SQB: Session-based Query Browsing for More Effective Query Reuse

Nodira Khoussainova, YongChul Kwon, Wei-Ting Liao,
Magdalena Balazinska, Wolfgang Gatterbauer, and Dan Suciu

Department of Computer Science and Engineering
University of Washington, Seattle, WA, USA
{nodira, yongchul, liaowt, magda, gatter, suciu}@cs.washington.edu

Abstract. Scientists are generating more data than ever before and are increasingly putting that data (or corresponding metadata) inside a relational database accessible through SQL. Composing non-trivial SQL queries, however, is difficult. Just like code reuse, query reuse can facilitate and accelerate the task of composing new queries. A key challenge is how best to facilitate this reuse. In this paper, we introduce Smart Query Browser (SQB), a tool for searching and browsing through a repository of SQL queries. SQB applies the idea of a *query session* to improve query browsing and reuse. Informally, a query session is a set of queries authored by a single user to achieve a specific information goal. To evaluate SQB, we conduct a user study with 16 users and find that SQB reduces the query composition time by 57% compared with a traditional, unassisted query composition setting.

1 Introduction

Scientists today are able to generate data at an unprecedented scale and rate [1]. Many scientific communities are taking this opportunity to create large-scale, high-quality, curated databases and make them publicly available. Some examples include the Sloan Digital Sky Survey (SDSS) [23], the Incorporated Research Institutions for Seismology (IRIS) [11], and the upcoming Large Synoptic Survey Telescope (LSST) [15]. Because these databases are extremely large, it is impractical and often even impossible for individual scientists to download them locally. Instead, these community datasets are typically stored in relational databases and made available over the Web through powerful query interfaces [26, 24]. Formulating non-trivial SQL queries, however, is a significant challenge for scientists because most are not database experts.

One approach to ease this problem is to offer form-based interfaces, where a user fills out some fields in a form and the application composes the SQL query to execute against the database. Such interfaces, however, significantly constrain how users can access the data and thus several scientific repositories enable users to write their own SQL queries in addition to using forms [26]. To leverage such a feature, scientists need the ability to write their own SQL queries.

Scientists are technically sophisticated users and can easily grasp simple SELECT-FROM-WHERE queries. To overcome the challenge of writing more complex SQL queries, users often rely on examples found in books, in online tutorials, or obtained from collaborators. Queries from database books and SQL tutorials illustrate generic SQL constructs while queries from collaborators show how to query the database of interest. Furthermore, several large databases (e.g., SDSS) offer a small set of sample queries that can serve as example and be modified to fit a user's specific needs.

Sample sets of queries and queries from colleagues can be helpful, but they offer only a limited set of examples. A method for giving users access to a larger number of sample queries would be to use a query browser system: a tool that would allow users to browse through all past queries executed over a database of interest. With access to such a query repository, users would have access to a larger number of potentially useful sample queries. Browsing through past queries, however, can be overwhelming. For example, the SDSS's query log contains more than 106 M queries [13]. Additionally, many queries in a query log can be of low-quality as previous users submitted multiple wrong queries before figuring out how to specify the query that they needed. To be effective, a query browser must thus *(1) allow users to efficiently locate relevant queries and (2) distinguish between high-quality and low-quality queries*.

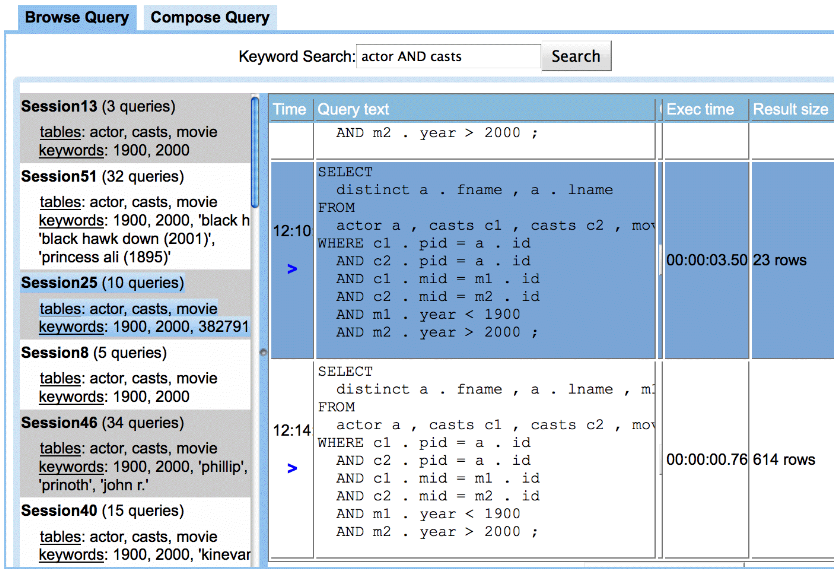
In this paper, we present the *Smart Query Browser* (SQB) system. SQB supports efficient retrieval of relevant queries using what we call *session-based browsing*. SQB enables a user to perform a keyword search over a query log. Instead of simply listing all matching queries, it presents the results as a *set of query sessions*. A query session, as introduced in our previous work [13], is a set of queries written by a user to achieve a single information goal. SQB thus allows users to view each result query in the context of the task that it aimed to complete. With this approach, SQB helps the user to more rapidly identify relevant queries because the user can decide on the relevance of entire sessions. It also helps users see how simple queries evolved into more complex ones. Finally, query sessions enable users to discriminate between high-quality and low-quality queries: queries that appear near the end of a session tend to be of higher quality because the author has spent time to edit and improve the query [13].

In this paper, we present the SQB system and show the results of its evaluation through a user study with 16 participants. The goal of the study is to investigate whether SQB speeds up the query formulation process by better supporting query reuse. We compare SQB to query formulation with no browser (traditional method) and to a straw-man query browser that provides keyword search over the query log but no notion of query sessions. We find that, on average, SQB allows users to complete their tasks 2.3 times faster than the traditional method for writing queries, and 1.3 times faster than with a basic query browser without sessions.

2 SQB Overview

We present an overview of SQB's design and implementation. In particular, we describe how SQB utilizes query sessions in order to improve the query browsing experience.

Query sessions. Intuitively, a query session is a set of queries that a single user wrote to achieve a specific information goal. For example, an astronomer may want to find in the SDSS database all the stars of a certain brightness in the r-band within 2



Browse Query | Compose Query

Keyword Search: actor AND casts Search

Session13 (3 queries)
tables: actor, casts, movie
keywords: 1900, 2000

Session51 (32 queries)
tables: actor, casts, movie
keywords: 1900, 2000, 'black h
'black hawk down (2001)',
'princess ali (1895)'

Session25 (10 queries)
tables: actor, casts, movie
keywords: 1900, 2000, 382791

Session8 (5 queries)
tables: actor, casts, movie
keywords: 1900, 2000

Session46 (34 queries)
tables: actor, casts, movie
keywords: 1900, 2000, 'phillip',
'prinoth', 'john r.'

Session40 (15 queries)
tables: actor, casts, movie
keywords: 1900, 2000, 'kinevar

Time | Query text | Exec time | Result size

```
  AND m2 . year > 2000 ;
```

12:10 — 00:00:03.50 — 23 rows

```
SELECT
  distinct a . fname , a . lname
FROM
  actor a , casts c1 , casts c2 , mov
WHERE c1 . pid = a . id
  AND c2 . pid = a . id
  AND c1 . mid = m1 . id
  AND c2 . mid = m2 . id
  AND m1 . year < 1900
  AND m2 . year > 2000 ;
```

12:14 — 00:00:00.76 — 614 rows

```
SELECT
  distinct a . fname , a . lname , m1
FROM
  actor a , casts c1 , casts c2 , mov
WHERE c1 . pid = a . id
  AND c2 . pid = a . id
  AND c1 . mid = m1 . id
  AND c2 . mid = m2 . id
  AND m1 . year < 1900
  AND m2 . year > 2000 ;
```

Fig. 1. Screenshot of SQB. User searches for queries written over the Internet Movie Database (IMDB) containing `actor` and `casts`. SQB returns several query sessions. e.g., the first session consists of 3 queries, which reference the tables `actor`, `casts`, and `movie`. The keywords `1900` and `2000` are constants that appear in the queries in the session.

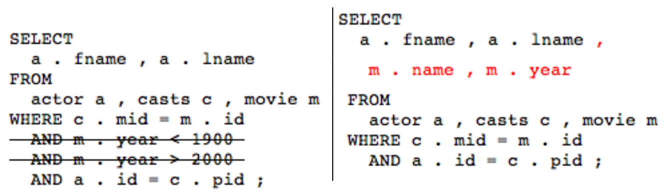
arc minutes (i.e., $\frac{1}{30}$th of $1°$) of a known star [13]. The user will likely need to write multiple SQL queries before reaching this goal.

To define a query session more formally, we first introduce the notion of a query *deriving* from another query. We say a query Q_2 derives from a query Q_1, if it occurs later in the query log and the user directly formulated query Q_2 by modifying Q_1 while working toward the same information goal. For example, the astronomer may first write a query that finds all stars with a certain brightness in the r-band (Q_1) and then add a predicate on distance to the known star (Q_2). Usually, a query Q_2 is derived from a single query Q_1, but can also be derived from multiple queries if, for example, the user merges together several queries to create Q_2 (perhaps using one of the earlier queries as a subquery in the other).

A *query derivation graph* is a directed acyclic graph constructed from a query log. Each node represents a logged query, and there is an edge from Q_1 to Q_2 if and only if Q_2 derives from Q_1. A *query session is a connected component of the query derivation graph*. If constructed correctly, there is exactly one query session per information goal.

Extracting query sessions. SQB is layered on top of a relational database management system (DBMS). SQB first uses the underlying DBMS's query logging functionality to accumulate a query log. Next, it executes the session extraction algorithm [13], which constructs the *query derivation graph* over the query log, and partitions that graph into query sessions. This part is executed offline.

Utilizing query sessions. We outline a typical user interaction with the SQB system, to demonstrate the different ways in which SQB leverages query sessions. Figure 1



```
SELECT
  a . fname , a . lname
FROM
  actor a , casts c , movie m
WHERE c . mid = m . id
~~  AND m . year < 1900~~
~~  AND m . year > 2000~~
  AND a . id = c . pid ;
```

```
SELECT
  a . fname , a . lname ,
  m . name , m . year
FROM
  actor a , casts c , movie m
WHERE c . mid = m . id
  AND a . id = c . pid ;
```

Fig. 2. SQB automatically computes and displays minimal edits between two queries.

Problem	1	2	3	4	5	6	7	8	9
Avg. # of SQL queries	7	2	11	7	27	39	15	23	18
Stdev. # of SQL queries	7.1	1.8	9.2	7.4	31.6	27.7	15.0	26.8	13.0

Fig. 3. Average and standard deviation of the number of queries per problem per student.

shows a screenshot of SQB used in the context of composing a query over the Internet Movie Database (IMDB), which is the scenario that we use later in the evaluation. To begin, the user types in keywords to search for queries (e.g., ‘actor AND casts’ in the screenshot in Figure 1). SQB then uses the full text search and ranking features offered by the underlying DBMS to identify relevant queries. For each query returned by the DBMS, SQB displays the corresponding session. If multiple queries belong to the same session, SQB shows that session only once. In order to rank sessions, SQB assigns each session the rank of its highest-ranked query. Within a session, all matching queries are marked with a blue arrow, and SQB brings to focus first the query that appears latest in the session. Because the queries are grouped according to their semantic information goal, the user needs to read through fewer results. Furthermore, SQB allows easy access to the last query in the session, which tends to be the one of highest quality.

To assist the user to quickly evaluate whether a session is potentially useful, SQB attaches a brief summary to each session. The summary includes the number of queries, the referenced tables, and the top five keywords in the session. The summaries are presented with the list of sessions, on the left panel of the GUI (Figure 1).

Once the user has selected a session to investigate, SQB lists the queries in execution timestamp order, with their execution duration and result size. Viewing a session at a time allows the user to easily see how queries evolved, so that she can go back to a simpler query or forward to a more complex one. To allow the user to more easily compare queries, SQB also computes and displays the minimal edits to transform one query to the other. Figure 2 shows an example.

In the following section, we investigate whether session-based browsing is indeed helpful in query formulation.

3 Evaluation

The evaluation dataset consists of all SQL queries written by students in an undergraduate database class, offered at the University of Washington in 2008. These queries were automatically logged as students worked on nine different problems for an assignment that used the IMDB database [5]. IMDB contains information about movies, actors, directors, and the connections between them. In total, our dataset contains 7294 queries, written by 52 students. Problem numbers give us ground truth for the query sessions. Figure 3 shows the overall distribution of the number of queries per problem and per student. For the user studies, we used a subset of these queries (492 queries).

We evaluate the benefits of SQB for query reuse through a user study. Overall, we find that for trivial queries, query browsing is not helpful and can be distracting. However, complex queries greatly benefit from query browsing.

Tasks: Participants were asked to write four SQL queries over the IMDB database (see [5] for the schema): the participant was given an English sentence and was asked to translate the sentence into a SQL query. We present the queries below.

Participants: SQB is designed to help novice to intermediate-level SQL users. Therefore, our study participants are graduate students who have either taken an introductory database class or learned SQL through their research. We had a total of 16 participants. *Each participant used one interface, and worked on all four tasks.*

Interfaces: We compare four variants of the SQB interface. The first is the $Compose$ interface, where participants have access to SQB's Compose Query tab (see Figure 1) but no browsing capabilities. It represents the current method for writing SQL queries without assistance. In the second, $Semantic$ interface, participants have access to the SQB tool with full capabilities. All queries are grouped into query sessions. The third and fourth interfaces are called the $Basic$ and $Syntactic$ interfaces, respectively. In these modes, the participants have access to the repository of queries. The queries are either not grouped at all ($Basic$ mode), or grouped according to the set of tables that the queries touched ($Syntactic$ mode). The goal of the $Syntactic$ mode is to compare session-based grouping to another form of query grouping.

Study Design: Each participant session was approximately one hour. First, the participant was presented with three examples of English sentences alongside their SQL translations. Next, the participant completed two practice tasks on paper. This exercise served two purposes: (1) to check that the participant had enough SQL knowledge, and (2) to verify that the groups had similar SQL competency, on average. The preparation queries were over a different database than the IMDB database. The table below shows the total completion times of the preparation exercises for each group.

Group	**Average Time**	**Min Time**	**Max Time**
$Compose$	4 m 8 s	3 m 29 s	5 m 33 s
$Semantic$	4 m 25 s	3 m 40 s	5 m 40 s
$Basic$	4 m	3 m 10 s	4 m 55 s
$Syntactic$	4 m 23 s	3 m 24 s	5 m 50 s

We see that the groups are comparable, with participants in the $Semantic$ group being slowest on average.

Following the preparation tasks, we gave each participant a tutorial of the interface they would be using and then let them work on the four SQL tasks on their own.

Study Results. To evaluate the interfaces, we compare the completion times for the SQL tasks. If a participant took longer than 30 min, or did not complete the task, we round down their time to 30 min. There were three such instances. One in the $Compose$ group for task three, and two in the $Syntactic$ group for task four. Also, since we have

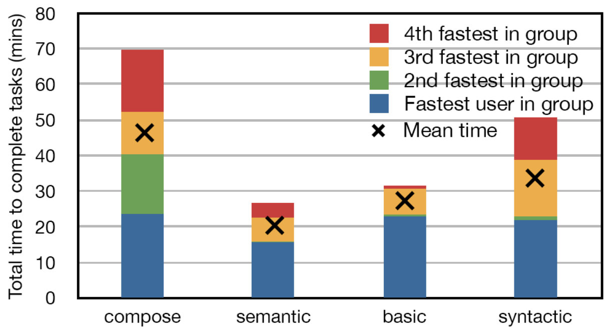


Fig. 4. Total completion time per user, grouped by interface.

only four users per interface, we include a participant's timing information even if their query is incorrect. There were a total of five mistakes; three in *Compose* and two in *Syntactic*. Most were minor mistakes (i.e., missing or mistyped predicate).

Figure 4 shows the total time to complete all four tasks for all sixteen participants. There is one stacked bar per interface group, showing the completion time of each participant in the group. For example, it took the second fastest person in the *Compose* group about forty minutes to complete all tasks. The second fastest person in the *Semantic* group, and the second fastest in the *Basic* group both took almost the same time as the fastest person within the respective group. Therefore, that stripe is hardly visible for those groups.

Looking at the left two bars of Figure 4, we see that the average time to complete all tasks with the *Semantic* interface is 2.3 times faster than the traditional interface for writing queries, *Compose*. This is a reduction from 46.4 minutes down to 20.1 minutes. Compared to the *Basic* and *Syntactic* interfaces, we see that SQB offers a 1.3 factor speed up, and a 1.7 factor speed up, respectively. Next, we examine Figure 5, which presents the average completion time per task.

Task 1: *Find all the 'Sci-Fi' movies that were released in a leap year.* As can be seen in Figure 5, this task is the easiest of the four. It requires a simple join, and no `GROUP BY` clause. Due to its simplicity, the participants in the *Compose* group do best because query browsing provides more of a distraction than help. This task shows that for trivial queries, which do not have exact matches within the repository, query browsing is not helpful.

Task 2: *List all directors who directed 200 movies or more, in ascending order of the number of movies they directed.* For this task, we begin to see significant benefits of query browsing. For example, the average completion time in the *Semantic* group is just under four minutes versus the fourteen minutes average for the *Compose* interface. This task leverages the query repository because there is a query that differs in the constant and `ORDER BY` clause. From this task, we learn that complex queries with good matches in the repository benefit greatly from query browsing.

Task 3: *List all the actors who acted in a film before 1950 and also in a film after 2005.* Again, this task demonstrates that the SQB tool offers great benefits. The most

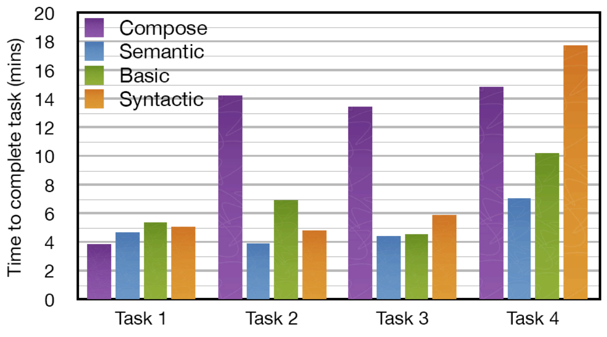


Fig. 5. Mean task completion time per interface, grouped by task.

similar query in the repository to this task only differs in constants. As expected, the participants in the *Semantic* group, on average, completed this task 3.6 times faster than the users in the *Compose* group.

The graph shows that the *Semantic* interface performed only marginally better than the *Basic* one. This was due to one outlier in the *Semantic* group; the slowest participant in that group took more than twice the time of the second slowest participant. If we were to remove the slowest person from the two groups, then the average completion time for the *Semantic* group would be 1.3 times faster than the *Basic* group.

A significant challenge that participants faced for this task was in realizing that you need two occurrences of the `ACTOR` and `CASTS` tables. Even those using the *Basic* and *Syntactic* versions of browser faced this problem because many of them chose to pick the query that was simpler (i.e. had fewer tables in the `FROM` clause. Those using the *Semantic* version of the tool did not face this problem, because they were able to see that in the remainder of the query session, this simpler (and incorrect) query was later replaced by a slightly longer but correct version.

Once again, we thus see that complex queries with good matches in the repository benefit greatly from session-based query browsing.

Task 4: *Find the actor with the most number of movies.* This task was the most difficult, despite the fact that we gave every participant a hint about the `TOP` keyword (SQL Server's version of the `LIMIT` keyword). The closest query in the repository found the *movie* with the most number of actors. Participants were expected to find this query and restructure it heavily. Ten out of twelve users working with a browser found it. One user from the *Basic*, and one from the *Syntactic* group did not find the query.

For this task, the *Semantic* group did 1.5, 2.1, and 2.5 times faster than the *Basic*, *Compose*, and *Syntactic* groups, respectively. It beat *Compose* because the query is a difficult one, and the *Semantic* group had available a good candidate query whose structure they could reuse. *Semantic* beat the other two browsing interfaces because in the *Semantic* interface users more quickly identified that the query they found was a good one. Whereas in the other two interfaces, the users continued to search for more queries even after they found the best one. In the *Semantic* group, participants used keyword search an average of 2 times for this task. For the *Basic*, and *Syntactic* groups, users had an average of 2.5, and 5.25 searches, respectively .

We see that for a complex task, even if the best query matches the task only structurally, SQB is still able to help users significantly. It beats the $Basic$ and $Syntactic$ versions because it helps users to more quickly evaluate the quality of a query.

4 Related Work

Web-based tools. If we relax the definition of a query, there are several tools that support query sharing on the web. CoScripter [4] is a web-based system for recording, automating and sharing web browser tasks (e.g., checking flight departure times). Yahoo! Pipes [28], Microsoft Popfly [16], Google Mashup Editor [8], and IBM Lotus Mashups [9] enable users to construct mashups of data from different sources on the web, and later share these mashups with the public. Woogle [7] is a tool for locating web services. Given a web service, Woogle finds similar ones based on the services' input and output schemas, as well as web services that compose with the specified one.

Commercial DBMS. Commercial tools currently provide query-by-example [19, 29], graphical tools for composing queries [2, 3], and query logging aimed at physical tuning [17, 25, 27]. None provide a query browsing tool aimed at helping user easily reuse, and learn from past queries.

Scientific workflows. There are several existing scientific workflow management systems. The MyExperiment project [20] envisions community oriented sharing of scientific workflows. The project is more focused on the social aspects of sharing rather than the automatic curation or browsing of activities within a system. Scriptome [22] is a repository of useful Perl scripts for computational biologists. It is well-curated and allows users to customize the scripts in the repository in predefined ways. Unlike Scriptome, SQB aims to minimize the level of human effort to curate such repositories. In a recent VisTrails paper, the authors demonstrate advanced capabilities of the system such as query-by-example and query-by-analogy with visualizations [21]. SQB is a first step toward such advanced capabilities for SQL queries.

Making databases usable. Jagadish *et al.* [12] discuss the pain points of using database systems today, and describe several usability challenges that database researchers should address. Nandi *et al.* [18] and Li *et al.* [14] present different techniques for phrase auto-completion on keyword searches over structured databases. SQB also strives to make databases more usable, however, we are specifically focused on providing users with capabilities that allow them to share, reuse, learn from the past queries. To allow users to more easily understand SQL queries, Ioannidis *et al.* [10] explore how to automatically translate SQL queries to natural language sentences. Danaparamita and Gatterbauer [6] also investigate this problem, but through the translation of the SQL query into a visual representation. Both projects are complementary to our work, and we hope to extend SQB with this capability in future work.

5 Conclusion

We presented SQB, a tool for browsing through past SQL queries. The key insight behind SQB's design is the concept of query sessions. We showed that query sessions help speed up query composition by organizing queries in a large repository in a manner that facilitates the identification of relevant, high-quality queries to use as example.

References

1. The fourth paradigm: Data-intensive scientific discovery, 2009.
2. BaseNow. Database front-end applications. About SQL Query Builder. `http://www.basenow.com/help/About_SQL_Query_Builder.asp`.
3. T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *J. of Visual Languages & Computing*, 8(2):215–260, 1997.
4. Coscripter. `http://coscripter.research.ibm.com/`.
5. IMDB course assignment. `http://www.cs.washington.edu/education/courses/cse444/08au/project/project1/project1.html`.
6. J. Danaparamita and W. Gatterbauer. Queryviz: helping users understand sql queries and their patterns. In *Proc. EDBT*, 2011.
7. X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proc. VLDB*, pages 372–383, 2004.
8. Google Mashup Editor. `http://code.google.com/gme/`.
9. IBM Lotus Mashups. `http://www.ibm.com/software/lotus/products/mashups/`.
10. Y. Ioannidis. From databases to natural language: The unusual direction. In *Proc. NLDB*, pages 12–16, 2008.
11. Incorporated Research Institutions for Seismology. `http://www.iris.edu`.
12. H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *Proc. SIGMOD*, pages 13–24, 2007.
13. N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: context-aware autocompletion for sql. *Proc. VLDB Endow.*, 4:22–33, October 2010.
14. G. Li, J. Fan, H. Wu, J. Wang, and J. Feng. DBease: Making databases user-friendly and easily accessible. In *Proc. of the 5th CIDR Conf.*, 2011.
15. Large Synoptic Survey Telescope. `http://www.lsst.org/`.
16. Microsoft Popfly. `http://www.popfly.com/`.
17. Microsoft TechNet. SQL Server TechCenter: Configuring the analysis services query log. `http://www.microsoft.com/technet/prodtechnol/sql/2005/technologies/config_ssas_querylog.mspx`.
18. A. Nandi and H. V. Jagadish. Effective phrase prediction. In *Proc. VLDB*, pages 219–230, 2007.
19. Reading objects using query by example. Oracle TopLink developer's guide 10g release 3 (10.1.3). `http://www.oracle.com/technology/products/ias/toplink/doc/1013/main/_html/qrybas002.htm`.
20. D. D. Roure, C. Goble, J. Bhagat, D. Cruickshank, A. Goderis, D. Michaelides, and D. Newman. myExperiment: Defining the social virtual research environment. *eScience, IEEE International Conference on*, 0:182–189, 2008.
21. C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and re-using workflows with vstrails. In *Proc. SIGMOD*, pages 1251–1254, 2008.
22. The Scriptome - Protocols for Manipulating Biological Data. `http://sysbio.harvard.edu/csb/resources/computational/scrptome/`.
23. Sloan Digital Sky Survey. `http://www.sdss.org/`.
24. IRIS SeismiQuery. `http://www.iris.edu/dms/sq.htm`.
25. Siebel business analytics server administration guide: Administering the query log. `http://download.oracle.com/docs/cd/E12103_01/books/admintool/admintool_AdministerQuery14.html`.

26. Sloan Digital Sky Survey/SkyServer. `http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp`.
27. Teradata Utility Pack. `http://www.teradata.com/t/page/94310/`.
28. Yahoo Pipes. `http://pipes.yahoo.com/pipes/`.
29. M. M. Zloof. Query-by-example: the invocation and definition of tables and forms. In *Proc. VLDB*, pages 1–24, 1975.