Automatic Example Queries for Ad Hoc Databases

Bill Howe¹, Garret Cole², Nodira Khoussainova³, Leilani Battle⁴
¹billhowe, ²gbc3, ³nodira, ⁴leiball]@cs.washington.edu, University of Washington, Seattle, WA, USA

“Here is my data. Where do I start?”

Tabular data extracted from files, spreadsheets, DBs, the web
• No schema available
• No query logs available
• No DBAs available
• Unknown relationships, semantics, utility
• Temporary, time-sensitive applications

“Ad Hoc Databases”

An ad hoc database is a collection of tables with unknown relationships gathered to serve a specific, often transient, often urgent, purpose.

Examples of Ad Hoc Databases

• 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 database from lists of addresses of nearby schools, locations of resources (e.g., ambulances), and contact information for emergency workers.
• A consulting business analyst assembles an ad hoc database from a set of spreadsheets provided by management for a short term engagement
• A security analyst assembles an ad hoc database from a set of application trace logs after an attack

Approach

1. Model each operator independently using curated sets of example queries from the web
2. Compose operators to generate a search space of example queries
3. Rank each set using scores derived from configurable patterns called idioms

Q: Are users willing and able to write SQL?
A: Yes! But they need access to high-quality examples (c.f. Gray, Szalay et al. 2005; Howe 2010)

SQLShare: Database-as-a-Service for Ad Hoc Data

1. Streamlined for a single workflow:
2. No DDL; schema inferred from data
3. Views as first-class citizens
4. Unfettered sharing; cloud-hosted
5. Full SQL; no restrictions

http://sqlshare.escience.washington.edu

Modeling Each Operator

Join
Finding: “Good” joins characterized by

- Feature: max/min of column i,
  - Expression: max(min(i))
- Cardinality difference: abs(i - j)
- Intersection cardinality: i \times j
- Jaccard similarity: \frac{ij}{i+j}

We train a decision tree over these features using existing sets of example queries (with > 80% precision and recall)
Build a graph (V,E) where V is the set of tables and E is the set of “good” joins.
Each query is a minimum spanning tree of a connected component of this graph.

Project
Finding: Important attributes appear near the far left or far right of the table.

Select
Finding: Good queries return around 5-10% of the tuples in the table/join.

Group by
Grouping column: a column is selected if it has manageable distinct values.
Aggregates: In most cases, project count(i). If a separate numeric column is also discovered, demonstrate functions sum, min, max, and avg.

Union
Idea: two tables are good candidates for a union if they share sequence of columns with matching data types.
Finds more matches than just considering column name matches.

Evaluating “Good” Queries

What makes a good set of queries? It is application-dependent. e.g. for a DB class, the queries should demonstrate various SQL concepts vs. if user is familiar with SQL but not the schema, better to have queries that refer to important tables or views.

What is an idiom? A function I : Q → [0, 1]. Takes a query and outputs score between 0 and 1. Examples:
- outputs 1 if query includes GROUP BY clause, 0 otherwise
- rewards queries with more joins
- outputs a score based on which important views the query references
- outputs 1 if query demonstrates a self-join, 0 otherwise

How to use idioms to select the set of starter queries?
Represent queries as vectors of idiom scores

<table>
<thead>
<tr>
<th>q₁</th>
<th>l₁(q₁)</th>
<th>...</th>
<th>l₀(q₀)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₂</td>
<td>l₁(q₂)</td>
<td>...</td>
<td>l₀(q₀)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Goals:
- maximize idiom scores
- select diverse set of queries

Score of a set of k queries:

w· \sum_{i=1}^{k} l_i(q_i) + (1 - w)· \sum_{i=1}^{k} d(q_i, q_j)

We use a greedy algorithm by selecting best query first, then iteratively add best additional query given the queries collected up to this point.