Advancing Declarative Query in the Long Tail of Science

Bill Howe
billhowe@cs.washington.edu

Daniel Halperin
dhalperi@cs.washington.edu

Abstract

Relational databases remain underused in the long tail of science, despite a number of significant success stories and a natural correspondence between scientific inquiry and ad hoc database query. Barriers to adoption have been articulated in the past, but spreadsheets and other file-oriented approaches still dominate. At the University of Washington eScience Institute, we are exploring a new "delivery vector" for selected database features targeting researchers in the long tail: a web-based query-as-a-service system called SQLShare that eschews conventional database design, instead emphasizing a simple Upload-Query-Share workflow and exposing a direct, full-SQL query interface over "raw" tabular data. We augment the basic query interface with services for cleaning and integrating data, recommending and authoring queries, and automatically generating visualizations. 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. Researchers in oceanography, molecular biology, and ecology report migrating data to our system from spreadsheets, from conventional databases, and from ASCII files. In this paper, we will provide some examples of how the platform has enabled science in other domains, describe our SQLShare system, and propose some emerging research directions in this space for the database community.

1 Introduction

The database community has been incredibly successful at delivering technology to IT professionals, but our tools and techniques remain remarkably underused in science. Despite prominent success stories [8, 24, 2, 11] and what is perhaps a natural correspondence between exploratory hypothesis testing and ad hoc query answering, scientists continue to rely primarily on scripts and files.

The gap is especially acute in the *long tail* of science [19], the small labs and individual researchers who have exploding data requirements but limited IT budgets (Figure 1). For these researchers, data management problems are beginning to dominate their activities: In an informal survey [13], several of our collaborators reported that the ratio of time they spend "handling data" as opposed to "doing science" is approaching 9 to 1!

It may be tempting to ascribe this underuse to a mismatch between scientific data and the models and languages of commercial database systems, but our experience is that standard data models and languages, such as SQL, are suitable for managing and manipulating a significant range of scientific datasets and tasks. We are finding that the key barriers to adoption lie elsewhere:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- The initial investment required for the conventional database design and loading process can be prohibitive. 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, even a small project may have data in dozens of formats.

- The corpus of data for a given project or lab accretes over time, with many versions and variants of the same information and little explicit documentation about connections between datasets and sensible ways to query them.

- Concerns about control of unpublished research data, at tension with the need for unfettered collaboration and sharing, complicate centralization and organization in a database.

Guided by these premises, we have deployed a new "delivery vector" for SQL targeting researchers called SQLShare [13, 12]. SQLShare de-emphasizes up-front schemas, physical database design, and destructive updates, instead emphasizing logical data independence, ad hoc queries, and controlled sharing. Users upload their data and immediately query it using SQL — no schema design, no reformatting, no DBAs.

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 represeting 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.

- 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.

Users create and share views as a first-class activity, one which users have applied to a variety of data tasks: cleaning raw data, integrating disparate sources, and separating public and private data. Some use cases were unexpected: In one case, our collaborators are using deeply nested hierarchies of views as a data processing pipeline, remarking that a stack of views provided several benefits over a sequence of scripts: nothing needs to be re-executed when the pipeline changes, and the view definitions themselves can be inspected directly to provide a form of documentation and provenance. We describe further uses in Section 2.

To help non-experts self-train in writing SQL, we typically provide a "starter kit" of SQL queries when data is first uploaded either manually translated from English questions provided by the researchers as part of our initial engagement protocol (a variant of Jim Gray's "20 questions" methodology [8]), or in some cases derived automatically [14]. These starter queries demonstrate the basic idioms for retrieving and manipulating data, and are saved within SQLShare so they can be copied, modified, and saved anew by the researchers. A cloud-based

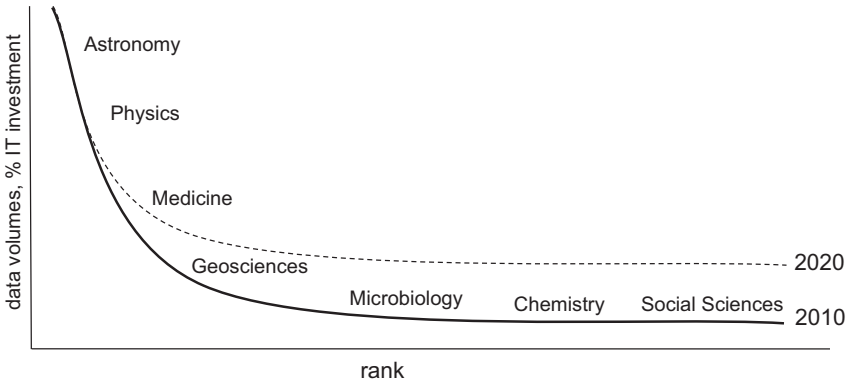

Figure 1: The long tail of science [19] — smaller labs and individual researchers with limited access to IT staff and resources but increasing data management needs and significantly more heterogeneity. These characteristics are more concentrated in some fields. Each line represents a different point in time; data volumes at all scales are trending upward.

deployment on Windows Azure has allowed us to establish a large interdisciplinary corpus of example queries that we can mine to assist in SQL authoring [5], organize and integrate data [3], and, going forward, study the way researchers interact with their data.

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 2000 views have been saved in the SQLShare system by over 200 users since its deployment. In this paper, we describe our initial experience with this system and outline some future directions to explore.

2 Declarative Query for Science: Some Use Cases

It may not be intuitively clear why the SQL language and the extensive use of views is appropriate for scientific inquiry. In this section, we describe a series of examples of how this platform can have an impact on science, especially in the long tail.

2.1 Example: Metagenomics as Set Manipulation

The goal of metagenomics is to correlate environmental conditions with genetic characteristics of the microorganism population as a whole — the metagenome [25]. Intuitively, metagenomics ask two questions: "Who is there?" and "What are they doing?" — i.e., what microorganisms are present in the sample, and what particular metabolic and regulatory activities are active?

Consider a comparative metagenomics experiment involving samples collected at different locations, depths, and times in a river estuary in order to characterize the coastal ecosystem (Figure 3). Each sample may be treated to analyze DNA (identifying organisms) or messenger RNA (identifying expressed genes that have been transcribed), then processed in a "next-gen" massively-parallel sequencing instrument to produce tens of millions of short reads around 35- to 1500-base-pairs long. Population analysis proceeds by using the DNA or RNA reads

to query public reference databases and verifying results against environmental context [17]. Reads without matches may indicate a new species — not unusual, given that hundreds of thousands of distinct species may be typically present in a sample, only a small fraction of which have been described in the literature.

After matching sequences from three distinct samples, one collaborator asked for help translating the following question into SQL: "I need to find the anomalies — genes that are found in at least one of three samples, but not in all of them."

Being a set-oriented declarative language, SQL can express this query rather elegantly: as the union of all samples less the intersection of all samples.

```
(SELECT gene FROM s1 UNION SELECT gene FROM s2 UNION SELECT gene FROM s3)
EXCEPT
(SELECT gene FROM s1 INTERSECT SELECT gene FROM s2 INTERSECT SELECT gene FROM s3)
```

Lessons Learned With our help, this query and others are written and saved within SQLShare, reusable by other researchers as examples. This process — to take users' data and English queries to bootstrap a solution — was described by Gray et al. in the context of the Sloan Digital Sky Survey [9]. Having captured thousands of queries in our system, most of which are written by the scientists themselves after some initial hand-holding, we corroborate SDSS findings that SQL is not the bottleneck to the uptake of relational technology. But we extend these findings to long tail science situations with limited access to dedicated database experts and limited IT funding.

2.2 Example: Integrating Ecological Field Measurements

Microorganisms sustain the biogeochemical cycling of nitrogen, one of the most important nutrient cycles on earth. A key step in this cycle, the oxidation of ammonia to nitrite by autotrophic microorganisms, is now attributed to the ammonia-oxidizing archaea (AOA), which are of high abundance in both marine and terrestrial environments. Understanding the environmental conditions in which these microorganisms thrive may have a significant impact on our ability to manage biodiversity [18].

To study the hypotheses related to how these organisms regulate and control the forms of nitrogen available to other microbial assemblages, field measurements are taken at different times and locations and combined with laboratory assays. Data collection is performed with spreadsheets for maximum flexibility. The raw field measurements are quality-controlled and integrated into "master" spreadsheets. Here the choice of technology breaks down: Shared files and frequent revisions to quality control procedures leads to multiple conflicting versions and poor provenance, processes for transforming data must be repeated by hand, the physical organization of the data into worksheets is inflexible and not always conducive to question answering.

The lab recognized the value in a database to organize these data, but did not want to sacrifice the flexibility of using spreadsheets for data collection. Out in the field, a fixed data entry form is too restrictive. Further, the (obvious) schema was fully expected to evolve steadily over time as they added more measurements.

Lessons Learned SQLShare became attractive as a "staging area" for the data as it came in from the field. With a few queries saved as views, the data could be integrated (the union of datasets from different researchers), cleaned (units standardized and incomplete records suppressed), and shared. Moreover, these views delivered a form of automatic provenance: the hierarchy of composed views, incrementally filtering, integrating, and cleaning the data, are visible to all and refresh automatically whenever the result is accessed. There is no need to "re-run" the data processing pipeline or workflow whenever the data changes. Finally, the use of SQLShare did not preclude the development of a more conventional database application down the road. In fact, a "rough cut" integration effort of the kind facilitated by SQLShare is a necessary first step.

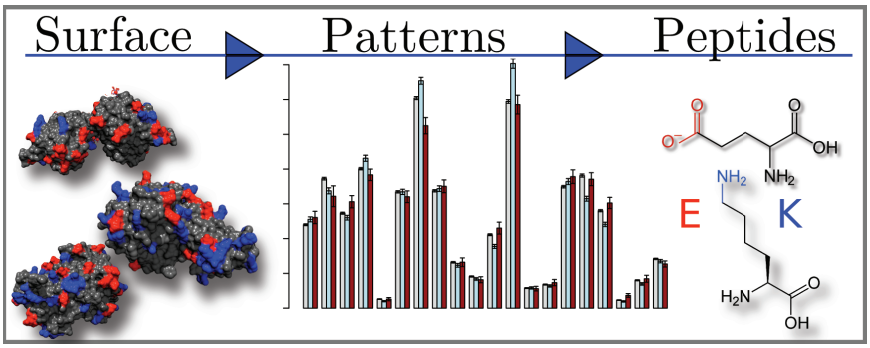

Figure 2: Illustration of the steps in an algorithm to (a) identify the surface of proteins, (b) calculate various statistics, and (c) synthesize "stealth" molecules that could mimic the protein surfaces. The use of SQLShare provided opportunities to exchange "a 10 minute 100 line script for 1 line of SQL."

2.3 Example: Drug Design and Stealth Proteins

A graduate student in Chemical Engineering was working with gigabytes of tabular data derived from analyses of protein surfaces. Given a set of proteins, a series of Python scripts identified their surface (Figure 2(a)), calculated statistics on them (Figure 2(b)), and synthesized molecules that could mimic the protein surfaces (Figure 2(c)). The application area is tissue engineering and implantable materials. The work involves creating surfaces that are "stealth," or invisible to the body because they look like proteins [20].

Loading the data into SQLShare allowed the student and an undergraduate assistant to access the data via the web, download query results for further analysis in R, and — crucially — "give reviewers access to data in publications." Previously, they report using "huge directory trees and plain text files." The adoption of SQL for basic processing allowed them to "accomplish a 10 minute 100 line script in 1 line of SQL [that ran in significantly less time]."

The previous implementation was considerably more complex:

> I ran a Python script that iterated through directories, each of which contained a protein. The Python script called shell scripts which called R scripts to generate data on each protein which was all copied into a total data single directory. Then the Python script calls another shell script which calls R scripts to generate statistics on the total data followed by another set of R scripts to generate plots. What was really getting out of control was that I ended up generating 50 files for each protein, which meant I had 125,000 files in a single directory in the course of the analysis.

Lessons Learned By implementing parts of this pipeline in SQL, the student was able to extract the table manipulation steps out of R and focus instead on the statistics. This separation of concerns is a key goal for SQLShare: it is designed not to supplant general purpose languages, but allow SQL to be as easy to use for quick scripting of table manipulation as R, Python, or MATLAB are for their own strengths. As a side effect, you will often realize significant performance gains even without physical database design and tuning by avoiding "quick and dirty" algorithms that lead to combinatorial explosions of file operations.

3 SQLShare System Details

SQLShare has three components [12]: 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

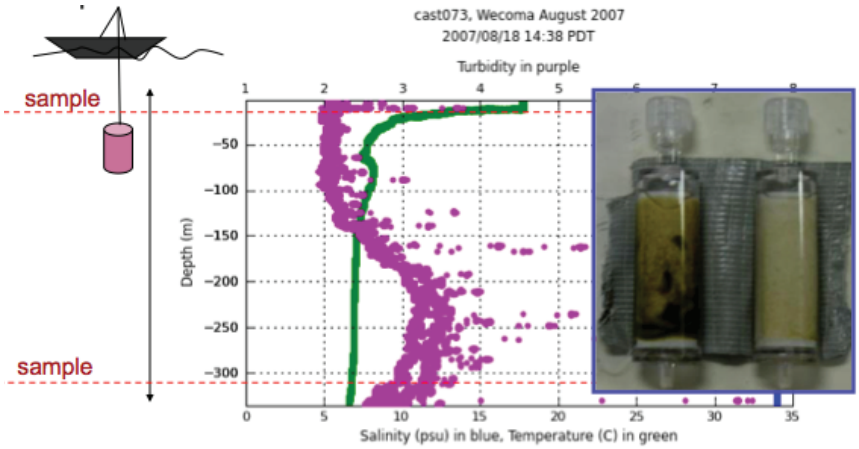

Figure 3: Oceanographic metagenomics involves sequencing entire microbial populations under different environmental conditions. Here, two samples are collected at different depths in the water column from a single *cast* of a Conductivity-Temperature-Depth sensor package (CTD). The sample near the surface has significantly more particulate matter, as is visible once the water is passed through a 2-micron filter (at right). These samples are frozen at sea, sequenced on shore, and computationally compared to correlate environmental conditions with population characteristics. This area of research may involve significantly more data and more samples than genomics techniques involving a single organism.

database is implemented using Microsoft's SQL Azure system, which is very similar to Microsoft's SQL Server platform.[1]

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.

Unifying Views and Tables Our data model consists of a single entity: the *dataset*. Both logical views and physical tables are presented to the user as datasets. By erasing the distinction, we reserve the ability to choose when views should be materialized for performance reasons. Since there are no destructive updates, 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, we create a pre-change snapshot of any views invalidated by the change. With this approach, no view will ever generate errors. However, these semantics may not be what the user expects; we are exploring alternatives for communicating these semantics to the user and allowing alternatives in some situations.

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.)

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.

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

Metadata and Tagging SQLShare encourages creating views liberally. Navigating and browsing hundreds of views 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, 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 upated to reflect the change.

Simplified Views We find views to be underused in practice. We hypothesize that the solution may be as simple as avoiding 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 — over 2000 views have been registered in the system by over 200 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 [16]. 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 [23]. 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 [14]. 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 Opportunities and Future Directions: Optimizing for Attention

SQLShare fits into an over-arching theme in eScience: database systems for scientists must be optimized for human attention. Consider Figure 4 (obtained from [1]): Both data volumes and computing capacity are growing exponentially over time, but human cognition has remained essentially flat. For the pace of science to keep up with the rate at which data is being generated, this computational power should be exploited to ensure that scientist attention is utilized to maximum effect.

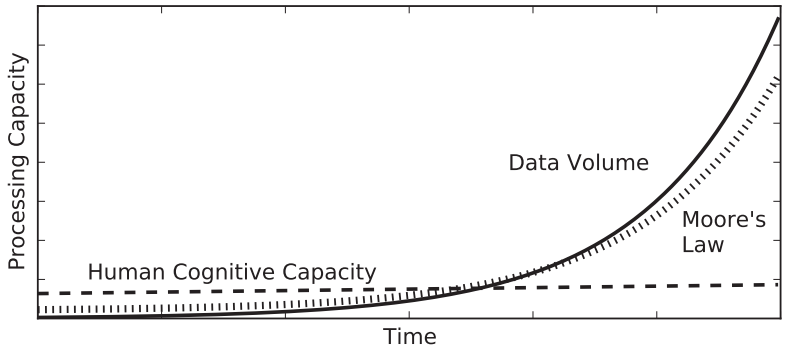

Figure 4: Data volumes grow exponentially, and computing resources have arguably kept pace. However, human cognition has remained essentially flat. This gap represents the dominant challenge for data-intensive science in the long tail.

One direction is to diminish the need for human intervention. SQLShare enables scientists to focus on asking science questions instead of on manual data processing/management or on database administration tasks. We are exploring new techniques that pursue aggressive automation of "human" tasks: SQL authoring [14, 5], visualization [16], and eventually statistical tests.

A second inefficient use of scientist attention is waiting for results. We make two observations about data use in the long tail that may help improve efficiency. First, data access is sparse and infrequent relative to the rate at which data volumes and computational capacity are growing. As a result, even shared data systems for science are typically pathologically underutilized [22]. Second, many science datasets are essentially read-only. Datasets registered as part of the scientific record are never intended to be updated (e.g., historical archives or data associated with a publication). Even prior to publication, data may be grouped into logical batches, then processed and quality-controlled as a unit before appending to an existing dataset (e.g., oceanographic research cruises [4]). These two observations suggest that there are opportunities to aggressively, speculatively, and perhaps wastefully consume resources for even small potential returns in runtime performance. We are exploring a variety of techniques to allow SQLShare and related systems to consume all available resources to improve performance and reduce human effort:

Eager Materialization All query results are cached and retained as materialized views that can be used to optimize future query plans. Even without adopting more sophisticated techniques to answer queries using materialized views [10], identical queries are often run multiple times and can benefit from this cache. In a typical scientific database, the lifetime of intermediate results is long and so is the expected utility of the cached results. This idea is related to database cracking [15], but simpler: Rather than physically reorganizing the data, we propose to simply keep multiple copies of the data. Several challenges emerge: Now that the entire available disk is potentially a cache, eviction policies based on current and anticipated query workload become interesting. These eviction policies can consider the time to recreate the view, partial materialization, and user-specified priorities as well. In addition, a practical implementation of techniques to answer queries using views is worthwhile.

Semantic Prefetching Caching relies on a "warm-up" period to fill the cache. Users receive no benefit querying "fresh" data, and at the frontier of research, most of the interesting data is fresh. Worse, the system will typically sit idle waiting for the human operator to express the next query. As data volumes and processing capabilities grow, the opportunity cost of this idle time becomes increasingly wasteful. We propose a second set of techniques, that we collectively refer to as *semantic prefetching* to exploit this idle time by anticipating unasked queries and speculatively generating possibly-useful results. We envision four levels of semantic prefetching:

1. In a system with geographically-dispersed, prefetch data that is known to already exist based on a model of

the user's task. This capability includes ranking results by similarity based on the user's recent browsing history ("more like this"), by results of interest to one's social network, by global popularity, and by compatibility with explicit user preferences. These techniques are already being explored in a variety of contexts.

2. Given an existing query $q(R)$ for some dataset R, apply q to some new dataset R' that has a compatible schema. For a typical database application, situations where this technique would be useful are essentially nonexistent. But the usage patterns of SQLShare suggest thousands of tables that are closely related (for example, multiple files telemetered from the same sensor, or different versions of the same data under different quality control assumptions). In these cases, users specifically request an operation of the form "do what I just did, but on this table." These opportunities are not difficult to recognize and proactively compute.

3. Even if we do not have an existing query to work from, we may be able to deduce some likely operations: candidates for joins, candidates for group bys, etc. We have begun to explore this approach in our work on automatic starter queries [14].

4. Perhaps neither the data nor the query is currently available in the system. In this case, it seems that all we can do is sit idle. However, if data acquisition itself is under the influence of the database system, as it is, for example, in the context of crowdsourced databases (c.f. Franklin et al. [7]), then there may be opportunities to speculatively acquire data based on predictions of the user's interests.

This last level warrants examples: Under what circumstances might the database be equipped to influence the acquisition of science data?

- Observational oceanographers sometimes make use of the concept of *vessels of opportunity* — vessels that are not under direction of the chief scientist, but that are in the right place at the right time to take an important measurement or sample. Empowering the system to automatically identify these opportunities and issue the request could significantly increase the value of data collected. Currently, scientists may not identify these situations until it is too late to exploit them.

- Citizen science projects provide volunteers with enough training and equipment to collect data on behalf of a research project. For example, volunteers for the Nature Mapping project [6] record observations of wildlife species in populated areas, dramatically improving the quality of the range maps used to inform public policy. A system that could proactively identify regions and species for which little data exists and issue standing requests for additional observations could amplify the effectiveness of these projects, and make the experience more rewarding for the volunteers.

- The term *adaptive sampling* refers to the capability of some sensors to receive commands while operating autonomously in the field. For example, an autonomous underwater vehicle (AUV) may adapt its trajectory based on commands issued from shore, or an atmospheric radar may rotate its antenna to face an incoming storm [21]. A system that can direct these resources automatically based not only on current observations but also based on the value of the potential derived products could increase the return on investment of deployment.

For all four levels, the search space is enormous and cannot be searched directly. Instead, we need to identify promising results by modeling importance to the user. While quantifying importance is difficult, a simplifying factor is that it is not necessarily important that the model is accurate when first deployed, as long as it is equipped to incorporate human feedback and learn statistically what is important and what is not. We envision an interface where scientists can browse the data products derived the previous night over their morning coffee, selecting a few for further review while rejecting the majority. This interaction provides a strong signal on which

to base a ranking algorithm. In some systems, storage resources may be limited. For these cases, cached and prefetched data can be pruned based on expected utility.

5 Summary

We have presented SQLShare, new platform for lightweight, aggressively automated data management designed to 1) allow SQL to compete with other lightweight languages and tools favored by scientists (Excel, Python, MATLAB, ASCII files, R), 2) increase collaborative data sharing by making it easy and fruitful to upload into a web-based system, 3) signficantly reduce the 9-to-1 ratio of time researchers report spending on data handling relative to science. Our progress so far is promising, with even non-programmers responding positively and becoming active users. In addition to the benefits to scientists and helping them explore their data, the use of a simplified interface for querying relational data not only motivate new research problems for the database community, but directly provide a corpus of real data and real queries that can catalyze the study of those problems.

6 Acknowledgments

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

References

[1] C. Aragon. Scientist-computer interfaces for data-intensive science. Microsoft eScience Workshop, 2010.

[2] S. Baker, J. Berger, P. Brady, K. Borne, S. Glotzer, R. Hanisch, D. Johnson, A. Karr, D. Keyes, B. Pate, and H. Prosper. Data-enabled science in the mathematical and physical sciences. Technical report, National Science Foundation, March 2010.

[3] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1), 2009.

[4] Center for Coastal Margin Observation and Prediction. `http://www.stccmop.org`.

[5] Collaborative query management. `http://db.cs.washington.edu/cqms`.

[6] K. Dvornich. Query NatureMapping data using SQLShare. `http://naturemappingfoundation.org/natmap/sqlshare/NM_sqlshare_1.html`.

[7] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering queries with crowdsourcing. In *Proc. of the SIGMOD Conf.*, pages 61–72, 2011.

[8] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *CoRR*, abs/cs/0502008, 2005.

[9] J. Gray and A. S. Szalay. Where the rubber meets the sky: Bridging the gap between databases and science. *CoRR*, abs/cs/0502011, 2005.

[10] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, Dec. 2001.

[11] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

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

[13] B. Howe and G. Cole. SQL Is Dead; Long Live SQL: Lightweight Query Services for Ad Hoc Research Data. In *4th Microsoft eScience Workshop*, 2010.

[14] 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.

[15] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *Proc. of the Third Biennial Conf. on Innovative Data Systems Research (CIDR)*, 2007.

[16] 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.

[17] R. Kodner, F. A. Matsen, N. Hoffman, C. Berthiaume, and E. V. Armbrust. A fast, phylogenetic based pipeline for shotgun environmental sequences analysis. *In prep*.

[18] S. N. Merbt, D. A. Stahl, E. Casamayor, E. Marta, G. Nicol, , and J. Prosser. Differential photoinhibition of bacterial and archaeal ammonia oxidation. *FEMS Microbiology Letters*, 327:41, 2012.

[19] 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.

[20] A. K. Nowinski, F. Sun, A. White, A. Keefe, and S. Jiang. Sequence, structure, and function of peptide self-assembled monolayers. *Journal of the American Chemical Society*, 134(13):6000–6005, 2012.

[21] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R. D. Clark, S. Yalda, D. A. Reed, E. Joseph, and V. Chandrasekar. CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting. *Computer*, 39(11):56–64, 2006.

[22] K. Ren, Y. Kwon, M. Balazinska, and B. Howe. Hadoop's adolescence: A comparative workload analysis from three research clusters. Technical Report UW-CSE-12-06-01, University of Washington, 2012.

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

[24] A. Szalay and J. Gray. 2020 computing: Science in an exponential world. *Nature*, 440(7083):413, 2006.

[25] S. Tringe, C. von Mering, A. Kobayashi, A. Salamov, K. Chen, H. Chang, M. Podar, J. Short, E. Mathur, J. Detter, P. Bork, P. Hugenholtz, and E. Rubin. Comparative metagenomics of microbial communities. *Science*, 308(5721):554–7, 2005 Apr 22.