SQLShare: Scientific Workflow via Relational View Sharing

Bill Howe
University of Washington
billhowe@cs.washington.edu

Francois Ribalet
University of Washington
ribalet@u.washington.edu

Daniel Halperin
University of Washington
dhalperi@cs.washington.edu

Sagar Chitnis
University of Washington
sagarc@cs.washington.edu

E. Virginia Armbrust
University of Washington
armbrust@u.washington.edu

Abstract

We consider a case study in using a web-based query-as-a-service platform as an alternative to script-based scientific workflows. The context is a project in observational biological oceanography to share and process data from a ship-based continuous profiler of microbial populations called SeaFlow. The representative tasks involve aggregating and cleaning SeaFlow measurements, integrating the cleaned dataset with other data sources, and sharing the results with colleagues for ad hoc interactive analysis. We find that expressing these tasks as hierarchies of queries instead of chains of scripts can reduce development effort while offering transparency, provenance, versioning, and automatic scalability beyond main memory. In order to realize these benefits, we relax assumptions made by conventional databases about the availability of up-front schemas and eliminate initial setup and configuration steps. Instead, we adopt SQLShare, a web-based application that emphasizes a simple upload-query-share protocol over conventional database design, and emphasizes ad hoc interactive query over general-purpose programming. In this paper, we describe the SQLShare system and our experience deploying it in this context.

Keywords:
H.2.8.n Scientific databases
H.2.8 Database Applications
H.2.4.p Workflow management

1 Introduction

Data management has replaced data acquisition as the bottleneck to scientific discovery. Consider an example: In the last decade, several high-frequency *flow cytometers* have been developed to study microorganism composition at very fine spatial and temporal scales, collecting several hundreds of samples per day for several weeks. A flow cytometer passes a fluid through a fine capillary, identifying particles in the fluid by analyzing the absorption and refraction patterns of various wavelengths of light. One of the largest flow cytometry datasets publicly available to marine biologists is produced by a new generation of flow cytometer created at the University of Washington, called SeaFlow, that continuously measures phytoplankton composition and abundance at a rate of

thousands cells per second [13]. The instrument generates the equivalent of 6700 samples, representing a dataset of 35-135 GB, after a typical 2-week long oceanographic cruise. To date, the dataset represents over 200 billion cells collected in different seasons and different environments, but only 10% of the data have been analyzed so far due to challenges in management and analysis. Specifically:

- Scripts for data processing and ad hoc analysis, typically written in R, must be manually re-executed, sometimes by multiple collaborators, when new data arrives or the logic changes.
- These scripts (written in R and comparable languages) assume all data can be loaded in main memory; redesigning one's algorithms to exploit secondary storage or multiple computers in parallel is often beyond the capabilities of domain researchers.
- New versions of scripts and/or the results they produce must be re-distributed among collaborators, adding opportunity for confusion. Older, deprecated versions of scripts and data cannot be "recalled" by their author, and may (silently) continue to be used to make wrong decisions.
- Given a dataset generated by some variant of these scripts, there is no unambiguous way to determine its provenance — the exact series of steps that produced it.
- Files must be read and written by each step in the pipeline, incurring a dependence on brittle file formats and requiring explicit paths that can complicate portability.

Some of these problems helped motivate the study of scientific workflow management systems (SWMS), which aim to raise the level of abstraction for expressing and executing science data processing pipelines by offering features for managing provenance, reusing components, and executing pipelines on heterogeneous platforms. These systems have enjoyed success in large-scale projects involving substantial investment in IT infrastructure [5], where the cost of workflow development can be amortized over many repeated executions. However, despite years of productive research, these systems have not enjoyed widespread adoption [4]. In particular, we find that workflow systems are difficult to deploy in the *long tail* of science [6, 10]: the smaller labs and individual researchers with relatively limited IT budgets but who operate at the "forward edge" of science characterized by exploding data acquisition rates, a short publication cycle, dynamic multi-institutional collaborations, and frequent use of heterogeneous shared data sources. In these scenarios, the initial cost to develop "pipelines" for data processing is not only prohibitively high due to limited access to in-house IT infrastructure and expertise, but there is also less opportunity to amortize this cost over time due to rapidly changing requirements. Instead, we seek a system that emphasizes ad hoc interactive analysis, and provides workflow-like features automatically as a side effect.

Relational databases support ad hoc interactive query and can comfortably handle the scale of the data, but they have not enjoyed significant uptake in these scenarios either. It is tempting to ascribe this underuse to a mismatch between scientific data and the models and languages of commercial database systems [1]. Our experience (which we describe throughout this paper) is that standard relational models and languages can effectively manage and manipulate a significant range of scientific datasets. We find that the barriers to adoption are associated with the assumption that before these models and languages can be used, one must engineer a database schema and populate it with clean, integrated, and well-structured data. Developing a definitive database schema for a project at the frontier of research, where knowledge is undergoing sometimes-daily revision, is a challenge even for database experts. Moreover, concerns about control of unpublished research data, at tension with the need for unfettered collaboration and sharing, complicate centralization and organization in a database. Researchers want fine-grained control over their data, and scripts and files give them this measure of control where databases, by default, do not.

Despite these weaknesses, the core technology of relational databases remains attractive for science. The datasets we encounter are naturally modeled as relations (csv files, "rectangular" spreadsheets, etc.). These

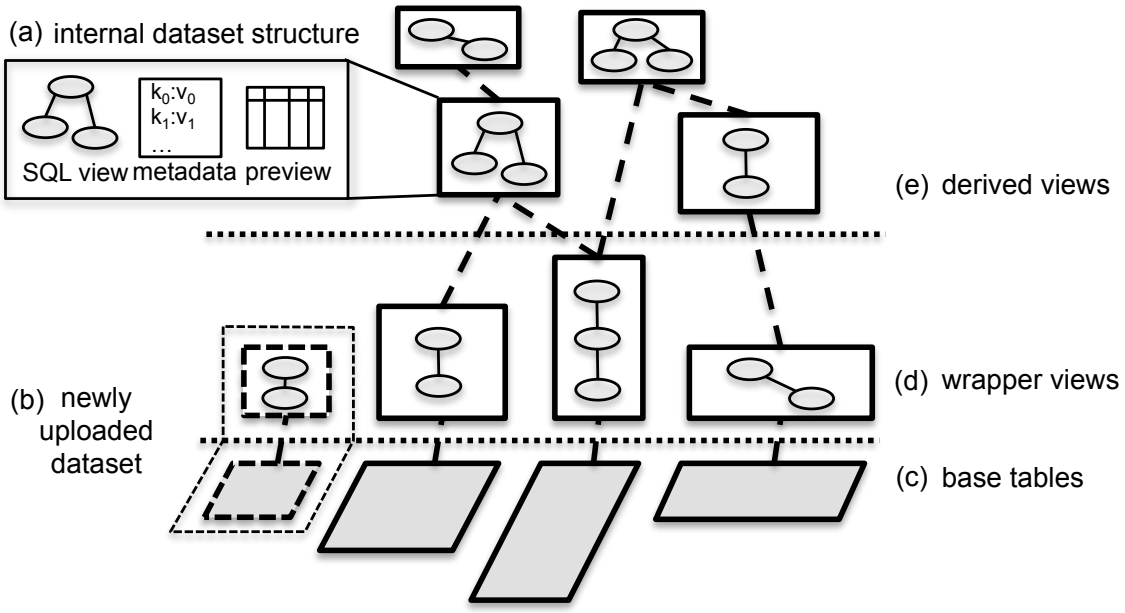

Figure 1: The SQLShare data model. (a) The internal structure of a dataset, consisting of a relational view, attached metadata, and a cached preview of the results. (b) A newly uploaded datasets creates both a physical base table and an initial (trivial) wrapper view. (c) The physical data are stored in base tables, but are never modified directly. (d) Wrapper views are indistinguishable from other SQLShare datasets, except that they reference physical base tables. (e) Derived datasets can be created by any user. Permissions are managed in the underlying database.

relations are frequently too large to be manipulated in main memory, but nearly all databases are equipped with out-of-core algorithms and can automatically select among them using cost-based algebraic optimization. Of course, performance is irrelevant if the programming interfaces and languages cannot be used to implement that kind of tasks researchers need to perform. But we and others have found that many of these analysis tasks can be easily expressed as SQL queries [3, 8]. For example, figure 3 illustrates binning a timeseries on 3-minute intervals for integration with other data streams, a task that was initially assumed to be inappropriate for implementation in the database.

Through a partnership between the University of Washington eScience Institute and the Armbrust Lab at the University of Washington, we are developing a new "delivery vector" for relational database technology called SQLShare, and studying how it can be used to satisfy both scientific workflow requirements and ad hoc interactive analysis. SQLShare is a web-based query-as-a-service system that eliminates the initial set up costs associated with conventional database projects and instead emphasizes a simple Upload-Query-Share protocol: users upload their table-structured files through a browser and can immediately query them using SQL — no schema design, no preprocessing, no database administrators. SQLShare users derive new datasets by writing and saving SQL queries. Each derived dataset is managed as a view in the underlying database: a named, permanently stored query. Each dataset is also equipped with descriptive metadata. Everything in SQLShare is accomplished by writing and sharing views: Users can clean data, assess quality, standardize units, integrate data from multiple sources, attach metadata, protect sensitive data, and publish results. The resulting network of related views is superficially similar to the "boxes-and-arrows" abstractions found in scientific workflow systems (Figure 1), but there are some important advantages that address the problems we have described:

- No special software need be installed. All data and processing logic are fully accessible through a web browser. Views can be edited directly in the browser as well.
- Data never needs to be re-processed explicitly. For most queries, the underlying database system can

efficiently and scalably re-generate results on demand whenever a view is accessed. For expensive queries, the results can be materialized and cached on disk automatically or on-demand.

- The only size limit for datasets is the storage capacity of the database itself. Unlike script-based processing, no task can ever crash due to "out-of-memory" errors (or thrash in virtual memory).
- All collaborators access the same version of each view, stored centrally in the database. Any change to a view is immediately and automatically reflected in all future results.There can be no confusion over which version is current or correct.
- SQLShare stores source code and data, coupled together, in a single web interface. Views and the input data they require cannot be separated, eliminating errors resulting from applying a script to the "wrong" data. In contrast, physical files can only be associated with the scripts that generated them through elaborate metadata schemes managed by provenance systems; there is no physical coupling.
- Ownership of a view affords precise control over access permissions. Views can be private, public, or shared explicitly with specific collaborators in a centralized manner.
- By storing all data in a centralized location, there is only one global namespace to manage. It is impossible to have two datasets with the same name but different content.
- A complex network of views can be inspected visually (Figure 4), providing an intuitive form of provenance.
- The SQL definition of each view provides a concise and declarative expression of the logic, providing independence from data representation issues [2].
- Ad hoc interactive exploration is supported directly. With sufficient permissions, any view can be copied, modified slightly, and executed, no matter who owns the original.

Our initial experience with SQLShare has led to some surprising findings: We can now reject the conventional wisdom that "scientists won't write SQL." Our experience is that they can and they will. We find that even non-programmers are able to create and share SQL views for a variety of tasks, including quality control, integration, basic analysis, and access control. We also find that researchers are using SQL not just to extract data, but also to exchange ideas and collaborate — that simply sharing SQL queries allows researchers to communicate in terms of science questions instead of computer programs.

In this paper, we consider a SQLShare case study in environmental flow cytometry, arguing that the platform offers significant benefits and lower overall costs than the alternatives: ad hoc scripts and files, scientific workflow systems, and conventional database applications.

2 Case Study: A Census of Microbial Ocean Populations with SeaFlow

The most abundant organisms in the world's oceans are microbes less than 20 μm in size. Together, these organisms drive biogeochemical cycling of major elements, with almost 50% of organic carbon production on Earth generated and recycled by these small microbes. Satellite images of chlorophyll-A concentrations in the surface ocean have transformed our view of the distribution of the photosynthetic microbes (the phytoplankton), although these data provide little information about specific groups of phytoplankton and finer-scale details are lost through the data aggregation needed to counteract cloud cover. Our ability to develop high-resolution maps of the distribution and abundances of these organisms is critical to understand ecosystem functions and the sensitivity of these processes to changes in the environment. Progress in this area has been limited in large part by a lack of observational tools for microbes.

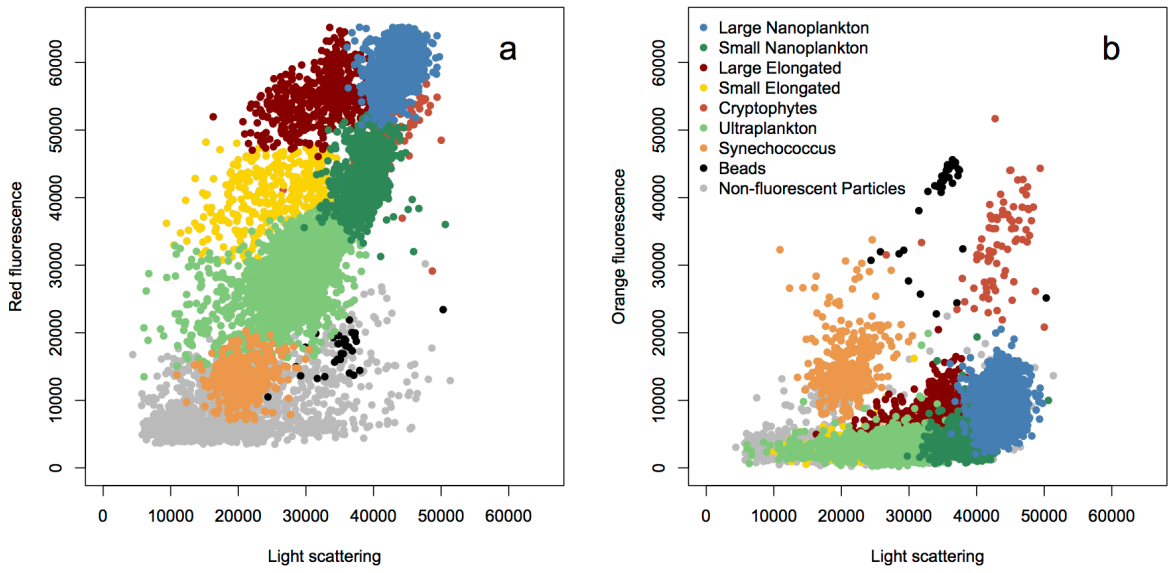

Figure 2: Example of flow cytometric signatures of phytoplankton populations in the North Pacific Ocean. (a) Red fluorescence from chlorophyll versus forward light scattering (a proxy of cell size) uniquely identified five distinct phytoplankton populations: large and small elongated phytoplankton, large and small nanoplankton, and ultraplankton. (b) Orange fluorescence from phycoerythrin versus forward light scattering was used to identify the cyanobacteria Synechococcus, cryptophytes and fluorescent microspheres (beads) added as an internal standard.

To address these issues, the SeaFlow project at the University of Washington has developed an entirely new flow cytometer [13]. The instrument taps into the seawater intake on ships and continuously (at a rate of up to 24,000 cells per second) measures abundance, cell size and fluorescence signals from pigments within individual phytoplankton cells (0.5-20 μm), while simultaneously logging underway temperature, and salinity and chlorophyll and fluorescence. Data files are created every three minutes, yielding a sampling resolution of 1 km along a cruise track.

To date, about 130 days of continuous cytometry data have been collected, sampling 60,000 km in the North Pacific ocean, a data set comparable to collecting 120,000 traditional flow cytometry samples. A set of software tools have been developed for automated clustering and analysis of cytometric populations [11]. This processed data can be accessed and visualized through our web portal.[1] Figure 2 is a *cytogram* that illustrates SeaFlow's ability to discriminate between microbial populations. A key remaining challenge is to create collaborative data processing, a management and analysis platform, with both on-ship and on-shore components, to allow scalable, interactive analysis of the rapidly expanding data sets. This challenge is the motivation for our work with SQLShare.

Data Processing Workflows for SeaFlow The SeaFlow instrument produces up to 24,000 particle counts per second, depending on ambient cell concentrations. Each particle is associated with 5 measurements corresponding to the intensity of scattered light (a proxy for cell size) and the fluorescence emitted by different pigments (e.g., chlorophyll a and phycoerythrin) within the organism. Cells are clustered into different microbial populations using a variant of k-means. Data size is then reduced by computing the summary statistics for each population. The summarized SeaFlow data are then merged with other data streams from the research cruise. The other sensors include a continuous flow-through thermosalinograph (TSG) to measure salinity and temperature and sometimes dissolved oxygen, latitude and longitude from the ship's navigation system, and depth profiles from

[1] `http://seaflow.ocean.washington.edu`

conductivity, temperature depth (CTD) packages. Additional sensors may be added depending on the cruise and the particular science mission. Certain variables such as location, time, temperature, salinity are integrated to the SeaFlow data during the cruise. Much of the other data must be processed first before being integrated with other datasets. This processing is done offline, after the cruise. The entire cruise dataset must then be integrated and shared with colleagues in a timely manner. It is at this phase that the script-oriented "pipeline" model breaks down, for the reasons described in the introduction: poor scalability beyond main memory, complications from multiple versions, and poor provenance. Consider these examples:

Example (Error Propagation): A bug was discovered in the initial R script responsible for computing summary statistics of the different microbial populations, a task on which the majority of the post processing depended. Resolving the bug was not an especially onerous task, but it took significant effort and communication with collaborators to ensure that everyone was using and sharing the corrected data. With SQLShare, this data is stored in one central place and is automatically re-generated on-demand as needed — no "stale" copies can exist.

Example (Scaling Data Integration): The R scripts used to process SeaFlow data pertain to exactly one cruise only; each cruise is processed independently. This approach complicates longitudinal analysis across multiple cruises. Combining files from different cruises and re-processing is one solution, but requires loading tens of GB into memory at once, which is not feasible without an investment in new hardware. In SQLShare, datasets from multiple cruises can be combined with a single `UNION` query (Figure 4). Thanks to the closure properties of the underlying relational algebra, we know that a set of cruises can be handled identically to a single cruise: everything is a relation. Evaluation is lazy; nothing is recomputed until the results are accessed. More importantly, no assumptions about available memory are made when manipulating these datasets, and performance is excellent. In this way, SQLShare facilitates a new class of longitudinal science questions that were previously de-emphasized due to logistical challenges. For example, macro-ecology questions involving the relationships between organisms and their environment at large spatial scales cannot be studied without convenient, efficient access to large-scale datasets.

To solve these problems and migrate the overall workflow to SQLShare, the processing scripts must be re-implemented in SQL. It may seem surprising that this is possible; SQL has a reputation for being a simple, inexpressive language. But we find in practice that most data processing tasks reduce to table manipulations, at which SQL excels.

For example, Figure 3 shows two implementations of the same task, one in R and one in SQL. The task is to average a timeseries of measurements by 3-minute intervals. The two implementations are of superficially comparable length and complexity and produce the same results, up to structural distinctions. The R script must read from and write to a file, exposing a parameter that is dependent on the user's local filesystem resources. The storage for the SQL version is implicit. The SQL version can also be optimized by the database and executed regardless of the available memory, whereas line 1 of the R version reads the entire file into memory — a critical limitation for long cruises.

3 SQLShare Architecture, Data Model, and Features

SQLShare has three components [7], all of which are cloud-hosted: a web-based UI, a REST web service, and a database backend. The UI is a Django Python application (a web framework similar to Ruby on Rails), and hosted on Amazon Web Services. The UI communicates with the backend exclusively through REST calls, ensuring that all client tools have full access to all features. The web service is implemented on Microsoft Azure as (one or more) Web Roles. The database is implemented using Microsoft's SQL Azure system, which is very similar to Microsoft's SQL Server platform.[2]

[2]The differences include: tables must have clustered indexes, non-SQL user-defined functions are not supported, and most distributed query features are not supported.

```r
table <- read.csv(''table.csv'')
# define 3 min time intervals
breaks <- seq(
          min(table$time),
          max(table$time),
           by=3)
# bin the table according to the breaks
b <- cut(table$time, breaks=breaks)

# calculate the mean of each variable
b.time <- tapply(table$time, b, mean))
b.Fluo <- tapply(table$Fluo, b, mean))
b.Temp <- tapply(table$Temp, b, mean))
b.Oxyg <- tapply(table$Oxyg, b, mean))
b.Nitr <- tapply(table$Nitr, b, mean))
b.Lat  <- tapply(table$Lat, b, mean))
b.Lon  <- tapply(table$Lon, b, mean))

binned.table <- data.frame(
          cbind(b.time, b.Fluo, b.Temp,
                b.Oxyg, b.Nitr,
                b.Lat, b.Lon))
write.csv(binned.table,''binned.csv'')
```

(a) An R implementation of the time-binning task

```sql
WITH data AS
  (SELECT * FROM [table.csv]),
    -- compute the minimum timestamp
    bounds AS
  (SELECT min(time) AS mintime FROM data),
    -- assign each timestamp a bin
    binned AS
  (SELECT bounds.mintime +
          floor((data.time - bounds.mintime)/3.0) *
          3.0 as binid
    FROM data, bounds)
-- compute the average of each bin
SELECT binid
   , avg(Fluo) as Fluo
   , avg(Temp) as Temp
   , avg(Oxyg) as Oxyg
   , avg(Nitr) as Nitr
   , avg(Lon) as Lon
   , avg(Lat) as Lat
   , avg(time) as time
FROM binned
GROUP BY binid
ORDER BY binid asc
```

(b) A SQL implementation of the time-binning task

Figure 3: Two implementations of a time-binning task, where each measured variable as averaged across 3-minute intervals. Variables from different sensors are binned to the same intervals, then joined to construct a single dataset. Data from multiple research cruises are similarly integrated, then merged into a single overall dataset, as seen in Figure 4.

A screenshot of SQLShare appears in Figure 5. The data model consists of a single entity: the dataset. A dataset consists of a named SQL query (implemented as a view in the underlying database), a free-text description, and a collection of metadata tags. We also compute and cache a preview of the results of each view to afford low-latency browsing of the relatively static science datasets we encounter in practice. When a new dataset is uploaded, both the physical *base table* and a trivial *wrapper view* of the form `SELECT * FROM table` are created by the system. Users may then modify the wrapper view or create *derived views*. Figure 1 illustrates the situation. Users rarely interact directly with the underlying physical tables: every dataset is associated with a view. By erasing the distinction between logical view and physical table, we preserve the ability to choose when views should be fully materialized (i.e., pre-computed and stored on disk). Since there are no destructive updates supported, we can cache view results as aggressively as space will allow. When a view definition changes, downstream dependent views may no longer be valid. In this case, before applying the change, we create a snapshot of any views that would have been invalidated by the change. With this approach, no change or access to any view will ever be rejected outright, though we do warn the user when dependent objects are affected.

Researchers can load data into SQLShare by uploading files through a browser. The system makes an attempt to infer the record structure and schema of the file by recognizing column names, identifying row and column delimiters, and inferring the type of the data in each column. Files with no column headers are supplied with default names. Various data quality issues are addressed automatically: files with an inconsistent number of columns or inconsistent data types among rows can still be uploaded successfully. The design goal was to be as tolerant as possible in getting data into the system and encourage the use of queries and views to repair quality problems. For example:

- Numeric data is often polluted with some string value representing NULL (e.g., “N/A”, “None”, or “-”), complicating automatic type inference. The situation is easy to repair by writing a simple view to replace these strings with a true NULL.

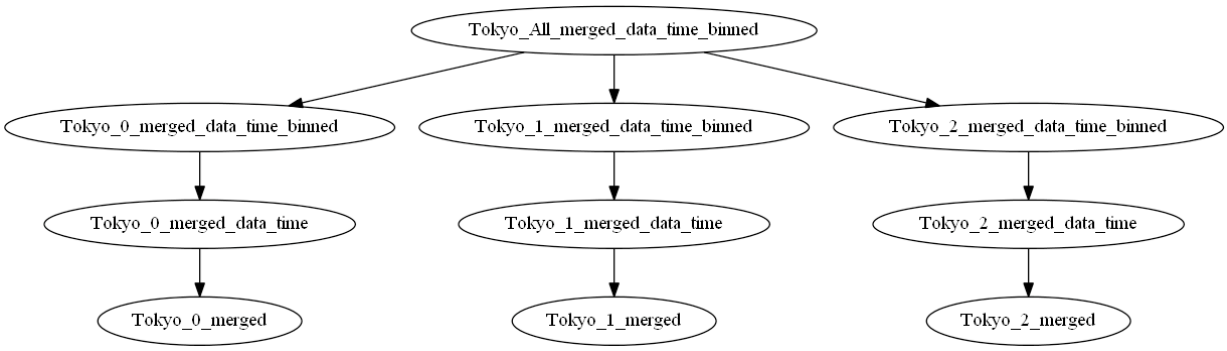

Figure 4: A set of related datasets in SQLShare representing the integration of data from three research cruises organized as a dependency graph. These dependencies are derived automatically from the queries themselves and need not be tracked explicitly by a separate system.

- Missing or non-descriptive column names can be replaced by using aliases in the SELECT clause.
- Bad rows and bad columns can be filtered out entirely with an appropriate WHERE clause or SELECT clause, respectively.
- Logical datasets that have been artificially decomposed into multiple files can be reconstructed with a UNION clause. For example, one week of sensor data may be represented as seven one-day files.

This idiom of uploading dirty data and cleaning it declaratively in SQL by writing and saving views has proven extremely effective: it insulates other users from the problems without resulting in multiple versions of the data accumulating, and without requiring external scripts to be written and managed: everything is in the database.

All permissions handling is pushed down into the database. Each SQLShare user is associated with a database user and a schema, and permissions changes in the UI are translated into `GRANT` and `REVOKE` statements in the database. Web authentication is handled through OAuth and Shibboleth; once authentication is confirmed, the service impersonates the user when issuing queries.

The SQLShare data model, API, and supported features are designed to lift certain database features (e.g., views) and suppress others (e.g., DDL and transactions). Here is a summary of the distinguishing features:

No Schema We do not allow `CREATE TABLE` statements; tables are created directly from the columns and types inferred in (or extracted from) uploaded files. Just as a user may place any file on a filesystem, we intend for users to put any table into the SQLShare "tablesystem," not just those that comply with a pre-defined schema.

Appends and Incremental Upload Datasets can be uploaded in chunks. This mechanism allows large files to be uploaded safely, but also affords support for appends: A chunk for a table can arrive at any time, and the table can be freely queried between chunks (the chunked upload is non-transactional.) Appends are handled in the view layer, and not be physical insertion into the underlying table. Each chunk is created as a separate table, and the base view is rewritten as a `UNION` of these chunks. There are two advantages to this approach. First, this organization is exactly what is required for distributed query processing in many vendors' systems, especially Microsoft SQL Server. Each chunk is placed on a separate disk or server, allowing them all to be scanned and filtered in parallel.[3] Second, the original partitioning of the data is preserved for provenance reasons. If a "bad" chunk is uploaded, it can be trivially removed or replaced by simply editing the query rather than editing

[3]Distributed query is not the same as true parallel query processing, but it can still significantly improve performance and be used to implement federated databases.

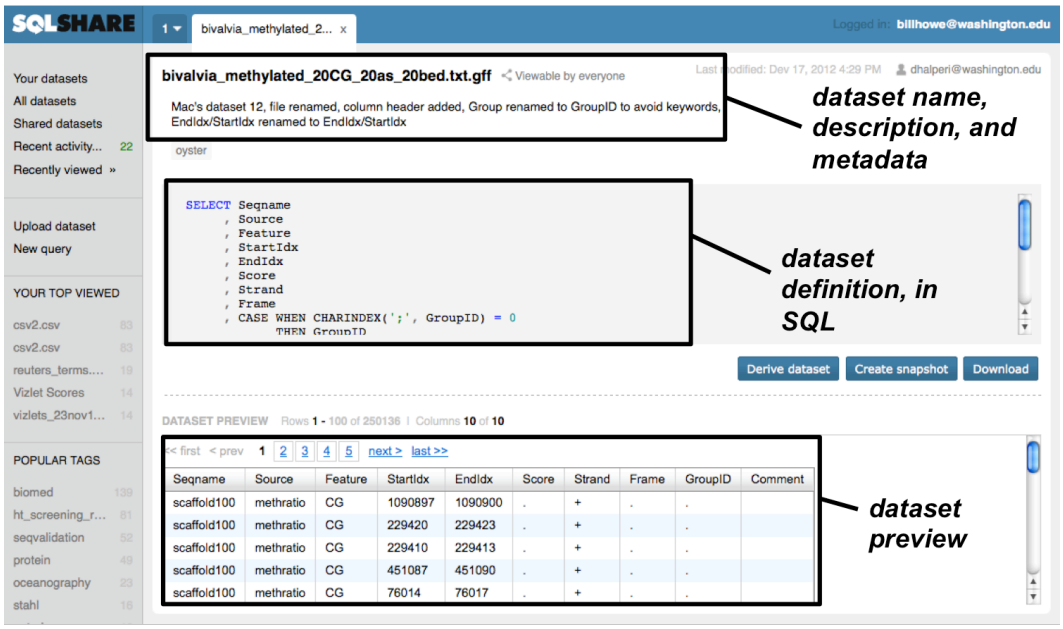

Figure 5: An annotated screenshot of the SQLShare system. The options at left support browsing datasets in typical ways: by popularity, by recency, by tags. Each dataset consists of its SQL definition, a text description, a set of tags, and a preview of its result.

the database. This approach emphasizes dataset-level operations over row-level operations, which we find to be the natural unit of processing for researchers and a closer analog to the file-oriented manipulation to which researchers are accustomed.

Tolerance for Structural Inconsistency Files with missing column headers, columns with non-homogeneous types, and rows with irregular numbers of columns are all tolerated. We find that data need not be pre-cleaned for some tasks (*e.g.,* counting records), and that SQL is an adequate language for many data cleaning tasks.

Metadata and Tagging SQLShare encourages creating views liberally. Navigating and browsing the hundreds of views that result from the use of SQLShare has emerged as a challenge not typically encountered in database applications. To help solve the problem, views can be named, described, and tagged through the UI and programmatically through the REST web service. The tags can be used to organize views into virtual folders. In future work, we are implementing bulk operations on virtual folders: download, delete, tag, and change permissions. We are also experimenting with a feature that would allow regex find-and-replace over a set of view definitions to simplify refactoring. We envision eventually evolving into a database-backed IDE-type environment for SQL and UDF development.

Append-Only, Copy-on-Write We do not allow destructive updates. Users insert new information by uploading new datasets. These datasets can be appended to existing datasets if the schemas match. Name conflicts are handled by versioning — the conflicting dataset is renamed, and views that depend on the old version are updated to reflect the change.

Simplified Views To make views simpler to use, we avoid the awkward CREATE VIEW syntax. In SQLShare, view creation is a side effect of querying — the current results can be saved by simply typing a name.This simple UI adjustment appears to be effective, with over 2500 views registered in the system by over 350 users.

Provenance Browsing We find that some users create deep hierarchies of rather simple, incremental views. This usage pattern is encouraged — the optimizer does not penalize you at runtime, and a composition of simple queries is easier to read and understand than one huge query. However, databases provide no natural way to browse and inspect a hierarchy of views. The catalog must be queried manually. In SQLShare, we are actively developing two features to support this use case: First, a *provenance browser* that creates an interactive visualization of the dependency graph of a hierarchy of composed views to afford navigation, reasoning, and debugging. Each node in the graph can be clicked to access the view definition in the existing SQLShare interface. Second, each table name in a view definition is rendered as a link if it refers to a view, affording more direct navigation through the hierarchy.

Semi-automatic Visualization An immediate requirement among frequent users of SQLShare is visualization. VizDeck is a web-based visualization client for SQLShare that uses a card game metaphor to assist users in creating interactive visual dashboard applications in just a few seconds without training [9]. VizDeck generates a "hand" of ranked visualizations and UI widgets, and the user plays these "cards" into a dashboard template, where they are automatically synchronized into a coherent web application that can be saved and shared with other users. By manipulating the hand dealt — playing one's "good" cards and discarding unwanted cards — the system learns statistically which visualizations are appropriate for a given dataset, improving the quality of the hand dealt for future users.

Automatic Starter Queries SQLShare users frequently do not have significant SQL expertise, but are fully capable of modifying example queries to suit their purpose [12]. For some collaborators, we seed the system with these "starter queries" by asking them to provide English questions that we translate (when possible) into SQL. But this manual approach does not scale, so we have explored automatically synthesizing good example queries from the structural and statistical features of the data [8]. Users upload data and example queries that involve reasonable joins, selections, unions, and group bys are generated automatically. We are in the process of deploying this feature in the production system.

4 Conclusions and Next Steps

The early response from our system has been remarkable. At the first demonstration, the results of a simple SQL query written "live" in less than a minute caused a post doc to exclaim "That took me a week!" meaning that she had spent a week manually cleaning and pre-filtering a handful of spreadsheets and then computing a join between them using copy-and-paste techniques. Within a day, the same post doc had derived and saved several new queries.

The experience was not isolated: the director of her lab has contributed several of her own SQL queries. She has commented that the tool "allows me to do science again," explaining that she felt "locked out" from personal interaction with her data due to technology barriers, relying instead on indirect requests to students and IT staff. She is not alone — over 2500 views have been saved in the SQLShare system by over 350 users since its deployment.

We originally focused on facilitating exploratory, interactive analysis of datasets that were outgrowing script-oriented solutions. However, as we have described in this paper, a pure SQL approach to scientific workflow realizes a variety of advantages, even relative to workflow management systems that are designed expressly for these situations. We find that the provenance, maintainability, lazy evaluation, and scalability provide a "white-box" workflow solution that is markedly superior to "black-box" approaches that rely on scripts and files.

Our next steps include surfacing the provenance information in the UI in a more direct way, allowing users to browse and interact with the hierarchy of views like that in Figure 4. We are also exploring a distributed deployment of SQLShare to allow multiple Universities to share data in a controlled way. To support sensitive

data, we are establishing a locally deployable version of SQLShare that is compliant with HIPAA, ITAR, and FERPA regulations. The system can be accessed at `http://sqlshare.escience.washington.edu`.

5 Acknowledgments

This research was sponsored by NSF award 1064505, the Gordon and Betty Moore Foundation, and Microsoft Research.

References

[1] P. G. Brown. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 963–968, New York, NY, USA, 2010. ACM.

[2] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[3] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *Proc. VLDB Endow.*, 2(2):1481–1492, Aug. 2009.

[4] S. Cohen-Boulakia and U. Leser. Search, adapt, and reuse: the future of scientific workflows. *SIGMOD Rec.*, 40(2):6–16, Sept. 2011.

[5] T. Crithclow, I. Altintas, G. Chin, D. Crawl, H. Iyer, A. Khan, S. Klasky, S. Koehler, B. Ludäescher, P. Mouallem, M. Nagappan, N. Podhorszki, A. Shoshani, C. Silva, R. Tchoua, and M. Vouk. Working with workflows: Highlights from 5 years building scientific workflows. In *SciDAC Conference*, Denver, Colorado, July 2011. Department of Energy.

[6] P. B. Heidorn. Shedding Light on the Dark Data in the Long Tail of Science. *Library Trends*, 57(2):–299, 2008.

[7] B. Howe. SQLShare: Database-as-a-service for long tail science. http://escience.washington.edu/sqlshare.

[8] B. Howe, G. Cole, E. Soroush, P. Koutris, A. Key, N. Khoussainova, and L. Battle. Database-as-a-service for long tail science. In *SSDBM '11: Proceedings of the 23rd Scientific and Statistical Database Management Conference*, 2011.

[9] A. Key, B. Howe, D. Perry, and C. Aragon. VizDeck: self-organizing dashboards for visual analytics. In *Proc. of the SIGMOD Conf.*, pages 681–684, 2012.

[10] P. Murray-Rust and J. Downing. Big science and long-tail science. `http://blogs.ch.cam.ac.uk/pmr/2008/01/29/big-science-and-long-tail-science/`, term attributed to Jim Downing.

[11] F. Ribalet, D. M. Schruth, and E. V. Armbrust. flowphyto: enabling automated analysis of microscopic algae from continuous flow cytometric data. *Bioinformatics*, 27(5):732–733, Mar. 2011.

[12] Sloan Digital Sky Survey. `http://cas.sdss.org`.

[13] J. Swalwell, F. Ribalet, and E. V. Armbrust. Seaflow: A novel underway flow-cytometer for continuous observations of phytoplankton in the ocean. *Limnology and Oceanography Methods*, 9:466–477, 2011.

Bill Howe is the Director of Research for Scalable Data Analytics at the University of Washington eScience Institute and holds an Affiliate Assistant Professor appointment in the Department of Computer Science & Engineering, also at UW. He leads a team studying data management, analytics, and visualization systems for science applications, with emphasis on domain-specific query languages and query algebras. He currently leads the SQLShare, VizDeck, and GridFields projects, and co-leads the Myria project that aims to provide a cloud-hosted service for large-scale data manipulation and analytics targeting scientists. Howe also serves on the board of ZettaScience and the Science Advisory Board of the SciDB project. He holds a Ph.D. in Computer Science from Portland State University, where he studied under Prof. David Maier, and a Bachelor's degree in Industrial & Systems Engineering from Georgia Tech.

Francois Ribalet received his Ph.D. from the Open University of London, UK, and the Stazione Zoologica Anton Dohrn in Naples, IT, where he first began to study the microbial ecology of marine ecosystems. He conducted postdoctoral research at the University of Washington, where he examined how physical and chemical gradients influence the distribution, abundance and activity of microbial communities. He uses advanced technology for the automated measurement of microbial populations and their activity at the single cell level, and develops tools to facilitate the analysis of the high volume of flow cytometry data. His current research focuses on determining the selective forces that shapes the patterns of microbial communities across ocean basins.

Daniel Halperin is a postdoc at the University of Washington eScience Institute. Prior to eScience, he completed his Ph.D. in wireless networking at UW's department of Computer Science and Engineering. His work on security for implantable medical devices won a best paper award at the 2008 IEEE Security and Privacy conference, and multiple projects have been featured in the New York Times and on PBS Nova ScienceNOW.

Sagar Chitnis is a research engineer at the University of Washington eScience Institute and currently leads development of SQLShare. He has expertise in cloud systems, automated testing, embedded computing, and managed deployment. He comes to UW from Microsoft, where he was a Senior Software Development Engineer on the Windows Home Server group. Prior to that, he worked at Qualcomm on projects ranging from desktop email to software interrupt mechanisms for embedded systems. Sagar has a Masters degree in Computer Science from the University of Southern California (USC) and a Bachelor Of Engineering (BE) in Electronics and Telecommunication from the University of Poona, India.

E. Virginia Armbrust is a Professor of Oceanography at the University of Washington in Seattle, WA. She received her Ph.D. from the Massachusetts Institute of Technology and the Woods Hole Oceanographic Institution, where she first began working with marine microbes. She conducted postdoctoral research at Washington University, where she studied the molecular genetics of chloroplast inheritance in green algae. Since her arrival at the University of Washington in 1996, she has focused on understanding the roles of diatoms in marine ecosystems. Her work combines genomics and physiological approaches to address questions that span a range of spatial and temporal scales. Current areas of interest include the basis of molecular interactions between diatoms and other microbes, the factors that control the abundance and distribution of marine algae across ocean basins, and the impact of past and future conditions on the distributions and interactions of diatoms in the sea.