

Turning Open Government Data Portals into Interactive Databases

by

Chang Liu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

© Chang Liu 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The launch of open governmental data portals (OGDPs), such as `data.gov`, `data.gov.in`, and `open.canada.ca`, have popularized the open data movement of the last decade, which now includes numerous other portals from other public or private institutions. These portals publish large numbers of datasets related to a very wide range of topics. Although the amount of datasets in OGDPs are increasing, the functionalities provided by most of the OGDPs to the end users are limited to finding the datasets based on the title and description, and downloading the actual files. This limitation hinders the end users, especially those without technical skills, to find the open data files and make use of them.

This thesis presents Governor, a web application developed to make open data tables more accessible to the end users in several ways. First, Governor facilitates searching the actual records in the original tables in OGDP. Second, Governor allows users to preview the tables in the web browser directly without downloading them. Third, Governor allows users to integrate multiple tables to form enriched datasets. A key feature here is automatically finding and suggesting joinable and unionable tables to users based on the latest state of their integrated tables. These operations are performed in the web browser interactively through a few clicks without using a programming language or a spreadsheet software. Lastly, Governor provides a set of features to summarize the provenance of integrated tables allowing users and their collaborators to easily trace back the values in integrated tables to the original tables in the OGDP.

Acknowledgements

First, I would like to thank my supervisor, Prof. Semih Salihođlu, for his constant support over the last two years. This thesis would not be possible without his advice and help. Working with Semih has been a great experience. I am grateful to be part of his research group and to have the opportunity to continue working with him as a PhD student.

I also would like to express my appreciation to Prof. Jian Zhao for his contributions to this project. His valuable insights and feedback played an important role in this project.

Moreover, I would like to thank my colleague Dr. Arif Usta for being part of this research project and helping with the recruitment of participants for the user study.

Last but not least, I would like to thank my thesis reader, Prof. Jimmy Lin, for spending time reading my thesis, attending the presentation, and providing valuable feedback.

Table of Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
2 Background	6
3 Related Work	8
3.1 Open Data Search Tools	8
3.2 Systems With Data Integration Capabilities	9
3.3 Techniques for Finding Related Tables	11
4 Usage Scenarios	13
4.1 Searching Research Grant Amounts	13
4.2 COVID 19 and Vaccination Analysis	15
5 Design Goals	19
6 Governor System	22
6.1 Overview	22
6.2 Search	24

6.3	Original Table Preview	25
6.4	Data Integration	28
6.5	Provenance Information	30
7	User Study	32
7.1	Participants and Apparatus	32
7.2	Tasks and Design	33
7.3	Procedure	34
8	Results	36
8.1	Task Performance	36
8.2	Questionnaire Ratings	38
8.3	Qualitative Results	40
9	Discussion	42
10	Conclusions and Future Work	45
	References	47

List of Figures

1.1	An Overview of Governor System	1
4.1	Search View	14
4.2	Original Table View	14
4.3	Working Table View	16
6.1	Navigation Flow between Different Views of Governor	22
6.2	Governor System Architecture	23
6.3	Column Statistics Pop-up	26
6.4	Suggestions for Completing the Working Table	29
8.1	Participants' ratings on the NASA TLX questionnaire for the Tasks (the lower the better)	38
8.2	Participants' ratings on the exit-questionnaire (the higher the better)	39

List of Tables

6.1	Comparison of the Loading Time (in Seconds) between Governor and Excel	27
-----	--	----

Chapter 1

Introduction

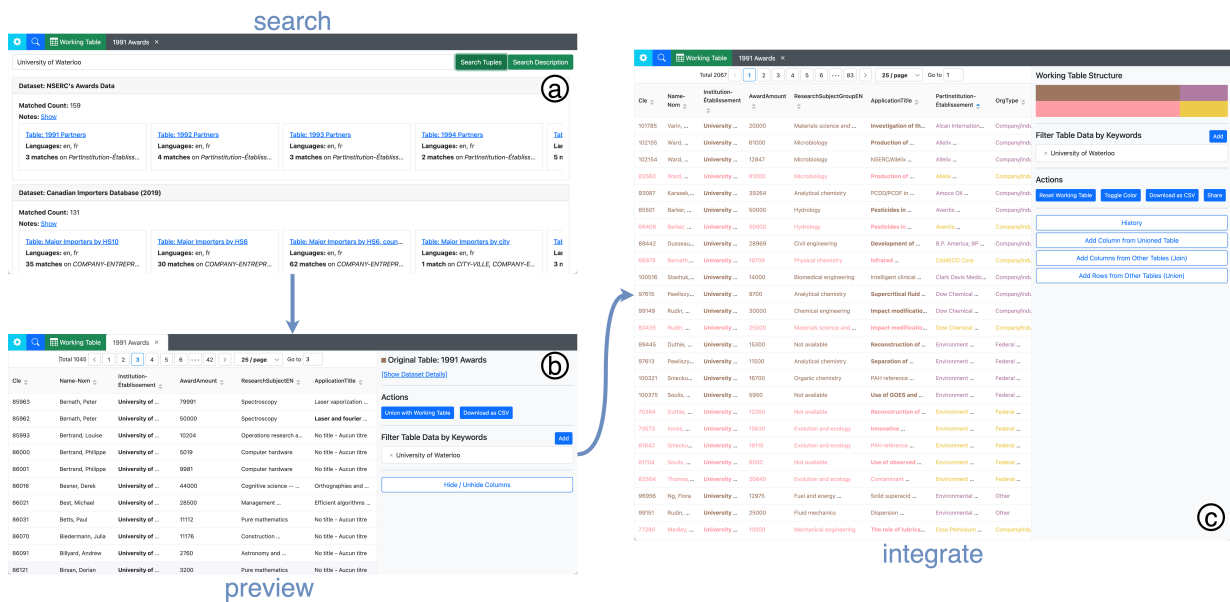


Figure 1.1: Governor: an interactive system to help users effectively search, preview, and integrate open data tables published by OGDs. The main interface of Governor consists of: (a) Search view which allows the user to find open data tables by the description of the dataset or the values stored in the table, (b) Original Table view which provides the user a preview of the original open data tables, (c) Working Table view which facilitates open data tables integration through the automatic detection of joinable and unionable tables.

The launch of open governmental data portals (OGDP), such as data.gov, open.canada.ca,

data.gov.in, data.gov.uk, have popularized the open data movement of the last decade. These portals publish large numbers of datasets about many aspects of government institutions and more broadly the countries these governments govern. The topics of these datasets are too varied to enumerate, but examples include how governments distribute research funds, how much meat they export, daily COVID-19 cases, or the CO₂ emissions of different factories. The core goal of opening government datasets to the public is to make governments more transparent, make governments more accountable, as well as fulfill the information needs of different members of the society, such as researchers or businesses [9]. In their usage vision, these datasets can be analyzed by policy analysts, journalists, researchers, and engaged citizens to find rooms for improvement in how governments operate and identify corruption and systematic injustices in a country.

In light of this vision, the potential societal value of making more government datasets open in the upcoming decades, and making these datasets easier to find, understand, and analyze by users can be invaluable to countries. Despite their clear societal importance, existing OGDs are primarily raw data publishing websites and fall short on delivering their potential value to their intended users due to several shortcomings:

- **Lack of record searching functionality:** Open datasets are published as datasets, where each datasets can encapsulate multiple data files, such as tables published in CSV or Excel file formats. These datasets have textual descriptions (aka dataset metadata), describing the datasets. Existing OGDs, including dataset search engines, such as Google Dataset Search [7], support searching only through these descriptions of the datasets and not the contents of the records in these datasets. Many simple questions that are of potential interest to users can be answered through a few records found in the datasets published by governments. Consider for instance the question: “How much NSERC research money was granted to Yoshua Bengio’s last Discovery Grant application?” or “How many refugees from Afghanistan did Canada accept in 2022?”. The answers to both of these questions are found in a few tables that contain the terms “Yoshua Bengio” or “Afghanistan” but a search of keywords, such as “NSERC”, or “refugees” in open.canada.ca, return over 100 and 50 tables, respectively. This shortcoming is ubiquitous across all major OGDs, such as data.gov, data.gov.in, or data.gov.uk. The ability to do keyword search also through records would make it easier to answer such questions for users.
- **Raw data downloading-oriented data access:** Existing OGDs primarily fulfill the functionality of publishing the dataset in some raw format, which are downloadable to the host machines of users over the web. They provide very limited forms of accessing the data through the portal for initial viewing. For example, a user

interested in the question about Yoshua Bengio, would need to potentially download many CSV files from open.canada.ca, e.g., the 2022 NSERC Awards or 2022 NSERC co-applicants. Then the user would need open and perform a search inside each of these files in a spreadsheet software. This style of interaction can be very laborious.

- **Lack of data integration functionalities:** Many of the datasets in OGDPs are ultimately intended to be analyzed sometimes directly but often by first integrating them into larger datasets. Existing OGDPs are solely publishing websites and do not provide tools to integrate datasets. For example, consider a journalist who would like to construct a Canada-wide dataset of COVID 19 cases and vaccination rates. In absence of any support for data integration in OGDPs, this could be accomplished through a tedious and manual task of finding related tables, downloading them and putting them together in spreadsheet software. Ideally, a data integration system could provide these integration functionalities in a common interface to users.
- **Lack of provenance information and management:** A natural need for users who have performed data integration and analyses using open government datasets is to verify the results of these analyses. Consider, as a purely hypothetical example, that our journalist has constructed a table of vaccination rates and COVID 19 cases in the Ontario province of Canada. Suppose the journalist finds that although initially the vaccination rates steadily increased in the province, there was a sharp decline in September 2021, after which it has remained very low. The journalist speculates that this may be connected to the beginnings of a strong anti-vaccine movement there around the same time. Before publishing this article, both the journalist as well as others, such as editors of the publication, would need to verify the soundness of the data and fact check the numbers against the raw data published in open.canada.ca. In absence of tools that can facilitate and manage data integration tasks, providing these functionalities requires manual and careful documentation by the journalist. A system that facilitated automatic provenance management could greatly simplify such usage scenarios.

Many of the above shortcomings would be addressed by the core functionalities of database management systems (DBMSs). For example, one can ask queries in DBMSs to search through records in the tables. Similarly, tables that integrate multiple other tables can be modeled as *views* [16] over the base tables in DBMS. DBMSs also store information about how the views have been constructed, which can serve as provenance information of these views and facilitate fact checking. Indeed several systems in literature [21, 25] provide a subset of these features by storing the data in data “lakes” in a DBMS or an advanced index

and making this data accessible through a general or DBMS programming language, such as SQL. However, such systems are not accessible to many potential users of OGDs who are not programmers. *This thesis argues that in order for OGDs to deliver their potential value to their users, they should evolve from mere publishing websites into systems with database capabilities with interactive interfaces that are familiar to non-programmers.*

The main contribution of this thesis is a browser-based system that we call *Governor* (Figure 1.1), which provides the above capabilities. Governor models an OGD as a database of published tables, which are accessed and integrated interactively and non-programmatically. Governor is designed to be accessible to users with the basic skills for exploring, integrating and analyzing datasets through spreadsheet software, who are not necessarily programmers.

Governor uses the open.canada.ca portal as a test-bed and indexes all of the records in the tabular corpus of open.canada.ca in an index. This lets users to search original records about specific entities, such as “Yoshua Bengio”, and preview the tables in which these records appear quickly in a single interface without downloading and browsing them using separate software. Starting from one of the original tables, users can start data integration sessions. By interactively performing two core relational operations, unions and joins of tables, through a few clicks, users can construct a “Working Table” that integrates multiple tables. These data integration capabilities can be very useful especially when putting together frequently and periodically published tables, such as monthly or daily published COVID-19 case datasets. The data integration sessions are generally guided through the system’s suggestions for tables to union and join with the Working Table, which respectively allow users to add more rows and columns to their Working Tables.

Governor keeps the full provenance information and provides a set of additional features to support provenance-related tasks. The system summarizes the integration task through color-guided provenance summaries that visually show the different tables that have been integrated, and how they have been integrated (see the top right side of the working table view in Figure 1.1c). Through a few clicks, users can view the original tables that have been integrated, as well as go to the original cells (in the original table) that correspond to the cells in the Working Table. To facilitate these at interactive levels, Governor uses the DuckDB relational database management system [5] on user’s browser to store and manage the integrated datasets. The logs of the data integration steps that a user has taken, which contain the necessary provenance information, can be persisted in Governor’s back end in MongoDB [8]. By persisting this provenance information, users can share Working Tables with other users and/or continue their data integration sessions without losing any information.

The rest of this thesis is organized as follows.

- Chapter 2 reviews how the open data tables are currently managed and structured in OGDPs.
- Chapter 3 discusses related work.
- Chapter 4 demonstrates two usage scenarios which motivate the design goals of Governor.
- Chapter 5 describes the main design goals of Governor.
- Chapter 6 discusses in detail the functionalities and implementation of Governor.
- Chapter 7 and Chapter 8 present the user study we conducted to assess the effectiveness of Governor and the result of the study, respectively.
- Chapter 9 covers the limitations of Governor.
- Chapter 10 discusses future work and conclusions.

Chapter 2

Background

In this section, we explain structure of OGDs, in particular, the portals maintained by governments. Many governments utilize an open-source content management system for open data named CKAN[3], which follows a certain structure for the collection of data to be published. The data published in OGDs are stored under *datasets*. In other words, an OGD is a set of datasets $D = \{d_1, d_2, \dots, d_n\}$, where d_i is the i^{th} dataset and n is the number of datasets. Users search for datasets to find information they seek. An example dataset is *NSERC's Awards Data*¹ from the open data portal of Canada.

Each dataset d_i has two main properties:

- Metadata information about the dataset such as title, publisher organization, publish date, format, keywords, etc.
- A list of *resource files* $F^i = \{f_1^i, f_2^i, \dots, f_m^i\}$, where f_j^i stores the actual data for dataset d_i . Each d_i can have any number of resources, m . Each f_j^i can be in different format such as HTML, PDF, CSV, etc. For instance, for the above example dataset, there are different resource files storing information for different years or storing different entities of the same dataset in same year such as *Awards, Co-applicants and Partners*.

One of the most common resource file formats in OGDs is *comma separated values (CSV)*. CSV files are also more preferable by users, especially data scientists, who have the intention to process these data files programmatically to gain insight. In Governor, we focus on resource files in CSV format due to their popularity and simple format, which

¹<https://open.canada.ca/data/en/dataset/c1b0f627-8c29-427c-ab73-33968ad9176e>

gives us enough data to form a test bed without requiring us to develop a complex file parsing and data processing pipeline.

We also keep the notion of datasets in Governor. Similar to the OGDPs, we present the search results by grouping tables from the same dataset together and show the metadata information stored at dataset level to the users. Additionally, we also use the source dataset of each table as a heuristic to limit the search space of joinable tables, which will be further discussed in [Chapter 5](#) and [Chapter 6](#).

Chapter 3

Related Work

In this chapter, we review prior work on three areas: (i) open data search systems; (ii) open data integration systems; and (iii) systems and algorithms for finding related tables.

3.1 Open Data Search Tools

Several systems support capabilities to search through large collections of enterprise or open data lakes. These include data integration systems that we will cover in Section 3.2. Here we cover systems whose primary functionalities are searching. These systems support search through one of two search modes: (i) keyword search; or (ii) by issuing actual tables as queries, which indicates that the user is searching for tables related to the query table. We refer to the latter type of search mode as table-as-a-query search. In addition, some systems let users navigate or browse through tables/datasets by moving from one table/dataset to another in contrast to an interaction in which the search results take the user to an external website that contains the dataset. We review the related systems below.

Google Dataset Search (GDS) [13] is a large-scale search engine that indexes both public and private datasets. GDS crawls webpages that contain special html tags indicating that the page contains a dataset, and indexes the metadata about the dataset, which can include the descriptions of these datasets and their tables, their publisher information, or the type of open data licence the dataset is published under. GDS supports keyword queries that search over this metadata. Unlike Governor, GDS is solely a search engine and does not index the contents of the records inside the tables in these datasets, nor supports data integration.

RONIN [20] is a data lake exploration system that supports both types of search modes we mentioned above. RONIN uses two indexes. First is the faiss index [18], that indexes one “semantic” vector for each column of each table in the data lake. This vector is used for keyword search. Second, RONIN uses an LSH Ensemble index [28] (discussed in more detail in Section 3.3) that also indexes each column, but is used efficiently to find two columns that have high overlaps in their values and so are joinable. The LSH ensemble index is used when a user performs a table-as-a-query search. The overall user interaction starts by a user issuing a keyword query, then getting back a list of tables, and then navigating the data lake starting from a selected result table, which can be used as a table query to further get a set of related/joinable tables. Similar to Governor, RONIN shows previews of the tables but is not designed to integrate tables and provide provenance capabilities.

Auctus [14] is similar to RONIN but contains more indices, including a keyword index of descriptions in Elastic Search (similar to Governor), an LSH-based index [15] to find set similarity over categorical data, as well as indexes for temporal and spatial columns. Auctus uses these indices to support keyword search, temporal or spatial search, or table-as-a-query search by manually uploading a table into the system. When a user uploads a table as a query to Auctus, the system also supports a very limited form of integration, where the table can be joined with one other table in the query results and the final join of these two tables can be downloaded by the users. Governor instead supports integrating large numbers of tables and is designed to manage these integrated tables and their provenance.

3.2 Systems With Data Integration Capabilities

Several prior work has developed interactive systems that have direct or indirect capabilities to support data integration in enterprise or open data lakes. We discussed Auctus [14] and its (limited) capability to join two tables above. Toronto Open Data Search [29] is similar to Auctus and allows users to find joinable tables and similarly only allows a single join of two tables for data integration.

Voyager [12] is a data search and discovery system for data lakes that is designed as an extension of Jupyter Notebook. As such, Voyager is accessed through a programming language, such as Python. Voyager supports both keyword queries over indexed tables/-datasets as well as table-as-a-query searching capabilities. Although not directly targeting data integration tasks, the results of these searches are returned as tables and can then be integrated with other tables all in the user’s programming language. Voyager assumes an interactive user experience, albeit the interaction happens through a (programmer) user

writing snippets of programs in a Jupyter Notebook, instead of Governor’s clicking-based web interface.

Juneau [25] is another system, whose functionalities overlaps with Voyager. Juneau is also designed as a Jupyter Notebook extension. We refer to these notebooks as Juneau notebooks. Users can perform table-as-a-query searches against source data lakes, and the results of these searches can be integrated with other tables in Juneau notebooks. However, the primary goal of Juneau is to manage data science pipelines. To achieve this, the Juneau notebooks, which contain both data integration as well as analytical codes, such as machine learning pipelines, are themselves stored and managed in Juneau’s server. At a high level, this server stores the computation graph of how the tables in these notebooks were constructed as provenance of these tables. This provenance information is used to suggest related tables to other users’ of Juneau notebooks. As a simple example, consider a user that is working on a Juneau notebook, and is in the middle of a data integration process that has joined two tables R and S . The user can get a suggestion from Juneau to further join table T , because a prior Juneau notebook stored in the system has done the same thing. Similar to Voyager, user interactions with Juneau are through programming a notebook. Similar to Governor, Juneau stores the provenance information for integrated tables in Juneau notebooks, but this information is used in suggesting related, e.g., joinable, tables to the user. Instead, Governor uses the provenance information to facilitate fast verification/fact-checking in the original tables published in OGDPs.

KGLac [17] is another data discovery system that also supports table-as-a-query search capabilities. KGLac generates a knowledge graph out of a data lake, whose nodes contain tables, datasets, and columns and edges contain different relationships such as how similar different columns are, or whether they represent primary-foreign key relationships. This knowledge graph is stored in an Resource Description Framework (RDF) database [23], which is a popular data model to model knowledge graphs. This knowledge graph can then be queried through the SPARQL query language from any host programming language. In addition, users can use the KGLac library in Python to perform table-as-a-query searches and find related tables, which can, similar to Voyager and Juneau, be integrated in Python.

Voyager, Juneau, and KGLac all provide data integration capabilities for users with programming skills, who are accessing these systems through a programmatic interface. Instead, Governor is a web application where data integration happens through a spreadsheet-like interface that is extended with interfaces showing data integration suggestions. Governor’s goal is to make data exploration and integration on OGDPs more accessible to potential OGDG users, which includes non-programmer users, such as journalists or policy analysts, or regular engaged citizens, who have basic proficiencies in data analyses on spreadsheet software. In addition, these systems do not provide features to summarize the

integration steps of an integrated table or link the data in these tables with original tables to facilitate fast data verification. We also note that some of the features provided in these systems are complementary to Governor. For example, Governor implements indexes and algorithms to find joinable and unionable tables to the user’s Working Table, which we discuss more in Section 3.3 and in much more detail in Chapter 6. Each of these systems also has different ways of finding joinable or unionable tables, each of which could replace Governor’s current approach. We review some more literature on these techniques below.

DICE [21] is another data discovery system in which users provide example tuples of a desired table to the system. The goal of DICE is to find SQL queries that could have generated those tuples and show users additional tuples that these queries would generate. These SQL queries effectively perform automatic data integration to generate a table that conforms to the example tuples provided by the user. Therefore, users can optionally issue these queries to the underlying data lake, generate new tables, and store them back in the data lake. In contrast, Governor allows users to perform the data integration interactively in its web interface and manages these integrated tables and their provenance.

3.3 Techniques for Finding Related Tables

There has also been prior research on the foundations of how to find related tables, specifically unionable and joinable tables, in data lakes. This literature assumes a setting in which a system supports table-as-a-query search. Since Governor also finds unionable and joinable tables with the users’ Working Table, these techniques are related to our work. Given a query column c , reference [28] proposes an efficient locality-sensitive hashing based index to find other columns that have high value overlap with c . This approach is approximate and can return false positives, which a follow-up work by the same authors addresses in reference [26]. Reference [19] has proposed a suite of indexing, searching, and ranking techniques for the problem of finding unionable tables. Some of these techniques are based on the values in the columns, some are based on ontology mappings of the columns of the tables (if they exist), and others are based on semantic similarity if the columns contain strings. Similar to our comment above on similar techniques implemented in Voyager, Juneau, and KGLac, these techniques are complementary to our work.

In Governor, we have chosen a set overlap-based technique for finding joinable tables. Governor only searches for joinable tables based on a key (or almost key) columns. Governor’s unionability suggestion algorithm searches for pairs of tables that have exactly the same schema. In our experience developing and using Governor, we realized that both of

these approaches were sufficient to find good related tables and accomplish the data integration scenarios that motivated our work. We therefore opted for simplicity here, since our focus in Governor is to provide a user-friendly system to make dataset exploration and integration more accessible to users, instead of developing advanced related table search functionality.

Chapter 4

Usage Scenarios

In this chapter we demonstrate two scenarios to motivate the design goals of Governor, which we cover in Chapter 5. Our scenarios expand on the Yoshua Bengio and COVID 19 and vaccination correlations examples from Chapter 1.

4.1 Searching Research Grant Amounts

Alma is final year graduate student who has been offered a faculty position from a Canadian university. She is interested in forming an opinion on the sizes of the research grants she can obtain from the Canadian government on her research topic of deep neural networks. She is aware of several faculty in Canadian universities who work on similar topics, one of which is Yoshua Bengio. She sets out to answer how much research grants has Yoshua Bengio obtained lately. She thinks this information must be published openly by the government, so goes to Governor’s landing page, which contains the “Search View”. Here, she types in “Yoshua Bengio”, and clicks “Search Tuples”, which returns 7 datasets with many tables (Figure 4.1). For example, she sees that under the NSERC awards dataset, she sees tables whose titles include years 1991 to 2019. So she infers that Bengio’s latest NSERC award must be from 2019. The table’s name is 2019-co-applicants. She clicks on this result, which opens a new tab that is in the Original Table View (Figure 4.2). On the left, she sees a spreadsheet-like interface with 2 tuples and 2 columns: Cle and CoApplicantName. Both of the tuples contain “Yoshua Bengio” under the CoapplicantName column. Reading the dataset description on the right hand-side, she understands that Bengio must be the co-PI on two grants in 2019.

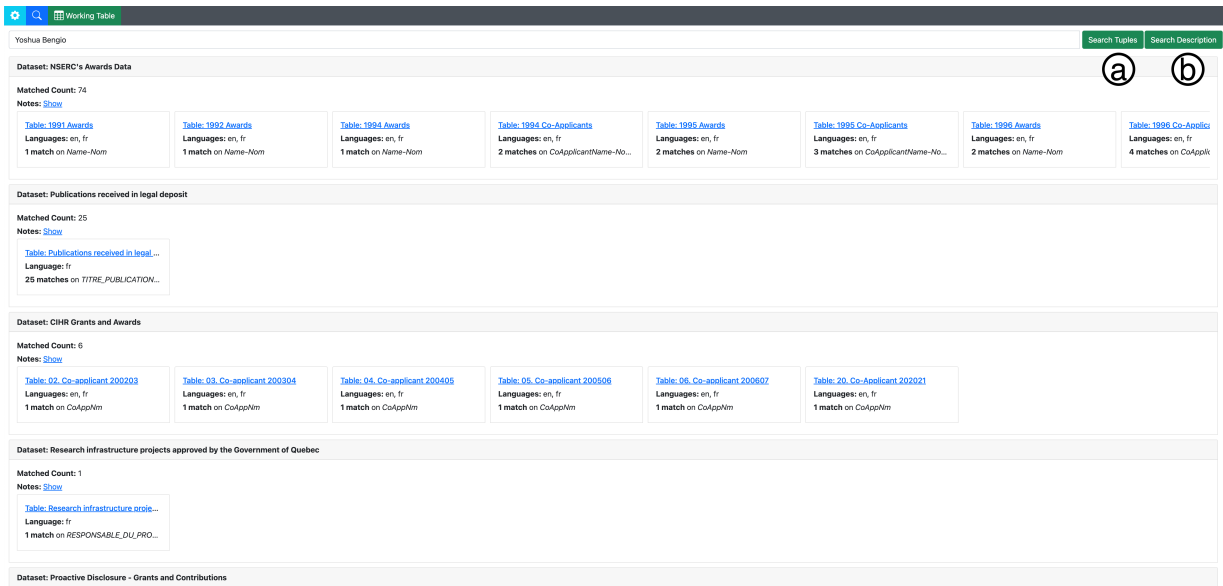


Figure 4.1: Search View

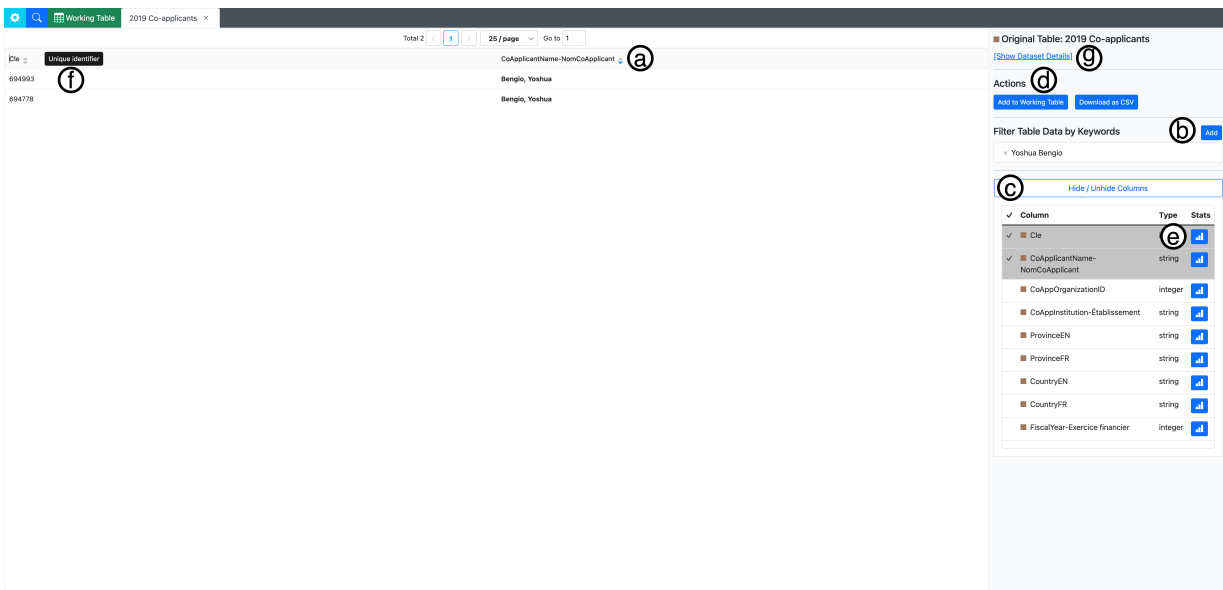



Figure 4.2: Original Table View

She then goes to the Hide/Unhide Columns sub-panel the Action Panel on the right side (Figure 4.2c) and notices that this table does not seem to contain any column that contains the amount of money given to this grant. She then looks at the spreadsheet again and sees that the top tuple has an identifier looking value “694993” under the Cle column. Hovering over the column, she sees a description of the column which says “Unique-identifier”¹. She thinks this might be an identifier for the grant and decides to search for this grant. She click on the  icon to go back to the Search View and types “694993”. This returns several tables², one of which is the 2019-co-applicants table she had already opened. Next to this table is a 2019-applicants table. She opens this table, which takes her back to a new tab in the Original Table View. Here she sees only a single tuple, the one with Cle value 694993. This is a table with 35 columns and Governor by default shows 5 of these columns, including Cle, Name (which is the PI’s name), and Application Title but not the amount, which she is interested in. She goes the to Hide/Unhide Columns (Figure 4.2e) sub-panel and scrolling down sees the AwardAmount column, which she clicks, and sees the amount on the table on the left. She repeats a similar process for several few recent grants obtained by Yoshua Bengio (and several other faculty she is interested in) to estimate the scales of the government research grants on her topics.

4.2 COVID 19 and Vaccination Analysis

Shufan is a journalist who has been tasked by his newspaper to investigate the latest debate about the effectiveness of the vaccine rollouts in Canada. Shufan is tasked with a specific question: Are the vaccinations reducing the COVID-19 cases in Ontario? Shufan thinks he can answer this question with a thorough analysis of 2 years of COVID-19 hospitalization cases and correlating this with the vaccination doses administered in Ontario. Shufan knows that this information is periodically published by the government and he decides to construct a dataset that consists of COVID 19 and vaccination cases of every day since April 2020 and decides to use Governor.

He types “COVID-19” on Governor’s landing page and clicks “Search Description”, which returns back several datasets and tables including a dataset of COVID-19 Vaccine Data in Ontario with many tables in it, since this information is published quarterly. He

¹If datasets have a structured description files, then Governor extracts such column descriptions and displays them on its interfaces (Figure 4.2f)

²Not surprisingly, numerical identifiers that are incremental and start from 0 and increase, are keys also in tables other than the 2019 NSERC tables in open.canada.ca.

The screenshot displays a 'Working Table' interface with a data table, a sidebar, and three callout boxes labeled a, b, and c.

Table Data:

report_date	total_doses_administered	icu	nonicu
2021-09-30	21750009	93	125
2021-10-01	21809713	94	110
2021-10-02	21847046	94	110
2021-10-03	21870930	84	95
2021-10-04	21890185	77	96
2021-10-05	21916657	78	106
2021-10-06	21948953	75	121
2021-10-07	21974193	74	95
2021-10-08	22004768	73	104
2021-10-09	22042483	64	92
2021-10-10	22071730	58	87
2021-10-11	22083843	64	86
2021-10-12	22089455	70	95
2021-10-13	22119312	68	101
2021-10-14	22148068	80	99
2021-10-15	22177830	82	97
2021-10-16	22208199	77	100
2021-10-17	22231210	79	99
2021-10-18	22243809	77	91
2021-10-19	22264919	77	104
2021-10-20	22280203	75	113
2021-10-21	22312892	65	128
2021-10-22	22338662	75	114
2021-10-23	22365606	71	104

Working Table Structure (a): A sidebar showing a table structure with columns for 'report_date', 'total_doses_administered', 'icu', and 'nonicu'. It also includes a 'COVID-19 Vaccine Data in Ontario' section with a 'Vaccine data 2020-Q2' entry.

Filter Table Data by Keywords (e): A section with an 'Add' button and a message: 'There is no filter. Click "Add" to apply a new filter.'

Actions (e): A section with buttons for 'Reset Working Table', 'Toggle Color', 'Download as CSV', and 'Share'. Below it is a 'History' section with options: 'Hide / Unhide Columns', 'Add Columns from Other Tables (Join)', and 'Add Rows from Other Tables (Union)'. Arrows from these options point to callout boxes b and c.

Callout Box (b): A list of 'Vaccine data' entries for quarters Q3, Q4, Q1, Q2, Q3, and Q4, each with 'Open' and 'Undo' buttons.

Callout Box (c): A 'Filter by dataset / column / table name...' dialog. It shows a search filter 'date' and a list of columns: 'icu' and 'nonicu', both with checkmarks. Below it, a table structure for 'Table: Case rates by age group' is shown with 'Columns' and 'Open' buttons.

Callout Box (d): A list of 'Add rows from table:' and 'Add column "icu"/"nonicu" from table:' options for various quarters (Q2, Q3, Q4, Q1), each with an 'Undo' button.

Figure 4.3: Working Table View

clicks on the Q2-2020 Vaccine Data table, the table with the earliest date, which opens the table in Original Table View. He notices that this table contains 91 tuples, one for each day of April, May, and June with a TotalDosesAdministered column storing the number of vaccines administered in each day of the second quarter 2020.

Shufan decides to enrich this table with data from other quarters and also hospitalization cases. To start this data integration task, he clicks on the “Add to Working Table” button (Figure 4.2d) on the right panel which copies this data to the “Working Table View” (Figure 4.3a). Next, Shufan wants to add more rows to this table by gathering vaccination data from other quarters. He inspects the “Add Rows From Other Tables (Union)” sub-panel in the Actions Panel, where Governor lists its suggestions of other tables (Figure 4.3b) that have the same schema, i.e., column names. These are the tables in open.canada.ca that can be integrated with the Working Table through a union operation. There he sees 7 suggestion, one for each quarter. He clicks on tables Q3-2020 Vaccine Data and Q4-2020 Vaccine Data and generates a table with hundreds of tuples and two columns, report_date, and total_doses_administered (Shufan hides several other columns he is not interested in).

Next, Shufan decides to integrate hospitalization data into the Working Table. Since this data is in separate files with different schema, Shufan inspects the “Add Columns From Other Tables (Join)” panel and sees several tables there, including “COVID-19 Hospitalizations”. The tables here all have close to perfect overlaps with a key (or almost key) column in the Working Table, i.e., that is unique for each tuple in the table. In Shufan’s current table, this is the report_date column. Shufan inspects the columns that he can gather from the Hospitalization by Vaccine Status table and sees two columns that look related to his task: icu and nonicu (Figure 4.3c). He infers that these must be reporting the number of hospitalizations that were in intensive care units (icu) or those that did not need icus (nonicu). He clicks on these 2 columns, which extends the Working Table with 6 new columns (and no new rows). Unlike the vaccination data, which is published periodically, the hospitalization data is in a single table, so this operation effectively joins the entire Working Table with a single other table. In another and a more common scenario, Governor supports separately joining each unioned table (e.g., only the tuples from the Q2-2020 Vaccine Data table) with another table.

Having integrated a total of 9 tables, Shufan then decides that this information is enough for him to do his analyses. He downloads the integrated table as a CSV file and opens it in Excel. He plots a line chart of total_doses_administered and icu and sees a sharp decline starting in the fourth quarter 2021, though without a clear effect on hospitalization. He thinks that it would still be worth raising the question of what has caused of this decline in vaccinations in an article. He prepares his article that contains the chart and a shareable

Governor link that contains the integrated table for the editors to fact-check this observed trend.

Upon opening the shared Governor link, the editor sees the integrated table and inspects the Working Table Structure (WTS) panel (Figure 4.3a) to understand the tables that got integrated. She sees that many tables have been unioned together, and inspecting the names of these tables on the WTS, understands that these correspond to quarterly vaccination data. She knows that Shufan's article mentions a decline in administered vaccinations in the fourth quarter of 2021, so finds the rectangle for Q4-2021 and clicks on it, which opens the original table in a new tab. Here she eyeballs the `total_doses_administered` column of the 92 tuples in this table and notices that they are in the few thousands, which indeed looks very low. She double checks the line chart in the article, and sees that the chart indeed plots numbers in the few thousands for the fourth quarter 2021. She then clicks on the Show Dataset Details (Figure 4.2g) and reads the dataset details to verify that these are COVID 19 vaccination data and published by Ontario's ministry of health. She then repeats this fact checking process for a few other quarters and lets Shufan know that she has verified that the article's data is sound.

Chapter 5

Design Goals

Governor’s design was guided by four goals. Some of these design goals were driven by our overarching goal of providing database management system capabilities over the datasets in OGDPs through a non-programming interface. Others were informed by the properties of the actual tables stored in OGDPs.

G1: Store and index the tuples inside the tables: In order to facilitate searching through tuples and any data integration task, a system needs to store and index the actual tuples inside the tables of OGDPs. An important property of the datasets in OGDPs is that they are relatively small in size. For example, the size of the largest OGDG, data.gov is 2120 GB in uncompressed raw file size and 434 GB when compressed. open.canada.ca, which is the OGDG we used is 345 GB in uncompressed raw file size and 128 GB when compressed. These sizes are in the scale that even for research projects undertaken in academic groups can have the resources to fully index the tuples in these datasets. Indexing the tuples was also needed to let users preview the datasets at interactive speeds, By managing the tables in our own servers, we could compress large tables and send them to user’s browsers more efficiently.

G2: Support core relational data processing operations interactively: In order for users to integrate multiple tables and construct larger tables, we needed an interactive user interface that supported several core relational operations of. Specifically we needed to support union and join operations for data integration. In addition we needed basic data cleaning and transformation operations, such as projection (removing columns), filtering (selection), and ordering. We wanted the interface to be a familiar spreadsheet interfaces where users clicked on buttons instead of a command line interface where they would type SQL quefries.

G3: Provide suggestions for related tables: Navigating through OGDPs is challenging for two reasons. First, the entire corpus can contain tens of thousands of tables. Second, there can be dozens or hundreds of related tables that users may potentially integrate together. There two common reasons for why the information users need can be spread across multiple tables: (i) periodic publishing: a common style of publishing a dataset in OGDPs is to periodically publish a dataset at certain time intervals, e.g., every week, month, or year; and (ii) normalization [22]: some information is published in a normalized form, i.e., partitioned into multiple tables to avoid some data redundancy. Manually finding the tables to integrate one by one cannot scale, so we needed to intelligently provide suggestions for tables that can unioned and joined with users’ Working Tables (i.e., the one they are constructing).

This meant that Governor had to implement algorithms for the table-as-a-query problem, that we discussed in Chapter 3. Here, we had options from numerous algorithms from literature for finding and suggesting unionable and joinable tables, which we also covered in Chapter 3. Here we opted for simple solutions. For unionability, we opted for a schema-based approach, where we required the schemas of tables to be the same to be considered unionable, as this is sufficient for capturing the most popular case of detecting periodically published tables. In order to consider two tables T_1 and T_2 joinable on columns c_1 and c_2 , we opted for a technique which required two constraints: (i) the sets of values to overlap with a very high-percentage; and (ii) that one of c_1 and c_2 to be a key. All existing work on joinability [26, 27, 28] is based on set-overlap metrics, and our requirement for high overlap was to avoid introducing null values for tuples that do not successfully join. Requirement (ii) was adopted to ensure that Working Tables do not grow in size after the join. If the join is a non-key-non-key join, then each tuple in the Working Table can join with multiple tables and produce a much larger table, which we observed in our initial iterations on the system. This is similar to how join-like functions in spreadsheet software, such as Excel’s VLOOKUP function, also only joins a tuple $t_1 \in T_1$ with the first matching tuple in $t_2 \in T_2$ by default. After adopting these two constraints, we further noticed that the overwhelming majority of the suggestions Governor makes is limited to pairs of tables that are both published as part of the same dataset.

G4: Manage integrated datasets and their provenance to support later fact checking and verification: Recall that an overarching goal of OGDPs is to increase government transparency and accountability, and one important use case is for these datasets to be used by journalists, policy analysts, or engaged citizens to find problems in the government. Therefore the outcomes of the analyses performed on these datasets can be sensitive. Similar to our usage scenario, we envisioned cases where users would construct an integrated table in Governor and later share these tables along with their analyses with

others. Then other people could verify the data in these integrated tables and easily (and interactively) go back to the original tables and cells in these tables, which is akin to exploring a database view in a DBMS by navigating into the base tables. This required Governor to have features to store and manage the provenance of the integrated tables, i.e., the sequence of operations that formed them.

Chapter 6

Governor System

In this section, we describe the functionalities and technical details of the Governor system. We start in Section 6.1 by providing an overview of the user interface components and the architecture of the system. Then, in Section 6.2 to Section 6.5, we go over the functionalities and implementation of each component in detail.

6.1 Overview

Governor is a single-page Vue.js [10] web application with three main functionalities: 1) searching through the collection of open data tables, 2) previewing the original open data table, and 3) integrating multiple open data tables together. Each functionalities is associated with one view of the user interface. The Search View (Figure 4.1) allows the users to

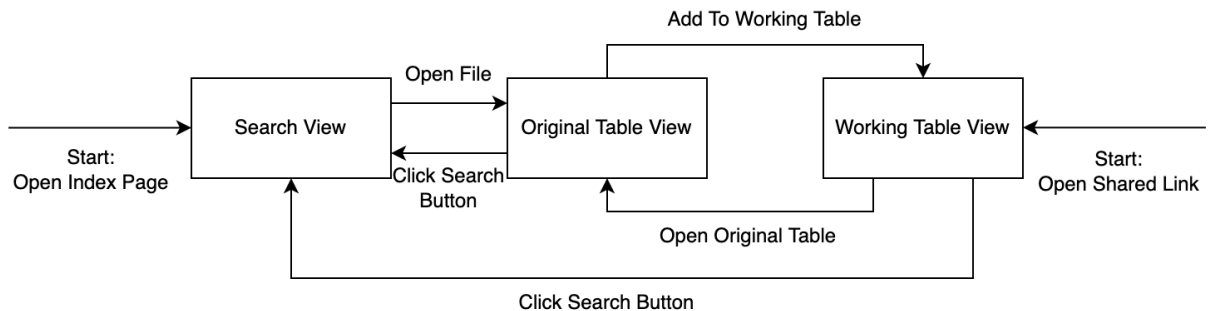


Figure 6.1: Navigation Flow between Different Views of Governor

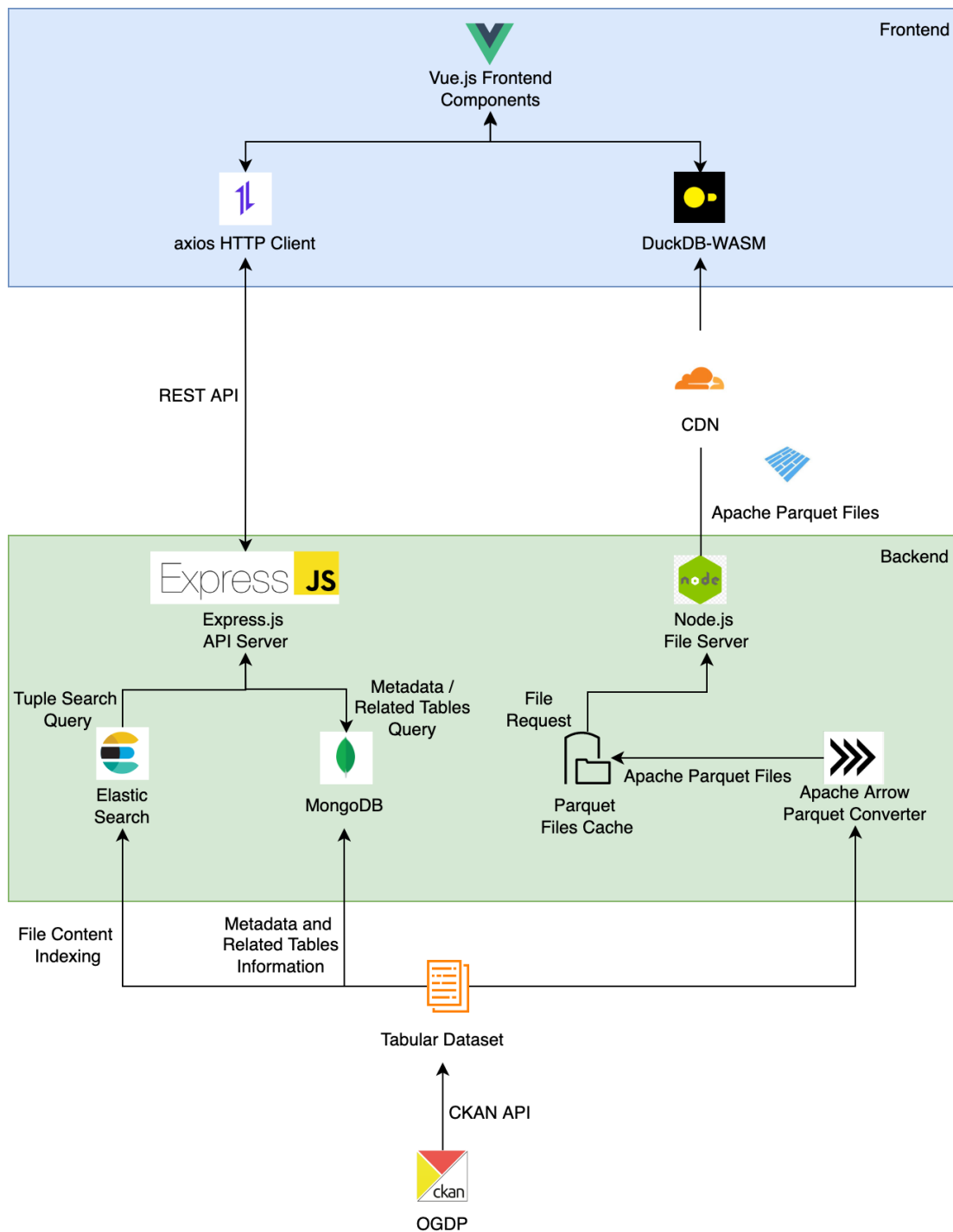


Figure 6.2: Governor System Architecture

perform table search by both the values of the table and the metadata description. Original Table View (Figure 4.2) allows the users to preview the open data tables directly without downloading and provides basic sorting and filtering functionalities. Finally, the Working Table View (Figure 4.3) allows the users to integrate data from multiple open data tables via union and join operations, and provides color-guided provenance summaries that visually show the different tables that have been integrated, and how they have been integrated.

A complete workflow of finding an interesting open data table and integrating it with other open data tables requires users to interact with all three views. Users can navigate between the three views through different interactions in the system as shown in Figure 6.1. When the users open the index page of Governor system initially, it will land on searching page by default, so that the users can immediately start looking for open data tables of interest. After finding a potential table of interest, users can click on the title of the table, which will lead to the original table view for previewing. If users are interested in expanding the original table by integrating with other tables in the collection, they can start this workflow by adding the original table to the Working Table, which will provide them with suggestions on unionable and joinable tables automatically. Finally, after the users are done with creating the Working Table, they can export the Working Table by creating a shareable link, (discussed in more detail in Section 6.4). When opening Governor system with a shareable link, the system will land on the Working Table page and automatically replay the previous operations to create the same Working Table.

To support the diverse functionalities of the web application, the back end of Governor combines multiple technologies and provides a unified interface for the front end via an HTTP server. The back end fetches the open datasets from the government data portal through CKAN[3] API, indexes the dataset with MongoDB [8] and Elasticsearch[6] for querying, and serves the data files by converting them into a compressed Apache Parquet [2] format using Apache Arrow [1], as illustrated in Figure 6.2.

6.2 Search

In Governor, the tuples of open data table files under a dataset are indexed by Elasticsearch[6], while the metadata information of the dataset, such as the name of the table, as well as the notes, publisher, category, and keywords of the dataset, is indexed by MongoDB. These two indices in the Governor back end provide two different search modes: “Search Tuple” (Figure 4.1a) and “Search Description” (Figure 4.1b), respectively.

The tuple search feature finds table by matching the actual values in the data table, which is more suitable for finding information about a specific entity such as human names, business names, and addresses. For example, in the usage scenario described in Section 4.1, by using “Yoshua Bengio” as a search keyword to perform a tuple search, as shown in Figure 4.1, the grants awarded to Yoshua Bengio and the publications submitted by her are found.

On the other hand, the description search finds the table by matching the metadata of the dataset that consists of the table. The description search feature acts similarly to Google Dataset Search and CKAN-based OGDPS, which is more suitable for finding tables by its domain. For example, by using the keyword “economy” to perform a description search, the users can find various tables related to the economy of Canada, such as the transfer payments, tax statistics, and major importers.

Users can perform a search by typing a keyword and selecting the search mode. The search results returned by the system are grouped by dataset. The users can also read the notes, subjects, and release date of each dataset directly from the search result page to determine the usefulness of the dataset.

6.3 Original Table Preview

In Governor, the original table preview consists of a few basic features of a spreadsheet software, including filtering (Figure 4.2b), sorting (Figure 4.2a), and basic column statistics (Figure 4.2e). It also provides basic statistics and a histogram visualization for each column (Figure 6.3). These features can help users look up simple facts without having to download the table and open it in a spreadsheet software as discussed in Section 4.1.

In the open.canada.ca’s tabular data corpus crawled in November 2021, 50% of the tables contain more than 11 columns. The large number of columns can be overwhelming if all of them are displayed by default. To make the user interface cleaner, we use two strategies to determine whether a column should be shown. If a filter is applied, we show all columns with at least one match to the filter keywords to make it easier for users to understand the fields that match the search keyword. If there is not a filter keyword, we use the uniqueness score to determine the importance of the column. The assumptions is that the columns with more unique values are more important. By default, we pick the top-5 columns with the most unique values to show. Optionally, the users can pick up the columns to show manually by clicking on the column title from “Hide / Unhide Column” section of the action panel.

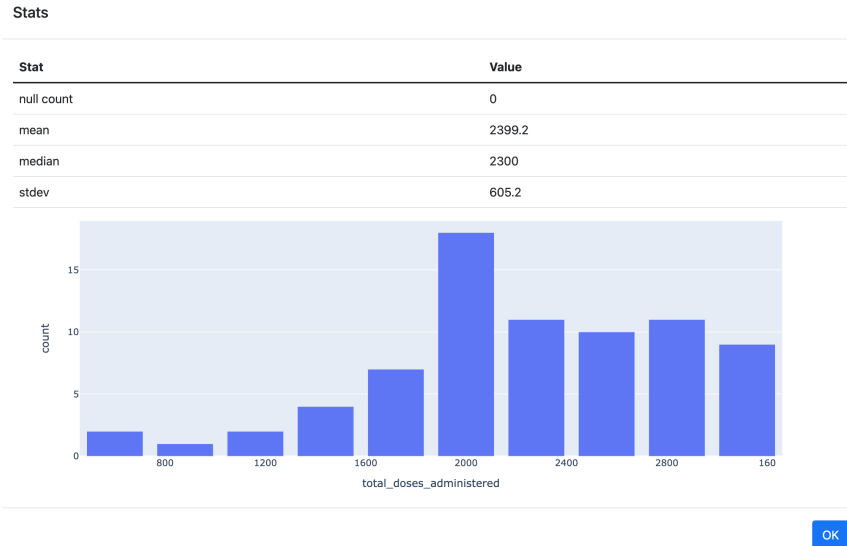


Figure 6.3: Column Statistics Pop-up

The original table preview feature of Governor is supported by the WebAssembly [11] version of DuckDB [5], which is a SQL OLAP database management system that runs inside the browser. When the users preview the original table, the entire table is loaded into the client-side DuckDB in a compressed Apache Parquet format. When the original table is sorted or filtered, the system simply issues a new query locally, and renders the results.

Loading the full tables and processing them at the front end database is a very uncommon practice in web applications today. Most web applications that provide similar functionalities would run the database at the back end and send the paginated results to the front end. We think this architecture is suitable for our system due to the following benefits:

- **Reduce Internet Traffic:** As discussed in Chapter 5, the open data tables are usually highly compressible, which is partially due to the large number of repeated values in data columns like “provinces”. By loading the entire table at once, the system can send the data in a format with a high compression ratio such as Apache Parquet, thus reduces the total Internet traffic required to load the table.
- **Reduce Server Load:** By processing the data at the client side, the server does not need to evaluate the queries for filtering and sorting. Instead, the server only

performs compression when a file is requested. Moreover, compressed files can be easily cached by load balancers and CDNs. In our deployment for the user study, we utilize Cloudflare CDN [4] to automatically cache frequently-accessed data files. According to the dashboard of Cloudflare, over 98% for the traffic in the user study is handled by the CDN without reaching our server.

One potential drawback of sending the full table to front end for processing is the slowdown of the initial loading. A system that loads data with server-side pagination can usually output the first page quickly, as only partial results are required by the front end. However, despite having to load the full table from a remote server, Governor can still load large table files at a speed comparable to native desktop spreadsheet software, such as Excel in many cases.

File	Original Size	Parquet Size	# of Rows	# of Cols	Excel Loading Time	Governor Loading Time
1991 Awards	10 MB	2 MB	18458	34	2	2.1
Historical DriveBC Events	91 MB	8 MB	201802	25	10.6	7.6
Complete file: 2009 to today	335 MB	31 MB	442690	46	29.7	17.6
Proactive Disclosure - Grants and Contributions	850 MB	68 MB	590021	37	58.6	39.2

Table 6.1: Comparison of the Loading Time (in Seconds) between Governor and Excel

In Table 6.1, we show a comparison of the loading time between Governor and Excel (version 16.63.1) for several tables sampled from open.canada.ca. In this comparison, the files loaded by Governor are pre-compressed and cached by Cloudflare, while the same original CSV files are loaded by Excel directly from the local file system. The benchmarks are performed on a Mac Pro computer with a 3.5 GHz 8-Core Intel Xeon W CPU and 32 GB of RAM. The Internet connection is throttled down to 50 Mbps, and the browser cache is disabled. As shown in Table 6.1, Governor can load all of the listed tables at a speed similar to or faster than Excel due to the high compression ratio and the near-native performance of WebAssembly.

6.4 Data Integration

In Governor, a key functionality is to automatically provide suggestions for joinable and unionable tables based on the Working Table, so that users can quickly integrate multiple related tables together, as discussed in Section 4.2. From the technical perspective, the key functionalities of the Working Table are: 1) automatically detecting the unionable and joinable tables, and 2) performing the data integration operations. Since the method for unionable table detection is covered in Chapter 4 and Chapter 5, in this section we expand on the method of detecting joinable tables and discuss how the data integration is performed by the system in detail.

As discussed in Chapter 3 and Chapter 5, Governor utilizes Jaccard set containment score similar to [26, 29, 28]. The metrics is defined as $\frac{|Q \cap X|}{|Q|}$ where Q is the query column for which the system needs to find joinable columns and X are the other candidate columns in the corpus. This containment score ranges from 0 to 1 where the closer to one the containment score is, the more value overlaps between the columns are.

While a high overlapping score between two columns indicates that they are joinable, it does not ensure that the joins are meaningful, especially for numerical columns. For example, the Jaccard overlap score between the research area code column from table “Ontario Research Funding – Summary” and the wind speed column from “Alpine Birds - Jasper” is greater than 0.7. However, suggesting joins like this would confuse the users, as joining these two columns together will produce a meaningless table due to the completely different meanings of the columns. In other systems that utilize a similar metric to find joinable columns, this issue is usually addressed by removing all the numerical columns from the search space, as it is much less likely for string values that have completely different meanings to match each other by coincidence. However, for Governor, a major motivation for providing the join feature is to handle de-normalized datasets as discussed in Chapter 5, which requires the system to be able to find joinable columns for integer key columns similar to the Cle column in Section 4.1. Therefore, instead of limiting the columns to strings, we limit the search space by only looking for joinable columns within the same dataset. Since all the tables under a dataset are published by the same government agency and are related to each other, it is much more likely that the codings and IDs are consistent between them, thus leading to a meaningful join in most of the cases.

As discussed in Section 4.2, in many use cases, the users need to separately joining each unioned table with another table to form the final Working Table, due to the periodical publishing of the tables. While in Governor the joins can be performed easily, it can still be a tedious process to perform a join for each of the unioned table manually. Instead,

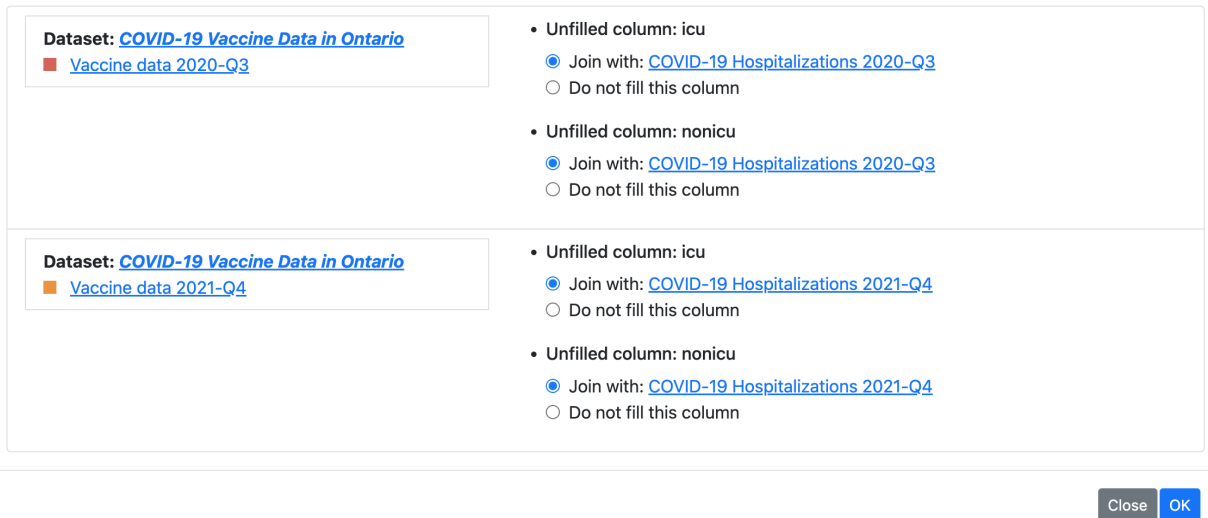


Figure 6.4: Suggestions for Completing the Working Table

Governor provides a feature to automatically handle this workflow. After the users adds a column c_1 to one unioned table T_1 , the system detects if the same columns can be added to another unioned table T_2 by matching the names of the c_1 with the schema of the joinable tables of T_2 and suggest these matches to the users. With these suggestions, users can easily construct a Working Table that involves many joins and unions by manually performing a join for one of the unioned tables and complete the rest of the Working Table based on the system’s suggestion. For example, if hospitalization data from Section 4.2 is also published quarterly, the user would only need to add icu and non-icu columns for Q2-2020, while the system would suggest the joins for all the other quarters, as shown in Figure 6.4.

Similar to the original table preview feature discussed in Section 6.3, the Working Table in Governor is implemented with DuckDB at the front end. Specifically, the Working Table is modeled as a *view* in DuckDB, which is created by a single SQL command compiled from all the joining, unioning, and filtering operations that users have performed in the system. When the users perform a new operation, the system first appends a log for the operation. Then, the system goes through its catalog to check if all the required tables are loaded to the local DuckDB and pulls any missing table from the server. After that, the system compiles all the logs into a single SQL command that first performs join, then unions all the joined components together, applies the filters, and finally projects the hidden columns out. Finally, the system drops the current Working Table view from DuckDB and recreates

it with the compiled SQL command, and re-renders the user interface based on the new Working Table view.

The approach of modeling the Working Table as a view simplifies the state management of the system, because the system does not need to maintain the data for the Working Table. It also enables the Working Table to be re-created easily, which facilitates the “shareable link” feature of Governor. When a shareable link is created by the users, the front end of Governor sends all the logs to the back end, which gets stored in the MongoDB. When the link is later used to reopen the Working Table, the front end simply retrieves the logs from the back end by the object id embedded in the shareable link, loads all the required data tables from the back end, then compiles the SQL query as usual to recreate the DuckDB view, and finally renders the user interface.

6.5 Provenance Information

The main visual cue provided by Governor to distinguish different tables that have been integrated is the color coding. When a table is added to the Working Table via a join or union, the system automatically assigns a color to it and uses the color consistently throughout the entire user interface. First, when users click the “Toggle Color” button in the Actions panel (Figure 4.3e), the cells of the working table are colored based on the color of the table from which it comes from. Moreover, the color coding is used in the Working Table Structure (WTS) panel (Figure 4.3a), which is a color-guided provenance summaries that visually show the different tables that have been integrated, and how they have been integrated as described in Chapter 4 and Chapter 5. The layout of WTS corresponds to the structure of the Working Table. The joined tables are displayed horizontally next to the main table it joined with, while the unioned tables are stacked vertically. The WTS also displays the name of the data table and the dataset it comes from when the users hover on a colored block.

In addition to WTS which summarizes the Working Table in a structured way, Governor also provides a history panel (Figure 4.3d) which keeps track of all the operations performed to construct the current working table in a linear order. The history panel also provides an “Undo” button for each operation performed, allowing users to recover from operations performed mistakenly.

Finally, Governor provides several ways for the users to go back to the original table from the Working Table. First, by clicking on the title of the each column, Governor shows a “Column Composition” modal, which lists all the original tables being unioned to form

the selected column. From this modal, the users can click on the title of any original table to open it . Second, when clicking on a block from WTS, the Working Table Component Detail modal pops up, which shows the title of the table and the columns under it. The users can click on the title of the table to open it. Lastly, when the users click on a cell from the working table, Governor opens the original table and locates the cell from the original table. If a cell in the Working Table corresponds to multiple cells in an original table due to joining, a modal will be displayed for the users to pick up which value they would like to locate.

Chapter 7

User Study

We conducted a user study to assess the effectiveness and usefulness of Governor. The general purpose of this study was to investigate how people use the system to perform ad hoc search and create an integrated table from multiple original tables that fulfill their goals and to understand the strengths and weaknesses of the system.

7.1 Participants and Apparatus

We recruited ten participants (five males and five females) via mailing lists at a local university and by reaching out to people from the open data community. Five participants are between age 18–34, four between age 35–54, and one between 55–74. All of the participants have a bachelor’s degree or higher (two Bachelor’s, six Master’s, and two PhDs) whose backgrounds include information technology, electrical engineering, law, public policy, and social services. Their self-reported familiarity with spreadsheet software (e.g. Excel, LibreOffice Calc, Numbers, Google Sheet) had a median of 4 and a mode of 4 on a scale of 1 to 5 (1: no familiarity; 5: advanced user of such software). Their self-reported familiarity with RDBMS (e.g. Oracle, PostgreSQL, MySQL, SQLite, Db2) had a median of 3 and a mode of 3 on a scale of 1 to 5 (1: no familiarity; 5: advanced user of such software). Four participants have prior experience working with open government data.

We conducted the study using remote video conferencing software. The system was deployed publicly as a web application, and participants accessed it from their personal computers.

7.2 Tasks and Design

As discussed in Chapter 3, two classes of existing tools partially provide Governor functions: tools that aim to make it easier for users to search and explore open data tables, and tools for integrating open data tables. A combination of open data search tools with a standard spreadsheet software such as Excel would enable our use cases and form a baseline, but switching between tools would require copy-pasting and would not be a fair comparison to Governor. Also, the tools that require users to write code to perform data integration [12, 25, 17] would not be comparable to Governor. The closest to our work are Auctus [14] and Toronto Open Data Search [29]. However, the table integration feature of both tools is limited to joining two tables and therefore cannot support efficient table integrations that require multiple joining and unioning operations. Thus, we decided not to include a baseline in our study design.

To evaluate Governor, we designed four tasks for our study:

T1: Search: Participants needed to find out how much money was granted by the Early Researcher Awards Program (Ontario province-level award) to the researcher “Ian Goldberg” from University of Waterloo. This task requires participants to employ the search and previewing feature of Governor, including finding and opening the table, adding additional columns, and applying a filter.

T2: Table Union: Participants needed to create an integrated table about how the border wait time at the Peace Bridge border office for travelers changed over 2015 and 2016 by unioning eight tables that are published quarterly together and applying a filter based on the keyword “Peace Bridge”. This task requires participants to interact with the Working Table view of Governor and utilize the unionable table suggestions feature.

T3.1: Pre-defined Table: Participants were presented with a pre-defined table with four columns via a shared link about how much funding was spent by charitable organizations in Canada on political activities in 2019, which was created by joining three tables. Then, we asked the participants the following questions:

- Can you describe what operations were performed to construct this table?
- From how many different tables does this table contain information? What are these tables?
- Can you open the original table that contains the “Description” column of charitable organizations?
- Can you show the following cell in the original table?

- “Legal Name: SecondStreet.org” (from one of the joined tables)
- “5030:6000” (from the main table)

This task tests whether participants can understanding a pre-defined table by interacting with the provenance features of Governor, such as color-coding and working table component detail modal. To test whether participants can locate a cell manually, we disable the “Locate in Original Table” feature in this task.

T3.2: Join + Union: Participants are required to extend the table presented in T3.1 to contain the data from 2017 to 2019. This task requires participants to understand the structure of table from T3.1, and also interact with both the joining and unioning feature of Governor.

T1 requires participants to find an answer from the open data table, which allows us to validate if participants could quickly find the record they are looking for via the table search and table preview feature of Governor. T2 requires participants to create an integrated table, which allows us to test if participants could easily integrate multiple tables with Governor. T3.1 and T3.2 require the participants to work with a pre-defined table, which allows us to test if the participants can understand the originality of the data with the provenance features of Governor. By having these four tasks, the study could examine all the features of the system comprehensively.

7.3 Procedure

We begin the user study using the “CIHR Grants and Awards”¹ dataset from open.canada.ca portal as an example. The structure of this dataset is similar to the NSERC dataset mentioned in Chapter 4, but focuses on the grants provided for healthcare research projects. The experimenter starts the introduction by searching for the keyword “CIHR” from the open.canada.ca portal, opens the dataset and shows the participant that the dataset consists of data from 20 years, which was published periodically in different data tables. Then the experimenter downloads the grants and awards table from 2000 and opens it in Excel to show the participants the columns and values stored in the table. The experimenter also picks up the value of one cell from the original table and shows the participants that the search feature of open.canada.ca cannot find the dataset by using this value. Next, as a contrast, the same search is performed with the tuple search feature in Governor, which shows the participants that the tuple search feature of Governor can find the table

¹<https://open.canada.ca/data/en/dataset/49edb1d7-5cb4-4fa7-897c-515d1aad5da3>

by a value stored in it. The experimenter also opens the table directly in Governor to demonstrate the table preview feature and shows participants that Governor supports hiding/unhiding columns, filtering, and sorting for the preview of the original table. Then, the original table is added to the working table and performed a table integration by unioning it with the table with grants and awards table from 2001 and joining it with the partner tables to add the name of the partner for each main grant application. Finally, the experimenter demonstrated the provenance features of Governor by showing the working table structure, color-toggling feature, working table component detail modal, and column composition modal with a focus on how to navigate to the original table from the working table. The purpose of the introduction session is to educate the participants about the background of the project to help them better understand the related concepts such as dataset and the purposes of Governor.

The participants were then instructed to perform two practice tasks. The first practice task asks the participants to find the large contracts between Microsoft and the government of Canada via the search feature, and the second practice task requires the participants to create a table very similar to that in the introduction session: NSERC Award winners from the University of Waterloo in 1991 and 1992 with their industrial partners. The detailed, step-by-step instructions are provided to participants on how to perform these tasks, and the experimenter can answer any questions raised by the participants. However, the participants were encouraged to think about how these tasks can be completed without first looking at the instructions. The above procedure ensured that the participants got some familiarity with the system and had adequate skills and knowledge to complete the actual tasks.

The participants were then asked to perform all the actual tasks. We did not give any direction or hint to complete the task unless the participants get stuck for a long time. After each task was completed, the participants filled in the NASA TLX questionnaire based on their experience during the task.

Finally, we conducted a semi-structured interview to collect their feedback. In the end, participants received \$20 Amazon gift card for their time and effort. The whole study lasted about 90 minutes for each participant. We screen-captured the task sessions and audio-recorded the interviews.

Chapter 8

Results

In this chapter, we report our results from the user study, including both quantitative measures and qualitative feedback. We refer the participant by number (e.g. P#) in the following sections.

8.1 Task Performance

T1: Search: On average, participants spent 2 min 57 s ($\sigma = 1$ min 9 s) for T1. Of all participants, eight perfectly completed the task by meeting all requirements, and two partially met the requirements by incorrectly reporting the total cost of the project as the government commitment. However, the mistakes due to misinterpretation of the column name and descriptions were mostly due to the ambiguity in the original open dataset and less of an indicator of the effectiveness of Governor.

An ideal solution to T1 would be: (1) use “Ian Goldberg” as a keyword to perform a tuple search, (2) open the table “Early Researcher Awards Program” , (3) unhide the column “Ontario Commitment”. Of the eight participants who completed the task well, only two followed a very similar approach, while six searched for “Early Researcher Awards” via tuple or description search. This reveals the flexibility of Governor, supporting multiple ways to find the record. Interestingly, two participants added the original table to the Working Table before applying the filter or un hiding the column. It is encouraging that the flexibility offered by Governor allows most participants to be effective in completing the task, even if they sometimes went off the “optimal” path. During the process, two participants were confused about the difference between the “tuple” and “description”

search modes and required clarification. These two search modes can potentially be merged into one universal search in the future.

T2: Table Union: On average, participants spent 6 min 20 s ($\sigma = 1$ min 30 s) for T2. All participants were able to complete T2 successfully. However, five got stuck a bit trying to figure out if they had the correct columns and three participants were confused because the data table for 2015 Q1 also contains the last few hours of data from December 31st of 2014. However, similar to T1, this is largely due to the non-descriptive column names and the data cleanness issue which existed in the original open dataset.

In this task, the participant can start by adding the data from one of the eight quarterly-published tables to the Working Table. Then, by expanding the panel “Add Rows from Other Tables (Union)”, the system can suggest the seven remaining data tables. The participant needs to click the “Union” button for each of the seven table to add them to the working table. All participants followed this approach. However, four participants complained during the process that the table integration requires too many clicks or the reloading after each click is annoying and asked if we had an alternative way to integrate all the eight tables at once.

T3.1: Predefined Table: On average, participants spent 8 min 16 s ($\sigma = 2$ min 5 s) for T3.1. All participants were able to answer the first two questions of T3.1 with the information provided by Governor and open the correct table for the third question. Among the ten participants, five found the required answer with the “Working Table Component Detail”, three used the color-coding of the table to determine the structure of the table, while two obtained the required answer from the history panel. This indicates that by providing the same provenance information in different ways, the system makes it intuitive for participants to understand the provenance of the data. Even without memorizing all the features of the system, the participant can still complete the tasks with the subset of features that they are familiar with.

On the other hand, the locating tasks (Q4) did not go smoothly for most of the participants. An optimal way to locate the cell from the original table is to open the original table and filter the data in it with the unique identifier (“BN”) of the row that contains the cell and then unhide the required column. However, only two participants followed this approach. Instead, the other eight participants tried to locate the cell by performing a filter based on the raw value of the cell. This approach works for the “SecondStreet.org” case, as there is only one row with this value in the original table. However, for the “6000” case, due to a large number of table cells that contain the same value, after filtering, the table still contains 2683 rows. Although in the end, of the eight participants, seven were able to locate the cell correctly by performing an additional filter, using sorting, or manually

matching information in other columns, the performance of the participants in this tasks indicates that it is not intuitive to locate a cell based on the unique identifier for most of the users. Being able to locate the cell automatically from the original table is still an important feature of the system.

Additionally, this task also reveals Governor’s weakness in filtering the data. Since Governor only features a global filter, participants cannot filter based on the column they are interested in. If the participant can perform the filter based on the “5030” column, they will be able to complete the locating task much quicker, as there will only be two rows remaining in the table after filtering.

T3.2: Join + Union: On average, participants spent 2 min 10 s ($\sigma = 1$ min 25 s) for T3.2. An optimal solution for this task is to first add rows from the “Financial data” table for 2017 and 2018, then use the system’s suggestions to automatically fill the unfilled blocks and complete the Working Table. Of the ten participants, nine followed this approach, while one got stuck trying to add more columns due to misunderstanding the instructions.

8.2 Questionnaire Ratings

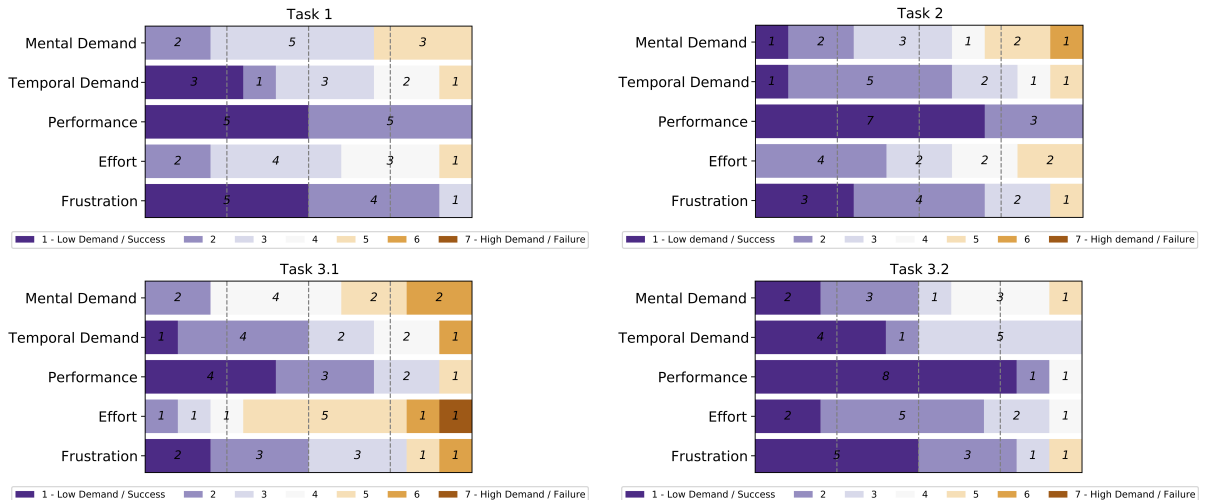


Figure 8.1: Participants’ ratings on the NASA TLX questionnaire for the Tasks (the lower the better)

Figure 8.1 shows the participants’ ratings on the NASA TLX questionnaires, respectively. We can see that for T1, most of the participants (at least 7 out of 10) rated 4 or less

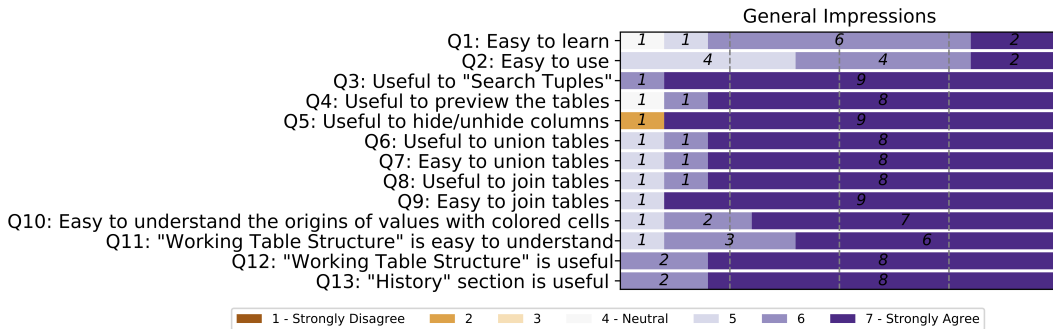


Figure 8.2: Participants’ ratings on the exit-questionnaire (the higher the better)

on each question, indicating that they were comfortable of using Governor for searching and felt successful in doing their tasks. All participants indicated that they had successfully completed the task, while the mental demand and effort involved depend on the search strategy they used. The results of T2 are similar to those of T1, with a slightly higher rating for frustration. This is consistent with the complaints that the system does not offer the bulk operating feature to perform the union, as discussed in Section 8.1. The results of T3.1 show a high level of effort and mental demand, as most of the participants were unable to locate the cell of “6000” at first attempt. While most of them are able to successfully locate the cell in the end, it is not a smooth experience. Finally, the results of T3.2 are slightly better than those of T1 and T2, which might be because participants got familiar with the table structure in T3.1 and also became more proficient with the system after performing the first three tasks.

Moreover, the results of the exit-questionnaire as shown in Figure 8.2, in general, indicate that the participants had a good experience of using Governor. Most of them thought that the system was easy to learn and use (Q1 and Q2). They thought all the core features of Governor, including searching (Q3), previewing the original tables (Q4), adding rows via unioning (Q6), and adding columns via joining (Q8) are useful. Furthermore, they thought that it was generally easy to perform these operations in Governor (Q7 and Q9). Participants especially appreciate Governor’s abilities to summarize the provenance of the integrated table with color-coding and Working Table Structure. They think that it is easy to understand the origins of values with colored cells (Q10), and the information provided in the Working Table Structure is useful and easy to understand (Q11 and Q12).

8.3 Qualitative Results

Our interview focused on collecting feedback from participants on Governor. In general, participants, especially those who work with open data as part of daily job, thought that the capabilities provided by Governor are useful and intuitive. *“The app is fantastic and I can’t wait to use it. Please let me know when it’s available.”* - P9. In this chapter, we report our results based on the design goals in Chapter 5.

G1: Store and index the tuples inside the tables Indexing the tuples of open data tables for search is a key feature of Governor. In general, participants had a good impression about the tuple search feature. For example, P1 said: *“while I do not work with open data, I can definitely see it (the tuple search feature) useful.”* This is echoed by P3 and P6. However, the distinction between Tuple Search and Description Search can be confusing for some users. For example, P5 commented that *“I still do not get the difference between the tuple and description search.”* Similarly, P10 said that *“The term tuple can be confusing to people from non-technical background.”* Finally, P4 suggested that we should merge the two search modes: *“What if I want to match both the description and the value? Can it do that?”*

G2: Support core relational data processing operations interactively Governor supports the core relational data processing operations, including union and join operations interactively via the front end DuckDB, as discussed in Chapter 6. From the user interaction perspective, we would like to study whether the key concepts in relational data processing such as union and join can be understood by the end users. In our study, five of the participants felt these terminologies easy to understand, while the other five participants thought that the terminology can be confusing to the end users and offered us alternative suggestions. For example, P2 suggested that we *“add a description or small example for joining”* in order to make it more understandable to the user. P3 offered a similar suggestion. P6 suggested that we use different words for “union”, such as “append” as *“‘union’ is a mathematical term”*. Similarly, P7 suggested us to replace the wording of “join” with “combine”. Lastly, P4 suggest us to use visual cues for the join and union: *“You could do something similar to Apple Numbers. I remember that they have a small ‘Add’ button at the last row and column of the table. You can do something similar for the Working Table Structure.”*

G3: Provide suggestions for related tables As discussed in Chapter 5 and Chapter 6, the system uses the set overlap score to detect unionable and joinable tables from the corpus. In general, the participants find the recommendation useful. For example, P3 said: *“It is amazing that it can do this (recommending joinable tables).”* However, one

concern raised by several users is the limitation of being able to integrate only the tables suggested by the system. For example, P7 asked for the feature of joining an arbitrary table: *“Hmm... Do I have to pick up from the list (suggested by the system)? Can I join the table based on my own choice?”* Similar questions are also raised by P4 and P9.

G4: Manage integrated datasets and their provenance to support later fact checking and verification In Governor, the main visual cue provided by to distinguish different tables that have been integrated is the color coding. Each integrated table is assigned a color automatically by the system automatically and the color is used consistently across multiple user interface component to help the user understand the provenance of the integrated tables. This design is favoured by most of the participants. For example, P2 said: *“I really like how you use the colors.”* The same opinion is echoed by P3, P5, P7 and P9. However, P8 complained that she is confused by the colors used by the system: *“There are like ten colors. I cannot remember which color corresponds to which table.”* P10 also brought up an interesting point regarding the accessibility of the system: *“There are more color-blind people than you know. You should think about them (when designing the system). Maybe you can add some patterns to the blocks.”*

Chapter 9

Discussion

While the results of the user study indicate the effectiveness of Governor for supporting participants to find open data tables via search, perform table integration, and understand the provenance of data in general, the system still has some drawbacks. Our user study has revealed three broad usability challenges of Governor including table integration process, spreadsheet functionalities, and searching. In this chapter, we first discuss these challenges and then discuss other limitations of Governor and our study.

Table Integration Process: Currently, Governor only supports constructing the working table step by step. After each operation is performed, the working table is reloaded to reflect the change immediately. For example, for T2, the users have to click the “Union” button for eight times, and also wait for the working table to be reloaded for eight times to construct the final table. Four participants requested the support of bulk operations for table assembling or suggested that reloading the working table after each operation is inefficient. While the step-by-step table integration may be more straightforward to the inexperienced users of the system by allowing them to review the results before performing the next step, it might be annoying and time-consuming for expert users who would like to integrate a large number of tables together.

Spreadsheet and Data Analytic Functions: Currently, Governor does not support advanced table or database operations, such as custom data aggregation, range selection, table pivoting, charts plotting, etc, which were requested or mentioned by several participants. For example, three participants requested a column-based table filter during the study, two mentioned the data aggregation capability in the interview section, and two requested the functionality of reordering the columns in the tables. While our focus in Governor is on searching and integrating open data tables, it is also clear that these

spreadsheet functionalities are necessary to provide a better experience for users integrating the tables and summarizing the data using Governor. Such functionalities would also enable the users to answer some interesting questions directly in Governor without having to export the data to a spreadsheet software. For example, if users would like to figure out the total amount of NSERC funds granted to a specific institution within a year using the current version of Governor, they have to export the data from Governor into a spreadsheet software in order to perform a sum. However, if the sum aggregation function is provided by Governor, this could be done within the system directly.

Search Feature: Currently, Governor provides the tuple search and the description search as separate features. This distinction is partially due to the back end architecture, which indexes the metadata of the datasets with MongoDB and the values from the original tables with Elasticsearch separately. Two participants find this distinction confusing and requested us to merge the tuple search and description search into one universal search.

Other Limitations: Governor is currently implemented with the corpus of CSV files crawled from open.canada.ca. Governor’s join and union features are also limited to the tables suggested by the system as discussed in Chapter 6. This did not create much problems in our user study as we limited the scope of table integration in the tasks for the user study. However, in a realistic setting, to gather data for certain analytic tasks, users may need to gather data from multiple open data portals, which is not currently supported by Governor. For example, to compare how the number of COVID-19 cases and vaccinations differs across provinces of Canada, the user may need to source the data from multiple provincial levels OGDs, such as the Ontario Data Catalogue¹ and the BC Data Catalogue².

Study Limitations: Our user study has few unavoidable limitations. First, we did not compare our system with any baseline systems because no prior system we are aware of provides the full functionality of Governor, though a baseline can be formed by allowing users to use a mix of systems, e.g., an open data search engine along with a spreadsheet software or a relational database management system. However, we did not think that is a fair comparison given that even these combinations could require a lot of copy pasting or prior experience with the SQL query language. Second, participants did not create tables based on their own needs; instead, we designed all of the tasks in a restrictive way. While this allowed us to better control what participants could do and derive consistent insights, we could miss many other factors about the usability of Governor. Third, our study was conducted in a lab setting and we did not have a large number of participants.

¹<https://data.ontario.ca/>

²<https://www2.gov.bc.ca/gov/content/data/bc-data-catalogue>

Future evaluation of Governor within a realistic setting would be necessary to reveal more potential usability issues of the system.

Chapter 10

Conclusions and Future Work

We have presented Governor, a web application developed to make open data tables more accessible to end users. Governor utilizes set overlap algorithms to automatically detect joinable and unionable tables within the OGDs and allows users to preview the data tables and integrate them with each other interactively at the front end by leveraging an embedded SQL OLAP database at the front end. Additionally, Governor manages the provenance information of the integrated table and provides a color-guided visualization for the users to understand how the tables are integrated. A user study was conducted to assess the strengths and weaknesses of Governor. The results indicate the effectiveness of Governor in open data table search and integration tasks and the usefulness of the provenance features.

We would like to continue extending the features of Governor to address the limitations discussed in Chapter 9, by providing data cleaning, analysing, and visualization features. These features would allow users to extract more interesting statistics from open data tables without exporting the table to another software. Also, developing a “professional mode” which allows the users to perform custom table unioning and joining beyond those suggested by the system, and construct the final working table by selecting multiple tables at once without intermediate previews would make the system more flexible and efficient for expert users.

In this thesis, we focused on open data tables crawled from open.canada.ca. In the future, we would like to enrich the data corpus of Governor by crawling and maintaining the open data from multiple OGDs as well as allowing the users to upload their own tables into the system. We can also combine the data published by the OGDs with knowledge graphs similar to [24], in order to allow the users to find and integrate additional

information that is not available in OGDs about an entity. For example, in the usage scenario described in Section 4.1, the age of Yoshua Bengio does not exist in any open data table, but this information can be easily found on DBPedia¹. Doing so would also raise several important challenges, such as knowledge graph management and entity alignment, which we identify as future research directions.

¹https://dbpedia.org/page/Yoshua_Bengio

References

- [1] Apache Arrow <https://arrow.apache.org/>, 2022.
- [2] Apache Parquet <https://parquet.apache.org/>, 2022.
- [3] CKAN <https://ckan.org/>, 2022.
- [4] Cloudflare <https://cloudflare.com/>, 2022.
- [5] DuckDB <https://duckdb.org/>, 2022.
- [6] Elasticsearch <https://www.elastic.co/>, 2022.
- [7] Google Dataset Search <https://datasetsearch.research.google.com/>, 2022.
- [8] MongoDB <https://mongodb.com/>, 2022.
- [9] Open Data Principles <https://open.canada.ca/en/open-data-principles>, 2022.
- [10] Vue.js <https://vuejs.org/>, 2022.
- [11] WebAssembly <https://webassembly.org/>, 2022.
- [12] Alex Bogatu, Norman W. Paton, Mark Douthwaite, and Andre Freitas. Voyager: Data Discovery and Integration for Data Science. In *EDBT*, 2022.
- [13] Dan Brickley, Matthew Burgess, and Natasha Noy. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. In *WWW*, 2019.
- [14] Sonia Castelo, Rémi Rampin, Aécio Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *PVLDB*, 14(12), 2021.

- [15] Raul Castro Fernandez, Jisoo Min, Demetri Nava, and Samuel Madden. Lazo: A Cardinality-Based Method for Coupled Estimation of Jaccard Similarity and Containment. In *ICDE*, 2019.
- [16] Alon Y. Halevy. Answering Queries Using Views: A Survey. *VLDBJ*, 10(4), 2001.
- [17] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. A Demonstration of KGLac: A Data Discovery and Enrichment Platform for Data Science. In *VLDB*, 2021.
- [18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.
- [19] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table Union Search on Open Data. *PVLDB*, 11(7), 2018.
- [20] Ouellette, Paul and Sciortino, Aidan and Nargesian, Fatemeh and Bashardoost, Bahar Ghadiri and Zhu, Erkang and Pu, Ken Q. and Miller, Renée J. RONIN: Data Lake Exploration. *PVLDB*, 14(12), 2021.
- [21] El Kindi Rezig, Anshul Bhandari, Anna Fariha, Benjamin Price, Allan Vanterpool, Vijay Gadepally, and Michael Stonebraker. DICE: Data Discovery by Example. *PVLDB*, 14(12), 2021.
- [22] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. Chapter 7: Relational database design. In *Database System Concepts, Seventh Edition*, pages 303–360. McGraw-Hill Book Company, 2020.
- [23] Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, and Sherif Sakr. Rdf data storage and query processing schemes: A survey. *ACM Comput. Surv.*, 51(4), 2018.
- [24] Siyuan Xia, Nafisa Anzum, Semih Salihoglu, and Jian Zhao. KTabulator: Interactive Ad Hoc Table Creation Using Knowledge Graphs. In *SIGCHI*, 2021.
- [25] Zhang, Yi and Ives, Zachary G. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*, 2020.
- [26] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*, 2019.

- [27] Erkang Zhu, Yeye He, and Surajit Chaudhuri. Auto-Join: Joining Tables by Leveraging Transformations. *PVLDB*, 10(10), 2017.
- [28] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. LSH Ensemble: Internet-Scale Domain Search. *PVLDB*, 9(12), 2016.
- [29] Erkang Zhu, Ken Q. Pu, Fatemeh Nargesian, and Renée J. Miller. Interactive Navigation of Open Data Linkages. *PVLDB*, 10(12), 2017.