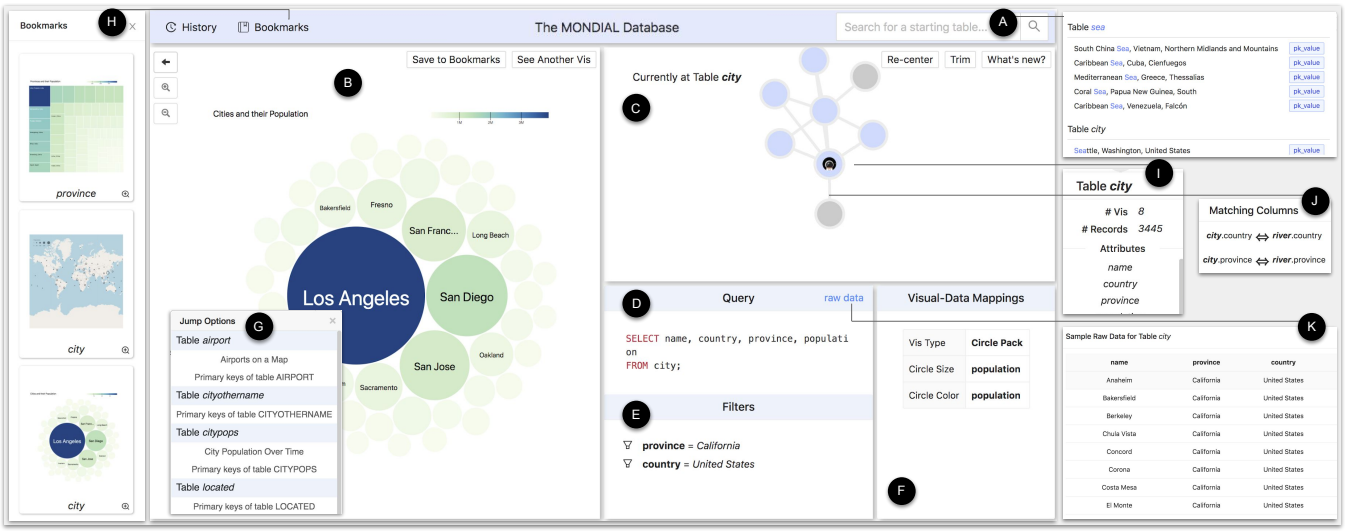

Figure 4: The Kyrix-J user interface

Table 1: Search tasks used in the user study.

Task 1	Is the following statement true? The 3rd largest ethnic group in the 4th largest country in the South America continent is European.
Task 2	a: Find the 4th most populated country in the world. b: Find the most populated city in that country. c: Find one organization headquartered at that city.
Task 3	Find the population (in 2011) of the 3rd most populated city in the most populated province in the world.
Task 4	Is the following statement true? Ukerewe is an island in the 3rd largest lake in the world.
Task 5	a: Find the country with the 2nd largest average city elevation. b: Find the 2nd most spoken language in that country.

usability of Kyrix-J on a 5-point Likert scale (1 – strongly disagree and 5 – strongly agree).

7.1 Results and Discussions

Task completion. All participants were able to complete all tasks with minimal guidance. The completion times for the tasks are shown in Table 2. The average total completion time for all tasks was 11.4 minutes.

Completions of the tasks were mostly accurate. Only two participants got one task wrong on the first try, which was due to inaccurate numerical comparison (e.g. mistaking the 4th lengthiest bar for the 3rd) and misunderstanding of the task. Both were able to quickly correct themselves.

Ease of use and learning. In the questionnaire, participants rated that Kyrix-J was easy to use ($\mu = 4.63, \sigma = 0.52$) and easy to learn ($\mu = 4.75, \sigma = 0.46$). Many participants gave positive comments on

Table 2: Average task completion times and standard deviations (in minutes).

	Task 1	Task 2	Task 3	Task 4	Task 5
μ	2.2	1.7	1.8	1.7	4.1
σ	0.6	0.7	0.6	1.0	2.0

the overall experience: *"it's very natural to jump from thing to thing* (P1)*"*, *"this (Kyrix-J) helps me figure out what tables are related to each other, what are the PK-FK relationships, where am I in filtering* (P2)*"*, *"the jumps help data discovery* (P7)*"*, *"I want to use this tool for my personal use if I can* (P8)*"*. Participants especially liked the ability to browse a lot of jump paths: *"I like the options for drill down (jumps), you can have multiple options so you are not limited* (P6)*"*, *"it helps showcase the possible ways you can filter different tables* (P7)*"*, *"the pace at which I can quickly jump from one visualization to another is amazing* (P8)*"*.

Comparison with alternative techniques. Not all participants had prior experiences with jumps, but they still provided insights on what alternative techniques they might use to complete similar tasks. Many participants mentioned using the SQL command-line interface offered by the DBMS as an alternative. However, they also pointed out that using a graphical interface like Kyrix-J would be easier and more intuitive: *"that (using a SQL command-line interface) would require knowing the schema a lot better. I need to pay a lot more attention* (P1)*"*, *"it (Kyrix-J) automates all the painful coding you have to do* (P2)*"*, *"I don't have to write 10 to 20 lines of code, everything (in Kyrix-J) is a click away. It's like shopping on Amazon* (P4)*"*.

One participant (P6) had used a variety of tools (e.g. Tableau[20], Chartio[1] and SSIS[4]) to perform jump-based data discovery in professional settings. She noted that a common limitation of the tools she used was that they only supported jumps in the same table: *"going to another chart is challenging because when they (tool creators) are building it, they often will grab the only the values you can find in that chart (table). When you want to grab something*

outside that chart (table), it's not easy to do". P6 further commented that *"Having drill down (jump) like Kyrix-J in one of those tools I'm using would be amazing."* P8 also compared using Tableau to do jumps versus using Kyrix-J and echoed that Kyrix-J offers a better experience.

8 RELATED WORKS

Prior data discovery systems such Aurum[9], Lyft's Amundsen[2] and Google's Goods[11] mainly allow users to discover data through text search. To our best knowledge, Kyrix-J is the first data discovery system that focuses on visual navigation of data connections.

A number of visualization systems[3, 5, 7, 8, 15, 16, 18, 20, 24] have been developed to support visual jumps. These systems mostly focus on jumps between visualizations for one data table. As such, they generally cannot be easily extended to a multi-table data lake setting. Moreover, *pivoted search* or simply *pivoting* are commonly used terms. In our data lake setting, we use the term *jump* to indicate that the interaction "jumps" across the table boundaries.

Kyrix[24] is a details-on-demand system we have built that can also support jumps. However, it requires writing tens of lines of Javascript to create a jump. Also, it does not offer UI support to help users stay oriented. Kyrix-J advances Kyrix by automatically generating jumps based on data connections, and offering a novel UI to enable effective exploration of a data lake.

9 DISCUSSIONS ON LIMITATIONS AND FUTURE WORK

Enabling visualization authoring. A desirable feature is to enable authoring/editing visualizations in the UI. Although it is straightforward to bring the current JSON-based command-line authoring paradigm into the UI, it currently only supports simple `SELECT` and `GROUP BY` queries as discussed in Section 5.1. We plan to lift this restriction in the future to support more types of SQL queries.

Tackling data lakes in the wild. Data lakes in the wild are very complex, consisting of a large number of tables that often lack maintenance[22]. To deploy Kyrix-J in the wild, we face the following challenges.

First, table and column names are often machine-generated. The quality of visualization titles may not be high if they are authored by users in a multi-user environment. These will make it harder to comprehend the meaning of a jump. We plan to address this challenge by integrating with metadata management systems (e.g. Google's Goods[11] and Lyft's Amundsen[2]) to ensure that popular tables are more visible to the users and get more quality crowdsourced annotations of tables and columns. We will also investigate quality control mechanisms such as allowing custom-made data visualizations by users to be rated by a pool of users so that we can make highly-rated jumps more visible and vice versa.

Second, the amount of jumps will grow tremendously at scale when there are hundreds of even thousands of visualizations. The current way of browsing the jumps is limiting in that it requires a user to scroll through a list of jumps. To enable more efficient browsing of the jumps, we plan to offer a functionality to allow the user to search a jump using keywords.

REFERENCES

[1] Chartio. https://chartio.com/. accessed: 2021/03.
[2] Lyft amundsen. https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9. accessed: 2021/03.
[3] More powerful data drilling. https://help.looker.com/hc/en-us/articles/360023589613--More-Powerful-Data-Drilling. accessed: 2021/03.
[4] Sql server integration services. https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver15. accessed: 2021/03.
[5] C. Ahlberg. Spotfire: an information exploration environment. *ACM SIGMOD Record*, 25(4):25–29, 1996.
[6] M. Backes, N. Grimm, and A. Kate. Data lineage in malicious environments. *IEEE Transactions on Dependable and Secure Computing*, 13(2):178–191, 2015.
[7] M. Dörk, N. H. Riche, G. Ramos, and S. Dumais. Pivotpaths: Strolling through faceted information spaces. *IEEE transactions on visualization and computer graphics*, 18(12):2709–2718, 2012.
[8] C. Dunne, N. Henry Riche, B. Lee, R. Metoyer, and G. Robertson. Graphtrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 1663–1672, 2012.
[9] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1001–1012. IEEE, 2018.
[10] A. Gupta, I. S. Mumick, et al. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.
[11] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 795–806, 2016.
[12] R. Ikeda and J. Widom. Data lineage: A survey. Technical report, Stanford InfoLab, 2009.
[13] W. May. Information extraction and integration with Florid: The Mondial case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from http://dbis.informatik.uni-goettingen.de/Mondial.
[14] N. Miloslavskaya and A. Tolstoy. Big data, fast data and data lake concepts. *Procedia Computer Science*, 88:300–305, 2016.
[15] C. North and B. Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the working conference on Advanced visual interfaces*, pp. 128–135, 2000.
[16] C. Partl, A. Lex, M. Streit, H. Strobelt, A.-M. Wassermann, H. Pfister, and D. Schmalstieg. Contour: data-driven exploration of multi-relational datasets for drug discovery. *IEEE transactions on visualization and computer graphics*, 20(12):1883–1892, 2014.
[17] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.
[18] M. Spenke and C. Beilken. Infozoom-analysing formula one racing results with an interactive data mining and visualisation tool. *WIT Transactions on Information and Communication Technologies*, 25, 2000.
[19] B. Stein and A. Morrison. The enterprise data lake: Better integration and deeper analytics. *PwC Technology Forecast: Rethinking integration*, 1(1-9):18, 2014.
[20] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.
[21] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *Cidr*, vol. 2013. Citeseer, 2013.
[22] M. Stonebraker, D. Deng, and M. L. Brodie. Database decay and how to avoid it. In *2016 IEEE International Conference on Big Data (Big Data)*, pp. 7–16. IEEE, 2016.
[23] W. Tao, X. Hou, A. Sah, L. Battle, R. Chang, and M. Stonebraker. Kyrix-s: Authoring scalable scatterplot visualizations of big data. *IEEE Transactions on Visualization and Computer Graphics*, 2020.
[24] W. Tao, X. Liu, Y. Wang, L. Battle, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive pan/zoom visualizations at scale. In *Computer Graphics Forum*, vol. 38, pp. 529–540. Wiley Online Library, 2019.
[25] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, 3(1-2):805–814, 2010.