Database-as-a-Service for Long-Tail Science

Bill Howe, Garret Cole, Emad Souroush,
Paraschos Koutris, Alicia Key, Nodira Khoussainova, Leilani Battle

University of Washington, Seattle, WA
{billhowe,gbc3,soroush,pkoutris,akey7,nodira,leibatt}@cs.washington.edu

Abstract. Database technology remains underused in science, especially in the long tail — the small labs and individual researchers that collectively produce the majority of scientific output. These researchers increasingly require iterative, ad hoc analysis over ad hoc databases but cannot individually invest in the computational and intellectual infrastructure required for state-of-the-art solutions.
We describe a new "delivery vector" for database technology called SQLShare that de-emphasizes pre-defined schemas and focuses on ad hoc integration, query, sharing, and visualization. To empower non-experts to write complex queries, we generate automatic example queries directly from the data and explore limited English hints to aaugment the process. We integrate iterative, ad hoc, collaborative visualization through a web-based service called VizDeck that uses automatic visualization techniques combined with a card game metaphor to allow creation of interactive visual dashboards in seconds with zero programming.
We present data on the initial uptake and usage of the production system and report preliminary results testing our new features against the datasets collected in the system during the initial pilot deployment. We conclude that the SQLShare system and associated services have the potential to increase uptake of relational database technology in the long tail of science.

1 Introduction

Relational database technology remains remarkably underused in science, especially in the *long tail* — the large number of relatively small labs and individual researchers who, in contrast to "big science" projects [23, 26, 32], do not have access to dedicated IT staff or resources yet collectively produce the bulk of scientific knowledge [5, 25]. The problem persists despite a number of prominent success stories [7, 19, 20] and an intuitive correspondence between exploratory hypothesis testing and the ad hoc query answering that is the "core competency" of an RDBMS. Some ascribe this underuse to a mismatch between scientific data and the models and languages of commercial database systems [9, 18]. Our experience (which we describe throughout this paper) is that standard relational data models and languages can manage and manipulate a significant range of scientific datasets. We find that the key barriers to adoption lie elsewhere:

1. *Setup.* Local deployments of database software require too much knowledge of hardware, networking, security, and OS details.
2. *Schemas.* The initial investment required for database design and loading 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, 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.

3. *SQL.* Although we corroborate earlier findings on the utility of SQL for exploratory scientific Q&A [32], we find that scientists need help writing non-trivial SQL statements.
4. *Sharing.* Databases too often ensconce one's data behind several layers of security, APIs, and applications, which complicate sharing relative to the typical status quo: transmitting flat files.
5. *Visualization.* While SQL is appropriate for assembling tabular results, our collaborators report that continually exporting data for use with external visualization tools makes databases unattractive for exploratory, iterative, and collaborative visual analytics.

As a result of these "S4V" challenges, spreadsheets and ASCII files remain the most popular tools for data management in the long tail. But as data volumes continue to explode, cut-and-paste manipulation of spreadsheets cannot scale, and the relatively cumbersome development cycle of scripts and workflows for ad hoc, iterative data manipulation becomes the bottleneck to scientific discovery and a fundamental barrier to those without programming experience.

Having encountered this problem in multiple domains and at multiple scales at the UW eScience Institute [34], we have released a cloud-based relational data sharing and analysis platform called SQLShare [21] that allows users to upload their data and immediately query it using SQL — no schema design, no reformatting, no DBAs. These queries can be named, associated with metadata, saved as views, and shared with collaborators. Beyond the basic upload, query, share capabilities, we explore techniques to partially automate difficult tasks through the use of statistical methods that exploit the shared corpus of data, saved views, metadata, and usage logs.

In this paper, we present preliminary results using SQLShare as a platform to solve the S4V problems, informed by the following observations:

- We find that cloud platforms drastically reduce the effort required to erect and operate a production-quality database server (Section 2).
- We find that up-front schema design is not only prohibitively difficult, but unnecessary for many analysis tasks and even potentially harmful — the "early binding" to a particular fixed model of the world can cause non-conforming data to be overlooked or rejected. In response, we postpone or ignore up-front schema design, favoring the "natural" schema gleaned from the filenames and column headers in the source files (Section 2).
- We find that when researchers have access to high-quality example queries, pertinent to their own data, they are able to self-train and become productive SQL programmers. The SQL problem then reduces to providing such examples. We address this problem by deriving "starter queries" — automatically — using the statistical properties of the data itself (as opposed to the logs, the schema, or user input that we cannot assume access to.) Moreover, we analyze the corpus of user-provided free-text metadata to exploit correlations between English tokens and SQL idioms (Section 3).
- We find that streamlining the creation and use of views is sufficient to facilitate data sharing and collaborative analysis (Section 2).
- We find the awkward export-import-visualize cycle can be avoided by eager pre-generation of "good" visualizations directly from the data (Section 4).

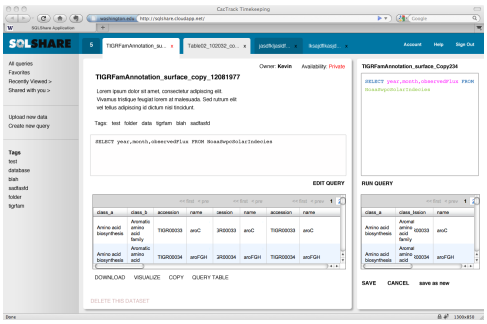


Fig. 1. Screenshot of SQLShare.

Number of uploaded datasets	772
Number of non-trivial views	267
Number of queries executed	3980
Number of users	51
Max (tables, views) for a user	(192,133)
Total size of all data	16.5 GB
Size of largest table, in rows	1.1M

Fig. 2. Early usage for the SQLShare system during a 4-month pilot period. We use data collected during the pilot to perform preliminary evaluation of advanced features.

2 SQLShare: Basic Usage and Architecture

The SQLShare platform is currently implemented as a set of services over a relational database backend; we rely on the scalability and performance of the underlying database. The two database platforms we use currently are Microsoft SQL Azure [29] and Microsoft SQL Server hosted on Amazon's EC2 platform.

The server system consists of a REST API managing access to database and enforcing SQLShare semantics when they differ from conventional databases. The "flagship" SQLShare web client (Figure 1) exercises the API to provide upload, query, sharing, and download services. The following features highlight key differences between SQLShare and a conventional RDBMS.

No Schema We do not allow *CREATE TABLE* statements or any other DDL; tables are created directly from the columns found in the uploaded files. Data is uploaded "as-is," without first defining any kind of schema. Just as a user may place any file on his or her filesystem, we intend for users to put any table into the SQLShare "tablesystem." By identifying patterns in the data (keys, union relationships, join relationships, attribute synonyms) and exposing them to users through views, we can superimpose a form of schema post hoc, incrementally — a *schema later* approach (Section 3).

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 many tasks (e.g., counting records), and that SQL is an adequate language for many data cleaning tasks.

Append-Only We claim that no science data should ever be destructvely updated. We therefore do not support tuple updates most forms of transaction, and we can cache results aggressively at multiple levels. Incorrect data can be logically replaced with dataset-level versioning.

Simplified Views Despite their power, we find views to be underused in practice. We hypothesize that the problem is as simple as avoiding the awkward CREATE VIEW syntax. View creation is a side effect of querying — the current results can be saved by typing a name. This trivial UI adjustment appears sufficient to encourage users to create views (Table 2).

Metadata Users can attach free text metadata to datasets; we use these metadata not only to support basic keyword search, but to bootstrap query recommendations by mining correlations between English tokens and SQL idioms (Section 3).

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. However, since datasets can be versioned, the semantics of views must be well-defined and presented to the user carefully. We find both snapshot semantics and refresh semantics to be relevant, depending on the use case. Currently we support only refresh semantics.

Provenance The DAG of views used to produce a given result is itself useful to communicate a simple form of provenance to collaborators.

3 Automatic Starter Queries

When we first engage a new potential user in our current SQLShare prototype, we ask them to provide us with 1) their data, and 2) a set of questions, in English, for which they need answers. This approach, informed by Jim Gray's "20 questions" requirements-gathering methodology for working with scientists [19], has been remarkably successful. Once the system was seeded with these examples, the scientists were able to use them to derive their own queries and become productive with SQL. The power of examples should not be surprising: Many public databases include a set of example queries as part of their documentation [16, 32]. We adopt the term *starter query* to refer to a *database-specific* example query, as opposed to examples that merely illustrate general SQL syntax. In our initial deployment of SQLShare, starter queries were provided by database experts, usually translated from English questions posed by the researchers. In this section we show preliminary results in generating a set of starter example queries from a set of tables by analyzing their statistical properties only — no schema, no query workload, and no user input is assumed to exist.

Our approach is to 1) define a set of heuristics that characterize "good" example queries, 2) formalize these heuristics into features we can extract from the data, 3) develop algorithms to compute or approximate these features from the data efficiently, 4) use examples of "starter queries" from existing databases to train a model on the relative weights of these features, and 5) evaluate the model on a holdout test set. In this context, we are given just the data itself: In contrast to existing query recommendation approaches, we cannot assume access to a query log [22] , a schema [35], or user preferences [2].

We are exploring heuristics for four operators: union, join, select, and group by. In these preliminary results, we describe our model for joins only. Consider the following heuristics for predicting whether two columns will join: (1) A foreign key between two columns suggests a join. (2) Two columns that have the same active domain but different sizes suggest a 1:N foreign key and a good join candidate. For example, a fact table has a large cardinality and a dimension table has a low cardinality, but the join attribute in each table will have a similar active domain. (3) Two columns with a high similarity suggest a join[1]. (4) If two columns have the same active domain, and that active domain has high entropy (large numbers of distinct values) then there is evidence in favor of a join. Conversely, if both attributes have small entropy, then this is perhaps evidence against a join.

[1] An important exception is the case of an "autoincrement" column that is sometimes used as a key, and may have no relationship to another autoincrement column.

Join heuristics 1-4 above all involve reasoning about the union and intersection of the column values and their active domains, as well as their relative cardinalities. We cannot predict the effectiveness of each of these heuristics a priori, so we train a model on existing datasets to determine the relative influence fo each. For each pair of columns x, y in the database, we extract each feature in Table 1 for both set and bag semantics.

Join card. estimate	$\frac{\|x\|_b \|y\|_b}{max(\|x\|,\|y\|)}$
max/min card.	$max/min(\|x\|, \|y\|)$
card. difference	$abs(\|x\| - \|y\|)$
intersection card.	$\|x \cap y\|$
union card.	$\|x \cup y\|$
Jaccard	$\frac{\|x \cap y\|}{\|x \cup y\|}$

Table 1. Features used to predict joinability of two columns, calculated for set and bag semantics.

	$user_1$	$user_2$	$user_3$	$user_4$	all
(join,join)	5	6	3	2	4
(thaps@phaeo,join)	-	-	1	-	18
(counts,join)	7	7	5	6	13
(flavodoxin,readid)	1	-	-	-	16
(tigr,gene)	7	-	-	4	16
(cog@kog,gene)	7	-	-	6	18

Table 2. Number of occurrences of the top 8 pairs of co-occurring tokens between English and SQL, for 4 users.

Preliminary Results To evaluate whether a particular join candidate should be included in the set of starter queries, ideally, we would have access to a decision tree that can weight these features appropriately for all databases. In these preliminary results, we train the model on the Sloan Digital Sky Survey logs (SDSS) [32], and then evaluate it on the queries collected as examples from the pilot period of SQLShare. The underlying model we use is an Alternating Decision Tree (ADTree) [15] consisting of internal decision nodes and prediction nodes at the root and leaves. To determine the class of a specific instance, we traverse the ADTree and sum the contributions of all paths that evaluate to true. The sign of this sum indicates the class.

For the SDSS database, we have the database, the query logs, and a set of curated example queries created by the database designers to help train users in writing SQL. We use the log to train the model, under the assumption that the joins that appear in the logs will exemplify the characteristics of "good" joins we would want to include in the starter queries.

For the SQLShare dataset, we use *the same model learned on the SDSS data* and see if it can be used to predict the example queries written for users. The key hypothesis is that the relative importance of each of these generic features in determining whether a join will appear in an example query do not vary significantly across schemas and databases. In this experiment, we find that the model classifies 10 out of 13 joins correctly, achieving 86.0% recall. To measure precision, we tested a random set of 33 non-joining pairs. The model classified 33 out of 37 of these candidates correctly, achieving 86.4% precision.

We observe that the model encoded several intuitive and non-intuitive heuristics. For example, the model found, unsurprisingly, that the Jaccard similarity of the active domains of two columns is a good predictor of joinability. But the tree also learned that similar columns with high cardinalities were even more likely to be used in a join. In low-similarity conditions, the model learned that very high numbers of distinct values in one or both tables suggests a join may be appropriate even if the Jaccard similarity is low. Overall, this simple model performed well even on a completly unrelated schema.

To improve this score, we are also exploring how to exploit the English queries provided by users as part of the 20 questions methodology. Natural language interfaces to databases typically require a fixed schema and a very clean training set, neither

of which we have. However, we hypothesize that term co-ocurrence between metadata descriptions of queries and the SQL syntax provides a signal that can be incorporated to our ranking function for starter queries. To test this hypothesis using the full set of queries saved in SQLShare, we first pruned examples with empty or automatically generated descriptions, as well as all descriptions that included the word "test." Second, we tokenized the SQL and English descriptions into both single words and pairs of adjacent words[2]. Finally, we computed the top k pairs of terms using a modified tf-idf measure. The support for these co-occurences appear in the cells of Table 2. We see that some term pairs refer to structural elements of the SQL (`join,join`). This rules may help improve our example queries for all users by helping us prune the search space. Other frequent co-occurring terms are schema dependent, which may help personalize query recommendations, or help determine which science domain is relevant to the user.

4 VizDeck: Semi-Automatic Visualization Dashboards

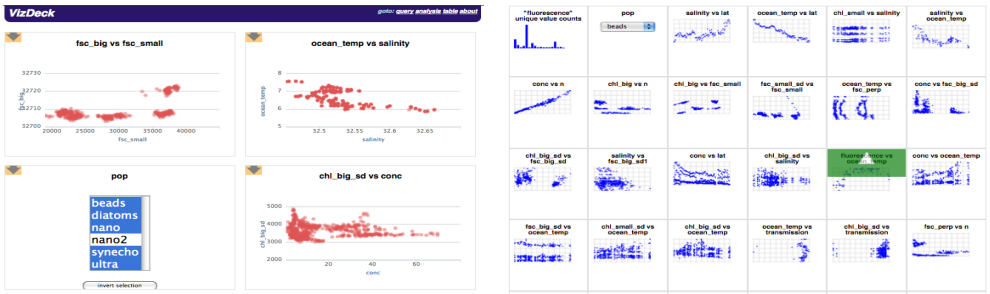
VizDeck is a web-based visualization client for SQLShare that uses a card game metaphor to assist users in creating interactive visual dashboard applications withiout programming. VizDeck generates a "hand" of ranked visualizations and UI widgets, and the user plays these "cards" into a dashboard template, where they are syncronized into a coherent web application. 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.

Figure 4 shows a pair of screenshots from the vizdeck interface. VizDeck operates on datasets retrieved from SQLShare; users issue raw SQL or select from a list of public datasets. After retrieving the data, VizDeck displays 1) a *dashboard canvas* (Figure 3(a)) and 2) a *hand of vizlets* (Figure 3(b)). A vizlet is any interactive visual element — scatter plots and bar charts are vizlets, but drop down boxes are also vizlets.

By interacting with this ranked grid, a user can *promote* a vizlet to the dashboard or *discard* it. Once promoted, a vizlet may be *demoted* back to the grid. As you select thumbnails and work with full-size visualizations, the thumbnails collectively respond to the input. Specifically, highlighting points in one chart highlights the corresponding points in other charts, and establishing a data filter in one chart filters the display in other charts. Multiple filtering widgets are interpreted as a conjunctive expression, while multiple selections in a single widget (as in Figure 3(a)) are interpreted as a disjunction. By promoting visualizations and widgets, simple interactive dashboards are constructed. Crucially, the user can *see* the elements they are adding to the dashboard before they add them — we hypothesize that this "knowledge in the world" [30] will translate into more efficient dashboard construction with less training relative to other sophisticated visualization and mashup systems [33, 24, 11]. The user study to test this hypothesis is planned for future work.

Ranking VizDeck analyzes the results of a query to produce the ranked list of vizlets heuristically. For example, a scatter plot is only sensible for a pair of numeric columns,

[2] In a separate experiment, we evaluated various tokenization methods on a synthetic dataset with respect to model performance.



(a) (b)

Fig. 3. Screenshots from the VizDeck application. (a) A VizDeck dashboard with three scatter plots and one multi-select box for filtering data. (b) A "hand" of vizlets that can be "played" into the dashboard canvas. The green arrow appears when hovering over a vizlet, allowing promotion to the dashboard with a click.

and bar charts with too many bars, or multiple bars of similar height, are difficult to read [27].

As part of ongoing work, we are incorporating user input into the ranking function. We interpret each promote action as a "vote" for that vizlet, and each discard action as a vote against that vizlet, then assign each vizlet a score and train a linear model to predict this score from an appropriate feature vector. We are currently collecting data to evaluate our preliminary model.

Preliminary Results A potential concern about our approach is that eagerly generating a large number of possible visualizations is prohibitively expensive. A related concern is that the number of potential vizlets is either too large (as to be overwhelming) or too small (making a ranking-based approach unnecessary). To test the feasibility of this approach, we ran VizDeck for all public datasets registered in the SQLShare system.

Figure 4(a) shows the elapsed time to both retrieve data and generate (but not render) the vizlets for each public query, where each query is assigned a number. Most queries (over 70%) return in less than one second, and 92% return in less than 8.6 seconds, the target response time for page loads on the web [8]. We have not included rendering time in the browser to avoid network and browser effects. Figure 4(b) shows the number of vizlets generated by each query. Most of the queries (73 out of 118) generated a pageful of vizlets or less (30 vizlets fit on a page at typical resolutions). Of the remaining 45 queries, most (29) return more than 90 vizlets, suggesting that our ranking-based approach is warranted.

5 Related Work

Database-as-a-service platforms target a different set of requirements (enormous scale, limited query capabilities) that make them unsuitable for long tail science [4, 29, 3]. Google Fusion Tables shares a very similar motivation with our work, but does not support SQL.

The VizDeck system builds on seminal work on automatic visualization of relational data using heuristics related to visual perception and presentation conventions [27, 31]. More recent work on intelligent user interfaces attempts to infer the user's task from

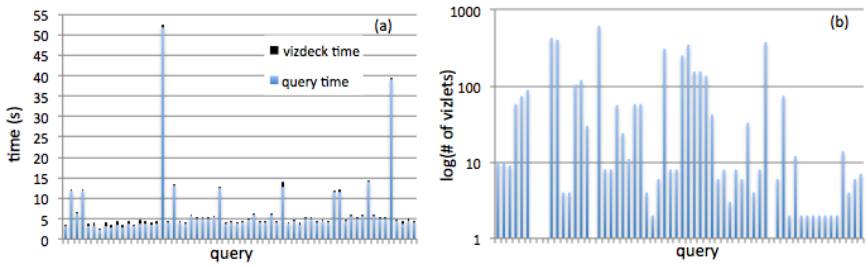


Fig. 4. Two experiments using VizDeck with the public queries in the SQLShare system. At left, we see the elapsed time for executing the query and generating (but not rendering) the vizlets. The total time is dominated by query execution rather than VizDeck analysis. At right, we see the number of vizlets generated or each query.

behavior and use the information to recommend visuaizations [17]. Dork et al. derive coordinated visualizations from web-based data sources [11]. Mashup models have been studied in the database community [13, 12, 1], but do not consider visualization ensembles and assume an organized repository of mashup components.

Query recommendation systems proposed in the literature rely on information that we cannot assume access to in an ad hoc database scenario: a query log [22], a well-defined schema [35], or user history and preferences [2]. The concept of dataspaces [14] is relevant to our work; we consider SQLShare an example of a (relational) Dataspace Support Platform. The Octopus project [10] provides a tool to integrate ad hoc data extracted from the web, but does not attempt to derive SQL queries from the data itself. The generation of starter queries is related to work on schema mapping and matching [28, 6]: both problems involve measuring the similarity of columns.

References

1. S. Abiteboul, O. Greenshpan, T. Milo, and N. Polyzotis. Matchup: Autocompletion for mashups. In *ICDE*, pages 1479–1482, 2009.
2. J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. Sql querie recommendations. *PVLDB*, 3(2):1597–1600, 2010.
3. Amazon Relational Database Service (RDS). `http://www.amazon.com/rds/`.
4. Amazon SimpleDB. `http://www.amazon.com/simpledb/`.
5. C. Anderson. The long tail. *Wired*, 12(10), 2004.
6. P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.
7. B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, January 2003.
8. A. Bouch, A. Kuchinsky, and N. Bhatti. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 297–304, New York, NY, USA, 2000. ACM.
9. 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.

10. M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1), 2009.
11. M. Dörk, S. Carpendale, C. Collins, and C. Williamson. Visgets: Coordinated visualizations for web-based information exploration and discovery. *IEEE Transactions on Visualization and Computer Graphics*, 14:1205–1212, November 2008.
12. H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin. Mashup advisor: A recommendation tool for mashup development. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pages 337–344, Washington, DC, USA, 2008. IEEE Computer Society.
13. R. J. Ennals and M. N. Garofalakis. Mashmaker: mashups for the masses. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1116–1118, New York, NY, USA, 2007. ACM.
14. M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *SIGMOD Record*, 34(4), December 2005.
15. Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *International Conference on Machine Learning*, 1999.
16. Gene ontology. http://www.geneontology.org/.
17. D. Gotz and Z. Wen. Behavior-driven visualization recommendation. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '09, pages 315–324, New York, NY, USA, 2009. ACM.
18. M. Graves, E. R. Bergeman, and C. B. Lawrence. Graph database systems for genomics. *IEEE Eng. Medicine Biol. Special issue on Managing Data for the Human Genome Project*, 11(6), 1995.
19. 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.
20. G. Heber and J. Gray. Supporting finite element analysis with a relational database backend; part 1: There is life beyond files. Technical report, Microsoft MSR-TR-2005-49, April 2005.
21. B. Howe. Sqlshare: Database-as-a-service for long tail science. http://escience.washington.edu/sqlshare.
22. N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: A context-aware sql autocomplete system. In *Proc. of the 37th VLDB Conf.*, 2011.
23. Large Hadron Collider (LHC). `http://lhc.web.cern.ch`.
24. J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '09, pages 97–106, New York, NY, USA, 2009. ACM.
25. Big science and long-tail science. `http://wwmm.ch.cam.ac.uk/blogs/murrayrust/?p=938`. term attributed to Jim Downing.
26. Large Synoptic Survey Telescope. `http://www.lsst.org/`.
27. J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5:110–141, 1986.
28. J. Madhavan, P. A. Bernstein, and E. Rahm. "generic schema matching with cupid. In *VLDB*, 2001.
29. Microsoft SQL Azure. `http://www.microsoft.com/windowsazure/sqlazure/`.
30. D. Norman. *The design of everyday things*. Doubleday, New York, 1990.
31. S. F. Roth and J. Mattis. Data characterization for intelligent graphics presentation. In *In Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '90)*, pages 193–200. ACM Press, 1990.
32. Sloan Digital Sky Survey. `http://cas.sdss.org`.
33. Tableau. `http://www.tableausoftware.com/`.
34. University of washington eScience institute. `http://escience.washington.edu/`.
35. a. D. X.Yang, C.M.Procopiuc. Summarizing relational databases. *Proc. VLDB Endowment*, 2(1):634£645, 2009.