applied sciences

MDPI

Article

Ad Hoc Data Foraging in a Life Sciences Community Ecosystem Using SoDa †

Kallol Naha ‡ and Hasan M. Jamil *,‡

Department of Computer Science, University of Idaho, 875 Perimeter Drive, Moscow, ID 83844, USA; naha7197@vandals.uidaho.edu

* Correspondence: jamil@uidaho.edu; Tel.: +1-(208)-885-9654

† This paper is an extended version of our paper published in Jamil, H.M. Supporting Data Foragers in Scientific Computing Community Ecosystems for Life Sciences. In Proceedings of the Information Integration and Web Intelligence—26th International Conference, iiWAS 2024, Bratislava, Slovak Republic, 2–4 December 2024.

‡ These authors contributed equally to this work.

Abstract: Biologists often set out to find relevant data in an ever-changing landscape of interesting databases. While leading journals publish descriptions of databases, they are usually not recent and do not frequently update the list that discards defunct or poor-quality databases. These indices usually include databases that are proactively requested to be included by their authors. The challenge for individual biologists, then, is to discover, explore, and select databases of interest from a large unorganized collection and effectively use them in their analysis without too large of an investment. The advocation of the FAIR data principle to improve searching, finding, accessing, and inter-operating among these diverse information sources in order to increase usability is proving to be a difficult proposition and consequently, a large number of data sources are not FAIR-compliant. Since linked open data do not guarantee FAIRness, biologists are now left to individually search for information in open networks. In this paper, we propose *SoDa*, for intelligent data foraging on the internet by biologists. SoDa helps biologists to discover resources based on analysis requirements and generate resource access plans, as well as storing cleaned data and knowledge for community use. SoDa includes a natural language-powered resource discovery tool, a tool to retrieve data from remote databases, organize and store collected data, query stored data, and seek help from the community when things do not work as anticipated. A secondary search index is also supported for community members to find archived information in a convenient way to enable its reuse. The features supported in SoDa endows biologists with data integration capabilities over arbitrary linked open databases and construct powerful computational pipelines using them, capabilities that are not supported in most contemporary biological workflow systems, such as Taverna or Galaxy.

Keywords: large language model; intelligent user interface; FAIR; wrapper generation; interoperability; ecosystem

check for updates

Academic Editor: Zhibin Lv

Received: 12 October 2024
Revised: 25 December 2024
Accepted: 27 December 2024
Published: 10 January 2025

Citation: Naha, K.; Jamil, H.M. Ad Hoc Data Foraging in a Life Sciences Community Ecosystem Using SoDa. *Appl. Sci.* **2025**, *15*, 621. https://doi.org/10.3390/app15020621

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The extraction techniques needed to gather data that are stored in databases behind firewalls are significantly different from browsing web page data on the internet [1]. The standard tools used to access deep web data are wrappers [2] and, more recently, RESTful APIs [3]. Word-of-mouth [4] and web crawlers [5] also play a significant role in the generation of resource indices, such as MBDL [6] or Pathguide [7]. While the compiled

resource indices help with the discovery and identification of interesting data sources and analysis tools that users need, the process is completely manual, sluggish, and limited. They are slow to update and to include new resources, and do not serve the unique needs that users may have. In particular, they do not actively find resources of interest that are not already indexed in the resource list.

Linked open data (LOD), on the other hand, has emerged as a promising approach to provide interconnected, machine-readable datasets, but its effectiveness is often hampered by issues related to data quality and accessibility. While LOD emphasizes openness, the FAIR (Findable, Accessible, Interoperable, and Reusable) principles [8] extend this concept, revealing significant gaps in compliance among LOD resources [9]. Tools such as the FAIR-Checker [10] have been developed to evaluate and improve FAIRness in LOD; by utilizing standardized metrics and criteria, this tool enhances transparency and reliability in LOD practices. However, the widespread adoption of FAIR principles remains a challenge, hindering the full realization of LOD's potential in supporting scientific inquiry and data-driven research. It is interesting to note that LOD does not imply FAIR, or vice-versa [11], and both of these concepts are not supported by set standards and thus are very hard to achieve. The question that remains open is how users can find the most useful and relevant resources accessible via the open internet?

In this paper, we introduce the concept of data foraging and sharing in a community resource ecosystem [12] that will be curated [13] and will evolve with time [14]. We believe that the ingredients required to construct such an ecosystem are in place and a careful integration of them is ultimately possible. We discuss the architecture and functionalities of the proposed system called *SoDa* (stands for Solo Data Forager). In SoDa, we integrate four basic subsystems—a resource recommender system, a data access protocol designer or a wrapper system, a query processing system with the help of a schema matcher to support interoperability, and a curation system for knowledge evolution (we invite readers to consult [15] for a formal exposure to the concept of wrappers, and [16] for a discussion on schema matchers. In the presentation to follow, we assume that readers are familiar with these standard concepts in database integration technology). We illustrate the functionality of SoDa using an example from life sciences research.

We would like to emphasize that while there are several prominent biological data integration and workflow systems available (e.g., Galaxy [17], Taverna [18], Kepler [19], VisFlow, etc.), they are largely limited in their capabilities, especially when integrating arbitrary databases and constructing workflows using them. They are also not capable of helping users find the data sources or tools needed to construct a computational pipeline. Some of them are very powerful and supportive since they restrict the resource type they support. For example, Taverna only supports web service compliant resources, while Galaxy largely supports databases that have pre-arranged cooperative arrangements with Galaxy. Such limitations hinder interesting data analysis and scientific expeditions using interesting resources not supported by these systems. For example, the gene-disease association database DisGeNet [20] is not supported by Galaxy or Taverna, nor can they help users discover this database or make them use it in any analysis pipeline. The issue we address in SoDa is as follows: with no prior knowledge or awareness, can a biologist find DisGeNET and use it in a workflow along with another database that potentially has schema disparity with it in a matter of minutes, all without writing a single line of code.

2. Related Research

Powerful data integration systems, such as Galaxy and Taverna, could be limiting for some applications. For example, finding new resources and integrating them into the Galaxy tool set requires significant expertise and coordination with the Galaxy developer

team, and thus is not suitable for ad hoc spur-of-the-moment usage for novel queries— it requires planning, coordination, and technical knowledge. Taverna, on the other hand, allows user-initiated resource integration, but requires web service compliance for resources. Even when they do comply, the required tooling is extensive and requires significant preparation and coding know-how. In both systems, implementing a scientific inquiry of the form discussed in Section 3.1 will be extremely difficult, if not impossible, by a biologist who is not computationally intelligent, and there are a large number of such biologists.

Recent efforts such as Voyager [21] for data discovery and integration though a step in the right direction, it does not support many needs that we address in SoDa, some of which are expressed in [22]. Such deficiencies in the contemporary system that supports an end-to-end data discovery, integration and querying [23] are leading to custom application developments such as BioBankUniverse [24] and DZL [25]. The landscape of data discovery, scientific inquiry, and integration is being reshaped by the emerging LLM technology. New application design platforms will need to support not only complex SQL queries post discovery and integration, they must support automatic computation of predictive and other forms of analyses only possible through the incorporation of machine learning capabilities with querying engines. The SoDa system we introduce in this paper approaches the science of scientific inquiry from these emerging perspectives.

3. SoDa Architecture

As shown in the conceptual model of SoDa in Figure 1, it has four major components: (i) ResCom resource recommender, (ii) CroW wrapper generation system, (iii) Crowd curation tool QCurator, and (iv) Application designer SoDaPro. ResCom helps identify databases of interest when prompted using a natural language description of the required data. It also helps generate a query plan. CroW is a semi-automatic wrapper generator that helps SoDa extract data from deep web databases. If users make mistakes or the CroW's database access instructions fail to function as expected, users are able to seek community help using QCurator to correct the errors. Finally, using SoDaPro subsystem's QBuild tool, users can construct queries or analytics and execute them.

In the sections to follow, we discuss these components on intuitive grounds. As a preamble to the discussion of the architecture of SoDa, we introduce a potential scientific expedition a biologist might want to carry out using our cloud-based analysis platform for open science over linked open data.

3.1. Scientific Inquiry

In their quest to establish a definitive link between defects in sperm and male infertility [26], a biologist might want to follow the steps below once semen samples from both fertile and infertile men are collected and sequenced (among several other steps before and after).

1. Perform differential expression analysis to identify RNAs that are significantly differentially expressed between fertile and infertile men using commonly used tools such as DESeq2, edgeR, or Limma.
2. Use tools like DAVID, Enrichr, or clusterProfiler to perform Gene Ontology (GO) and pathway enrichment analysis on the differentially expressed RNAs.
3. Identify biological processes, molecular functions, and pathways that are significantly associated with the differentially expressed RNAs.

This project could be approached at multiple different levels depending on the biologist or their expertise. For example, a more well informed and prepared researcher in fertility research may already have collected sperm samples from both fertile and infertile men, isolated RNA from these sperm samples, identified and quantified the RNA molecules

present in the sperm samples using RNA sequencing and learned about the RNA profiles of fertile and infertile men before entering differential expression analysis.

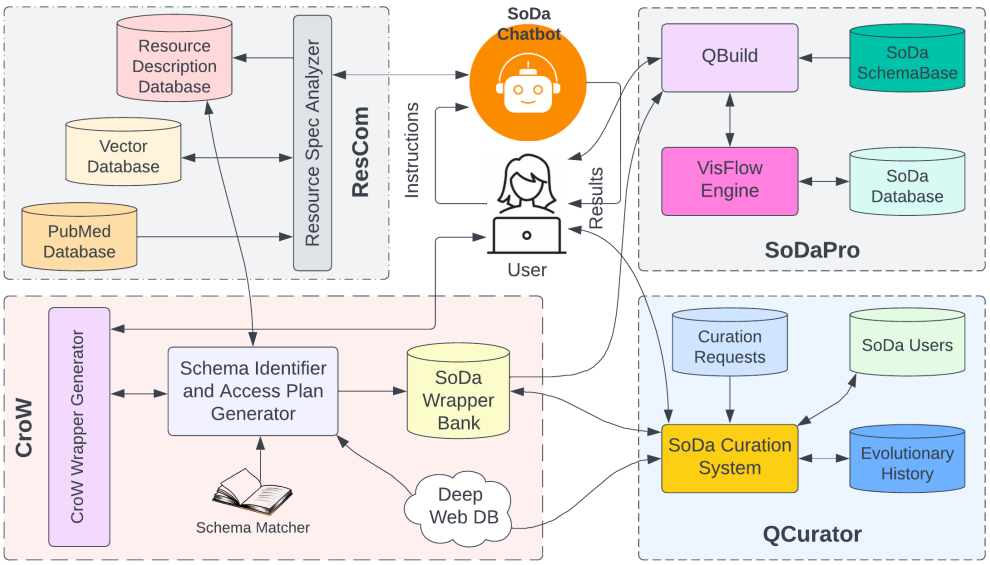


Figure 1. SoDa Architecture: the SoDa system consists of four interconnected sub-systems (i) ResCom resource recommender, (ii) CroW wrapper generation system, (iii) Crowd curation tool QCurator, and (iv) Application designer SoDaPro. ResCom parses natural language user requests or queries and generates a query plan or workflow. It also identifies the required internet resources to compute the query. CroW is a semi-automatic wrapper generator that helps SoDa extract data from deep web databases. If users make mistakes or the CroW access instructions fail to function as expected, users are able to seek community help using QCurator to correct the errors. Finally, using SoDaPro subsystem's QBuild query design tool, users can construct queries or analytics and execute them.

An investigator who just wants to understand a possible link between sperm defects and its impact on fertility may not be as prepared as the researcher discussed above. Therefore, an alternative to reach this stage is to use publicly available data in GEO [27] or SRA [28] databases and start right away. However, the querying abstraction levels could be highly varied. In a recent study on ProAb, we explored the interesting possibility of asking this query at the highest possible abstraction level in natural English, likely as follows:

> *Is there a link between human spermatozoal RNA and infertility? Could I establish the link computationally?*

We consider the question, could a candidate workflow be developed that could be executed fully automatically? However, in a cognate research in BioNursery, we demonstrated that significant technological and knowledge gaps exist that hinder this approach, and often, substantial human involvement is necessary.

In SoDa, the abstraction levels supported help find relevant data and tools automatically but requires users' involvement in selecting, isolating, and generating a plan of how to compute the response to a scientific query. Steps 1 through 3 (described earlier) on how to link sperm defects with male infertility is a plan that we treat as a computational information extraction procedure that can be used to automate the query computation. Once collected and archived, and once users already have a well articulated computational process in mind, they are able to develop a computational pipeline to implement a scientific inquiry using the smart SoDa GUI as described in the following sections.

It is perhaps interesting to note here that traditionally, users visit these database sites, submit queries by hand and manually collect data by scrapping, copying or downloading data, and then saving them in some way before analyzing them. A significant amount of

time is expended in the manual search of databases and tools, as well as the collection and processing of data. In contrast, in SoDa, users are aided by resource identification tools, data extraction, and query processing tools and the queries are computed almost instantly.

3.2. Resource Recommendation

Existing resources in SoDa can be browsed from a searchable index, or a specification in the form of a paragraph for the resource needs can be used as a search key in natural English. For example,

> *Need to find normalized sperm RNA-seq expression data for differential expression analysis (using DESeq2, edgeR, or Limma).*

The Resource ReCommender system ResCom accepts this request and suggests a ranked list of databases, prioritizing the most relevant at the top. Not only does ResCom suggest the databases, it also generates a possible scheme of a table that can be accessed via a web link, i.e., a URL.

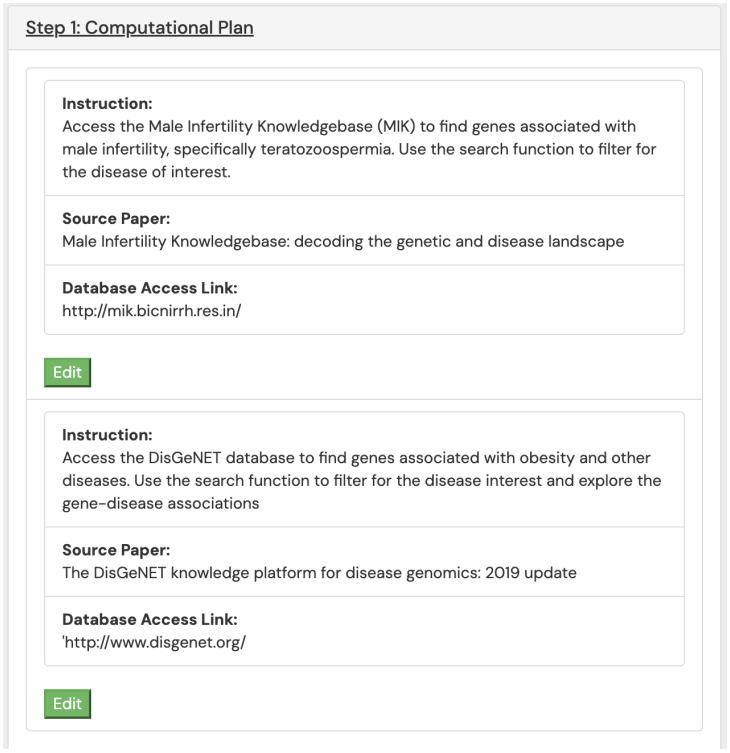
In response to the above request, as an example, ResCom will first search for a matching data set in SoDa archive. In SoDa, all table schemes are semantically described in some detail using natural language and their possible usage is also included in these descriptions. These descriptions are stored in SoDa's vector database for a possible linguistic analysis using an LLM, such as ChatGPT, to ascertain query relevance.

In the event a table could not be identified, or the user requests an external search, ResCom uses PubMed abstracts as the first level of descriptors of data to find a match. If a sufficient match is found, the list of abstracts are organized, ranked in order of relevence, and vectorized for semantic matching. Database links found in the abstracts are explored exhaustively using a link, hoping to identify a database with the closest match and presented to the user for review with two options—accept or search next. The search ultimately ends in a success or failure.

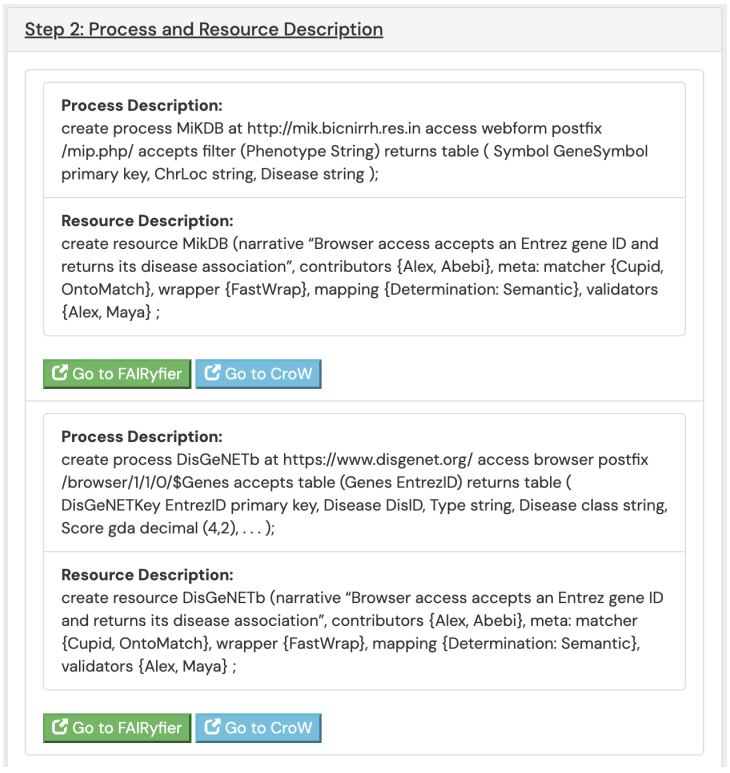
The process of successfully identifying resources undergoes a secondary step. In this step, a machine-readable and editable description on the nature and capabilities of the resource is generated (see Figure 2 showing this process for the resource description generation for the DisGeNET (https://www.disgenet.org/ (accessed on 1 December 2024) database). The resource description can be used in multiple different ways. The major use of the descriptions, however, is by the workflow generation system Needle [29], as discussed in Section 3.5. Needle uses resource descriptions to identify schema heterogeneity and schema matchers for their resolution, suitable wrappers for data access, and the trustworthiness of the resource. A crowd computing approach is adopted to ensure the accuracy and usefulness of the generated resource description within a community knowledge sharing ecosystem.

3.3. Accessing Resources

Once an external SoDa resource is identified, a wrapper needs to be generated to facilitate real-time online access and ensure successful extraction. In SoDa, resources are of two types—a deep web database (a database that can only be accessed via query form submission or through an API), or an online analysis tool that returns a table on the appropriate submission of input parameters, again likely using forms. In SoDa, the underlying data model is relational, and thus all its operations are conceived using a tabular representation of data, although the engine is capable of processing TXT, CSV, XML, and JSON formatted data.



(a) Computational plan generation.



(b) Generating process description.

Figure 2. Resource discovery and resource description generation in ResCom: These figures show how ResCom's Resource Specification Analyzer finds relevant resources from PubMed database and helps generate an execution plan and resource descriptions for the QBuild query builder using an LLM with the help of users and the wrapper generation system CroW.

However, developing an access protocol for such resources is mostly a manual process, largely because each one is unique, but follows a simple mathematical relation as follows.

$$\gamma : \mathcal{U} \rightarrow (\mathcal{T} \rightarrow \mathcal{T})$$

where γ is a function (the wrapper generator) that maps a resource $u \in \mathcal{U}$ (represented by a URL) into a function $\omega \in \Omega$ of the form

$$\Omega \subseteq \{\omega | \omega : \mathcal{T} \rightarrow \mathcal{T}\}$$

Thus, $\gamma(u) : \mathcal{T} \rightarrow \mathcal{T}$ or, $\gamma(u)(t) = t \in \mathcal{T}$, where $u \in \mathcal{U}$ is a URL and $t \in \mathcal{T}$ is a table. The access plan we develop is a learned function, ω from $\gamma(u)$, that would transform an input table t into a result table t' using the resource at the URL u.

To be able to develop the function ω, we use a GUI-enabled public wrapper generation system called *CroW* (stands for Crowd Wrapper Generator). CroW supports visual tools and functions to help CroW to learn the form of behavior and data layouts so that a formula can be learned, which CroW can then use to recreate the access plan for a site u when requested by a query in real time (the wrapper generation process for the DisGeNET database is shown in Figure 3). Once a wrapper is generated and tested, it can be archived in a searchable wrapper bank (a wrapper bank is a storage, or a database, of wrapper definitions in a machine-readable format, and thus can be reused as needed to retrieve information from a deep web database) for community use, as shown in Figure 4.

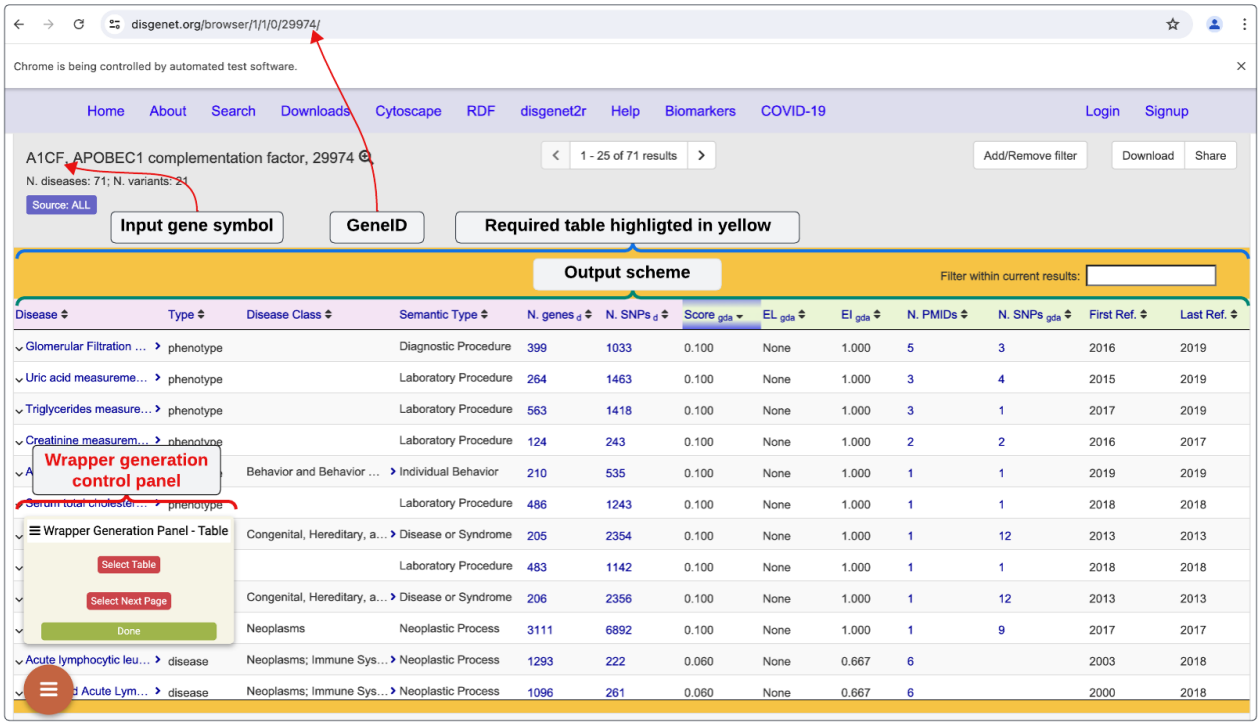


Figure 3. CroW wrapper generation toolkit: CroW allows a deep web database page to be loaded in HTML and manually identifies interesting features by marking them with a pointing device. It also allows users to characterize the marked elements on the page. Once marked and requested, CroW can generate resource descriptions in Needle, which SoDa can use to extract the data. This picture shows such a process for the DisGeNET database.

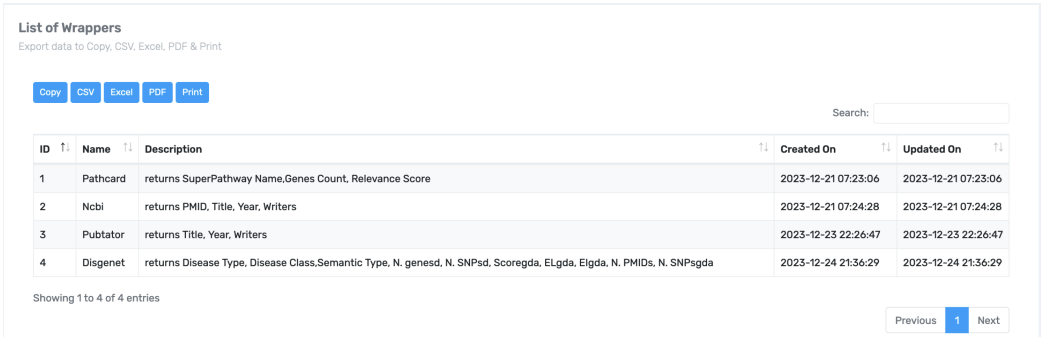


Figure 4. SoDa wrapper bank: the wrappers generated using CroW, as in Figure 3, can be materialized and stored for posterity. The table in this picture shows a searchable index of materialized wrappers ready to be used in user applications.

3.4. Application Design

Application design using SoDa always involves registered resources in SoDa, and never involves resources that SoDa has generated no access plans for. SoDa supports a graphical query builder and allows fairly complex query generation, and computed view materialization, either temporarily or indefinitely. The application designer is called *QBuild* (stands for Query Builder). Except for the directly stored tables, all references to a table are virtual, which means, a reference to a table on the internet is made through the use of a wrapper available in a SoDa wrapper database.

3.4.1. SoDa Query Language

QBuild uses our previously developed BioFlow query language construct extract to access the referenced tables in real time and treats the extracted tables as traditional tables in the back-end relational database, MySQL. The extract statement has a general form, as shown in Figure 5.

extract $A_1, A_2, \ldots, A_n$
using wrapper W, mapper M, filler F
at φ
submit r

Figure 5. BioFlow extract statement syntax: the extract statement can retrieve and return a table with columns $A_1, A_2, \ldots, A_n$ from a deep web database at the URL φ by submitting each row in table r as parameters to the database. Schema disparities are resolved automatically using the schema matcher M, and tables are extracted using the wrapper W. If a form filling operation is required, it is carried out using the filler F.

In this statement, a wrapper is a function that hides the internal details of a database or web site and offers an abstract view of it. In this view, a deep web database or a web site can be viewed as a table returned by the database or a site when it is supplied with a set of input values. For example, the table in Figure 3 is returned by the DisGeNet database when the GeneID 29974 (gene symbol A1CF) is submitted to it. A wrapper would accept this GeneID and scrape the table from this HTML page and return a table in .txt or .CSV format. A schema matcher, on the other hand, would resolve the schema disparities between two views. For example, a user may refer to a column name as Gene, while the DisGeNET database may refer to it as G_ID. A schema matcher may establish the correspondence that says Gene and G_ID are identical. Finally, a filler is a function that helps mimic form filling in web sites, such as submitting the GeneID 29974 to the DisGeNET database form at the following URL: https://disgenet.com/ (accessed on 1 December 2024).

For any table r, the set of column names by which it is defined is called the scheme or r, denoted as $r(R)$ where R is the set of column names. The extract statement above returns

a table t with column names $A_1, A_2, \ldots, A_n$ when it is supplied with a table r with columns $B_1, B_2, \ldots, B_m$. The scheme $S_o = \{A_1, A_2, \ldots, A_n\}$ is called the output scheme of the extract statement, while the scheme of r, i.e., $S_i = \{B_1, B_2, \ldots, B_m\}$, is called the input scheme of the extract statement.

BioFlow approaches the deep web resource querying problem declaratively. In this approach, a deep web database is modeled as a function in an internet repository φ. This function, when prompted with a table r_i over a scheme S_i, returns a table r_o over a scheme S_o. The extract statement above provides the syntax needed to implement this function. The interesting and unique component of the extract statement is its using clause. This option specifies the set of required tools needed to access a hidden web database. The application of these tools provides the needed abstraction of the function and hides the complexity of retrieving data resolving possible schema disparity using a schema matcher and possible scraping of tables (in case it is returned as an HTML document) using the wrapper W from the returned document. The wrapper essentially captures all necessary access details, as discussed in Section 3.3, including converting returned data by database d at φ into a tabular form. Any schema heterogeneity and mismatch between the input scheme $S_i = R = \{B_1, B_2, \ldots, B_m\}$ and the output scheme $S_o = \{A_1, A_2, \ldots, A_n\}$ are resolved by the schema matcher M, and the form filler F helps with the construction of the endpoints needed to process each element in the input set. Both BioFlow and SQL play a significant role in QBuild in the construction and execution of user workflow queries as discussed in the following sections.

3.4.2. Query Builder Interface

As alluded to earlier, SoDa supports two types of query constructs—extract statement for deep web data extraction, and select statement for querying tables. It also supports workflow orchestration using these two query types by allowing the logical sequencing of them. It does so using a query construction toolbox similar to SQL's QBE query builders [30], called *QBuild*, as introduced in the SoDaPro component of the SoDa architecture in Figure 1. QBuild comprises two main GUI interfaces—one to construct the equivalent of the extract statements to access deep web databases, and the other to write SQL queries to interrogate the locally stored databases in SoDa. Figure 6 shows the first of the two screens of the QBuild query builder. Using this interface, users are able to retrieve data from any deep web databases. The second interface to query locally stored data is shown in Figure 7. We only briefly outline some of its features due to limited space. This interface is rich in features and powerful compared to most contemporary visual SQL query builders; it deserves a more complete discussion and is outside the scope of this article. Only a brief discussion is presented below.

In both of these interfaces, the top bar above the multi-functional canvas shows function selector buttons as icons that are active one at a time (in a similar way to radio buttons). The icons are "Run" or "Play", "Pause", "StepThrough", "Download" or "Save", "Delete" or "Restart", and "Share". Below this bar, there are five tabs that allow an extract query, SQL query, SQL subquery, and a correlated SQL subquery to be written; a workflow can then be assembled using extract and SQL queries. The screen below, called the canvas or workspace, changes based on the selected tab as they have unique features and functions. During execution, the results are shown as a pop-up window (as shown in Figure 8), and execution trace is shown at the side panel that opens up on the right. When execution terminates or is paused, the canvas returns to Edit mode.

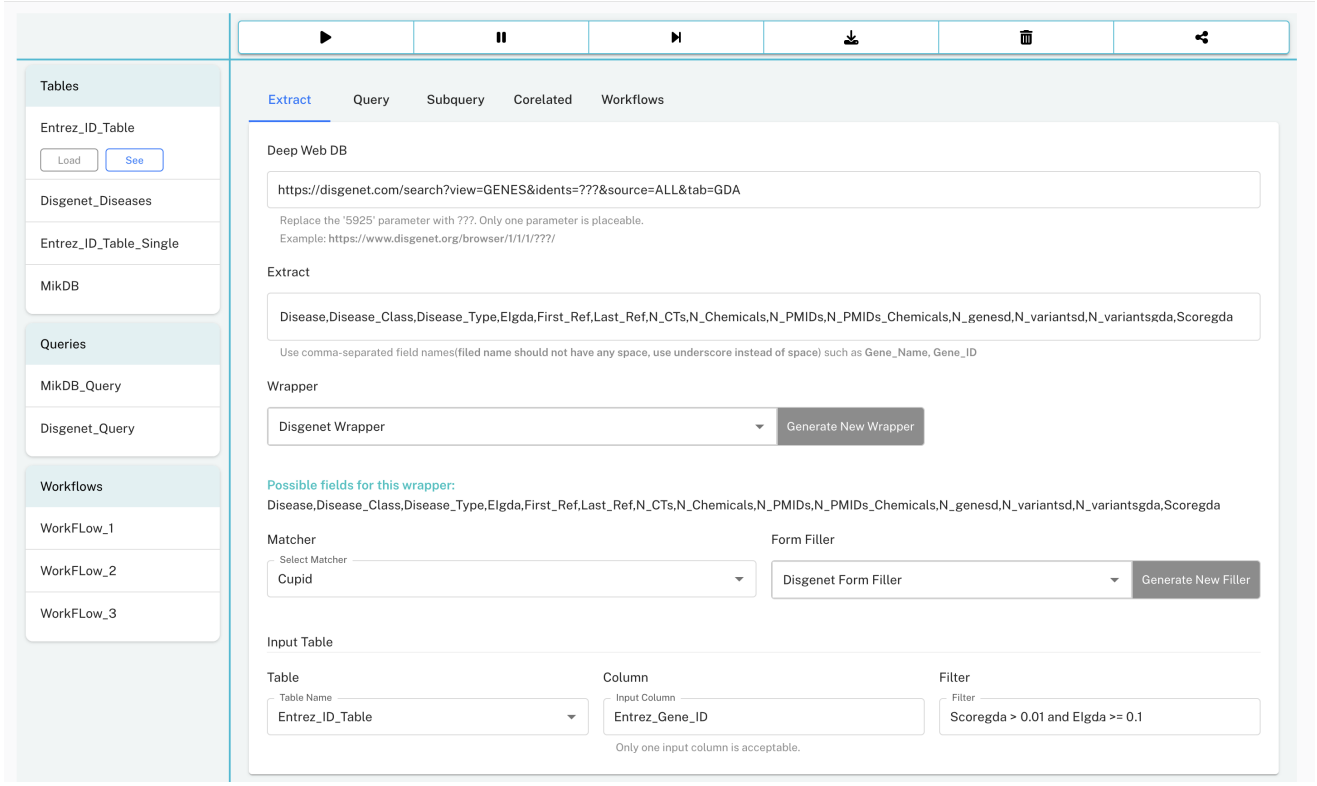


Figure 6. QBuild interface: Using this graphical interface, users can retrieve data from deep web databases without writing a single line of code. They can use the wrapper-constructed tool CroW and stored schema matchers from the system libraries to construct a Needle query to extract data, and store it for subsequent use.

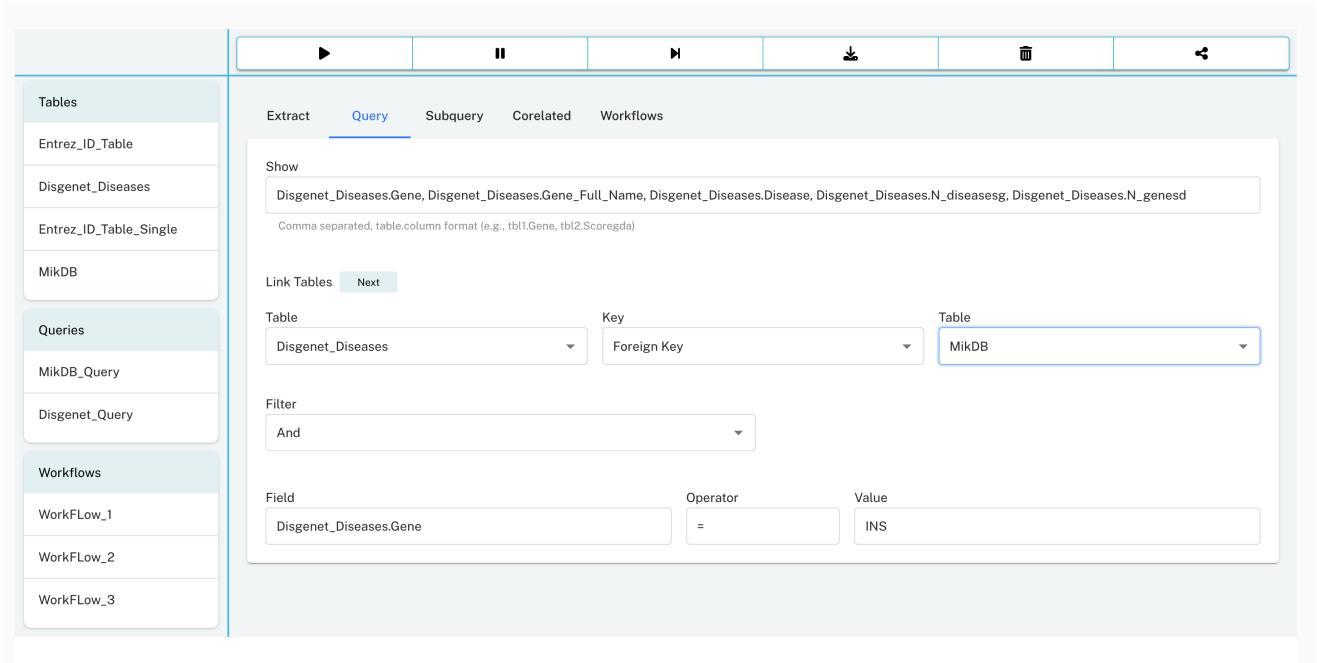
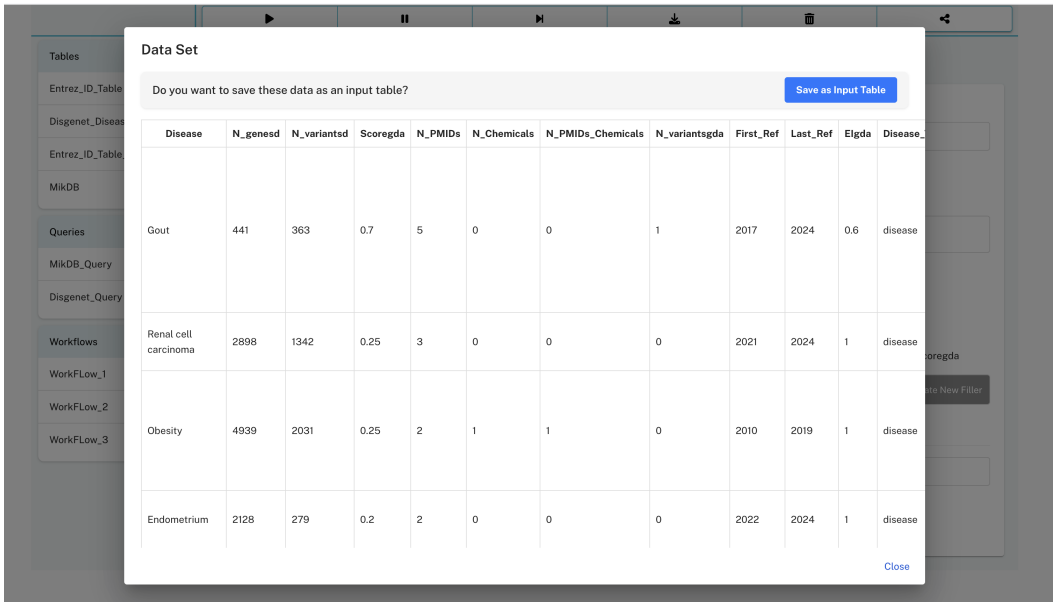


Figure 7. QBuild SQL query builder interface: This interface allows users to construct queries and analytics using stored tables in ways similar to traditional relational database QBE [30] interfaces.



Disease	N_genesd	N_variantsd	Scoregda	N_PMIDs	N_Chemicals	N_PMIDs_Chemicals	N_variantsgda	First_Ref	Last_Ref	Elgda	Disease_
Gout	441	363	0.7	5	0	0	1	2017	2024	0.6	disease
Renal cell carcinoma	2898	1342	0.25	3	0	0	0	2021	2024	1	disease
Obesity	4939	2031	0.25	2	1	1	0	2010	2019	1	disease
Endometrium	2128	279	0.2	2	0	0	0	2022	2024	1	disease

Figure 8. QBuild interface: The table shows the execution of the extract query constructed in Figure 6. The table can be stored in multiple formats—TXT, CSV, JSON, or as a relational table.

The side panel on the left has three selectors—Tables, Queries, and Workflows. These electors list stored tables in the SoDa database as well as previously constructed and stored queries and workflows. Clicking on a table name shows two options, *See* and *Load*. "See" shows the column names and a sample row of data to help understand the table. The "Load" option, on the other hand, pulls up the entire table and lets users inspect the table in full. However, unless the table is stored as a relational table, the queries (extract or SQL) are usually stored (not the computed tables) and when requested, the queries are executed live and the results are shown when Load option is chosen. This approach reduces the space needed to store materialized tables at the cost of time to compute it live.

Both Queries and Workflow entries load the corresponding definitions into the workspace or canvas in edit mode. Users are allowed to edit and execute queries or workflows. An edited version can be stored—as a new query or workflow, or replace the existing entries.

Deep Web Data Extraction

Figure 6 shows the Extract function in Edit mode. The fillable blocks in this interface resemble the extract statement options of BioFlow. The interface in Figure 6 shows several fillable entries that correspond to the extract statement in Figure 5. The Deep Web DB entry allows entering the database URL φ in the extract statement in Figure 5. The Extract entry enables listing the columns that users wish to retrieve from the deep web database. This is called the "user view" of the database. It is interesting to note that these column names in the user view need not match exactly with the column names in the deep web database at φ, called the "database view". The semantic match between the two views are resolved and a proper correspondence is established using the schema matcher selected by the user using the Schema Matcher entry just below. For this edition of SoDa, the schema matcher Cupid [16] is a default choice.

Every deep web data extraction requires a wrapper. A wrapper can be pre-built and stored using the CroW wrapper generation tool or can be constructed live using CroW following the "Generate New Wrapper" option. The drop-down menu of the Wrapper entry option allows a stored wrapper to be selected instead. The Form Filler entry helps an end-point constructor to be selected from the stored Needle description of a deep web

database. Usually, a form filler generates individual end-points from the entries in the input table *r* of the extract statement. Finally, the input table *r* can be selected from the drop-down menu found toward the bottom in this interface. The current edition of SoDa allows only one column as input parameter to the deep web database (Future editions will allow multiple parameters and remove this restriction).The filter entry allows a Boolean condition to be written to exclude unwanted parameters in the input table.

SQL Queries

While writing an extract query is straightforward, with only a few options, building an SQL query, on the other hand, could turn complex. Figure 7 shows construction of a simple join query using two stored tables—Disgenet_Diseases and MikDB, selected from the drop-down menus. A Boolean filter condition can be specified using the filter constructor below. The top Show option allows the columns that the users want to see to be listed. Note that, as discussed earlier, the tables selected from the drop-down menus are usually stored query specifications; the corresponding tables are computed live, and they are potentially extract queries.

Though a relatively large class of SQL queries are simple and involve only select–project–join (SPJ) operations over a set of relations, many require complex subqueries and correlated subqueries. The logic and construction process of such queries are pretty tedious and require substantial expertise in SQL. To simplify query construction using these SQL features, SoDa supports two interfaces—subquery and correlated subquery. Basically, these options allow queries constructed using the Query option to be linked in a systematic way using a graphical user interface. A discussion on subquery and correlated subquery is involved but is outside the scope of this paper. An independent and separate article on query construction using QBuild is being planned for publication soon, and we will not discuss the details in this article any further. Figure 9 shows the result of the SPJ query shown in Figure 7.

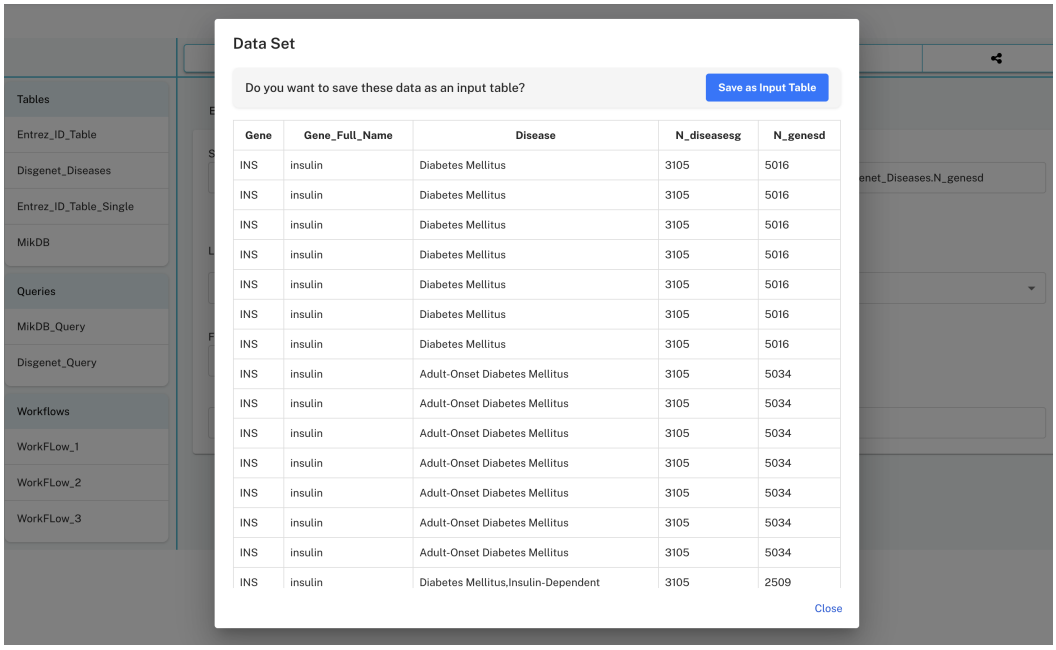


Figure 9. QBuild SQL query builder interface: table returned by SoDa upon execution of the query in Figure 7.

3.5. Workflow Query Construction

SoDa supports workflow orchestration using a cognate and separate system called VisFlow (For a detailed discussion on VisFlow, we invite readers to consult the first edition

of VisFlow article in ACM TCBB. We note that the new and second edition of VisFlow is currently under construction). While VisFlow has its own user base and maintains a resource registry, SoDa shares its resource registry (wrapper bank: all stored data and their Needle description) with VisFlow, and helps users construct workflow queries using the full power of VisFlow. In the case of SoDa, simple workflows may involve a successive set of BioFlow/SQL queries written using the QBuild application builder. Users build a query, likely using multiple resources (SoDa treats each of the resources as a table or one that returns a table) and save the results in a named view, then reusing the view in a later query. All workflows in SoDa have a named script that can be retrieved from the left side panel. In the event that VisFlow is used to construct a workflow, it can be stored in VisFlow, but cannot be executed in SoDa. The query processor of SoDa can be viewed as a fusion of the QBuild and VisFlow workflow systems, and together they form the *SoDaPro* system in which VisFlow is accessed through QBuild. The Workflow option in Figure 7 can be used to sequence named queries already built by dragging them from the left bar under the query tab. Or, users can hop over to VisFlow system to use the SoDa resources to construct more powerful workflows.

Since queries in SoDaPro potentially involve deep web databases, or online tools, that are accessed at query time, the process may fail for various reasons. The reasons for this include poor construction of the wrappers, failure of the schema matcher to resolve heterogeneity, resource site evolution or modification, logical errors, or simply network errors. All of the preceding causes are non-database operations that largely depend on software or tool applications with non-deterministic behaviors. In the event that they breakdown, debugging and fixing could be complicated simply because the clues could be hidden inside many moving parts of the query execution apparatus. One less frustrating way to fix such breakage or query errors is to seek help from more experienced or willing users in the community using crowd computing, as discussed in Section 3.7.

3.6. Workflow Meta-Language

While the BioFlow query language discussed in Section 3.4.1 serves as the main vehicle for query processing, its automatic construction requires additional engineering of meta-information about the internet resources. This meta-information is generated by the *ResCom* sub-system in Figure 1. It uses the FAIRBridge algorithm discussed in Section 3.4 to generate two specific meta-information—*process description* and *resource description* expressed in a language similar to *Needle*. We briefly illustrate the idea of how Needle helps generate workflows in BioFlow below.

3.6.1. Process Descriptions

The process description of a resource captures *how* the resource functions behaviorally, i.e., the structure of the table it returns, what it needs to compute a table, and how the resource can be accessed. Typically, a workflow consists of a series of resources in which information from one resource moves to the next, each computed using an extract statement in BioFlow. To aid in the automatic construction of such a workflow node, we introduced the create process statement along the lines of Needle's create webtable statements as follows.

The create process statement below describes the functionalities of the browser-based access to the DisGeNET database. It outlines how this database, when invoked, will return a gene-disease association (GDA). The four required components are the process identifier, the resource address and its features, and the input and output table schemes.

```
% process identifier
create process DisGeNETb
% access protocol
```

```
        at https://www.disgenet.org/
        access browser
        postfix /browser/1/1/0/$Genes
    % input table scheme
        accepts table (
            Genes EntrezID) (
    % output table scheme
        DisGeNETKey EntrezID primary key,
        Disease DisID,
        Type string,
        Disease_class string,
        Score_gda decimal (4,2),
        ...
    );
```

This statement helps to programmatically construct the endpoint (such as https://www.disgenet.org/browser/1/1/0/29974/ (accessed on 1 December 2024) for a gene, e.g., A1CF, using its Entrez ID 29974. We would like to note that the DisGeNET database recently changed their access protocol and now this end-point must be constructed as https://disgenet.com/search?view=GENES&idents=29974&source=ALL&tab=GDA (accessed on 1 December 2024)), which is needed to access the gene–disease association of a given gene from the DisGeNET database. The $ sign in the postfix clause provides possible substitutions in the accepts clause. On the other hand, the process description for API access by genes, is constructed as follows in a similar but distinct way. Depending on the access type we wish to adopt, an algorithmic construction of retrieval protocol is now possible from either of these statements.

```
create process DisGeNETa
    at https://www.disgenet.org/api/
    access API
    postfix /gda/gene/
    authorization keyK
    accepts table (
        Genes EntrezID) (
    DisGeNETKey EntrezID primary key,
    Disease DisID,
    Type string,
    Disease_class string,
    Score_gda decimal (4,2),
    ...
);
```

3.6.2. Resource Descriptions

In contrast to the process description, a resource description captures *which* computational tools are appropriate for a resource to function accurately, and how to determine the reliability of the resource so that its candidacy as a trusted information source can be reasoned. Thus, the intended usage of the resource description is to help automatically find, assemble, and construct executable workflows that are semantically the most likely implementation of a scientific inquiry. It does so by identifying the needed resources and providing tools to seamlessly collect the required data and query them in a pipeline. Recall that most of the resource descriptions in SoDa are contributed by a wide range of community members, some with limited domain expertise or credibility as curators.

For example, FAIRBridge constructs the following resource description for the DisGeNET site automatically from the corresponding process description.

```
% resource identifier
create resource DisGeNETb (
% resource narrative for machine consumption
    narrative "This browser access accepts an Entrez
      gene ID and returns its disease association",
% process contributors
    contributors {Alex, Abebi},
% applicable data integration tools
    meta:
      matcher {Cupid, OntoMatch},
      wrapper {FastWrap},
      mapping {Determination Process: Semantic Type},
% process testers and validators
      validators {Alex, Maya},
);
```

Resource descriptions such as *DisGeNETb* provide higher level details of process descriptions. Resource descriptions typically include operational guidance such as the types of schema matchers and wrappers that work best for the resource or exceptions that must be used for schema heterogeneities, e.g., the pair Determination Process: Semantic Type under mapping is one such mapping. The meta entries listing validators can be used to determine the credibility of the resource.

Using ResCom, we are now also able to generate process descriptions for the MiKDB [31] database and MapBase ID mapping system as follows.

```
create process MIKDB
    at http://mik.bicnirrh.res.in/mip.php
    access browser
    postfix /mip.php/
    accepts filter (
        Phenotype String) (
    Symbol GeneSymbol primary key,
    ChrLoc string,
    Disease string
);
```

and

```
create process MapBase
    at https://www.mapbase.smartdblab.org
    access browser (
    Symbol GeneSymbol primary key,
    GeneID string
);
```

3.6.3. Automated Workflow Construction

To compute the following interesting query

Q: *Find all genes implicated in obesity related male infertility using the gene list in the table crrews.*

Using a set of genes (gene symbols) in a table called *crrews* (with scheme *crrews(Symbol)*) and the MiKDB [31] and DisGeNET databases, we proceed as follows. We first collect a

set of genes (gene symbols) from MiKDB database for a specific phenotype (e.g., teratozoospermia). We then submit those genes (Entrez IDs) to the DisGeNET database to find the genes in the set that are also associated with obesity. The technical issue is that MiKDB returns the gene list in the form of gene symbols, and DisGeNET requires the gene list in the form of Entrez GeneIDs, necessitating an ID conversion step, which we carry out using MapBase. The entire workflow query generated by SoDa is as follows:

```
select Disease, Type, N_genes, Score_gda, EL_gda, N_PMIDs, First_Ref
from (extract Type, N_genes, Score_gda, EL_gda, N_PMIDs, First_Ref
        using matcher S-match wrapper Web-Prospector
        from https://www.disgenet.org/browser/1/1/0/
        submit (extract GeneID
                using matcher S-match wrapper Web-Prospector
                from https://www.mapbase.smartdblab.org
                submit (with mikgenes as (extract Symbol, Phenotyoe
                                using matcher S-match wrapper Web-Prospector
                                from http://mik.bicnirrh.res.in/mip.php
                                )
                        select crrews.Symbol
                        from crrews natural join mikgenes
                        where Phenotype = "teratozoospermia"
                        )
                )
        )
where Disease = 'obesity' and Score_gda > 0.01;
```

The execution of this query in SoDa will return the partial table shown in Figure 10. In the above statement, S-Match [32] is a schema matcher, and Web-Prospector [33] is a wrapper. An interesting observation is that SQL's select statements and BioFlow's extract statements both uniformly accept each other where a table is expected, and thus allows nesting, as in this query. In the event that the query needs to be broken down into smaller queries and then strung together, SQL's create view construct can be used in the usual way.

Disease	Type	N_genes	Score_gda	EL_gda	N_PMIDs	First_Ref
obesity	disease	2821	0.040	1.000	4	1996
obesity	disease	2821	0.200	1.000	1	1996
obesity	disease	2821	0.030	0.667	3	2012
obesity	disease	2821	0.110	1.000	1	2015

Figure 10. Results of query *Q*.

3.7. Crowd-Enabled Curation of Workflows When Pipelines Break

Platforms such as StackOverflow (https://stackoverflow.com/ (accessed on 1 December 2024)) or GitHub (https://github.com/ (accessed on 1 December 2024)) aid developers with code development projects. LLM-based systems, such as CoPilot (https://copilot.microsoft.com/ (accessed on 1 December 2024)) and Amazon Q (https://aws.amazon.com/q/ (accessed on 1 December 2024)), are offering a more intelligent and faster alternative to the traditional code development and debugging practices. Unfortunately, when the applications, languages, and knowledge needed to code and debug are not in the mainstream, such as BioFlow or Needle, such platforms need to be trained with enough data to be useful. In particular, even a trained LLM will likely need access actual resources to be able to offer debugging or even coding assistance [34,35].

SoDa's crowd curation system, called *QCurator*, takes a more active support approach for error tracing and bug fixing using a crowd computing approach. SoDa users have the option to push a code segment or query to a community discussion board for help. The discussion board has a special notification system that alerts relevant users of an available help request as a ticket. Users may sign up as a participant crowd in the research area or topic of choice. They are able to participate in the discussion by fixing problems and errors, executing code fragments to see if they work, or opting out, and until they explicitly exit the ticket, it stays active in their notification queue. The ticket is resolved until the user who initiated the ticket exits from it, or all active participants do so. Note that the ticket is also available for all members of the community in the general discussion board. However, the general discussion board notification goes off only when the initiating user exits the ticket. Finally, the discussion with and solutions of special users are also visible in the general discussion board.

The look and feel of the QCurator discussion board is not much different than the Stack Overflow or GitHub discussion boards, but the difference is more critical when it comes to the notification and debugging approaches. Notifications are owned by the users to whom they are specifically sent, and the notification to the general discussion board is owned by the ticket-initiating user. Therefore, unless all of the owners exit the ticket, it stays active in each of the owners' dashboards. Finally, every user in the community has access to the discussion board and is able to debug, modify, and execute the code fragments on the forum directly from their dashboards.

4. Discussion

Technological limitations play a significant role in the design and functioning of SoDa. Though possible in principle, SoDa's objective of maintaining a no-coding environment makes it difficult to fully achieve some of its goals. We highlight two major issues for which SoDa has limited capability to address.

4.1. Complex Wrapper Generation

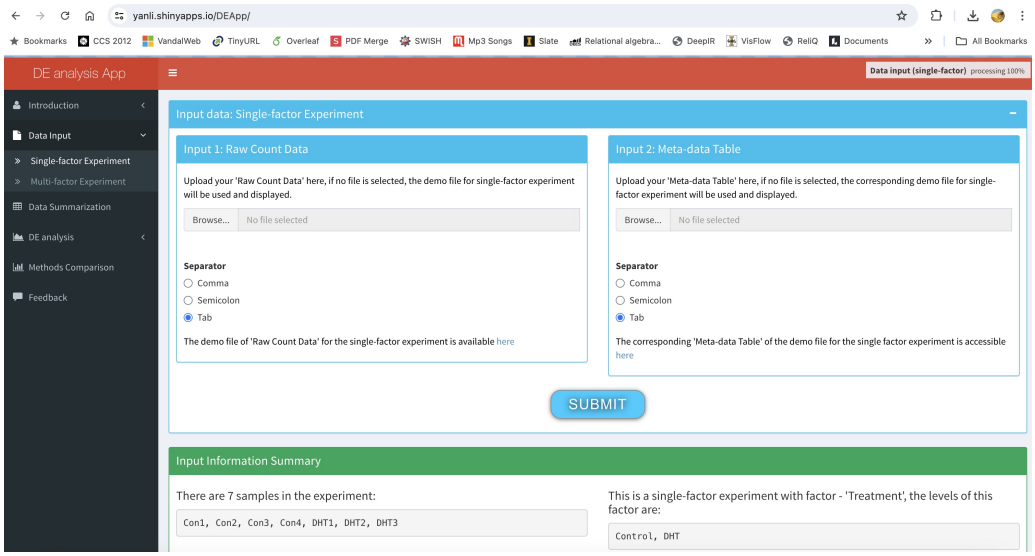
All internet resource access in SoDa requires an appropriate wrapper from its wrapper bank. As discussed in Section 3.3, SoDa supports the wrapper generation tool, CroW. While complex and innovative wrappers can be constructed using it, CroW currently cannot design wrappers for multi-page forms for a database that uses JavaScript for form design. For example, the online differential expression analyzer DEApp (https://yanli.shinyapps.io/DEApp/ (accessed on 1 December 2024)) [36] could be an excellent alternative to expression analyzers, DESeq2, edgeR, and Limma, available in libraries such as Python or R However, these libraries are difficult to use in a no-coding environment; even though they are self-contained, relatively easy to identify and codify, or written by CoPilot-type coding systems, they still need significant programming experience to incorporate in procedures that can be run on a system, such as SoDa.

The differential expression analysis in DEApp progresses in four steps—data input (Figure 11a); low-expression removal (Figure 11b); DE analysis using libraries such as DESeq2, edgeR, and Limma (Figure 12a); and finally DE analysis comparison (Figure 12b). Though a multi-step process and somewhat complex, source complication is not how this functions; it is how the interface is designed. In Figure 11b, the URL address at the top shows that it did not change from Figure 11a, because it was designed using JavaScript, which internally changes the form submission parameters that it collects from the user. Though CroW is able to design multi-page hopping wrappers, it can only do so when the forms are designed using traditional AJAX/JavaScript free technologies. Thus, for now,

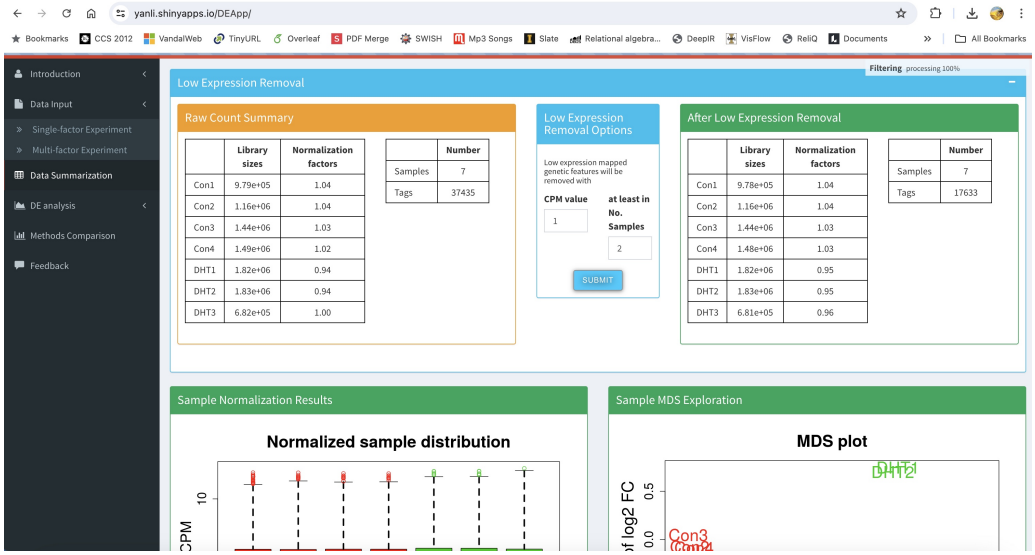
SoDa only supports the inclusion of resources that are either one hop, multi-page-designed with AJAX or JavaScript free technologies, or are API-based.

4.2. Complex Query Construction

The application design system discussed in Section 3.4 can be used to design fairly complex BioFlow extract and SQL select queries. The extract statements are regarded as resource access statements, and thus are not true querying instruments. Once the resources are accessed and data extracted using extract, they can be queried with the full power of SQL's select constructs. Typical SQL queries involve sub- and correlated queries, as well as aggregate functions. While designing query builder GUIs for SQL's aggregate functions is not too difficult, seamlessly incorporating all these features into the query builder is difficult, especially when the users are assumed "naive" and have no true experience in direct complex SQL query writing.

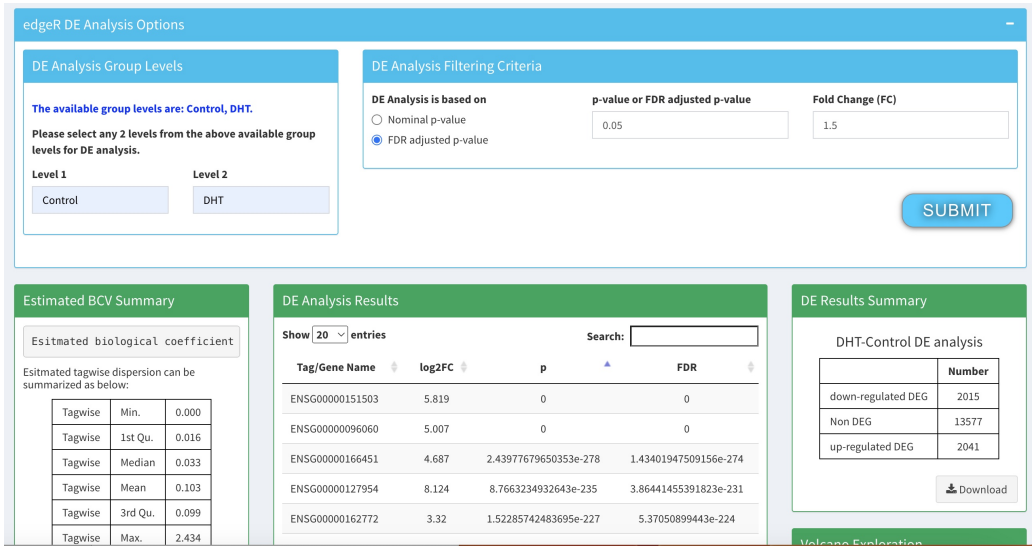


(a) Data input step.

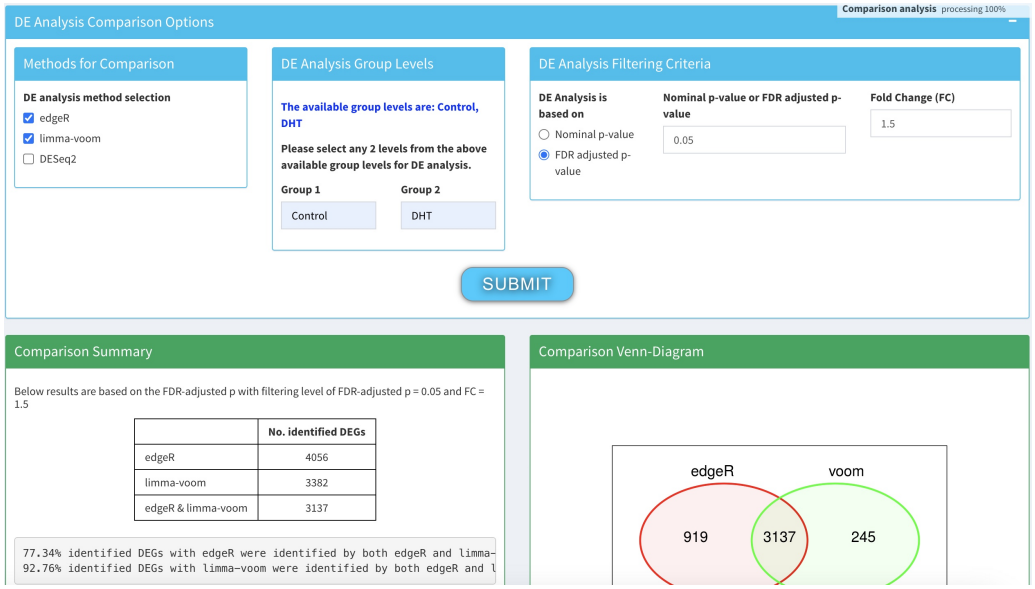


(b) Filtering step.

Figure 11. Part 1: Stepwise differential expression analysis using DEApp: (**a**) shows the data input step of the online differential gene expression analysis tool DEApp [36]. Once the input data and their format are chosen, a filtering operation can be carried out at step 2, as shown in (**b**).



(a) DE analysis step.



(b) Inter-analysis comparison step.

Figure 12. Part 2: Stepwise differential expression analysis using DEApp: DEApp allows computing differential gene expression analysis using multiple algorithms and different filter conditions that users are able to select using the interface in (**a**). Finally, a comparison of the expression analysis using multiple methods can be compared, as shown in (**b**). The important observation is that all of these four steps within DEApp are in sequence and have to be followed in succession, creating a dependence or order.

Designing no-coding SQL interfaces has a long and rich research history [37,38]. While there are numerous SQL query builders, both academic and commercial, most assume that users are familiar with complex SQL syntax and semantics and aim to aid and expedite query building. In particular, they are difficult to use when the query interleaves aggregate functions and correlated subqueries to a too deep level [39]. Designing a query builder for the general public, or biologists, for arbitrary database and intended arbitrary query has its challenges. The recent emergence of LLM-based Text2SQL efforts [40] probably performs better than most no-coding interfaces. Our interface too has limitations, but we are currently exploring how to enhance our interface with LLM-powered SQL query generation by fine tuning an LLM to BioFlow and Needle.

5. Conclusions

The main objective of SoDa is to support biologists in gathering data from online resources in a searchable repository so that other biologists can also readily use these data. The search for resources is truly flexible using text analysis of resource needs. As opposed to fully automatic ProAb, SoDa is manual with a human-in-the-loop principle, and offers relatively much lower failure possibilities. The downside is that users must know exactly what they want to compute but not necessarily how to do it. SoDa supports a declarative way of computing internet workflow queries involving heterogeneous resources. The current edition of SoDa is experimental and does not support user data archiving for guest users. Future editions of SoDa will support user accounts and allow data archival options along with advanced query building options, likely using LLMs. Finally, as discussed in a related article on DeepIR [41], the wrapper generation and their usage in data scraping has its downsides. Not all deep web sites allow scraping, and designing wrappers to scrape data is not efficient, even when building wrappers is not too difficult. However, these problems are not an issue when deep web databases support data access through APIs, which SoDa supports.

6. Software Availability

The first edition of SoDa is available for public use at http://dblab.nkn.uidaho.edu/soda/ (accessed on 1 December 2024). Also, a preliminary version of this article appeared in iiWAS 2024 proceedings [42]. While parts of the system are still under development, and are in flux, the basic functions are active. The link with the sister workflow construction system, VisFlow, and the wrapper generation system, Crow, are under construction and will be made available soon. The authors also wish to acknowledge the contributions of Syed N Sakib, who helped to develop the resource identification component ResCom (see Figures 1 and 2) and the crowd curation system QCurator, as discussed in Sections 3.3 and 3.7, respectively.

Author Contributions: Conceptualization, H.M.J.; Methodology, H.M.J.; Software, K.N.; Formal analysis, H.M.J.; Investigation, H.M.J.; Writing—original draft, H.M.J.; Writing—review & editing, K.N.; Supervision, H.M.J.; Funding acquisition, H.M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by a National Institutes of Health IDeA grant P20GM103408, a National Science Foundation CSSI grant OAC 2410668, and a US Department of Energy grant DE-0011014.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interests.

References

1. Liu, J.; Lin, L.; Cai, Z.; Wang, J.; Kim, H. Deep web data extraction based on visual information processing. *J. Ambient Intell. Humaniz. Comput.* **2024**, *15*, 1481–1491. [CrossRef]
2. Liu, D.; Ma, L.; Liu, X. Research on Adaptive Wrapper in Deep Web Data Extraction. In Proceedings of the IOV 2015, Chengdu, China, 19–21 December 2015; Volume 9502, pp. 409–423.
3. Bülthoff, F.; Maleshkova, M. RESTful or RESTless—Current State of Today's Top Web APIs. In Proceedings of the ESWC 2014, Anissaras, Crete, Greece, 25–29 May 2014; Volume 8798, pp. 64–74.
4. Rodrigues, T.; Benevenuto, F.; Cha, M.; Gummadi, P.K.; Almeida, V.A.F. On word-of-mouth based discovery of the web. In Proceedings of the ACM SIGCOMM IMC '11, Berlin, Germany, 2 November 2011; pp. 381–396.

5. Li, Y.; Wang, Y.; Tian, E. A New Architecture of an Intelligent Agent-Based Crawler for Domain-Specific Deep Web Databases. In Proceedings of the WI 2012, Macau, China, 4–7 December 2012; pp. 656–663.
6. Burks, C. Molecular Biology Database List. *Nucleic Acids Res.* **1999**, *27*, 1–9. [CrossRef]
7. Bader, G.D.; Cary, M.P.; Sander, C. Pathguide: A Pathway Resource List. *Nucleic Acids Res.* **2006**, *34*, 504–506. [CrossRef] [PubMed]
8. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 1–9. [CrossRef] [PubMed]
9. Pattyn, F.; Wulbrecht, B.; Knecht, K.; Constandt, H. Assessment of FAIRness of Open Data Sources in Life Sciences. In Proceedings of the (SWAT4LS 2017), Rome, Italy, 4–7 December 2017; Volume 2042.
10. Gaignard, A.; Rosnet, T.; De Lamotte, F.; Lefort, V.; Devignes, M.D. FAIR-Checker: Supporting digital resource findability and reuse with Knowledge Graphs and Semantic Web standards. *J. Biomed. Semant.* **2023**, *14*, 16–20. [CrossRef]
11. Soiland-Reyes, S.; Goble, C.; Groth, P. Evaluating FAIR Digital Object and Linked Data as distributed object systems. *PeerJ Comput. Sci.* **2024**, *10*, e1781. [CrossRef] [PubMed]
12. Wandl-Vogt, E.; Ostojic, D.; Piringer, B.; Rainer, H.; Zsaytseva, K. Designing Collaborative Ecosystems and community organization: Introducing the multidisciplinary portal on "Biodiversity and Linguistic Diversity: A Collaborative Knowledge Discovery Environment". In Proceedings of the DH 2017, Montréal, QC, Canada, 8–11 August 2017.
13. Antonazzo, G.; Urbano, J.; Marygold, S.J.; Millburn, G.H.; Brown, N.H. Building a pipeline to solicit expert knowledge from the community to aid gene summary curation. *Database J. Biol. Databases Curation* **2020**, *2020*, baz152. [CrossRef] [PubMed]
14. Gendarmi, D.; Abbattista, F.; Lanubile, F. Fostering Knowledge Evolution through Community-based Participation. In Proceedings of the (CKC 2007)@(WWW2007), Banff, AB, Canada, 8 May 2007; Volume 273.
15. Dalvi, N.N.; Kumar, R.; Soliman, M.A. Automatic Wrappers for Large Scale Web Extraction. *Proc. VLDB Endow.* **2011**, *4*, 219–230. [CrossRef]
16. Madhavan, J.; Bernstein, P.A.; Rahm, E. Generic Schema Matching with Cupid. In Proceedings of the VLDB 2001, 27th International Conference on Very Large Data Bases, Rome, Italy, 11–14 September 2001; pp. 49–58.
17. Jalili, V.; Afgan, E.; Gu, Q.; Clements, D.; Blankenberg, D.J.; Goecks, J.; Taylor, J.; Nekrutenko, A. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update. *Nucleic Acids Res.* **2020**, *48*, W395–W402. [CrossRef] [PubMed]
18. Damkliang, K.; Tandayya, P. Middleware for running and debugging Taverna workflows utilising RESTful web services. *Int. J. Simul. Process. Model.* **2020**, *15*, 546–561. [CrossRef]
19. Altintas, I.; Berkley, C.; Jaeger, E.; Jones, M.B.; Ludascher, B.; Mock, S. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), Santorini Island, Greece, 21–23 June 2004; pp. 423–424. [CrossRef]
20. Pinero, J.; Ramirez-Anguita, J.M.; Sauch-Pitarch, J.; Ronzano, F.; Centeno, E.; Sanz, F.; Furlong, L.I. The DisGeNET knowledge platform for disease genomics: 2019 update. *Nucleic Acids Res.* **2019**, *48*, D845–D855. [CrossRef]
21. Bogatu, A.; Paton, N.W.; Douthwaite, M.; Freitas, A. Voyager: Data Discovery and Integration for Onboarding in Data Science. In Proceedings of the EDBT 2022, Edinburgh, UK, 29 March–1 April 2022; pp. 2:537–2:548.
22. Diepenbroek, M. The Application of Semantic Resources and Technologies for the Discovery and Integration of Geo- and Biosciences Data (invited paper). In Proceedings of the The Bolzano Summer of Knowledge @ FOIS 2021, and ICBO 2021, Bolzano, Italy, 11–18 September 2021; Volume 2969.
23. Ouzzani, M.; Tang, N.; Fernandez, R.C. Data civilizer: End-to-end support for data discovery, integration, and cleaning. In *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*; Brodie, M.L., Ed.; Morgan & Claypool: San Rafael, CA, USA, 2019; Volume 22, pp. 291–300.
24. Pang, C.; Kelpin, F.D.L.; van Enckevort, D.; Eklund, N.; Silander, K.; Hendriksen, D.; de Haan, M.; Jetten, J.; de Boer, T.; Charbon, B.; et al. BiobankUniverse: Automatic matchmaking between datasets for biobank data discovery and integration. *Bioinformatics* **2017**, *33*, 3627–3634. [CrossRef]
25. Majeed, R.W.; Stohr, M.R.; Ruppert, C.; Gunther, A. Data Discovery for Integration of Heterogeneous Medical Datasets in the German Center for Lung Research (DZL). In Proceedings of the (GMDS e.V.) 2018, Osnabruck, Germany, 2–6 September 2018; Volume 253, pp. 65–69.
26. Burl, R.B.; Clough, S.; Sendler, E.; Estill, M.; Krawetz, S.A. Sperm RNA elements as markers of health. *Syst. Biol. Reprod. Med.* **2018**, *64*, 25–38. [CrossRef]
27. Clough, E.; Barrett, T.; Wilhite, S.E.; Ledoux, P.; Evangelista, C.; Kim, I.F.; Tomashevsky, M.; Marshall, K.A.; Phillippy, K.H.; Sherman, P.M.; et al. NCBI GEO: Archive for gene expression and epigenomics data sets: 23-year update. *Nucleic Acids Res.* **2023**, *52*, D138–D144. [CrossRef] [PubMed]
28. Katz, K.; Shutov, O.; Lapoint, R.; Kimelman, M.; Brister, J.R.; O'Sullivan, C. The Sequence Read Archive: A decade more of explosive growth. *Nucleic Acids Res.* **2021**, *50*, D387–D390. [CrossRef] [PubMed]

29. Jamil, H.; Naha, K. Mapping Strategies for Declarative Queries over Online Heterogeneous Biological Databases for Intelligent Responses. In Proceedings of the SAC 2023, Tallinn, Estonia, 27–31 March 2023.
30. Yen, M.Y.; Scamell, R.W. A Human Factors Experimental Comparison of SQL and QBE. *IEEE Trans. Softw. Eng.* **1993**, *19*, 390–409. [CrossRef]
31. Joseph, S.; Mahale, S.D. Male Infertility Knowledgebase: Decoding the genetic and disease landscape. *Database* **2021**, *2021*, baab049. [CrossRef]
32. Giunchiglia, F.; Autayeu, A.; Pane, J. S-Match: An open source framework for matching lightweight ontologies. *Semant. Web* **2012**, *3*, 307–317. [CrossRef]
33. Mir, S.; Staab, S.; Rojas, I. Web-Prospector—An Automatic, Site-Wide Wrapper Induction Approach for Scientific Deep-Web Databases. In *BTW*; Gesellschaft für Informatik e.V.: Bonn, Germay, 2009; pp. 87–106.
34. Sakib, F.A.; Khan, S.H.; Karim, A.H.M.R. Extending the Frontier of ChatGPT: Code Generation and Debugging. *CoRR* **2023**, abs/2307.08260. [CrossRef]
35. Saben, C.; Chandrasekar, P. Enabling BLV Developers with LLM-driven Code Debugging. *CoRR* **2024**, abs/2401.16654.
36. Li, Y.; Andrade, J. DEApp: An interactive web interface for differential expression analysis of next generation sequence data. *Source Code Biol. Med.* **2017**, *12*, 2. [CrossRef] [PubMed]
37. Miedema, D.; Fletcher, G. SQLVis: Visual Query Representations for Supporting SQL Learners. In Proceedings of the IEEE VL/HCC 2021, St. Louis, MO, USA, 10–13 October 2021; pp. 1–9.
38. Murakawa, T.; Nakagawa, M. Graphical Expression of SQL Statements Using Clamshell Diagram. *IEICE Trans. Inf. Syst.* **2010**, *93-D*, 713–720. [CrossRef]
39. Taipalus, T. The effects of database complexity on SQL query formulation. *J. Syst. Softw.* **2020**, *165*, 110576. [CrossRef]
40. Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; et al. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In Proceedings of the NeurIPS 2023, New Orleans, LA, USA, 10–16 December 2023.
41. Naha, K.; Jamil, H. A Declarative Query Language Enabled Autonomous Deep Web Search Engine. In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, 8–12 April 2024; pp. 305–312.
42. Jamil, H.M. Supporting Data Foragers in Scientific Computing Community Ecosystems for Life Sciences. In Proceedings of the Information Integration and Web Intelligence—26th International Conference, iiWAS 2024, Bratislava, Slovakia, 2–4 December 2024; Proceedings, Part II; Volume 15343, pp. 118–123. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.