# Automatic Example Queries for Ad Hoc Databases

Bill Howe Garret Cole Nodira Khoussainova Leilani Battle

University of Washington
{billhowe,gbc3,nodira,leibatt}@cs.washington.edu

## 1. INTRODUCTION

We explore automatic generation of example queries from an *ad hoc database*. An ad hoc database is a collection of tables with unknown relationships gathered to serve a specific, often transient, often urgent, purpose. Consider these examples:

- A researcher assembles an ad hoc database of recent experimental results to prepare a paper or proposal.
- Emergency workers responding to a natural disaster assemble an ad hoc data-base from lists of addresses of nearby schools, locations of resources (e.g., ambulances), and contact information for emergnecy workers.
- A consulting business analyst assembles an ad hoc database from a set of spreadsheets provided by management for a short term engagement

Analysts who assemble ad hoc databases frequently do not have significant SQL expertise, but we find that by providing a rich set of examples is sufficient to empower non-experts to use SQL for data analysis [8]. This finding should not be surprising: Many public databases include a set of example queries as part of their documentation [6, 11], suggesting that the strategy is effective. We adopt the term *starter query* to refer to a database-specific example query, to distinguish them from examples that merely illustrate SQL syntax abstractly.

Analysts use starter queries in several ways: They *browse* them to learn basic idioms of SQL (joins, in particular, are often frequently unfamiliar). They *execute* them to explore the data itself. They *modify* them by adding or removing *snippets* [9]: predicates in the where clause, tables in the from clause, columns in the select clause. They *compose* them to derive new queries — each saved query is automatically registered as a view and is referenceable as a table. They *share* them during collaboration — each query (and its result) has a unique url that can be bookmarked or emailed to colleagues, who can then add comments, derive their own queries, etc. Our preliminary results suggest that this query-oriented collaborative analysis is an effective model for working with ad hoc databases. To bootstrap this model, we need only "seed" the collaboration with a set of starter queries. In our existing system, these examples queries are provided by database experts. In this demonstration, we show how a set of starter example queries can be derived from a set of tables just be analyzing their statistical properties — no schema, no query workload, and no user input is assumed to exist. We will also demonstrate how this capability fits into the larger context of an Ad Hoc Database Management System called SQLShare [8].


*SIGMOD 2011* Athens, Greece


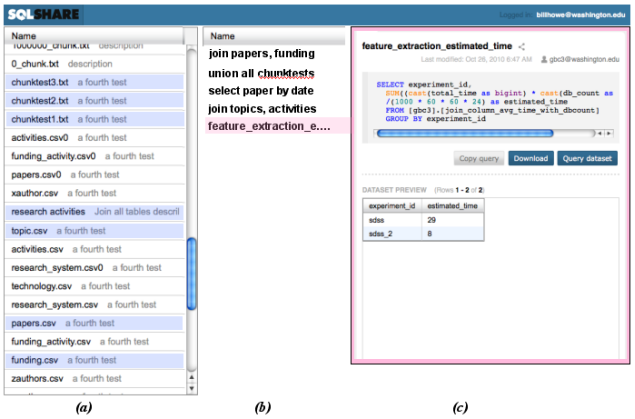


**Figure 1: Demonstration interface for exploring how different ad hoc databases lead to different sets of starter queries. (a) The user selects the tables to act as the ad hoc database. (b) A list of starter queries is dynamically updated based on the user's selection of tables. (c) The user can view, modify, and execute the query definition.**

## 2. WHAT WILL THE USER SEE AND DO?

Figure 1 illustrates the interface that users will interact with in this demonstration. In the left-hand column, the user will highlight a subset of the available tables in the system. Each table is a link to the In the middle column, the starter queries derived from the selected tables are displayed. The user can select a query and view, modify, and execute the query definition. Finally, the user can view the decision tree that led to this query being recommended, with the relevant path highlighted.

We allow the user to select different sets of tables to 1) simulate the assembly of an ad hoc database and 2) illustrate the relationship between the database extent and set of generated starter queries. For example, a bioogist we work with uploaded three tables: $experimentA$, $experimentB$, and $metadata$, where $experimentB$ subsumes $experimentA$ and both experiment tables joined with $metadata$. In our approach, selecting $experimentA$ and $metadata$ as the basis for an ad hoc database will, in general, produce a different set of starter queries than selecting $experimentB$ and $metadata$. Further, selecting both $experimentA$ as well as $experimentB$ — corresponding to the common occurrence where an ad hoc database includes redundant data — may produce a starter query expressing the union of the two datasets, having observed that they share the same schema and verlap significantly. This exploration of the influence on the starter queries from the *data* — as opposed to query logs, the schema, or user input — is the key mechanic of our

demonstration and the hallmark of our approach.

The starter queries are generated on the fly based on the user's selection, but the statistical features that inform the decision will be pre-computed. For example, the likelihood of a join on $X.x = Y.y$ appearing in a starter query depends on the cardinality of the intersection between $X.x$ and $Y.y$, among other things. This cardinality and other relevant features will already have been extracted to ensure interactivity.

# 3. MOTIVATION

Conventional relational database management software remains underused for ad hoc databases, especially in the *long tail* of science: the large number of relatively small labs and individual researchers who, in contrast to "big science" projects, have limited IT funding, staff, and infrastructure yet collectively produce the bulk of scientific knowledge. Despite prominent success stories of scientific databases [7] and an intuitive correspondence between exploratory hypothesis testing and database query answering, scientists frequently over-rely on file-oriented tools such as spreadsheets. However, as rate of data acquisition continues to increase, these tools cannot keep pace. Indeed: in an informal survey, researchers report that the ratio of time spent "manipulating data" as opposed to "doing science" is approaching 9 to 1!

Having encountered this problem in multiple science domains and at multiple scales, we find that although the relational model (tables) and de facto language (SQL) are perfectly adequate for many scientists' needs, the costs associated with deployment and use of database software are prohibitive. In particular, the need to pre-define a schema before any data can be loaded or any queries issued is infeasible in this context. A database schema represents a shared consensus about the domain being modeled. But at the frontier of research, such consensus does not exist, by definition ("if it were well-understood, it wouldn't be research"). Further, 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 — these issues complicate schema design, even for an expert.

As an alternative, we advocate explicit support for ad hoc databases, where tabular data can be loaded as is, under whatever schema can be inferred from the column names and data types present in the source file. These tables can be queried directly, and the queries saved as views. This incremental view creation and reuse facilitates pay-as-you-go integration, cleaning, analysis, and schema-building [4].

This approach relies on another key finding: the conventional wisdom that states "scientists won't write SQL" is simply false, corroborating the findings of the Sloan Digital Sky Survey [11, 7] Our experience is that scientists can and will write even very complex queries, *provided they are given sufficient relevant examples from which to build.*

Guided by these premises, we have constructed a platform for managing ad hoc databases called SQLShare[1] [8] that allows users to upload their data and immediately query it using SQL — no schema design, no reformatting, no DBAs, no obstacles. Queries can be named, saved, shared, and commented on — anything you can do with a YouTube video, you can do with a saved query. Each saved query is also registered as a view and can be referenced by other queries.

[1] `http://escience.washington.edu/sqlshare`

The early response from the initial prototype has been remarkable. At the first demonstration, the results of a simple SQL query written "live" in less than a minute caused a biologist 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 via copy-and-paste. Within a day, the same researcher had derived and saved several new queries of her own. 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.

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 [7], has been remarkably successful. Once the system was seeded with these *starter queries*, the scientists were able to use the examples to derive their own queries and start analyzing data.

We found that translating these English questions into SQL was rather routine [8]. These queries typically involve common idioms of using the data: how to combine two tables, how to retrieve basic data, how to perform an aggregation. The common patterns suggested that the process could be automated, deriving queries directly from the data itself. This automation is the focus of this demonstration.

# 4. METHOD

Our approach to the problem of automatically deriving starter queries for an ad hoc database is to 1) define a set of heuristics that characterize "good" example queries, 2) formalize these heuristics into quantities we can calculate 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, 5) evaluate the model on a holdout test set, and 5) deploy the model in the production SQLShare application. 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 [9] , a schema [12], or user preferences [1].

We explore heuristics for identifying four idioms: union, join, select, and group by. The demonstration uses all four idioms to make decisions. Here, we describe only our model for detecting joins to explain the approach.

**Detecting Joins** To detect join candidates, we derive a scoring function by combining a set of heuristics:

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. More generally, two columns with a high similarity offer evidence in favor of a join[2].

[2] 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.

4. If two columns have the same active domain, and that active domain has high entropy (large numbers of distinct values) then this evidence in favor of a join. Conversely, if both attributes have small entropy, then this is evidence against a join.

Join heuristics 1-4 above all involve reasoning about the union and intersection of the column values and their active domains. For example, heuristic 1 identifies foreign key relationships. Given two columns $x$ and $y$ (modeled as bags), a foreign key relationship relationship exists if $x \subset y$. Heuristic 2 adds the condition that $|x| << |y|$. Heuristic 3 relaxes the strict subset condition and invokes Jaccard similarity with bag semantics: $\frac{|x \cap y|}{|x \cup y|}$. Heuristic 4 sets conditions on the relative sizes of the active domains: $|\pi(x)| << |x|$, where $\pi(x)$ indicates the set derived by removing duplicates from $x$. For example, a fact table has a large cardinality and a dimension table has a low cardinality, but the join key will typically have a highly similar active domain.

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 ad hoc database, we extract each feature in Table 1 for both set and bag semantics.

**Table 1: Features extracted to estimate the joinability**

| Feature | Expression |
|---|---|
| max/min cardinality | $max/min(\|x\|, \|y\|)$ |
| cardinality difference | $abs(\|x\| - \|y\|)$ |
| intersection cardinality | $\|x \cap y\|$ |
| union cardinality | $\|x \cup y\|$ |
| Jaccard similarity | $\frac{\|x \cap y\|}{\|x \cup y\|}$ |

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 the preliminary results used to drive this demonstration, we train the model on the Sloan Digital Sky Survey logs (SDSS) [11], and then evaluate it on a completely unrelated database, the Gene Ontology database (GO) [6]. The underlying model we use is an Alternating Decision Tree (ADTree) [5] 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. Note that although we hypothesize that the relative influence of these features are agnostic with respect to the database used, we of course must still extract the features from each database we wish to study.

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. To sanity check our results, we evaluate our training model in a ten-fold evaluation. In this evaluation, we trained the model on 90% of the data and tested the model on the remaining 10%. The average of ten repetitions achieved a recall of 91.1% and a precision of 91.2%.

For the GO database, we have the database itself and a set of starter queries provided in the GO documentation. Here, we use *the same model learned on the SDSS data* and see if it can be used to predict the sample queries defined by the GO developers. 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 are *consistent across all schemas and all databases.* In this experiment, we find that the model classifies 28 out of 30 joins correctly, achieving 93.3% recall. To measure precision, we tested a random set of 12 join pairs that did not appear in the starter queries. The model classified 11 out of 12 of these candidates correctly, achieving 96.6% 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, the model performed well even on a completly unrelated schema.

# 5. RELATED WORK

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 [9], a well-defined schema [12], or user history and preferences [1]. The concept of dataspaces [4] is relevant to our work; we consider SQLShare an example of a (relational) Dataspace Support Platform. The Octopus project [3] 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 [10, 2]: both problems involve measuring the similarity of columns. However, ther goals and therefore techniques are different: we prioritize coverage of useful query idioms as opposed to finding all semantic matches.

# 6. REFERENCES


[1] 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.

[2] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

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

[4] 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.

[5] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *International Conference on Machine Learning*, 1999.

[6] Gene ontology. http://www.geneontology.org/.

[7] 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.

[8] 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.

[9] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: A context-aware sql autocomplete system. 2011.

[10] J. Madhavan, P. A. Bernstein, and E. Rahm. "generic schema matching with cupid. In *VLDB*, 2001.

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

[12] a. D. X.Yang, C.M.Procopiuc. Summarizing relational databases. *Proc. VLDB Endowment*, 2(1):634£645, 2009.