SQuID: Semantic Similarity-Aware Query Intent Discovery

Anna Fariha Sheikh Muhammad Sarwar Alexandra Meliou
College of Information and Computer Sciences
University of Massachusetts
Amherst, MA, USA
{afariha,smsarwar,ameli}@cs.umass.edu

ABSTRACT

Recent expansion of database technology demands a convenient framework for non-expert users to explore datasets. Several approaches exist to assist these non-expert users where they can express their query intent by providing example tuples for their intended query output. However, these approaches treat the structural similarity among the example tuples as the only factor specifying query intent and ignore the richer context present in the data. In this demo, we present SQuID, a system for **S**emantic similarity-aware **Qu**ery **I**ntent **D**iscovery. SQuID takes a few example tuples from the user as input, through a simple interface, and consults the database to discover deeper associations among these examples. These data-driven associations reveal the semantic context of the provided examples, allowing SQuID to infer the user's intended query precisely and effectively. SQuID further *explains* its inference, by displaying the discovered semantic context to the user, who can then provide feedback and tune the result. We demonstrate how SQuID can capture even esoteric and complex semantic contexts, alleviating the need for constructing complex SQL queries, while not requiring the user to have any schema or query language knowledge.

1 INTRODUCTION

Accessing data through traditional database systems requires expertise in query languages and knowledge of the database schema — skills that non-expert users typically lack. Query by Example (QBE) systems [4, 6, 9] assist non-expert users through an alternative mechanism: users can provide example tuples as representatives of their expected query output to express their intent [7]. For relational databases, existing QBE systems [3, 8, 9] focus on the structure and data content of the provided examples, but they don't look deeper into the semantic similarity of these examples. As a result, these systems typically miss important context and infer queries that are too general. A few approaches attempt to improve their inference by requiring additional information beyond the example tuples (e.g., labeling proposed tuples as positive or negative [1, 2], providing join paths for example tuples [3], and providing database-result pairs [5, 10]). However this information is often non-trivial and challenging for non-expert users.

id	title	year
100	**Inception**	2010
101	**Interstellar**	2014
102	Titanic	1997
103	Forrest Gump	1994
104	Rambo	2008
105	**The Matrix**	1999
106	Top Gun	1986
107	Terminator 2	1991

(a) Relation: `movie`

movie_id	genre
100	Sci-Fi
100	Thriller
101	Sci-Fi
102	Romance
102	Drama
103	Drama
104	Action
105	Sci-Fi
106	Action
107	Sci-Fi

(b) Relation: `movie_to_genre`

Figure 1: Excerpt of two relations of the IMDb database

Example 1.1. The IMDb database contains information related to movies and television programs, such as cast, production crew, roles, etc.[1] Figure 1 shows an excerpt of two relations from this database: `movie` and `movie_to_genre`. A user of a QBE system over the IMDb database provides the following example tuple set: `{Inception, Interstellar, The Matrix}`. The system looks for candidate queries based only on the structural similarity and data content of these examples, and produces the generic query `Q1`:

```
Q1: SELECT title FROM movie
```

This query is, however, too general. In fact, any set of titles from the `movie` table would lead the QBE system to `Q1`. Nevertheless, there is more specific context in the chosen examples: *they are all Sci-Fi movies*. Even though this semantic context is present in the data (by associating titles with genre information in the `movie_to_genre` relation), existing QBE systems fail to capture it. The more specific query that better represents the semantic similarity among the example tuples is `Q2`:

```
Q2: SELECT title FROM movie, movie_to_genre
    WHERE movie_to_genre.movie_id = movie.id AND
          movie_to_genre.genre = 'Sci-Fi'
```

In this demonstration, we present SQuID, a system that exploits the rich semantic context already present in a dataset, to perform *Semantic similarity-aware Query Intent Discovery*. SQuID derives the semantic similarities in a set of example tuples automatically, and uses this context to infer query intent more specifically and more effectively than existing QBE methods. Automatic discovery of the semantic context, without any additional information, is

[1] http://www.imdb.com

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193548

challenging. Some associations are related to attributes directly linked with the example tuples (e.g., movie year), and are simple to infer. Other associations require joins with other tables (e.g., movie genres), which complicate the candidate queries and increase the search space. And, finally, some associations are esoteric and hard to express precisely with a structured query; we highlight such a challenging case in Example 1.2.

Example 1.2. A user accessing the IMDb database using SQuID, provides the following set of example tuples: {Adam Sandler, Eddie Murphy, Jim Carrey}. A human with basic awareness of the works of these actors would easily identify them as comedians; in fact, these actors appear in most "Funny Actors" lists[2]. However, there is no explicit attribute in the IMDb database that identifies these actors as "comedians" or "funny"; expressing a query to retrieve all "funny actors" is not straightforward, even for someone with knowledge of the schema and query language expertise.

Nevertheless, relevant information exists in the database, though the semantic context is now *implicit*, derived through deeper associations with other entities. In this case, SQuID makes connections with several other relations (castinfo, movie, movie_to_genre, and genre) and automatically discovers a trend in the number of Comedy movies that each of the example actors has appeared in. This semantic context is complex, as the similarity among the actors is based on an aggregate trend of a different entity (movies each actor appeared in). SQuID automatically derives and explores explicit and implicit semantic properties and contrasts them with overall statistics of the data to infer the most probable semantic context for the provided examples.

In our demonstration, participants will observe how SQuID captures the semantic similarity of example tuples through a scenario based on Example 1.2. We proceed to discuss, at a high level, the inner workings of SQuID and its architecture, and we conclude with a detailed outline of our demonstration.

2 SOLUTION SKETCH

In this section, we use the IMDb dataset as our running example to describe, at a high-level, how SQuID works. SQuID assumes that queries focus on the *entities* of a dataset; in our running example, these entities are person and movie. The dataset contains information on semantic properties for these entities (e.g., the year of a movie). *Basic semantic properties* are properties that are associated with an entity directly, even if the association requires joins across tables (e.g., the genre of a movie). *Derived semantic properties* of an entity are aggregates over basic semantic properties of an associated entity. For example, genre is a basic semantic property of movie; the total number of comedy movies is a derived semantic property of person. Figure 2 shows example basic and derived semantic properties for person entities. We call the value of a derived semantic property its *association strength* to indicate how strongly an entity is associated with a property. For example, Adam Sandler is strongly associated with comedies.

SQuID models queries as a collection of basic and derived *filters* over the entities' basic and derived semantic properties. We denote a filter with the notation $\phi_{P,V}$, where P stands for a semantic property and V is a set or range of allowed values for P. The filter $\phi_{P,V}$ retrieves tuples from a relation that satisfy the filter condition $P \in V$. For example, Adam Sandler passes the filter $\phi_{\texttt{country},\{\texttt{USA}\}}$ but Hugh Jackman does not.

[2]For example: http://www.imdb.com/list/ls051583078

	Basic property		Derived property	
name	gender	country	#comedy	#drama
Adam Sandler	M	USA	**114**	36
Al Pacino	M	USA	11	38
Hugh Jackman	M	Australia	12	10
Jim Carrey	M	Canada	68	21
Jonah Hill	M	USA	**60**	13
Leonardo DiCaprio	M	USA	9	35
Tom Cruise	M	USA	13	12

Figure 2: Entities with computed semantic properties

Case A		Case B	
$\phi_{\#\texttt{comedy}}$	$[30,\infty)$	$\phi_{\#\texttt{comedy}}$	$[12,\infty)$
$\phi_{\#\texttt{drama}}$	$[25,\infty)$	$\phi_{\#\texttt{drama}}$	$[10,\infty)$
$\phi_{\#\texttt{thriller}}$	$[3,\infty)$	$\phi_{\#\texttt{thriller}}$	$[10,\infty)$
$\phi_{\#\texttt{action}}$	$[2,\infty)$	$\phi_{\#\texttt{action}}$	$[9,\infty)$
$\phi_{\#\texttt{mystery}}$	$[1,\infty)$	$\phi_{\#\texttt{mystery}}$	$[9,\infty)$

Figure 3: Two cases for derived semantic property filters

Given a set of semantic properties, SQuID creates filters based on the values of the user's examples for each property. For example, given the properties of Figure 2 and the set {Adam Sandler, Jonah Hill} as example tuples, SQuID will create the filters $\phi_{\texttt{gender},\{\texttt{M}\}}$, $\phi_{\texttt{country},\{\texttt{USA}\}}$, $\phi_{\#\texttt{comedy},[60,114]}$, and $\phi_{\#\texttt{drama},[13,36]}$. SQuID then identifies the filters that better distinguish the chosen examples from the rest of the data, using the notion of *selectivity*. Roughly, a filter that many entities satisfy has low selectivity; this is the case for $\phi_{\texttt{gender},\{\texttt{M}\}}$, $\phi_{\texttt{country},\{\texttt{USA}\}}$, and $\phi_{\#\texttt{drama},[13,36]}$. In contrast, $\phi_{\#\texttt{comedy},[60,114]}$ is the most selective of the filters, as only 3 of the example entities satisfy it. Finally, SQuID constructs the query using the selected filters as predicates.

Offline Preprocessing: Many of the basic semantic properties of an entity are readily available within a single relation, but others require joins, and derived properties require more complex queries and aggregations. While these can be expensive operations, they only need to be computed once. SQuID's offline precomputation module builds relations similar to the one in Figure 2, along with an inverted column index for fast reverse look-up of example tuples, database statistics, and selectivity information. These allow SQuID to achieve real-time performance.

Filter Pruning and Tuning: SQuID employs four main heuristics to refine and prune filters: (1) It sets a selectivity threshold to reject filters that are too general. (2) It sets a threshold on the association strength to reject filters that map to loose associations of the example tuples with a particular property. (3) It relaxes filter bounds (e.g., transforms $\phi_{\#\texttt{comedy},[60,114]}$ to $\phi_{\#\texttt{comedy},[60,\infty)}$); the initial bounds, derived from the values of the examples, are in a way arbitrary and loosening them better represents the target trends. (4) It groups derived filters of the same attribute (e.g., genre) and

evaluates the skewness of the distribution of the filters' values. A distribution that is close to uniform (e.g., Case B in Figure 3) means that, intuitively, this property is not significant. In contrast, if some filters of the set are outliers, this indicates a significant property of the example set (e.g., the top two filters of Case A in Figure 3).

3 THE SQUID ARCHITECTURE

Figure 4 illustrates the architecture of SQuID. The system consists of two main modules: *offline precomputation* and *online intent discovery*. The offline precomputation module uses information about the database schema and admin-provided metadata to compute derived relations, semantic properties, and related statistics. The function of the online intent discovery module is to discover the query intent from the user-provided example tuples, provide explanations for the discovered query intent, and translate it to a SQL query. This module consults the augmented database and the precomputed semantic property relations to identify semantic similarities among the example tuples. The module then uses the precomputed statistics to infer the most likely query intent and displays the chosen filters as an explanation to the user. The user can provide feedback directly on the explanation to tune the predicted query. SQuID executes the final SQL query to generate the result tuples and presents them to the user.

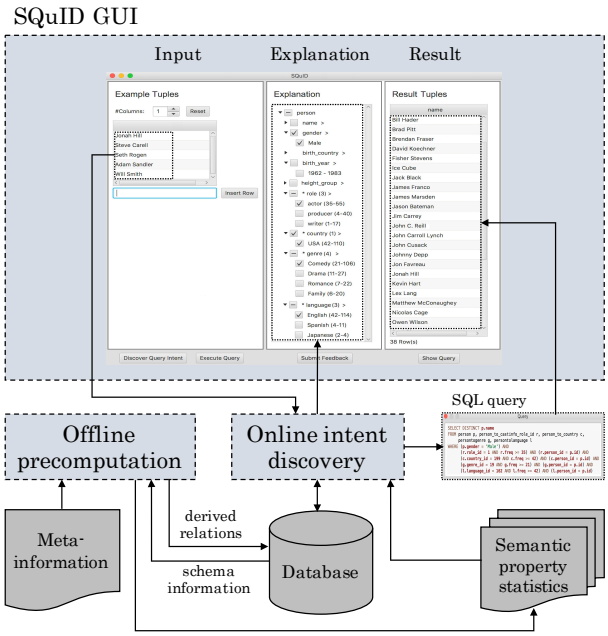

Figure 4: SQuID architecture

4 DEMONSTRATION OUTLINE

We will demonstrate the effectiveness of SQuID in query intent discovery using the real-world IMDb dataset. We expect that most participants will be familiar with the domain of the data (actors, directors, and films), allowing them to interact with the system more freely. We aim to show that SQuID can understand the user-provided example tuples semantically and infer the query intent insightfully. Moreover, we will demonstrate how SQuID improves its prediction with additional and more diverse examples.

4.1 Online Intent Discovery

The online intent discovery module of SQuID presents a simple interface consisting of three panels. Figure 5 shows screenshot of the system. The leftmost panel is the *Input Panel* where the user provides the example tuples. The center panel is the *Explanation Panel*, which is responsible for explaining the query intent that SQuID discovers. The rightmost panel is the *Result Panel*, which presents the result tuples to the user after executing the query corresponding to the predicted intent. Our demonstration will guide the participants through the scenario of Example 1.2 (search for funny actors) through five steps. We have annotated each step with a circle in Figure 5.

Step 1 (Providing example tuples): In the input panel, first the user specifies the number of columns in the example table. For our scenario, we only need one column. The user then enters, one row at a time, three examples of contemporary comedians. For our guided scenario, we choose `Jonah Hill`, `Steve Carell`, and `Adam Sandler` (Figure 5).

Step 2 (Intent discovery): Once the user completes entering the list of example tuples, she can submit a request to SQuID for query intent discovery. SQuID then presents the *basic* and *derived* semantic property filters that it deems significant in the explanation panel. SQuID infers the best query intent by pre-selecting the most likely semantic similarities. The explanation panel of Figure 5 illustrates the selected semantic similarities observed in the example tuples. Just from the three examples, the system is able to identify the semantic similarity of the actors' being comedians, reflected in the derived semantic property filter $\phi_{\text{genre}^{\text{comedy}},[40,114]}$; this is the only genre-related filter that SQuID selects. SQuID also identifies further similarities, reflected in the other selected semantic properties. Since all of the example tuples are male, contemporary Hollywood actors, the system also selects the property filters: $\phi_{\texttt{gender},\{\texttt{Male}\}}$, $\phi_{\texttt{birth\_country},\{\texttt{USA}\}}$, and $\phi_{\texttt{birth\_year},\{1962-1983\}}$.

Step 3 (Explanation feedback): SQuID's explanation panel allows users to easily reason about the inferred query and provide feedback by selecting or de-selecting filters. Naturally, users have knowledge biases, leading them to provide biased samples of their intent. In this case, the user is likely more familiar with American comedians; since the user's examples demonstrate this bias, SQuID identified it as a similarity, even though it was not part of the query intent. The user can easily note and correct this directly on the explanation panel by unchecking the box that constrains the birth country to `USA`.

Step 4 (Additional example): Biases in the provided examples can affect SQuID's inference. While the user can correct for these incorrect inferences due to bias through explanation feedback, we will demonstrate how diversifying the example set can improve SQuID's results. Our initial examples of funny actors happen to include individuals of similar height. SQuID detects this as a similarity and pre-selects the property of the specific height group. Adding `Vince Vaughn` to the list of examples, provides a more diverse example set, allowing SQuID to automatically detect that

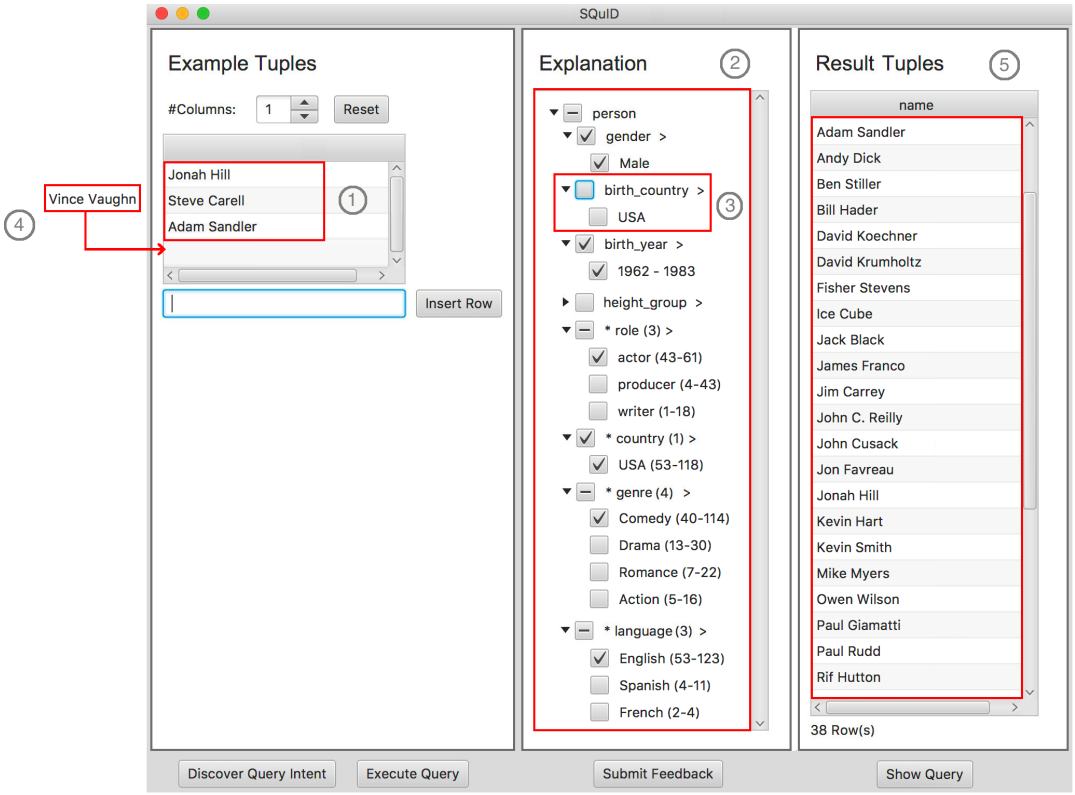

Figure 5: The SQuID demo: input, explanation, and result panels

```
Query
SELECT DISTINCT p.name
FROM person p, person_to_castinfo_role r, perosn_to_country c,
     person_to_genre g, person_to_language l
WHERE (p.gender = 'Male')  AND (p.birth_year >= 1962 AND p.birth_year <= 1983)
      (r.person_id = p.id) AND (r.role = 'actor' AND r.freq >= 43) AND
      (c.person_id = p.id) AND (c.country = 'USA' AND c.freq >= 53) AND
      (g.person_id = p.id) AND (g.genre = 'Comedy' AND g.freq >= 40) AND
      (l.person_id = p.id) AND (l.language = 'English' AND l.freq >= 53)
```

Figure 6: SQL query predicted by SQuID to capture intent

the height filter is not significant. (Figure 5 displays the explanation panel after the addition of the new example.)

Step 5 (Result retrieval): Our demonstration participants will be able to examine the query that SQuID generates (Figure 6), and execute and observe its result over the IMDb data. The result panel of Figure 5 presents the result set, which includes well-known and popular comedians such as `Jim Carrey`, `Ben Stiller`, and `Owen Wilson`.

Demo engagement: After our guided demonstration, participants will have the option to use and test SQuID with their own example tuples. IMDb is a rich dataset that includes information about 6 million artists and 1 million movies along with semantic properties, such as `nationality` of artists and `genre` of movies. We expect that the broad familiarity and appeal of the data domain will allow participants to engage with ease and little assistance.

Through this demo scenario, we will showcase how SQuID is effective at capturing the query intent by exploiting the semantic similarity with very few user-provided example tuples. We will also demonstrate how quickly SQuID's inference improves with the addition of examples or explanation feedback. One of the key takeaways that the conclusion of our demonstration will highlight, is that inferring semantic similarity among examples tuples can assist in expressing intent that may be obscure and vague, facilitating data retrieval for non-expert and expert users alike.

Acknowledgements. This material is based upon work supported by the National Science Foundation under grants IIS-1421322 and IIS-1453543.

REFERENCES

[1] A. Bonifati, R. Ciucanu, and S. Staworko. 2014. Interactive Inference of Join Queries. In *EDBT*. 451–462.
[2] A. Bonifati, R. Ciucanu, and S. Staworko. 2016. Learning Join Queries from User Examples. *ACM Trans. Database Syst.* 40, 4 (2016), 24:1–24:38.
[3] D. Deutch and A. Gilad. 2016. QPlain: Query by explanation. In *ICDE*. 1358–1361.
[4] G Diaz, M Arenas, and M Benedikt. 2016. SPARQLByE: Querying RDF Data by Example. *Proc. VLDB Endow.* 9, 13 (Sept. 2016), 1533–1536.
[5] H. Li, C. Chan, and D. Maier. 2015. Query from Examples: An Iterative, Data-driven Approach to Query Construction. *Proc. VLDB Endow.* 8, 13 (Sept. 2015), 2158–2169.
[6] S. Metzger, R. Schenkel, and M. Sydow. 2017. QBEES: query-by-example entity search in semantic knowledge graphs based on maximal aspects, diversity-awareness and relaxation. *J. Intell. Inf. Syst.* 49, 3 (2017), 333–366.
[7] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. 2014. Exemplar Queries: Give Me an Example of What You Need. *Proc. VLDB Endow.* 7, 5 (Jan. 2014), 365–376.
[8] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri. 2015. S4: Top-k Spreadsheet-Style Search for Query Discovery. In *SIGMOD*. ACM, 2001–2016.

[9] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. 2014. Discovering Queries Based on Example Tuples. In *SIGMOD*. ACM, 493–504.
[10] C Wang, A Cheung, and R Bodik. 2017. Interactive Query Synthesis from Input-Output Examples. In *SIGMOD*. ACM, 1631–1634.