# Collaborative Science Workflows in SQL

*SQLShare is a Web-based application for collaborative data analysis that emphasizes a simple upload-query-share protocol over conventional database design and ad hoc interactive query over general-purpose programming. Here, a case study examines the use of SQLShare as an alternative to script-based scientific workflows for a project in observational biological oceanography.*

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 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 SeaFlow, a new generation of flow cytometer created at the University of Washington that continuously measures phytoplankton composition and abundance at a rate of thousands of cells per second.[1] The instrument generates the equivalent of 6,700 samples, representing a dataset of 35–135 Gbytes, after a typical two-week oceanographic cruise. To date, the instrument's dataset represents more than 200 billion cells collected in different seasons and different environments, but only 10 percent of the data has been analyzed so far due to challenges in management and analysis. (See the related sidebar for more details.)

To contend with such challenges, through a partnership between the University of Washington's eScience Institute and the Armbrust Lab, we're 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. Here, we present a case study of SQLShare use in environmental flow cytometry. We argue 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.

1521-9615/13/$31.00 © 2013 IEEE
Copublished by the IEEE CS and the AIP

Bill Howe, Daniel Halperin, Francois Ribalet, Sagar Chitnis, and E. Virginia Armbrust
*University of Washington*

## SQLShare: Collaborative Science via Relational View Sharing

SQLShare is a Web-based query-as-a-service system that eliminates the initial setup costs associated with conventional database projects and instead emphasizes a simple upload-query-share protocol: users can upload their table-structured files through a browser and immediately query them using SQL without needing schema design, preprocessing, or the assistance of database administrators. SQLShare users derive new datasets by writing and saving SQL queries; these derived datasets can be queried and manipulated in the same way as uploaded datasets. 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 (see Figure 1), but there are some important differences:

- No special software is required—all data and processing logic are fully accessible through a Web browser, and views can be edited directly in the browser as well.
- Data never needs to be reprocessed explicitly—for most queries, the underlying database system can efficiently and scalably regenerate 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 (nor thrash due to limited 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, eliminating confusion over which version is current or correct.
- SQLShare stores source code and data, coupling them in a single Web interface: Views and the input data they require can't be separated, so errors resulting from applying a script to the wrong data can't occur (in contrast, physical files are generally decoupled from the scripts that generate them, and can only be associated explicitly through elaborate metadata schemes or sophisticated provenance systems).
- Ownership of a view affords precise control over access permissions: Views can be public, private, or shared explicitly with specific collaborators in a centralized manner.
- Because all data are stored in a centralized location, there's only one global namespace to manage—it's impossible to have two datasets with the same name but different content.
- A complex network of views can be inspected visually (see Figure 2), providing an intuitive form of provenance.

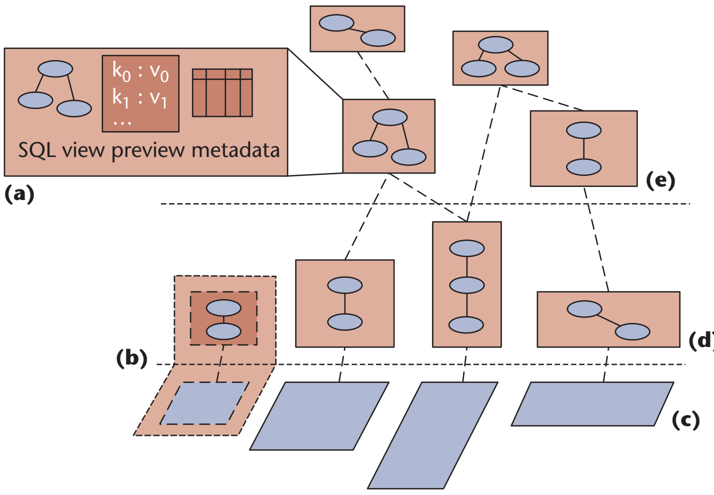
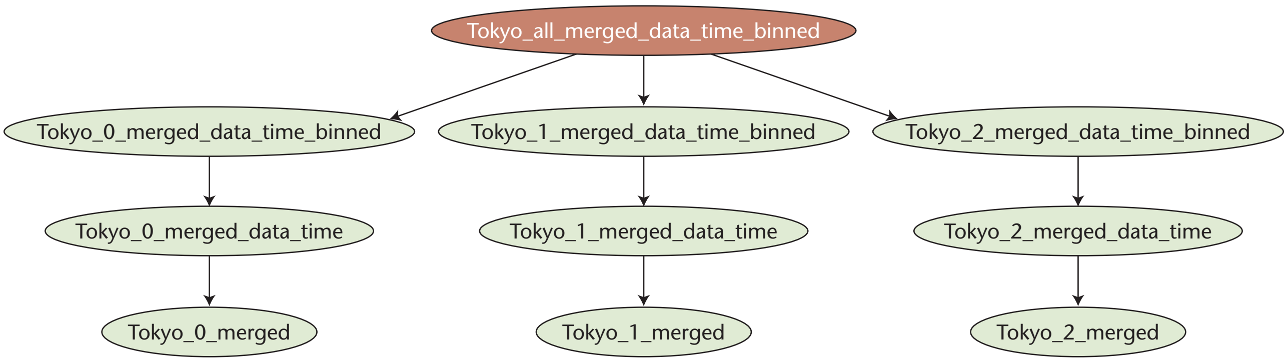


Figure 1. The SQLShare data model. (a) A dataset's internal structure consists of a relational view, attached metadata, and a cached preview of the results. (b) A newly uploaded dataset creates both a physical base table and an initial (trivial) wrapper view. (c) The physical data are stored in base 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.



Tokyo_all_merged_data_time_binned
Tokyo_0_merged_data_time_binned
Tokyo_1_merged_data_time_binned
Tokyo_2_merged_data_time_binned
Tokyo_0_merged_data_time
Tokyo_1_merged_data_time
Tokyo_2_merged_data_time
Tokyo_0_merged
Tokyo_1_merged
Tokyo_2_merged

Figure 2. 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.

## Why Use SQL for Collaborative Data Analysis?

Why was the script-based approach for analyzing the SeaFlow data problematic? Here are the issues cited by our colleagues in Oceanography and echoed by our other collaborators:

- 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 assume all data can be loaded in the main memory; redesigning 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 redistributed among collaborators, creating the opportunity for confusion; also, older, deprecated versions of scripts and data can't be "recalled" by their authors, and thus might (silently) continue to be used to make wrong decisions.
- Given a dataset generated by some variant of these scripts, there's no unambiguous way to determine its *provenance*—that is, 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 motivated 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,[1] where the cost of workflow development can be amortized over many repeated executions. However, despite years of productive research, these systems haven't been widely adopted.[2] In particular, we find that workflow systems are difficult to deploy in the *long tail* of science[3,4]—that is, the smaller labs and individual researchers who may operate at the "forward edge" of science where rapidly changing requirements, exploding data volumes, and limited access to in-house IT infrastructure and expertise make development of reusable pipelines for data processing prohibitively expensive. As a result, we seek a system that emphasizes simple, interactive analysis instead of software engineering, yet automatically provides workflow-like provenance and reuse features as a side effect.

Our observation is that although relational databases support interactive analysis and can comfortably handle the scale of the data, they haven't enjoyed significant uptake in these scenarios. It's tempting to ascribe this underuse to a mismatch between scientific data and the models and languages of commercial database systems.[5] But our experience is that relational models and languages aren't the problem.[6] Instead, we find that the barrier to adoption is the assumption that, before these models and languages can be used, we must engineer a database schema and populate it only with clean, integrated, and well-structured data. But it's clear that 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 controlling unpublished research data—a concern that conflicts 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) don't.

Despite these weaknesses, the core technology of relational databases remains attractive for science. The datasets we encounter are naturally modeled as relations—such as comma-separate values (CSV) files and "rectangular" spreadsheets. These 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 the algorithms using cost-based algebraic optimization. Of course, performance is irrelevant if we can't use the programming interfaces and languages to implement the tasks that researchers must perform. But we (and others) have found that many of these analysis tasks

- The SQL definition of each view provides a concise and declarative expression of the logic, eliminating data-representation issues.[2]
- Ad hoc interactive exploration is supported directly: With sufficient permissions, any view can be copied, modified slightly, and re-executed—no matter who owns the original.

Our initial experience with SQLShare has allowed us to reject the conventional wisdom that "scientists won't write SQL." Our experience is that they can and will. We find that even nonprogrammers can create and share SQL views for a variety of tasks, including those considered the turf of general-purpose scripting languages: quality control, data integration, basic statistical analysis, sampling, binning and aggregation, data cleaning, and reshaping in preparation for visualization. We also find that researchers are using SQL not just to manipulate data themselves, but

```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")
```

```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
```

Figure A. Two implementations of a time-binning task, where each measured variable is averaged across three-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.

can be easily expressed as SQL queries.[6,7] As an example, Figure A illustrates binning a time series on three-minute intervals for integration with other data streams—a task that was initially assumed to be inappropriate for implementation in the database.

### References

1. T. Crithclow et al., "Working with Workflows: Highlights from 5 Years Building Scientific Workflows," *Proc. Scientific Discovery through Advanced Computing* (SciDAC) *Conf.*, 2011; www.mcs.anl.gov/uploads/cels/papers/scidac11/final/Critchlow_2011-Scidac-poster-paper-rev-5.pdf.
2. S. Cohen-Boulakia and U. Leser, "Search, Adapt, and Reuse: The Future of Scientific Workflows," *SIGMOD Record*, vol. 40, no. 2, 2011, pp. 6–16.
3. P.B. Heidorn, "Shedding Light on the Dark Data in the Long Tail of Science," *Library Trends*, 2008, vol. 57, no. 2, article no. 299.
4. P. Murray-Rust and J. Downing, "Big Science and Long-Tail Science," blog, 29 Jan. 2008; http://blogs.ch.cam.ac.uk/pmr/2008/01/29/big-science-and-long-tail-science.
5. P.G. Brown, "Overview of SciDB: Large Scale Array Storage, Processing and Analysis," *ACM Sigmod Conf.*, ACM, 2010, pp. 963–968.
6. B. Howe et al., "Database-as-a-Service for Long-Tail Science," *Proc. 23rd Scientific and Statistical Database Management Conf.*, Springer-Verlag, 2011, pp. 480–489.
7. J. Cohen et al., "MAD Skills: New Analysis Practices for Big Data," *Proc. Very Large Databases Endowment*, VLDB Endowment, vol. 2, no. 2, 2009, pp. 1481–1492.

also to exchange ideas and collaborate—sharing SQL queries lets researchers communicate in terms of science questions instead of computer programs.

Now that we've offered a sense of our approach to the problem, let's consider how this works in practice. The following case study shows how scientists use SQLShare as an alternative to script-based scientific workflows in observational biological oceanography.

## Case Study: A Census of Microbial Ocean Populations Using SeaFlow

The most abundant organisms in the world's oceans are microbes less than 20 micrometers ($\mu$m) in size. Together, these organisms drive biogeochemical cycling of major elements, with almost 50 percent 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

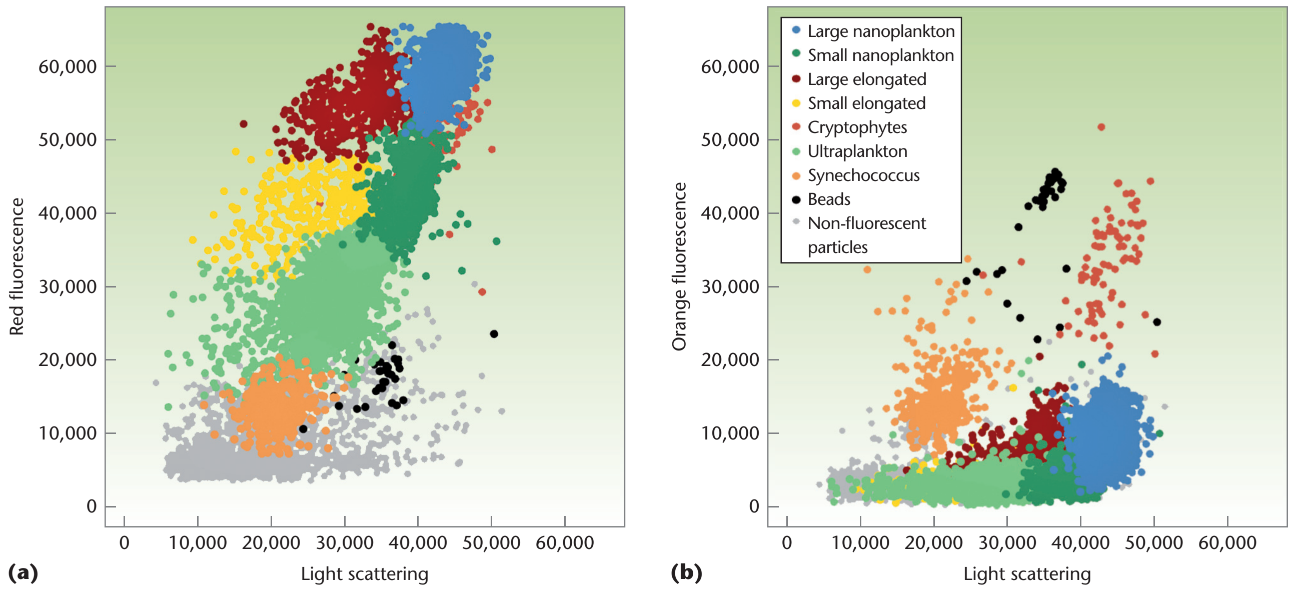


Figure 3. Example of flow cytometric signatures of phytoplankton populations in the North Pacific Ocean. (a) Red fluorescence from chlorophyll versus forward light scattering (a proxy of cell size) identified five distinct phytoplankton populations: large and small elongated phytoplankton, large and 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.

our view of photosynthetic microbe (phytoplankton) distribution, but provide little information about specific populations of phytoplankton. Also, finer-scale details are lost due to the data aggregation required to counteract cloud cover. Our ability to develop high-resolution maps of the distribution and abundance of these organisms is critical to understanding ecosystem functions and the sensitivity of these processes to environmental changes. Progress in this area has been limited largely because we lack observational tools for microbes.

## SeaFlow Overview

To address these issues, the University of Washington's SeaFlow project has developed an entirely new flow cytometer.[1] SeaFlow 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 $\mu$m), while simultaneously logging underway temperature, salinity, chlorophyll, and fluorescence. Data files are created every three minutes, yielding a sampling resolution of one kilometer (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 dataset comparable to collecting 120,000 traditional flow cytometry samples. A set of software tools has been developed to automatically cluster and analyze cytometric populations.[3] This processed data can be accessed and visualized through our Web portal (http://seaflow.ocean.washington.edu). Figure 3 is a cytogram that illustrates SeaFlow's ability to discriminate between microbial populations. A key remaining challenge is to create collaborative data processing and a management and analysis platform, with both on-ship and onshore components, to allow scalable, interactive analysis of the rapidly expanding datasets. This challenge is the motivation for our collaborative 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 five measurements corresponding to the intensity of scattered light (a proxy for cell size) and the fluorescence emitted by different pigments (for example, chlorophyll a and phycoerythrin) within the organism. Cells are clustered into different microbial populations using a variant of *k*-means (a cluster analysis method aimed at partitioning *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean). 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.

Additional SeaFlow measurements 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 conductivity, temperature, and depth (CTD) packages. More sensors can be added depending on the cruise and the particular science mission. Certain variables such as location, time, temperature, and salinity are integrated into the SeaFlow data during the cruise. Much of the other data must be processed 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's at this phase that the script-oriented "pipeline" model breaks down due to poor scalability beyond main memory, complications from multiple versions, and poor provenance. Consider these examples.

***Example 1: 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 wasn't 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 regenerated on demand as needed.

***Example 2: Scaling data integration.*** The R scripts used to process SeaFlow data pertain to only one cruise; each cruise is processed independently. This approach complicates longitudinal analysis across multiple cruises. Combining and reprocessing files from different cruises is one solution, but it requires loading tens of gigabytes into memory at once, which isn't feasible without an investment in new hardware.

In SQLShare, datasets from multiple cruises can be combined with a single `UNION` query (see Figure 2). 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 the performance is excellent. In this way, SQLShare facilitates a new class of longitudinal science questions that were previously deemphasized because of logistical challenges. For example, macroecology questions involving the relationships between organisms and their environment at large spatial scales can't 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 reimplemented in SQL. It might 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.

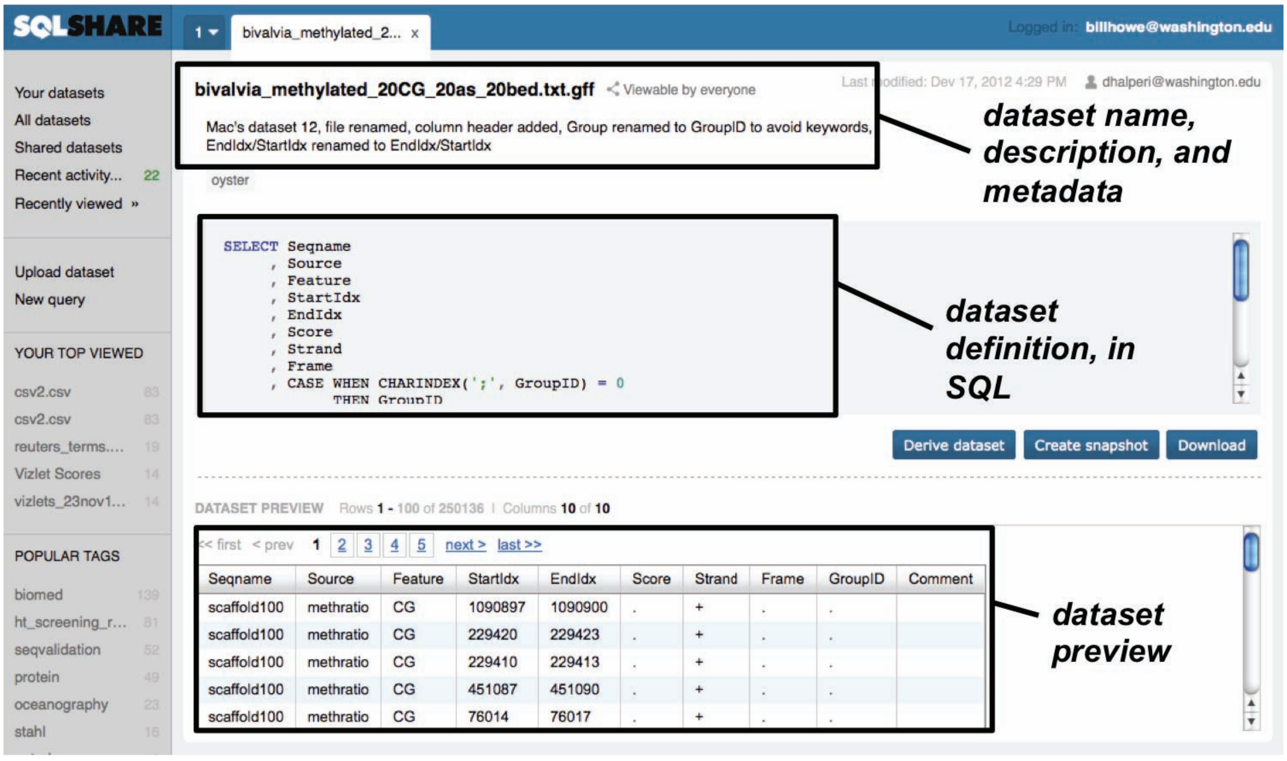
For example, Figure A (see the sidebar) shows two implementations of the same task, one in R (a free software programming language for statistical computing and analysis) and one in SQL. The task is to average a series of measurements by three-minute intervals. The two implementations are of superficially 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's dependent on the user's local file system 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.

## SQLShare Design and Implementation

SQLShare (see http://escience.washington.edu/sqlshare) has three components, all of which are cloud-hosted: a Web-based user interface (UI), a Representational State Transfer (REST) Web service, and a database back end. The UI is a Django Python application (a Web framework similar to Ruby on Rails), hosted on Amazon Web Services. The UI communicates with the back end 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 also is implemented using Microsoft's SQL Azure system, which is very similar to Microsoft's SQL Server platform.

### Data Model

Figure 4 shows a screenshot of SQLShare. 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 each



**bivalvia_methylated_20CG_20as_20bed.txt.gff**

Mac's dataset 12, file renamed, column header added, Group renamed to GroupID to avoid keywords, EndIdx/StartIdx renamed to EndIdx/StartIdx

```
SELECT Seqname
     , Source
     , Feature
     , StartIdx
     , EndIdx
     , Score
     , Strand
     , Frame
     , CASE WHEN CHARINDEX(';', GroupID) = 0
         THEN GroupID
```

| Seqname | Source | Feature | StartIdx | EndIdx | Score | Strand | Frame | GroupID | Comment |
|---|---|---|---|---|---|---|---|---|---|
| scaffold100 | methratio | CG | 1090897 | 1090900 | . | + | . | . | |
| scaffold100 | methratio | CG | 229420 | 229423 | . | + | . | . | |
| scaffold100 | methratio | CG | 229410 | 229413 | . | + | . | . | |
| scaffold100 | methratio | CG | 451087 | 451090 | . | + | . | . | |
| scaffold100 | methratio | CG | 76014 | 76017 | . | + | . | . | |

Figure 4. Annotated screenshot of the SQLShare system. The options on the left support browsing datasets in typical ways: by popularity, recency, and tags. Each dataset consists of its SQL definition, a text description, a set of tags, and a preview of its result.

view's results to afford low-latency browsing of the relatively static science datasets we encounter in practice.

When a new dataset is uploaded, the system creates both the *physical base table* and a *trivial wrapper* view of the form `SELECT * FROM TABLE`. Then, users can 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 (that is, precomputed and stored on disk). Because 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 might 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, although 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 file's record structure and schema by recognizing column names, identifying row and column delimiters, and inferring the data type in each column. Files with no column headers receive 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 is to be tolerant in getting data into the system and encourage the use of queries and views to repair quality problems. Here are some examples:

- Numeric data is often polluted with string values representing `NULL` (such as "N/A," "None," or "-"), complicating automatic type inference; we can repair this situation easily by writing a simple view to replace these strings with a true `NULL`.
- We can replace missing or nondescriptive column names with aliases in the `SELECT` clause.
- We can filter out bad rows and columns entirely with an appropriate `WHERE` or `SELECT` clause, respectively.
- We can reconstruct logical datasets that have been artificially decomposed into multiple files with a `UNION` clause—for example, we can represent one week of sensor data 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.

Further, 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 an open standard for authentification (OAuth) and Shibboleth. Once authentication is confirmed, the service impersonates the user when issuing queries.

### Key Features

The SQLShare data model, API, and supported features are designed to lift certain database features, such as views, and suppress others, such as document-description language (DDL) and transactions. Following are summaries of the distinguishing features.

***No schema.*** We don't allow `CREATE TABLE` statements; tables are created directly from the columns and types are inferred in (or extracted from) uploaded files. Just as users might place any file on a filesystem, we let them put any table into the SQLShare "tablesystem"—not just those tables that comply with a predefined schema.

***Appends and incremental upload.*** Datasets can be uploaded in chunks. This mechanism lets large files 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 physically inserted 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's required for distributed query processing in many vendors' systems, especially Microsoft SQL Server. Each chunk is placed on a separate disk or server, allowing each to be scanned and filtered in parallel. Distributed query isn't the same as true parallel query processing, but it can still significantly improve performance and be used to implement federated databases. 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 the database. This approach emphasizes dataset-level operations over row-level operations; we find dataset-level operations to be researchers' natural unit of processing and a closer analog to the file-oriented manipulation to which they are accustomed.

***Tolerance for structural inconsistency.*** Files with missing column headers, columns with nonhomogeneous types, and rows with irregular numbers of columns are all tolerated. We encourage researchers to put data into SQLShare as early as possible in the pipeline, and use SQL itself to clean the data to improve provenance and transparency.

***Metadata and tagging.*** SQLShare encourages creating views frequently and 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'll implement bulk operations—such as download, delete, tag, and change permissions—on virtual folders. We're also experimenting with a feature that would allow regular expression (regex) find-and-replace over a set of view definitions to simplify refactoring. We envision SQLShare evolving into a database-backed integrated development environment (IDE) for SQL and user-defined function (UDF) development.

***Append-only, copy-on-write.*** We don't 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 more than 2,500 views registered in the system by more than 350 users.

***Provenance browsing.*** We find that some users create deep hierarchies of simple, incremental views. This usage pattern is encouraged: the optimizer doesn't 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're actively developing two features to support this use case. First, we're developing a *provenance browser* that creates an interactive visualization of the dependency graph of a hierarchy of composed views to afford navigation, reasoning, and debugging. Users can click each node in the graph to access the view definition in the existing SQLShare interface. Second, we're rendering each table name in a view definition as a link if it refers to a view, affording more direct navigation through the hierarchy.

***Semiautomatic visualization.*** Visualization is an immediate requirement among frequent users of SQLShare. 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.[4] VizDeck generates a *hand* of ranked visualizations and UI widgets, and the user plays these *cards* into a dashboard template, where the program automatically synchronizes the cards into a coherent Web application that the user can save and share with other users. By manipulating the hand dealt—that is, playing the "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 lack significant SQL expertise, but they're fully capable of modifying example queries to suit their purposes (see, for example, the Sloan Digital Sky Survey; http://cas.sdss.org). For some collaborators, we seed the system with *starter queries* by asking them to provide English questions that we translate (when possible) into SQL. This manual approach doesn't scale, however, so we've explored automatically synthesizing good example queries from the data's structural and statistical features.[5] Users upload data and example queries that involve reasonable joins, selections, and unions; SQLShare automatically generates `GROUP BYS`. We're in the process of deploying this feature in the production system.

The early response to our system has been remarkable. One postdoctoral fellow was pretty excited during the first demonstration when a simple SQL query that was written live in less than a minute produced a result she had spent a week creating manually by cleaning and prefiltering a handful of spreadsheets and then computing a join between them using copy-and-paste techniques. Within a day, the same researcher had derived and saved several new queries.

This isn't an isolated experience: the director of her lab has contributed several of her own SQL queries. She commented that the tool "allows me to do science again," explaining that she felt "locked out" from personal interaction with her data because of technology barriers, relying instead on indirect requests to students and IT staff. She's not alone—as we mentioned, more than 2,500 views have been saved in the SQLShare system by more than 350 users since its deployment.

We originally focused on facilitating exploratory, interactive analysis of datasets that were outgrowing script-oriented solutions. However, as we've described here, 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's 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, letting users browse and interact with the hierarchy of views like that in Figure 2. We're also exploring a distributed deployment of SQLShare to allow multiple universities to share data in a controlled way. To support sensitive data, we're establishing a locally deployable version of SQLShare that's compliant with the Health Insurance Portability and Accountability Act (HIPAA), International Traffic in Arms (ITAR), and Family Education Rights and Privacy Act (FERPA) regulations. Visit http://sqlshare.escience.washington.edu to access this system.

## Acknowledgments

*This research was sponsored by US National Science Foundation award 1064505, the Gordon and Betty Moore Foundation, and Microsoft Research.*

## References

1. 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*, 2011, vol. 9, pp. 466–477.
2. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, 1970, vol. 13, no. 6, pp. 377–387.

3. F. Ribalet, D.M. Schruth, and E.V. Armbrust, “Flow-Phyto: Enabling Automated Analysis of Microscopic Algae from Continuous Flow Cytometric Data,” *Bioinformatics*, vol. 27, no. 5, 2011, pp. 732–733.
4. A. Key et al., “VizDeck: Self-Organizing Dashboards for Visual Analytics,” *Proc. ACM Sigmod Conf.*, ACM, 2012, pp. 681–684.
5. B. Howe et al., “Database-as-a-Service for Long-Tail Science,” *Proc. 23rd Scientific and Statistical Database Management Conf.*, Springer-Verlag, 2011, pp. 480–489.

***Bill Howe*** *is the Director of Research for Scalable Data Analytics at the University of Washington eScience Institute and an affiliate assistant professor in the Department of Computer Science and Engineering. His research interests include data management, analytics, and visualization systems for science applications, with an emphasis on domain-specific query languages and query algebras. Howe has a PhD in computer science from Portland State University. 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. Contact him at billhowe@cs.washington.edu.*

***Francois Ribalet*** *is a research assistant professor of oceanography in the Armbrust Lab at the University of Washington. His research interests include microbial communities across ocean basins. Ribalet has a PhD in biological sciences from the Open University of London, UK. Contact him at ribalet@u.washington.edu.*

***Daniel Halperin*** *is a postdoctoral researcher at the University of Washington eScience Institute, where he contributes to SQLShare, Myria, and other projects. His prior research focused on wireless networking and security. Halperin has a PhD in computer science and engineering from the University of Washington. His research has earned best paper awards at the 2008 IEEE Security and Privacy conference and 2013 USENIX Networking Systems Design and Implementation conference, and several of his projects have been featured in the* New York Times *and on* PBS NOVA ScienceNOW. *Contact him at dhalperi@escience.washington.edu.*

***Sagar Chitnis*** *is a research engineer at the University of Washington eScience Institute and currently leads the development of SQLShare. His research interests include cloud systems, automated testing, embedded computing, and managed deployment. Chitnis holds a master's degree in computer science from the University of Southern California (USC). Contact him at sagarc@cs.washington.edu.*

***E. Virginia Armbrust*** *is a professor of oceanography at the University of Washington. Her research interests include marine microbes and ecosystems. Ambrust has a PhD in biological oceanography from the Massachusetts Institute of Technology and the Woods Hole Oceanographic Institution. Contact her at armbrust@u.washington.edu.*

AIP
www.aip.org

The American Institute of Physics (AIP) is a not-for-profit membership corporation chartered in New York State in 1931 for the purpose of promoting the advancement and diffusion of the knowledge of physics and its application to human welfare. Leading societies in the fields of physics, astronomy, and related sciences are its members.

In order to achieve its purpose, AIP serves physics and related fields of science and technology by serving its member societies, individual scientists, educators, students, R&D leaders, and the general public with programs, services, and publications—*information that matters*.

The Institute publishes its own scientific journals as well as those of its member societies; provides abstracting and indexing services; provides online database services; disseminates reliable information on physics to the public; collects and analyzes statistics on the profession and on physics education; encourages and assists in the documentation and study of the history and philosophy of physics; cooperates with other organizations on educational projects at all levels; and collects and analyzes information on federal programs and budgets.

The scientists represented by the Institute through its member societies number more than 134,000. In addition, approximately 6,000 students in more than 700 colleges and universities are members of the Institute's Society of Physics Students, which includes the honor society Sigma Pi Sigma. Industry is represented through the membership of 37 Corporate Associates.

**Governing Board:** Louis J. Lanzerotti* (chair), Richard Baccante, Barry Barish, Malcolm R. Beasley, G. Fritz Benedict, J. Daniel Bourland,* Terri Braun, Robert Byer, Timothy A. Cohn, Beth Cunningham,* Bruce H. Curran,* Robert Doering, Michael D. Duncan,* H. Frederick Dylla*(ex officio), David Ernst, Janet Fender, Judith Flippen-Anderson,* Brian J. Fraser,* Jaime Fucugauchi, A. Jeffrey Giacomin,* Mark Hamilton, John Haynes, Paul L. Kelley, Angela R. Keyser, James T. Kirby Jr, Kate Kirby,* Rudolf Ludeke,* Jim Marshall, Kevin B. Marvel,* Christine McEntee, Catherine O'Riordan, Elizabeth A. Rogan, Charles E. Schmid,* Benjamin B. Snavely* (ex officio), David Sokoloff, Scott Sommerfeldt, Gene Sprouse, Gay Stewart, Hervey (Peter) Stockman, Michael Turner.
*Executive Committee member.

**Management Committee:** H. Frederick Dylla, Executive Director and CEO; Richard Baccante, Treasurer and CFO; Theresa C. Braun, Vice President, Human Resources; John S. Haynes, Vice President, Publishing; Catherine O'Riordan, Vice President, Physics Resources; Benjamin B. Snavely, Secretary.